Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB University Enterprises Cooperation
In Game Industry In Ukraine

ПРОЕКТ ЕРАЗМУС+
GAMEHUB: «СПІВРОБІТНИЦТВО МІЖ УНІВЕРСИТЕТАМИ
ТА ПІДПРИЄМСТВАМИ В СФЕРІ ГРАЛЬНОЇ ІНДУСТРІЇ В УКРАЇНІ»

ERASMUS+ PROJECT
GAMEHUB: «UNIVERSITY-ENTERPRISES COOPERATION
IN GAME INDUSTRY IN UKRAINE»

# МОДЕЛІ ТА МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ У КОМП'ЮТЕРНИХ ІГРАХ

Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB University Enterprises Cooperation In Game Industry In Ukraine

GAMEHUB: «СПІВРОБІТНИЦТВО МІЖ УНІВЕРСИТЕТАМИ
ТА ПІДПРИЄМСТВАМИ В СФЕРІ ГРАЛЬНОЇ ІНДУСТРІЇ В УКРАЇНІ»

# МОДЕЛІ ТА МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ У КОМП'ЮТЕРНИХ ІГРАХ

Харків
«Друкарня Мадрид»
2018

УДК 004.9
ББК 32.973-018.2
в6
Н62

Проект ЕРАЗМУС+
GameHub: «Співробітництво між університетами та підприємствами
в сфері гральної індустрії в Україні»
№ 561728-EPP-1-2015-1- ES-EPPKA2-CBHE-JP

**Нікітіна Л.О.**

Н 62 Моделі та методи штучного інтелекту у комп'ютерних іграх. / Л.О. Нікітіна, С. О. Нікітін. - Х.: «Друкарня Мадрид», 2018. - 102 с.

ISBN 978-617-7683-02-4

Довідник модуля «Моделі та методи штучного інтелекту у комп'ютерних іграх» написаний згідно з останніми освітньо - кваліфікаційними вимогами до підготовки магістрів зі спеціальності «Інформаційні технології». З урахуванням кредитно-модульної системи навчальна дисципліна формується як система змістовних модулів. Довідник модуля складається з інформації стосовно модуля та критеріїв оцінювання, лекційних та лабораторних робіт. Призначено для студентів всіх спеціальностей вузів, викладачів курсу «Моделі та методи штучного інтелекту у комп'ютерних іграх», фахівців у галузі «Інформаційні технології», які підвищують кваліфікацію.

Бібліографічні описи здійснено відповідно до існуючих державних та міждержавних стандартів: ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».

Manual of the "Models and methods of artificial intelligence in computer games" module is written in accordance with the latest educational qualification requirements for the preparation of masters in the "Information Technologies" specialty. Taking into account the credit-module system, the discipline is formed as a system of content modules. The module manual consists of information on the module and criteria for evaluation, lecture and laboratory work. It is intended for students of all specialties of universities, lecturers of the "Models and Methods of Artificial Intelligence in Computer Games" course, specialists in the field of "Information Technologies", who enhance their qualification.

This publication has been funded with support from the European Union. The publication reflects the views only of the authors, and the Union cannot be held responsible for any use which may be made of the information contained therein.

УДК 004.9
ББК 32.973-018.2



ISBN 978-617-7683-02-4

ЗМІСТ

## ДОВІДНИК МОДУЛЯ



## ЛЕКЦІЇ



## ЛАБОРАТОРНІ РОБОТИ

# Довідник модуля

# 1 ВСТУП

Предметом навчальної дисципліни є вивчення теоретичних основ застосування моделей та методів штучного інтелекту (ШІ) у компонентах ігрових додатків; оволодіння методологією пошуку рішень інтелектуальних задач (ІЗ) та моделей поведінки програмних агентів; придбання практичних навичок створення програмного забезпечення для імплементації моделей та методів ШІ у ігрових додатках.

## МЕТА ДИСЦИПЛІНИ

Метою викладання навчальної дисципліни «Моделі та методи штучного інтелекту» є формування у студентів цілісної системи знань про інтелектуальні системи (ІС), технології та методи їх розробки та програмної імплементації.

## ОЧІКУВАНІ РЕЗУЛЬТАТИ

Основними завданнями вивчення дисципліни «Моделі та методи штучного інтелекту» є набуття знань про сучасні підходи до класифікації та побудови ІС, моделі, методи та алгоритми роботи ІС та їх компонентів. Важливе місце посідають знання про застосування моделей та методів ШІ у ігрових додатках різних категорій та класів, а саме, концепції поведінки ігрових персонажів та об'єктів і пошук рішень.

**Змістовний модуль** «Моделі та методи штучного інтелекту у комп'ютерних іграх» дисципліни «Моделі та методи штучного інтелекту» орієнтовний на оволодіння студентами практичних навичок використання теоретичних знань алгоритмів, моделей та методів пошуку рішень у просторах задач та станів ігрового середовища, формального опису об'єктів та станів та подання його у програмному забезпеченні.

Модуль має метою:

- забезпечити знання моделей представлення знань та алгоритмів отримання логічних виведень та вміння використовувати їх при побудові сучасних програмних ігрових систем.
- отримання студентами відомостей про сучасні системи штучного інтелекту;
- вивчення методів формалізації знань;
- отримання знань про сучасні технології та засоби розробки ігрових систем з штучним інтелектом;
- отримання умінь та навичок розробки та програмної реалізації компонентів ігрових систем з штучним інтелектом.

# 2 ОПИС МОДУЛЯ

Галузь знань: 12 "Інформаційні технології".
Спеціальність: 122 "Комп'ютерні науки та інформаційні технології"
Рівень: бакалавр.
Назва дисципліни: Методи та системи штучного інтелекту.
Назва змістовного модуля: Моделі та методи штучного інтелекту у комп'ютерних іграх
Семестр: **7**
Кількість кредитних одиниць: дисципліна - 4,0, модуль - 1,0
Орієнтовна кількість годин: дисципліна - 120, модуль - 30
Викладач: доцент, к.т.н. Нікітіна Л.О.

# 3 ПЕРЕЛІК КОМПЕТЕНТНОСТЕЙ ТА РЕЗУЛЬТАТИ НАВЧАННЯ[1]

## СПЕЦІАЛЬНІ (ФАХОВІ) КОМПЕТЕНТНОСТІ

| | |
|---|---|
| ПК-5 | Здатність до системного мислення, застосування методології системного аналізу для дослідження складних проблем різної природи, методів формалізації та розв'язанні системних задач, що мають суперечливі цілі, невизначеності та ризики. |
| ПК-6 | Здатність застосовувати теоретичні та практичні основи методології та технології моделювання, реалізовувати алгоритми моделювання для дослідження характеристик і поведінки складних об'єктів і систем, проводити експерименти за програмою моделювання з обробкою й аналізом результатів. |
| ПКс8-1 | Здатність до математичного та логічного мислення, формулювання та досліджування математичних моделей, зокрема дискретних математичних моделей, обґрунтовування вибору методів і підходів для розв'язування теоретичних і прикладних задач в галузі комп'ютерних наук, інтерпретування отриманих результатів. |
| ПКс8-2 | Здатність до побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проектування, розроблення та аналізу алгоритмів, оцінювання їх ефективності та складності, розв'язності та нерозв'язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем. |

[1] Профіль програми освітнього ступенів бакалавра. Спеціальність 122 Комп'ютерні науки та інформаційні технології, спеціалізація 122-08 Системи штучного інтелекту

## ПРОГРАМНІ РЕЗУЛЬТАТИ НАВЧАННЯ

## ПРОФЕСІЙНА ПІДГОТОВКА

| | |
|---|---|
| РН-5 | Знання методології системного аналізу для системного дослідження детермінованих та стохастичних моделей об'єктів і процесів, проектування та експлуатації інформаційних систем, продуктів, сервісів інформаційних технологій, інших об'єктів професійної діяльності. |
| РН-6 | Знання моделей систем масового обслуговування, мереж Петрі; методології ймовірнісного та імітаційного моделювання об'єктів, процесів і систем; планування та проведення експериментів з моделями, прийняття рішень щодо досягнення мети за результатами моделювання. |
| ПКс8-1 | Знання теоретичних і прикладних положень неперервного та дискретного аналізу, включаючи аналіз нескінченно малих, інтегральне числення, лінійну алгебру, аналітичну геометрію, диференційні рівняння, функціональний<br>аналіз, комбінаторику, теорію графів, больову алгебру. |
| ПКс8-2 | Знання базових понять теорії алгоритмів, формальних моделей алгоритмів, примітивно рекурсивних, загально-рекурсивних та частково-рекурсивних функцій, питань обчислюваності, розв'язності та нерозв'язності масових проблем, понять часової та просторової складності алгоритмів при розв'язанні обчислювальних задач. |

## ПЕРЕЛІК КОМПЕТЕНТНОСТЕЙ ТА РЕЗУЛЬТАТИ НАВЧАННЯ МОДУЛЬ «МОДЕЛІ ТА МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ У КОМП'ЮТЕРНИХ ІГРАХ»

| | |
|---|---|
| Спеціальні (фахові) компетентності | ПК-5, ПК-6, ПКс8-1, ПКс8-2 |
| Професійна підготовка | РН-5, РН-6, ПКс8-1, ПКс8-2 |

# 4 МІЖДИСЦИПЛІНАРНІ ЗВ'ЯЗКИ

Для засвоєння матеріалу використовується такий перелік забезпечуючих дисциплін:

Алгоритмізація та програмування;

Алгоритми та структури даних;

Об'єктно-орієнтоване програмування;

Технологія створення програмних продуктів;

Мови програмування систем штучного інтелекту.

# 5 МЕТА ТА ПЕРЕДБАЧУВАНІ РЕЗУЛЬТАТИ ВИВЧАННЯ МОДУЛЯ

## 5.1 МЕТА МОДУЛЯ

Мета модуля – формування у студентів системи знань про ІС, навчання студентів способам подання інтелектуальних задач (ІЗ), методам та алгоритмам пошуку рішень, застосування набутих знань та практичних навичок у комп'ютерних ігрових додатках.

## 5.2 РЕЗУЛЬТАТИ НАВЧАННЯ

### ЗНАННЯ ТА ЇХ ВИКОРИСТАННЯ

У разі успішного оволодіння матеріалами модуля студент буде вміти використовувати сучасну теорію побудови ІС та їх компонентів, методи та алгоритми їх розробки. Отримані знання моделей представлення знань, алгоритмів отримання логічних виведень дадуть змогу студентам будувати сучасні програмні ігрові системи.

### ДОСЛІДНИЦЬКІ НАВИЧКИ

У разі успішного вивчення модуля студент буде вміти аналізувати предметну область, виконувати постановку задач та пропонувати варіанти їх рішення при створенні програмних ігрових додатків, оцінювати складність розроблених рішень.

### СПЕЦІАЛЬНІ ВМІННЯ

У разі успішного вивчення модуля студент буде вміти:

- визначати підходи та методи для створення інтелектуальних компонентів комп'ютерних ігор;
- використовувати методи, моделі та алгоритми для створення програмних ігрових додатків;
- брати участь у розробці комп'ютерних ігрових додатків.

### СОЦІАЛЬНІ ВМІННЯ

У разі успішного вивчення модуля студент буде вміти брати участь у командній роботі: обговорювати в групі шляхи побудови комп'ютерних ігрових додатків.

### ОСОБИСТІ ЯКОСТІ

У разі успішного вивчення модуля студент буде вміти:

- самостійно вирішувати питання, що стосуються побудови та створення комп'ютерних ігрових додатків;
- аналізувати сучасну науково-технічну літературу з питань створення комп'ютерних ігрових додатків.

# 6 КАЛЕНДАРНИЙ ПЛАН СЕМЕСТРУ І СТРУКТУРА МОДУЛЯ

## ІНФОРМАЦІЙНЕ НАПОВНЕННЯ ЗМІСТОВНОГО МОДУЛЯ

| Номер | Номер тижня | Тема та зміст модуля |
|---|---|---|
| 1 | 3 | Створення штучного інтелекту для ігор.<br>Мета та задачі модуля. Означення основних понять «комп'ютерна гра», «ігровий персонаж», «ігровий штучний інтелект», «інтелектуальна поведінка» та ін. Призначення та сфери застосування систем штучного інтелекту (СШІ) у програмних ігрових додатках.<br>Подання інтелектуальної задачі. |
| 2 | 4 | Способи подання ІЗ, алгоритми та методи пошуку рішень.<br>Поняття інтелектуальної задачі (ІЗ). Підходи до подання задач у інтелектуальних системах. Характеристики стратегій та методів пошуку рішення ІЗ.<br>Подання ІЗ у програмній системі. Виконання пошуку рішень у просторі станів. |
| 3 | 5-6 | Методи пошуку рішень ІЗ у просторі станів.<br>Поняття теорії графів, процесами пошуку на графі. Методи повного перебору, перебору в «глибину», «ширину». Можливості використання оцінних функцій, оптимальних алгоритмів перебору та евристик при пошуку рішень.<br>Подання ІЗ у програмній системі. Виконання пошуку рішень у просторі задач. |
| 4 | 7-8 | Методи пошуку рішень ІЗ у просторі задач.<br>Поняття «І-АБО» графів, дерев рішень. Методи пошуку задач у разі зведення задач до сукупності підзадач. |

# 7 ФОРМИ НАВЧАННЯ

Навчальне навантаження модуля складається з аудиторної та самостійної роботи. Аудиторна робота включає 6 лекцій та 3 лабораторних роботи. Самостійна робота студентів передбачає підготовку до аудиторних занять і лабораторних робіт, а також підготовку до тесту та оформлення залікового звіту з лабораторних робіт.

Підготовкою до поточних аудиторних занять є аналіз літератури, інтернет-матеріалів за темами лекцій і лабораторних робіт, підготовка до тестів.

Контактні години передбачають індивідуальні консультації та контроль студентів в онлайн режимі.

Заліковий звіт – опис та презентація за результатами виконання лабораторних робіт.

# 8 ПОРЯДОК ПРОВЕДЕННЯ АТЕСТАЦІЇ

Загальний принцип оцінювання підсумкових знань студента з курсу "Моделі та методи штучного інтелекту" полягає в опитуванні студентів на лекціях, оцінці поточної практичної роботи студентів у навчальному семестрі на лабораторних роботах та оцінки контрольного заходу у формі іспиту, у результаті яких студент отримує сумарну оцінку в балах.

За результатами вивчення кожного змістовного модуля передбачено оцінка виконання залікового завдання. Оцінка включає: результати тестування студентів на лекціях, результати поточного опитування при виконанні лабораторних робіт модуля, оцінку за виконання залікового завдання. Сумарно оцінка кожного змістовного модуля складає до 10 або до 20 балів. Максимальна оцінка іспиту 40 балів.

## ОЦІНКА РЕЗУЛЬТАТІВ ВИВЧЕННЯ ДИСЦИПЛІНИ В ЦІЛОМУ

| **Поточне тестування** | **Балів** |
|---|---|
| Змістовний модуль 1 | до 10 |
| Змістовний модуль 2 | до 10 |
| Змістовний модуль 3 | до 20 |
| Змістовний модуль 4 | до 10 |
| Змістовний модуль 5 | до 10 |
| Іспит | до 40 |
| Разом | До 100 |

## ГРАФІК ПРОВЕДЕННЯ ПОТОЧНОГО ОЦІНЮВАННЯ МОДУЛЯ

| **Номер тижня** | **Оцінювання** |
|---|---|
| 4 | Оцінка виконання лабораторної роботи 1 |
| 5-6 | Оцінка виконання лабораторної роботи 2 |
| 7-8 | Оцінка виконання лабораторної роботи 3 |
| 9 | Оцінка залікового звіту |

## ПРЕДСТАВЛЕННЯ ЗВІТУ ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

При представленні звіту з лабораторних робіт необхідно виконувати наступне. При умові виконання кожної окремої лабораторної роботи єдиний звіт надається студентом на 15 тижні семестру. Продовження терміну можливе лише при наявності поважної причини, передбаченої порядком навчання студентів у вищій школі. При цьому:

електронна версія єдиного звіту за результатами проходження модуля надається викладачеві через систему Інтернет на 3, 8, 14, 15 тижні;

за кожний день відтермінування представлення та здачі єдиного звіту по модулю знімається 1 бал (не більш ніж 5 днів – 5 балів).

## МЕТОД ОЦІНКИ ЗМІСТОВНОГО МОДУЛЯ

Кількість балів в загальній оцінці змістовного модуля відповідає наступному:

| | |
|---|---|
| Тестування з теми 1 | максимально 1 бал. |
| Тестування з теми 2 | максимально 1 бал. |
| Виконання лабораторної роботи 1 | максимально 5 балів. |
| Тестування з теми 3 | максимально 1 бал. |
| Виконання лабораторної роботи 2 | максимально 5 балів. |
| Тестування з теми 4 | максимально 1 бал. |
| Виконання лабораторної роботи 3 | максимально 5 балів. |
| Оцінка залікового звіту | максимально 1 бал. |

Усі набрані бали підсумовуються (максимально 20 балів), штрафні бали за запізнення в представленні єдиного звіту (максимально мінус 5 балів) віднімаються. Сумарна оцінка (від 0 до 20 балів) є індивідуальна оцінка студента освоєння змістовного модуля.

## МЕТОД ОЦІНКИ ДИСЦИПЛІНИ В ЦІЛОМУ

Оцінки студентів за результатами вивчення змістовних модулів 1 – 4 підсумовуються. Оцінка іспиту (максимально 40 балів) додається. Таким чином розраховується сумарна оцінка студента в балах за дисципліною.

Сумарна оцінка в балах переводиться за нижченаведеною шкалою оцінювання в національну та ЄКТС- оцінку:

| Сума балів за всі види навчальної діяльності | Оцінка ECTS | Оцінка за національною шкалою |
|---|---|---|
| 90 – 100 | A | Відмінно |
| 82 – 89 | B | Дуже добре |
| 74 – 81 | C | Добре |
| 64 – 73 | D | Задовільно |
| 60 – 63 | E | Посередньо |
| 35 – 59 | FX | Не зараховано з можливістю повторного складання |
| 0 – 34 | F | Не зараховано з обов'язковим повторним вивченням дисципліни |

## КРИТЕРІЇ ОЦІНЮВАННЯ ПРЕЗЕНТАЦІЇ РЕЗУЛЬТАТІВ РОБОТИ

| | Високий | Середній | | Низький |
|---|---|---|---|---|
| | 4 | 3 | 2 | 1 |
| організація | Організаційна структура (конкретне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) чітко і послідовно спостерігалася, вміло представлена і робить зміст презентації цілісним. | Організаційна структура (конкретне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) чітко і послідовно проглядається в презентації. | Організаційна структура (специфічне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) періодично проглядається в презентації. | Організаційна структура (специфічне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) не спостерігається в презентації. |
| Мова | Вибір мови є вражаючим, що запам'ятовується, є привабливим і підвищує ефективність презентації. Мова в презентації підходить аудиторії. | Вибір мови є продуманим і в цілому підтримує ефективність презентації. Мова в презентації підходить аудиторії. | Вибір мови є повсякденним і звичайним і частково підтримує ефектність презентації. Мова в презентації підходить аудиторії. | Вибір мови незрозумілий і мінімально підтримує ефективність презентації. Мова в презентації не підходить аудиторії. |
| Подача | Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію переконливою, а доповідач виглядає підготовленим, впевненим. | Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію цікавою, і динаміка доповідача виглядає впевнено. | Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію зрозумілою, а доповідач виглядає не дуже впевнено. | Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію незрозумілою, а доповідь виглядає невпевнено. |
| Допоміжні матеріали | Різноманітні типи допоміжних матеріалів (пояснення, приклади, ілюстрації, статистика, аналоги, цитати з відповідних джерел) дають посилання на інформацію або аналіз, які в значній мірі підтримують презентацію і підтверджують авторитет доповідача в представленій темі. | Допоміжні матеріали (пояснення, приклади, ілюстрації, статистика, аналогії, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які в цілому підтримують презентацію і підтверджують авторитет доповідача в представленій темі. | Допоміжні матеріали (пояснення, приклади, ілюстрації, статистичні дані, аналоги, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які частково підтримують презентацію і авторитет доповідача в представленій темі. | Недостатні допоміжні матеріали (пояснення, приклади, ілюстрації, статистика, аналоги, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які мінімально підтримують презентацію і авторитет доповідача в представленій темі. |

| | Високий | Середній | | Низький |
|---|---|---|---|---|
| | 4 | 3 | 2 | 1 |
| Головна думка | Головна думка є переконливою (точно сформульована, відповідним чином повторюється, запам'ятовується і міцно підтримується). | Головна думка є ясною і узгоджується з допоміжним матеріалом. | Головна думка є в принципі зрозумілою, але не часто повторюється і не запам'ятовується. | Головна думка може бути виведена, але явно не вказана в презентації. |

# 9 ЗВОРОТНИЙ ЗВ'ЯЗОК

Інформація щодо результатів тестування, виконання лабораторних робіт та загальна оцінка змістовного модуля надається кожному студенту як індивідуально, так і групі в цілому.

Інформація щодо результатів тестування надається студентам по завершенню тестування. Загальні результаті надаються на 16 тижні навчання.

Інформація щодо оцінки виконання лабораторної роботи надається студентові під час заняття.

Інформація щодо оцінки змістовного модуля в цілому надається студентам на 2 тижні навчання.

Контактні дані для on-line допомоги та консультування:

**Викладач:** доцент, к.т.н. Нікітіна Л.О., e-mail: nikitinalud@gmail.com

# 10 ВИКЛАДАЦЬКИЙ СКЛАД ТА ДОПОМІЖНІ ДЖЕРЕЛА

## ОБОВ'ЯЗКИ ВИКЛАДАЧІВ

- подача матеріалів модуля згідно з програмою;
- оцінка результатів тестування та виконання лабораторних робіт.

## ОБОВ'ЯЗКИ КООРДИНАТОРА ДИСЦИПЛІНИ

- планування та внесення змін до модуля;
- координація і управління професорсько-викладацьким складом;
- координація проведення тестування, лабораторних робіт та іспиту.

### ОБОВ'ЯЗКИ ДОПОМІЖНОГО ПЕРСОНАЛУ

Допоміжний персонал здійснює підготовку комп'ютерної техніки до виконання лабораторних робіт студентами та надає технічну підтримку студентів під час виконання лабораторних робіт.

### КОНТАКТНІ ДАНІ ВИКЛАДАЧІВ

доцент, к.т.н. Нікітіна Л.О., e-mail: nikitinalud@gmail.com

### КОНТАКТНІ ДАНІ КУРАТОРА

доцент, к.т.н. Касілов О.В., e-mail: o.kasilov@gmail.com

## 11 НАВЧАЛЬНА ПРОГРАМА І МАТЕРІАЛИ

### 11.1 ТЕМА 1: ВСТУП ДО МОДУЛЯ. ОСНОВНІ ПОНЯТТЯ ТА ОЗНАЧЕННЯ

#### АНОТАЦІЯ

Лекція знайомить з метою та задачами модуля. Даються означення основних понять «комп'ютерна гра», «ігровий персонаж», «ігровий штучний інтелект», «інтелектуальна поведінка» та ін. Наводиться призначення та сфери застосування систем штучного інтелекту (СШІ) у програмних ігрових додатках.

#### МЕТА ЛЕКЦІЇ

Ознайомити студентів з метою та задачами, темами та змістом модуля, знаннями та вміннями, передбаченими у результаті вивчення дисципліни. Надати формулювання основних понять комп'ютерних ігор, ігрового штучного інтелекту, визначити сутність інтелектуальної поведінки. Пояснення призначення та основних сфер застосування ШІ у комп'ютерних іграх, місця моделей та методів ШІ у ігрових технологіях.

#### ОЧІКУВАНІ РЕЗУЛЬТАТИ

Студент має усвідомити призначення модуля та можливості застосування отриманих знань та вмінь при розробці ігрових додатків. Студент повинен ознайомитися з основними поняттями ігрового ШІ. Має отримати розуміння сфери застосування ШІ у комп'ютерних іграх, методів та моделей ШІ у ігрових технологіях.

#### КОНТРОЛЬНІ ЗАПИТАННЯ

- Дайте відомі вам визначення поняття комп'ютерної гри.
- Поясніть термін «ігровий персонаж».
- Що таке ігровий штучний інтелект?
- Дайте визначення інтелектуальної поведінки.

- Поясніть призначення та сфери застосування ШІ у комп'ютерних іграх.
- Наведіть приклади використання ШІ у іграх.
- Назвіть відмінності інтелектуальної системи та традиційної програмної системи.
- Назвіть характеристики систем на основі правил.
- Як застосовуються кінцеві автомати в якості ШІ?
- Що таке «Адаптивний ШІ»?
- Як виконується сприйняття і пошук шляхів у ігрових додатках?
- Як виконується навігація ШІ?
- Охарактеризуйте тактичний та стратегічний ШІ.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

- HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.2 ТЕМА 2: СПОСОБИ ПОДАННЯ ІЗ. ХАРАКТЕРИСТИКИ СТРАТЕГІЙ ТА МЕТОДІВ ПОШУКУ РІШЕННЯ ІЗ.

### АНОТАЦІЯ

Лекція знайомить з поняттям інтелектуальної задачі (ІЗ). Характеризуються підходи до подання задач у інтелектуальних системах. Наводяться характеристики стратегій та методів пошуку рішення ІЗ.

### МЕТА ЛЕКЦІЇ

Ознайомити студентів основними поняттям ІЗ. Навести загальну характеристику підходів до подання задач у інтелектуальних системах. Охарактеризувати основні стратегії та методів пошуку рішення ІЗ.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

Студент має зрозуміти сутність поняття та характеристики ІЗ. Повинен знати підходи до подання задач у інтелектуальних системах – подання задач у просторі станів, просторі задач, у вигляді теорем. Студент повинен вміти визначити спосіб подання ІЗ відповідності до її характеристик та особливостей предметної області, вибрати відповідну стратегію пошуку рішень.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Дайте визначення ІЗ.
- Охарактеризуйте спосіб подання задач у просторі станів.
- Поясніть формальне подання задачі у просторі станів.
- Наведіть приклади подання задачі у вигляді графа.
- Охарактеризуйте спосіб подання задач у просторі задач.
- Поясніть формальне подання задачі у просторі задач.
- Що таке граф редукції задачі та І-АБО граф?
- Охарактеризуйте спосіб подання задач у вигляді теорем.
- Поясніть формальне подання задачі у вигляді теорем.
- Що таке формальна система?

- Чому формальну систему можна розглядати як генератор нових знань?
- Що таке ППФ?
- Прокоментуйте процес рішення задачі, поданої у вигляді теореми.
- Прокоментуйте переваги та недоліки способів подання ІЗ.
- Охарактеризувати основні стратегії та методів пошуку рішення ІЗ.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.2 ТЕМА 3: МЕТОДИ РІШЕННЯ ЗАДАЧ ЗА СТРАТЕГІЄЮ ПОШУКУ У ПРОСТОРІ СТАНІВ

### АНОТАЦІЯ

Лекція знайомить з основними поняттями теорії графів, процесами пошуку на графі. Характеризуються методи повного перебору, перебору в «глибину», «ширину». Пояснюється можливість використання оцінних функцій, оптимальних алгоритмів перебору та евристик при пошуку рішень.

### МЕТА ЛЕКЦІЇ

Ознайомити студентів з основними поняттями теорії графів. Пояснити сутність процесів пошуку на графі. Охарактеризувати та пояснити методи повного перебору, перебору в «глибину», «ширину». Пояснити можливість використання оцінних функцій, оптимальних алгоритмів перебору та евристик при пошуку рішень.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

Студент має бути ознайомленим з основними поняттями теорії графів. Отримати розуміння методів повного перебору, перебору в «глибину», «ширину». Студент повинен вміти визначити можливість використання оцінних функцій, оптимальних алгоритмів перебору та евристик при пошуку рішень.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Дайте визначення основних понять теорії графів.
- Розкрийте сутність поцесів пошуку на графі.
- Поясніть метод повного перебору.
- Поясніть метод пошуку в «глибину».
- Поясніть метод пошуку в «ширину».
- Що таке оцінна функція?
- Як складати оцінну функцію?
- Назвіть та поясніть основні оптимальні алгоритми перебору.
- Що таке евристики?
- Поясніть використання евристик при пошуку рішень.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

- HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.2 ТЕМА 4: МЕТОДИ ПОШУКУ РІШЕНЬ ЗАДАЧ У РАЗІ ЗВЕДЕННЯ ЗАДАЧ ДО СУКУПНОСТІ ПІД ЗАДАЧ. ПОШУК РІШЕНЬ ЗА ДОПОМОГОЮ МЕТОДІВ ЛОГІЧНОГО ПІДХОДУ.

### АНОТАЦІЯ

Лекція знайомить з поняттями «І-АБО» графів, дерев рішень. Характеризуються методи пошуку задач у разі зведення задач до сукупності підзадач.

### МЕТА ЛЕКЦІЇ

Ознайомити студентів основними поняттями поняттями «І-АБО» графів, дерев рішень. Навести характеристику методів пошуку на «І-АБО» графах. Навести способи обчислення вартості дерев рішень. Пояснити застосування макисмінного методу, методів альфа-бета відсікання, Ченга і Слейгла, планування загального вирішувача задач. Ознайомити з пошуком рішень за допомогою методів логічного підходу.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

Студент має зрозуміти сутність методів пошуку рішень задач у разі зведення задач до сукупності підзадач. Вміти застосовувати означені методи пошуку рішень на практиці.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Дайте визначення «І-АБО» графа.
- Дайте визначення дерева рішень.
- Назвіть та охарактеризуйте методи пошуку на «І-АБО» графах.
- Поясніть способи обчислення вартості дерев рішень.
- Поясніть та охарактеризуйте макисмінний метод, методи альфа-бета відсікання, Ченга і Слейгла, планування загального вирішувача задач.
- Як виконується пошук рішень за допомогою методів логічного підходу?
- Як виконується пошук рішень у великих просторах станів?
- Як виконується пошук рішень в умовах невизначеності?

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.3 ЛАБОРАТОРНА РОБОТА № 1. СПОСОБИ ПОДАННЯ ІЗ

### АНОТАЦІЯ

Лабораторна робота орієнтована на ознайомлення студентів з основними способами подання ІЗ – у просторі станів, у просторі задач, у вигляді теореми.

### МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Освоїти основні способи подання ІЗ, виконати графічне представлення задачі та показати можливі варіанти її рішення.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

Успішне виконання лабораторної роботи надає студенту вміння розуміти предметну область, застосовувати підходи до формального подання задачі, знаходити варіанти можливих рішень та виконувати загальну їх оцінку.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Охарактеризуйте спосіб подання задачі у просторі станів.
- Поясніть процес подання задачі у просторі станів згідно з індивідуальним завданням.
- Охарактеризуйте спосіб подання задачі у просторі задач.
- Поясніть процес подання задачі у просторі задач згідно з індивідуальним завданням.
- Охарактеризуйте спосіб подання задачі у вигляді теореми.
- Поясніть процес подання задачі у вигляді теореми згідно з індивідуальним завданням.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.4 ЛАБОРАТОРНА РОБОТА №2. ПОДАННЯ ІЗ У ПРОГРАМНІЙ СИСТЕМІ. ВИКОНАННЯ ПОШУКУ РІШЕНЬ У ПРОСТОРІ СТАНІВ.

### АНОТАЦІЯ

Лабораторна робота орієнтована на оволодіння студентами навичок розробки програмного забезпечення для подання ІЗ у просторі станів та пошуку можливих рішень.

### МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Розробка програмного забезпечення для подання ІЗ як компоненту ігрового додатку у просторі станів та пошуку можливих рішень.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

У разі успішного виконання лабораторної роботи студент має вміти подати у ігровому додатку задачу у просторі станів та реалізувати алгоритми пошуку рішень.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Як можна описати ІЗ у просторі станів?
- Назвіть та охарактеризуйте можливі способи представлення графів у програмній системі.
- Поясніть алгоритм методу повного перебору.
- Поясніть алгоритм пошуку «у глибину».
- Поясніть алгоритм пошуку «у ширину».
- Поясніть алгоритм пошуку з перевагою.
- Поясніть алгоритм пошуку за методом гілок і меж.
- Поясніть алгоритм Мура.
- Поясніть алгоритм Дейкстри.
- Поясніть алгоритм Дорана ті Мічі.
- Поясніть алгоритм Харта, Нільсона та Рафаеля.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

## 11.5 ЛАБОРАТОРНА РОБОТА №3. ПОДАННЯ ІЗ У ПРОГРАМНІЙ СИСТЕМІ. МЕТОДИ ПОШУКУ РІШЕНЬ ІЗ У ПРОСТОРІ ЗАДАЧ.

### АНОТАЦІЯ

Лабораторна робота орієнтована на оволодіння студентами навичок розробки програмного забезпечення для подання ІЗ у просторі задач та пошуку можливих рішень.

### МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Розробка програмного забезпечення для подання ІЗ як компоненту ігрового додатку у просторі задач та пошуку можливих рішень.

### ОЧІКУВАНІ РЕЗУЛЬТАТИ

У разі успішного виконання лабораторної роботи студент має вміти подати у ігровому додатку задачу у просторі задач та реалізувати алгоритми пошуку рішень.

### КОНТРОЛЬНІ ЗАПИТАННЯ

- Як можна описати ІЗ у просторі задач?
- Назвіть та охарактеризуйте можливі способи представлення графів у програмній системі.
- Яка процедура подання задачі у просторі задач?
- Як подати задачу у вигляді І/АБО-графу?
- Охарактеризуйте алгоритм повного перебору.
- Охарактеризуйте алгоритм пошуку «у глибину».
- Охарактеризуйте алгоритм впорядкованого перебору для І/АБО дерева.
- Охарактеризуйте алгоритм оцінювання вартості вирішувальних дерев.
- Охарактеризуйте мінімаксного алгоритм пошуку рішень.
- Охарактеризуйте алгоритм альфа-бета відсікання.
- Охарактеризуйте алгоритм Ченга і Слейгла.

### МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

# 12 СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ


1. Вступ до експертних систем : навч. посібник / В. О. Кравець [та ін.] ; Харківський політехнічний ін–т, нац. техн. ун–т. — Харків : НТУ «ХПІ», 2006. — 232 с.
2. Artificial intelligence in video games [Electronic resource] // Wikipedia: the free encyclopedia. — Mode access: https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games (reference date: 06.01.2017).
3. Bostrom N. Superintelligence. Paths, Dangers, Strategies / N. Bostrom. — Oxford University Press, 2016. — 432 p.
4. Champandard A. J. AI Game Development / A. J. Champandard. — New Riders Games, 2003. — 731 p.
5. Champandard A. J. Getting started with decision making and control systems / A. J. Champandard. — AI Game Programming Widsom, 2008. — Vol. 4. — P. 257–263.
6. Champandard A. J. The behavior tree starter kit / Alex J. Champandard, Philip Dunstan // Game AI Pro: Collected Widsom for Game AI Professionals / Ed. by Steven Rabin. — Boca Raton, FL : A K Peters/CRC Press, 2013. — P. 73–91.
7. Luger G. F. Artificial intelligence : structures and strategies for complex problem solving / G. F. Luger. —6th ed. — Pearson Addison Wesley, 2008. — 754 p.
8. Nilsson N. J. The quest for artificial intelligence: a history of ideas and achievements [Electronic resource] / Nils J. Nilsson. — Cambridge University Press, 2009. — Mode access: http://ai.stanford.edu/~nilsson/QAI/qai.pdf (reference date: 06.01.2017).
9. Poole D. Artificial Intelligence: foundations of computational agents [Electronic resource] / D. Poole, A. Mackworth. — Cambridge University Press, 2010. — Mode access: http://artint.info/html/ArtInt.html (reference date: 10.01.2017).
10. Russel S. Artificial Intelligence: A Modern Approach / S. Russel, P. Norvig. — 3rd ed. — Pearson Education Limited, 2014. — 1099 p.

# Лекції

# ЛЕКЦІЯ 1. СТВОРЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГОР

Комп'ютерна гра – комп'ютерна програма, що служить для організації ігрового процесу (геймплея), зв'язку з партнерами по грі, або сама виступає в якості партнера.

В даний час в ряді випадків замість терміну комп'ютерна гра може використовуватися термін «відеогра». Терміни «Комп'ютерна гра» та «відеогра» можуть вживатися як синоніми і бути взаємозамінними. У комп'ютерних іграх, як правило, ігрова ситуація відтворюється на екрані дисплея або звичайного телевізора (в цьому випадку комп'ютерні ігри одночасно є і відеоіграми), але в той же час комп'ютерна гра може бути звуковою, телетайповою та ін.

Комп'ютерні ігри настільки істотно впливають на суспільство, що в інформаційних технологіях відзначена стійка тенденція до Гейміфікації для неігрового прикладного програмного забезпечення.

Термін «ігровий персонаж» застосовується для позначення персонажа в комп'ютерних іграх, який управляється людиною-гравцем. Управління людиною відокремлює ігрові персонажі від неігрових, керованих ігровим штучним інтелектом. У переважній більшості випадків ігровий персонаж є протагоністом (головним героєм) гри.

Ігровий штучний інтелект (Game artificial intelligence) – набір програмних методик, які використовуються в комп'ютерних іграх для створення ілюзії інтелекту в поведінці персонажів, керованих комп'ютером. Ігровий ШІ, крім методів традиційного штучного інтелекту, включає також алгоритми теорії управління, робототехніки, комп'ютерної графіки та інформатики в цілому.

Інтелектуальною поведінкою ігрового персонажа є його здатність досягати поставленої мети. Інтелектуальність поведінки залежить від запасу використовуваних основних принципів і якостей процесів міркування загального вигляду. Використовувані системою основні принципи характеризують широту охоплення ігрової області. Важливим є відображення відомих системі принципів на ступінь відомих їй подробиць. Якість міркувань залежить від доступності фактів, які мають відношення до ситуації, і принципів і повноти процедури виведення та ефективності її реалізації.

У традиційних дослідженнях в галузі ШІ метою є створення справжнього інтелекту, хоча і штучними засобами. В таких проектах, як Kismet Массачусетського технологічного інституту (MTI) робиться спроба створити ШІ, здатний до навчання і до соціальної взаємодії, до прояву емоцій. Наразі в MTI ведеться робота над створенням ШІ, що має рівень здібностей маленької дитини, і результати цієї роботи дуже перспективні.

З точки зору ігор справжній ШІ далеко виходить за рамки вимог розважального програмного проекту. В іграх така потужність не потрібна. Ігровий ШІ не повинен бути наділений почуттями і самосвідомістю, йому немає необхідності навчатися чогось за межами рамок ігрового процесу. Справжня мета ШІ в іграх – імітація розумної поведінки і в наданні гравцеві переконливої, правдоподібної мети та постановки відповідної задачі.

## 1.1 ПРИЙНЯТТЯ РІШЕНЬ

Основним принципом, що лежить в основі роботи ШІ, є прийняття рішень. Для вибору при прийнятті рішень система повинна впливати на об'єкти за допомогою ШІ. При цьому такий вплив може бути організовано у вигляді «мовлення ШІ» або «звернень об'єктів».

У системах з «мовленням ШІ» система ШІ зазвичай ізольована у вигляді окремого елемента ігрової архітектури. Така стратегія часто приймає форму окремого потоку або декількох потоків, в яких ШІ знаходить найкраще рішення для заданих параметрів гри. Коли ШІ приймає рішення, воно передається усім учасникам об'єктів. Такий підхід найкраще працює в стратегіях реального часу, де ШІ аналізує загальний хід подій у всій грі.

Системи зі «зверненнями об'єктів» краще підходять для ігор з простими об'єктами. В таких іграх об'єкти звертаються до системи ШІ кожного разу, коли об'єкт «думає» або оновлює себе. Такий підхід найкраще підходить для систем з великою кількістю об'єктів, яким не потрібно «думати» занадто часто, наприклад в шутерах. Така система також може скористатися перевагами багатопотокової архітектури, але для неї потрібно більш складне планування.

## 1.2 БАЗОВЕ СПРИЙНЯТТЯ СЕРЕДОВИЩА

Щоб штучний інтелект міг приймати осмислені рішення, йому необхідно якимось чином сприймати середовище, в якому він знаходиться. У простих системах таке сприйняття може обмежуватися простою перевіркою положення об'єкта гравця. У більш складних системах потрібно визначати основні характеристики і властивості ігрового світу, наприклад можливі маршрути для пересування, наявність природних укриттів на місцевості, області конфліктів.

При цьому організатори повинні придумувати спосіб виявлення і визначення основних властивостей ігрового світу, важливих для системи ШІ. Наприклад, укриття на місцевості можуть бути заздалегідь визначені дизайнерами рівнів або заздалегідь обчислені при завантаженні або компіляції карти рівня. Деякі елементи необхідно обчислювати на льоту, наприклад карти конфліктів і найближчі загрози.

## 1.3 СИСТЕМИ НА ОСНОВІ ПРАВИЛ

Найпростішою формою штучного інтелекту є система на основі правил. Така система стоїть найдалі від справжнього штучного інтелекту. Набір заздалегідь заданих алгоритмів визначає поведінку ігрових об'єктів. З урахуванням різноманітності дій кінцевий результат може бути неявною поведінковою системою, хоча така система насправді зовсім не буде «інтелектуальною».

Класичним прикладом ігрового додатку, де використовується така система, є Pac-Man. Гравця переслідують чотири привиди. Кожний привид діє, підкоряючись простому набору правил. Один привид завжди повертає вліво, інше завжди повертає вправо, тре-

тій повертає в довільному напрямку, а четвертий завжди повертає в бік гравця. Якби на екрані привиди з'являлися по одному, їх поведінку було б дуже легко визначити і гравець зміг би легко від них рятуватися. Але оскільки з'являється відразу група з чотирьох привидів, їх рухи здаються складним і скоординованим вистежуванням гравця. Насправді ж тільки останній з чотирьох привидів враховує розташування гравця.

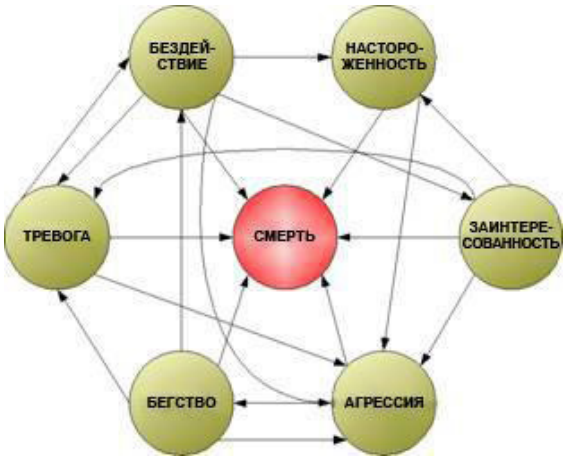
З цього прикладу випливає, що правила не обов'язково повинні бути жорстко заданими. Вони можуть грунтуватися на стані, що сприймається у ході гри (як у останнього привида), або на редагованих параметрах об'єктів. Такі змінні, як рівень агресії, рівень сміливості, дальність огляду та швидкість мислення, дозволяють отримати більш різноманітну поведінку об'єктів навіть при використанні систем на основі правил.

У більш складних і розумних системах як основа використовуються низки умовних правил. У тактичних іграх правила керують вибором використовуваної тактики. В стратегічних іграх правила управляють послідовністю об'єктів, що будуються, і реакцією на конфлікти. Системи на основі правил є фундаментом ШІ.

## 1.4 КІНЦЕВІ АВТОМАТИ В ЯКОСТІ ШІ

Кінцевий автомат (машина з кінцевим числом станів) є способом моделювання і реалізації об'єкта, що володіє різними станами протягом свого життя. Кожний «стан» може представляти фізичні умови, в яких знаходиться об'єкт, або, наприклад, набір емоцій, які висловлюються об'єктом. Тут емоційні стани не мають ніякого відношення до емоцій ШІ, вони належать до заздалегідь заданих поведінкових моделей, які вписуються в контекст гри.

Схема станів в типовому кінцевому автоматі, стрілки представляють можливі зміни стану:




Існують два простих способи реалізації кінцевого автомата з системою об'єктів. Перший спосіб: кожний стан є змінною, яку можна перевірити (найчастіше це робиться за допомогою великих інструкцій перемикання). Другий спосіб: використовувати покажчики функцій (на мові С) або віртуальні функції (С ++) і інші засоби об'єктно-орієнтованих мов програмування).

## 1.5 АДАПТИВНИЙ ШІ

Якщо в грі потрібна більша різноманітність, якщо у гравця повинен бути сильніший і динамічний противник, то ШІ повинен мати здатність розвиватися, пристосовуватися і адаптуватися.

Адаптивний ШІ часто використовується в бойових і стратегічних іграх зі складною механікою та великою кількістю різноманітних можливостей в ігровому процесі.

## 1.6 ПЕРЕДБАЧЕННЯ

Здатність точно передбачати наступний хід супротивника вкрай важлива для адаптивної системи. Для вибору наступної дії можна використовувати різні методи, наприклад розпізнавання закономірностей минулих ходів або випадкові здогадки.

Одним з найпростіших способів адаптації є відстеження рішень, прийнятих раніше, і оцінка їх успішності. Система ШІ реєструє вибір, зроблений гравцем в минулому. Всі прийняті в минулому рішення потрібно якимось чином оцінювати (наприклад, в бойових іграх в якості запобіжника успішності можна використовувати отриману або втрачену перевагу, втрачене здоров'я чи перевагу за часом). Можна збирати додаткові відомості про ситуацію, щоб утворити контекст для рішень, наприклад відносний рівень здоров'я, колишні дії і положення на рівні (люди грають по-іншому, коли їм вже нікуди відступати).

Можна оцінювати історію для визначення успішності колишніх дій і прийняття рішення про те, чи потрібно змінювати тактику. До моменту створення списку колишніх дій об'єкт може використовувати стандартну тактику або діяти довільно. Цю систему можна пов'язати з системами на основі правил і з різними станами.

В тактичній грі історія минулих боїв допоможе вибрати найкращу тактику для використання проти команди гравця, наприклад ШІ може грати від оборони, вибрати наступальну тактику, атакувати всіма силами незважаючи на втрати або ж обрати збалансований підхід. У стратегічній грі можна для кожного гравця підбирати оптимальний набір різних бойових одиниць в армії. В іграх, де ШІ керує персонажами, що підтримують гравця, адаптивний ШІ зможе краще пристосуватися до природного стилю гравця, вивчаючи його дії.

## 1.7 СПРИЙНЯТТЯ І ПОШУК ШЛЯХІВ

До цих пір мова йшла про найпростіші способи рішень, прийнятих інтелектуальними агентами; так в дослідженнях в області штучного інтелекту називаються об'єкти, що використовують ШІ. Далі нашому герою (або чудовиську, або ігровому об'єкту будь-якого іншого типу) надається контекст для прийняття рішень. Інтелектуальні агенти повинні виявляти області інтересу в ігровому світі, а потім думати про те, як туди дістатися.

Тут ми вже досить близько підходимо до дійсно штучного інтелекту. Всім інтелектуальним агентам потрібна базова здатність сприйняття свого середовища і будь-які засоби навігації та переміщення в оточуючому світі (реальному або будь-якому іншому). Ігро-

вим об'єктам потрібно те ж саме, хоча підхід сильно відрізняється. Крім того, по відношенню до ігрового світу можна шахраювати, і гравцеві доведеться це робити, щоб все працювало швидко і плавно.

## 1.8 ШІ ТА СПРИЙНЯТТЯ НАВКОЛИШНЬОГО СВІТУ

### Зір

Якщо агент збирається приймати досить виважені рішення, йому необхідно знати, що відбувається навколо. У системах ШІ, що застосовуються в робототехніці, значний обсяг досліджень присвячений комп'ютерному зору: роботи отримують здатність сприймати навколишній світ за допомогою об'ємного тривимірного зору точно так же, як люди. Але для наших цілей такий рівень досконалості, зрозуміло, надмірний.

Віртуальні світи, в яких відбувається дія більшості ігор, мають величезну перевагу над реальним світом з точки зору ШІ і його сприйняття. На відміну від реального світу нам відома кількість буквально всього, що є в віртуальному світі: десь серед ресурсів гри є список, де перераховується все, що існує в грі. Ви можете шукати за цим списком, вказавши потрібний запит, і миттєво отримати інформацію, яку ваш агент зможе використовувати, щоб приймати більш обґрунтовані рішення. При цьому можна або зупинитися на першому ж об'єкті, який буде представляти інтерес для вашого агента, або отримати список всіх об'єктів в межах заданої дальності, щоб агент зміг прийняти оптимальне рішення щодо навколишнього світу.

Такий підхід непогано працює для простих ігор, але, коли стиль гри ускладнюється, ваші агенти повинні бути більш розбірливими щодо того, що вони «бачать». Якщо ви не хочете, щоб агенти вели себе так, ніби у них очі на потилиці, можна провести вибірку в списку потенційних об'єктів, які перебувають в межах радіусу зору агента. Це можна зробити досить швидко за допомогою нескладної математики.

1. Розрахуйте вектор між агентом і потрібним об'єктом шляхом вирахування положення цілі з положення агента.

2. Обчисліть кут між цим вектором і напрямом погляду агента.

3. Якщо абсолютне значення кута перевищує заданий кут поля зору агента, то ваш агент не бачить об'єкт.

У більш складних іграх потрібно враховувати, що гравець або інші об'єкти можуть перебувати за будь-яким укриттям. Для таких ігор може знадобитися побудувати рухомі промені (так званий метод ray casting), щоб дізнатися, чи не загороджена чимось можлива мета. Побудова рухомих променів – це математичний спосіб перевірити, перетинається промінь з будь-якими об'єктами, починаючи з однієї точки і рухаючись в заданому напрямку.

Описаний вище метод дозволяє дізнатися, чи загороджує що-небудь центр цілі, але цього може бути недостатньо, щоб приховати вашого агента. Адже може бути і так, що центр агента прихований, але зате його голова найзручнішим (для супротивника) чином стирчить над укриттям. Використання декількох променів, що біжать, спрямованих на певні точки цілі, допоможе не тільки визначити, чи можна потрапити в ціль,

але і куди саме можна вразити ціль.

### Слух

Якщо ви можете бачити об'єкт, то, безумовно, ви можете його і чути. Дійсно, якщо ваш агент помітив об'єкт, агент може активно виявляти всі дії об'єкта до тих пір, поки об'єкт не виявиться поза полем зору. Проте якщо додати агентам додатковий рівень слуху, то і зір буде працювати ефективніше. Відстеження шуму, видаваного об'єктами, – це найважливіший рівень сприйняття для будь-яких ігор з підкраданням.

Як і в випадку із зором, спочатку потрібно отримати список об'єктів, що знаходяться поруч. Для цього можна знову просто перевірити дистанцію, але вибірка потрібних об'єктів з цього списку відбувається вже зовсім інакше.

З кожною дією, яку може виконати об'єкт, зв'язується певний рівень звуку. Можна заздалегідь задати рівні звуку (для оптимізації ігрового балансу) або розраховувати їх на основі фактичної енергії звукових ефектів, пов'язаних з тими чи іншими діями (це дозволяє домогтися високого рівня реалізму, але навряд чи необхідно). Якщо вироблений звук голосніше заданого порогу, то ваш агент помітить об'єкт, що видає звук.

Якщо потрібно враховувати перешкоди, то можна знову скоротити список об'єктів і за допомогою рухомих променів визначити, чи є якісь перешкоди на шляху звуку. Але лише деякі матеріали повністю звуконепроникні, тому слід більш творчо підійти до скорочення списку.

Базова функціональність, необхідна для додання вашим агентам зору і слуху, може використовуватися і для імітації інших органів відчуття. Наприклад, нюху. (Можливість відстеження гравців інтелектуальними агентами по запаху існує в сучасних іграх, таких як Call of Duty 4). Додавання нюху в гру не викликає особливих труднощів: досить призначити кожному ігровому об'єкту унікальний номер запаху і його інтенсивність. Інтенсивність запаху визначає два чинники: радіус запаху і силу запаху сліду, який залишається позаду. Активні об'єкти гравців часто відслідковують свої попередні положення за рядом причин. Однією з таких причин може бути використання об'єктів із запахом. З плином часу сила запаху сліду зменшується, слід «остигає». Коли дані агента про запах змінюються, він повинен перевірити наявність запаху точно так же, як наявність звуку (з урахуванням радіуса і перешкод). Успішність нюху обчислюється на основі інтенсивності запаху і сили нюху агента: ці значення порівнюються з об'єктом і його слідом.

### Дотик

Дотик в іграх підтримується спочатку, оскільки в будь-якій грі вже є система автоматичної обробки зіткнень об'єктів. Досить домогтися того, щоб інтелектуальні агенти реагували на події зіткнень і пошкодження.

Здатність відчувати навколишній світ – це чудово, але що саме повинні відчувати агенти? Необхідно вказувати і ідентифікувати доступні для сприйняття речі в налаштуваннях агентів. Коли ви розпізнаєте те, що бачите, агенти зможуть реагувати на це на основі правил, які керують даним об'єктом.

## 1.9 ТИМЧАСОВІ ОБ'ЄКТИ

Їх іноді називають частками, спрайтами або спецефектами. Тимчасові об'єкти є візуальними ефектами в ігровому світі. Тимчасові об'єкти схожі зі звичайними об'єктами в тому, що одна загальна структура класу визначає їх все. Різниця ж полягає в тому, що тимчасові об'єкти не думають, не реагують і не взаємодіють ні з іншими об'єктами ігрового світу, ні один з одним. Їх єдина мета – красиво виглядати, протягом деякого часу покращувати деталізацію світу, а потім зникнути. Тимчасові об'єкти використовуються для таких ефектів, як сліди від куль, дим, іскри, бризки крові і навіть відбитки підошов на землі.

В силу природи тимчасових об'єктів для них не потрібно значного обсягу обчислень і виявлення зіткнень (за винятком дуже простих зіткнень з навколишнім світом). Проблема полягає в тому, що деякі тимчасові об'єкти надають гравцеві наочні підказки про події, що нещодавно відбулися. Наприклад, кульові отвори і сліди обгорання можуть вказувати, що недавно тут був бій; сліди в снігу можуть призвести до потенційної мети. Чому б і інтелектуальним агентам не використовувати такі підказки?

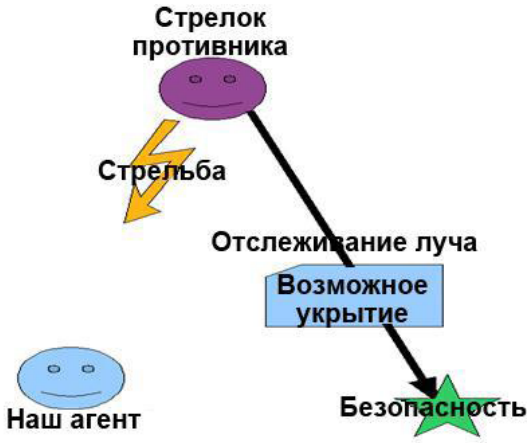
Цю задачу можна вирішити двома способами. Можна або розширити систему тимчасових об'єктів, додавши підтримку рухомих променів (але при цьому буде спотворений весь сенс системи тимчасових об'єктів), або схитрувати: розміщувати порожній об'єкт недалеко від тимчасових об'єктів. Цей порожній об'єкт не буде здатний думати, з ним не будуть пов'язані ніякі графічні елементи, але ваші агенти зможуть його виявляти, а тимчасовий об'єкт буде мати у своєму розпорядженні пов'язану інформацію, яку зможе отримати ваш агент. Отже, коли ви малюєте на підлозі тимчасову калюжку крові, Ви також можете розташувати там невидимий об'єкт, по якому ваші агенти дізнаються, що тут щось сталося. Що стосується відбитків: це питання вже вирішується за допомогою сліду.

## 1.10 УКРИТТЯ

У багатьох іграх зі стрілянином було б добре, якби агенти вміли ховатися за укриттями, якщо вони є поблизу, а не просто стояти під вогнем противника на відкритому місці. Але ця проблема дещо складніше всіх названих раніше проблем. Яким же чином агенти зможуть визначати, чи є поруч якісь відповідні укриття?

Ця проблема насправді складається з двох задач: по-перше, потрібно правильно розпізнати укриття на основі геометрії навколишнього світу; по-друге, потрібно правильно розпізнати укриття на основі об'єктів навколишнього світу. Щоб визначити, чи здатне укриття захищати від атак, можна просто один раз порівняти розмір граничної межі агента з розмірами можливого укриття. Потім слід перевірити, чи зможе ваш об'єкт поміститися за цим укриттям. Для цього потрібно провести промені від відмінностей в положеннях вашого стрілка і укриття. За допомогою цього променя можна визначити, чи є вільним місце, що знаходиться за укриттям (якщо дивитися з боку стрілка), а потім позначити це місце як наступну мету переміщення агента.

**Приклад.** На схемі наш агент визначив, що в місці, позначеному зеленої зірочкою, можна буде сховатися в безпеці:




## 1.11 НАВІГАЦІЯ ШІ

Прийнявши рішення, інтелектуальний агент повинен зрозуміти, як рухатися з точки А в точку Б. Для цього можна використовувати різні підходи, вибравши оптимальний залежно від характеру гри і від потрібного рівня продуктивності.

Алгоритм під умовною назвою «Зіткнутися та повернути», є одним з найпростіших способів формування маршруту руху об'єкта. Ось як це працює.

1. Рухайтеся в напрямку мети.

2. Якщо зіткнетеся зі стіною, поверніться в напрямку, при якому ви опинитеся найближче до мети. Якщо жоден з доступних для вибору варіантів не має очевидних переваг, вибір робиться довільним чином.

Такий підхід непогано працює для нескладних ігор. У величезній кількості ігор чудовиська застосовують цей алгоритм, щоб вистежувати гравця. Але при використанні алгоритму «Зіткнутися та повернути» об'єкти, що полюють за гравцем, опиняються за увігнутими стінами або за кутами. Тому такий алгоритм ідеальний хіба що для ігор з зомбі або для ігор без стін та інших перешкод.

Якщо ж агенти в грі повинні діяти не настільки безглуздо, можна розширити просту подію зіткнення і забезпечити агентів пам'яттю. Якщо агенти здатні запам'ятовувати, де вони побували, то вони зможуть приймати більш грамотні рішення щодо того, куди повертати далі. Якщо ж повороти в усіх можливих напрямках не привели до успіху, агенти зможуть повернутися назад і вибрати інший маршрут руху. Таким чином, агенти будуть виробляти систематичний пошук шляху до мети. Ось як це працює.

1. Рухайтеся в напрямку мети.

2. Якщо шлях розгалужується, виберіть один з можливих напрямків.

3. Якщо шлях приводить в глухий кут, повертайтеся до останнього розгалуження і виберіть інший напрямок.

4. Якщо всі можливі шляхи пройдені безрезультатно, відмовляйтеся від подальшого пошуку.

Перевага цього методу полягає в невисокому навантаженні на обчислювальні ресурси. Це означає, що можна підтримувати велику кількість рухомих агентів без уповільнення гри. Цей метод також може використовувати переваги багатопотокової архітектури. Недоліком є витрата величезного обсягу пам'яті даремно, оскільки кожен агент може відслідковувати цілу карту можливих шляхів.

Втім, нераціонального використання пам'яті можна уникнути, якщо агенти будуть зберігати відслідковувані шляхи в загальній пам'яті. В цьому випадку можуть виникнути проблеми у зв'язку з конфліктом потоків, тому рекомендуємо зберігати шляхи об'єктів в окремому модулі, в який всі агенти будуть відправляти запити (у міру переміщення) і оновлені дані (при виявленні нових шляхів). Модуль карти шляхів може аналізувати отриману інформацію, щоб не допускати конфліктів.

## 1.12 Пошук шляхів

Карти, на яких шляхи прокладаються за допомогою алгоритму «Зіткнутися та повернути», дозволяють пристосуватися у картах до умов, що змінюються. Але в стратегічних іграх гравці не можуть чекати, поки їх війська розберуться з прокладанням маршрутів. Крім того, карти шляхів можуть бути дуже великими, і на вибір правильного шляху на таких картах буде витрачатися дуже багато ресурсів. У таких ситуаціях на допомогу приходить алгоритм пошуку шляхів.

Пошук шляхів можна вважати вже давно і успішно вирішеною проблемою в розробці ігор. Навіть в таких старих іграх, як перша версія легендарної гри Starcraft (Blizzard Entertainment), величезні кількості ігрових об'єктів могли визначати шляхи руху за великими і складними картами.

Для визначення шляхів руху використовується алгоритм під назвою A* (вимовляється а-стар). З його допомогою можна знаходити оптимальний шлях між двома будь-якими точками в графі (в даному випадку – на карті). Простий пошук в Інтернеті видає чистий алгоритм, який використовує вкрай «зрозумілі» описові терміни, такі як F, G і H.

Спочатку потрібно створити два списки: список вузлів, які ще не перевірені (Unchecked), і список вже перевірених вузлів (Checked). Кожен список включає вузол розташування, передбачувану відстань до цілі і посилання на батьківський об'єкт (вузол, який помістив даний вузол в список). Спочатку списки порожні.

Тепер додамо початкове розташування в список неперевірених вузлів, не вказуючи нічого в якості батьківського об'єкта. Потім вводимо алгоритм.

1. Обираємо в списку найбільш підходящий вузол.

2. Якщо цей вузол є метою, то пошук завершено.

3. Якщо цей вузол не є метою, додаємо його до списку перевірених.

4. Для кожного вузла, сусіднього з цим вузлом:

- якщо цей вузол непрохідний, ігноруємо його;

- якщо цей вузол вже є в будь-якому зі списків (перевірених або неперевірених), ігноруємо його;

- в іншому випадку додаємо його в список неперевірених, вказуємо поточний вузол в якості батьківського і розраховуємо довжину шляху до мети (досить просто обчислити відстань).

Коли об'єкт досягає поля мети, можна побудувати шлях, відстеживши батьківські вузли аж до вузла, у якого немає батьківського елементу (це початковий вузол). При цьому ми отримуємо оптимальний шлях, по якому може переміщатися об'єкт.

Цей процес працює тільки коли агент отримує наказ або самостійно приймає рішення про рух, тому тут можна з великою вигодою використовувати багатопоточність. Агент може відправити запит в потік пошуку шляху, щоб отримати виявлений шлях, не впливаючи на продуктивність ШІ. У більшості випадків система може швидко отримати результати. При завантаженні великої кількості запитів шляхів агент може або почекати, або, не чекаючи видачі шляхів, просто почати рухатися в потрібному напрямку (наприклад, за алгоритмом «Зіткнутися і повернути»). На дуже великих картах можна розділити систему на області і заздалегідь обчислити всі можливі шляхи між областями (або точки маршруту).

У цьому випадку модуль пошуку шляхів просто знаходить найкращий шлях і негайно повертає результати. Потік карти шляхів може просто відстежувати зміни на карті (наприклад, коли гравець будує стіну), а потім знову запускає перевірку шляхів у міру необхідності. Оскільки цей алгоритм працює у власному потоці, він може адаптуватися, не впливаючи на продуктивність решті гри.

Нить може підвищити продуктивність навіть всередині підсистеми пошуку шляхів. Цей підхід широко застосовується у всіх стратегіях в реальному часі (RTS) і в системах з великою кількістю об'єктів, кожен з яких намагається виявити потенційно унікальний шлях. У різних потоках можна одночасно знаходити множину шляхів. Зрозуміло, система повинна відстежувати, які шляхи виявляються. Кожен шлях досить виявити тільки один раз.

## 1.13 ТАКТИЧНИЙ ТА СТРАТЕГІЧНИЙ ШІ

Тепер настав час поговорити про те, як віддавати агентам більш складні накази. Агенти повинні навчитися справлятися з ситуацією, в якій вони опинилися. Тобто, перейдемо до штучного інтелекту, здатного працювати з більш широкими цілями і сприймати обстановку більш масштабно.

### 1.13.1 ТАКТИЧНИЙ ШІ

Роль тактичного ШІ полягає в координації зусиль груп агентів в грі. Групи ефективніші, оскільки члени групи можуть підтримувати один одного, можуть діяти як єдиний підрозділ, обмінюватися інформацією і розподіляти дії з отримання інформації.

Принцип тактичного ШІ побудований навколо динаміки групи. Гра повинна відстежувати різні групи об'єктів. Кожну групу необхідно оновлювати окремо від індивідуальних об'єктів. Для цього можна використовувати виділений модуль оновлення, що

відслідковує різні групи, їх цілі та їх склад. Недолік цього методу полягає в тому, що потрібна розробка окремої системи для ігрового движка. Тому краще використовувати метод командира групи.

Одному юніту в складі групи можна призначити роль командира групи. Всі інші члени групи пов'язані зі своїм командиром, їх поведінка визначається інформацією, отриманою з наказів командира. Командир групи обробляє всі обчислення тактичного ШІ для всієї групи.

Рух групи: пошук шляхів

Рух об'єктів можна поліпшити за допомогою динаміки групи. Коли кілька агентів діють як єдиний підрозділ, їх рух можна зробити більш ефективним і реалістичним.

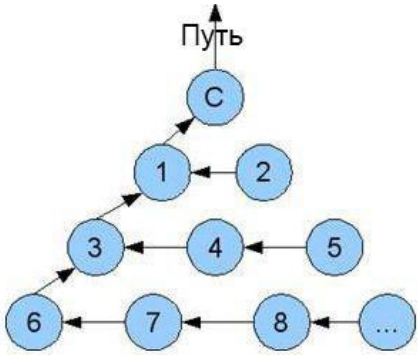
Пошук шляху може займати чимало часу навіть при прискоренні з допомогою заздалегідь обчислених карт шляхів і багатопоточного ШІ. Динаміка груп дозволяє істотно знизити навантаження, створене системою пошуку шляхів.

Коли групі юнітів віддається наказ на переміщення (гравцем або штучним інтелектом), найближчий до мети юніт призначається командиром групи, а всі інші члени групи слідують за командиром. При оновленні командира групи він запитує систему шляхів. При наявності шляху командир групи починає рух до мети. Всім іншим юнитам в групі досить просто прямувати за своїм командиром.

Для більш упорядкованого пересування застосовується стрій. При використанні строю група пересувається впорядковано, наприклад вишикувавшись фалангою або трикутником.

Управляти строєм дуже просто, для цього достатньо дещо розширити сферу дій командира групи. Кожен юніт в строю виконує певну роль. При побудові кожному члену групи призначається місце в строю точно так же, як одному з юнітів призначається роль командира групи. Мета кожного юніта – зберегти своє місце на відносній відстані до інших членів групи.

Для прикладу візьмемо рядових в грі Overlord. Вони пересуваються трикутним строєм. На малюнку нижче по маршруту повинен рухатися тільки командир групи (позначений літерою «С»). Юніт 1 слідом за юнітом «С» з такою ж швидкістю ззаду і трохи лівіше. Юніт 2 стежить за юнітом 1, рухаючись трохи збоку. Юніт 3 робить те ж саме, що і юніт 1, але слідом за юнітом 1, а не за командиром. Всі члени групи дотримуються цього порядку.



### ПОРЯДОК РУХУ ТРИКУТНИМ СТРОЄМ

Тактика груп

Тактика, зрозуміло, не обмежується ходьбою строєм, а включає також підтримку і ведення бою групою як єдиною командою. Командир приймає на себе обов'язки з планування та координації роботи команди, адже саме командир несе відповідальність за життя всіх підлеглих в своєму загоні.

Для реалізації тактики груп можуть використовуватися раніше описані системи, наприклад системи на основі правил або кінцеві автомати. Типові приклади поведінки груп в іграх: підтримка лікуванням (лікарі залишаються поруч з юнітами, які з найбільшою ймовірністю будуть атаковані), розвідка, прикриття вогнем, жертвування (заслін цінних юнітів менш цінними).

Крім того, в групі може стати в нагоді ще один рівень аналізу – аналіз можливостей кожного члена групи. Командиру важливо знати, в яких ситуаціях група може бути ефективна, коли група отримає перевагу, а коли група експертів повинна відступити.

Наприклад, в стратегії реального часу Starcraft * компанії Blizzard існують наземні війська і льотні війська. При цьому не всі види наземних військ можуть стріляти по льотних військах. Групі важливо знати про наявність такої можливості. Якщо в групі немає жодного юніта, здатного вести вогонь по льотних юнитах, то при виявленні льотного юніта найкраще втекти. Але якщо в групі є юніти, здатні вражати льотного противника, навіть якщо таких юнітів і небагато, краще не відступати, а зупинитися і оборонятися (якщо в цій групі є допоміжні юніти, здатні лікувати тих, які ведуть вогонь по повітряних цілях).

Залежно від наявності різних можливостей і від кількості юнітів, які мають такі можливості, можна оцінювати бойову ефективність групи в різних ситуаціях. Групи, в яких ці фактори беруться до уваги, будуть вести бій набагато ефективніше.

### 1.13.2 СТРАТЕГІЧНИЙ ШІ

Стратегічний ШІ – це ШІ вищого порядку, він управляє цілою армією і виробляє оптимальні стратегії.

Стратегічний ШІ зазвичай застосовується в стратегіях в реальному часі, але останнім часом його все частіше реалізують і в тактичних шутерах від першої особи. Керований гравцем командир може бути окремою системою або може бути налаштований як порожній об'єкт, який не має місця і графічного зображення, але оновлюється міркує.

Командири підкоряються ієрархічним системам правил і кінцевим автоматам, які керують такою діяльністю як збір ресурсів, вивчення дерева технологій, створення армії і т.д. Як правило, для базової підтримки ігрових елементів не потрібно особливо складних роздумів. Інтелект потрібен головним чином при взаємодії з іншими гравцями.

Управління цією взаємодією (або боєм) – ось де перш за все працює ШІ. Командир повинен вивчити карту гри, щоб виявити гравця, виявити основні області інтересу, наприклад вузькі проходи, вибудувати оборону і проаналізувати оборону іншого гравця. Як саме це зробити? Очевидної відповіді на це питання немає, але для спрощення можна використовувати карти рішень.

### Карти рішень

Карти рішень є двомірні масиви, приблизно відповідні ігровій карті. Кожна частина масиву відповідає певній області в грі і містить важливу інформацію про цю область. Ці карти допомагають стратегічному ШІ приймати грамотні рішення щодо гри в цілому.

### Карти ресурсів

Карти ресурсів містять інформацію про розташування ресурсів в стратегічній грі. Дані про місця зосередження ресурсів на карті можуть вплинути на багато рішень командира: де розширювати базу, де розгортати додаткові бази (ресурси поруч з базою командира), де противник з найбільшою ймовірністю буде розширювати свою територію (ресурси поруч з базою противника), в яких місцях найбільш вірогідні зіткнення за оволодіння ресурсами (ресурси посередині між своєю базою і базою противника).

Отримання інформації про кількість можливих доступних ресурсів також впливає на рішення про те, які юніти слід підтримувати і як потрібно розгортати армію. При дефіциті ресурсів слід обережніше використовувати кожен юніт, оскільки менша ймовірність отримати поповнення. При надлишку ресурсів можна використовувати стратегії масового створення дешевих юнітів або створення дорогих потужних юнітів.

### Карти цілей

Ці карти містять інформацію про цілі командира, наприклад розташування баз противника, постановку цілей на карті (підірвати об'єкт такий-то, захистити об'єкт такий-то, зламати комп'ютер там-то і т.п.) та про найважливіші юніти в армії нашого командира (головна база, юніти-герої і т. д.). Відстеження цієї інформації допомагає командиру ефективніше управляти своєю армією. Місця, які потребують захисту, слід оточувати оборонними спорудами і поруч з ними завжди слід розташовувати загони військ. Цілі, що підлягають атаці, слід розвідувати і вивчати, як вони захищені. Аналіз оборони, влаштованої навколо цілей, потрібно виконувати для вироблення оптимального способу подолання цієї оборони. На основі цих даних утворюється наріжний камінь всіх військових ігор – карти конфліктів.

### Карти конфликтів

Карти конфліктів використовуються і оновлюються набагато частіше, ніж всі названі вище карти. На картах конфліктів відслідковуються всі битви на даному рівні гри. Кожного разу, коли один з юнітів командира вступає в бій з противником, цей юніт оновлює карту конфліктів, передаючи такі дані як тип конфлікту, його силу, можливості і кількість юнітів.

Аналіз цієї інформації допоможе зробити потрібні висновки про ефективність розгорнутої оборони і нападу, а також про необхідні контрзаходи (залучення додаткових юнітів).

## 1.14 СТВОРЕННЯ ТА ЗАСТОСУВАННЯ КАРТ

Карти складаються юнітами зі складу армії командира. Правила, що керують штучним інтелектом, повинні передбачати відправку розвідників на якомога більш ранньому

етапі, оскільки це дозволить приступити до створення карт. Продуманий ШІ періодично перевіряє актуальність карт. На ранніх етапах гри, коли підтримкою карт займається всього кілька юнітів, оновлення не повинно утворювати істотне навантаження на ігровий движок. На більш пізніх етапах гри, коли інформацію одночасно видають десятки або сотні юнітів, можливе зниження продуктивності.

Втім, добитися швидкого оновлення карт рішень не так важко. Досить помістити систему карт рішень в окремий потік. В ідеалі кожен гравець, керований штучним інтелектом, повинен мати у своєму розпорядженні власним потоком для обробки власного набору карт рішень. Значний приріст продуктивності буде досягнутий в тому випадку, якщо всі об'єкти вже розділені на кілька потоків. Потоки з картами рішень будуть обробляти тільки запити від повідомлень поновлення розпаралелених об'єктів.

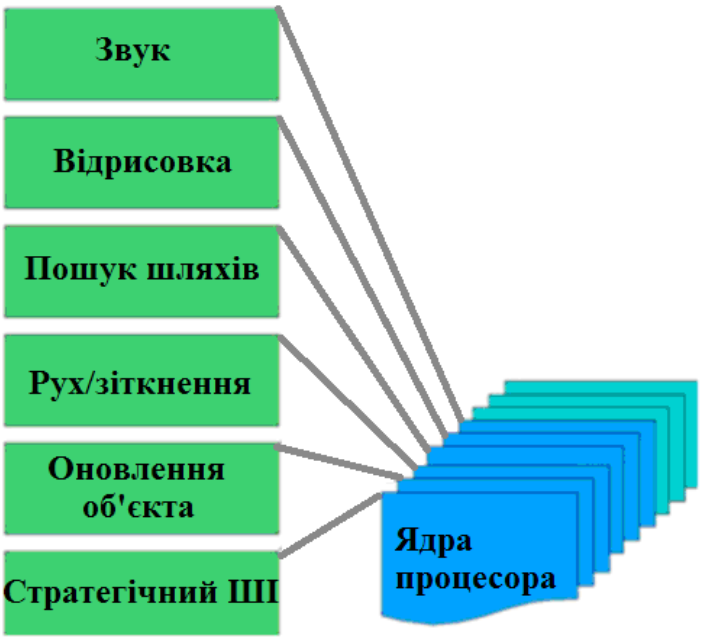
## 1.15 НАЙБІЛЬША ЕФЕКТИВНІСТЬ ШІ: ОБРОБКА ПОТОКІВ

Якою би прекрасною не була ваша система штучного інтелекту, вона марна, якщо уповільнює гру. Ефективне програмування та різні прийоми по оптимізації забезпечують деяке прискорення, але одних лише цих заходів недостатньо.

При роботі з системою, де встановлено кілька процесорів (процесор з декількома ядрами), можна розділити роботу між ними. Для цього є два способи: розпаралелювання задач (функціональне розпаралелювання) і розпаралелювання даних.

### 1.15.1 РОЗПАРАЛЕЛЮВАННЯ ЗАДАЧ

Найпростіший спосіб пристосувати додаток до багатопотокової архітектурі – розділити його на окремі задачі.




Залежно від потреб гри може бути багато різних завдань, кожному у тому числі можна надати окремий потік. Тут ми розглядаємо три такі завдання: пошук шляхів, стратегічний ІІ і власне система об'єктів.

### Пошук шляхів

Система пошуку шляхів може бути реалізована таким чином, що кожен об'єкт, який намагається знайти шлях, викликає свій власний алгоритм пошуку шляхів кожен раз, коли в цьому виникає необхідність. Такий метод буде працювати, але в цьому випадку движок буде чекати завершення роботи алгоритму пошуку шляхів при кожному запиті шляху. Якщо виділити пошук шляхів в окрему підсистему, можна вивести її в окремий потік. Тепер система пошуку шляхів буде працювати як диспетчер ресурсів, в якому шляхи є ресурсами.

Будь-який об'єкт, якому потрібно знайти шлях, відправляє запит на пошук шляхів і негайно отримує «квитанцію» від системи пошуку шляхів. Ця квитанція є просто унікальний дескриптор, який система пошуку шляхів може використовувати для роботи. Після цього об'єкт продовжує займатися своїми справами до наступного кадру в ігровому циклі. Об'єкт може перевірити, оброблена чи його квитанція. Якщо так, то об'єкт отримує розрахований шлях; в іншому випадку об'єкт продовжує займатися своїми справами в очікуванні обробки шляху.

В системі пошуку шляхів квитанція використовується для відстеження запитів шляхів, поки система працює над ними, не впливаючи на продуктивність інших компонентів. У такого підходу є цікаве перевага – автоматичне відстеження всіх виявлених шляхів. Тому під час вступу запиту на знайдений раніше шлях система пошуку шляхів може просто видати квитанцію на вже існуючий шлях. Цей метод чудово підходить для систем, де безліч об'єктів запитують шлях, оскільки всі знайдені шляхи з великою ймовірністю будуть запитані багаторазово.

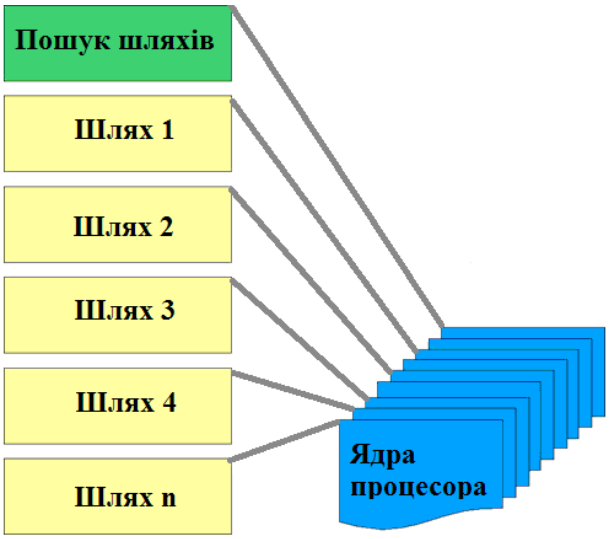
### Стратегічний ШІ

Добре, якщо система ШІ, що керує всім ходом гри в цілому, буде працювати у власному окремому потоці. Ця система зможе аналізувати ігрове поле і віддавати команди різних об'єктів, які зможуть отримувати і розпізнавати ці команди.

Система об'єктів у власному потоці буде зайнята роботою по збору інформації для карт рішень. Отримана інформація буде відправлена в систему стратегічного ШІ у вигляді запитів на оновлення карт рішень. При оновленні стратегічний ШІ буде аналізувати ці запити, оновлювати карти рішень і приймати рішення. При цьому не має значення, чи працюють ці дві системи (стратегічний ШІ і об'єкти) синхронно: будь-яка рассинхронизация буде незначною і не вплине на рішення ШІ (йдеться про розсинхронізації в межах 1/60 секунди, тобто, з точки зору гравця, реакція ШІ не забарилася ні на один кадр).

## 1.15.2 РОЗПАРАЛЕЛЮВАННЯ ДАНИХ

Функціональне розпаралелювання є дуже ефективним і використовує можливості систем з декількома ядрами. Але, на жаль, якщо в системі кількість ядер перевищує кількість завдань, то програма не використовує всю доступну обчислювальну потужність Тому переходимо до розпаралелювання даних, при якому одна функція може скористатися всіма доступними ядрами:




При функціональному розпаралелюванні ми брали автономний модуль і надавали йому окремий потік. Тепер ми розділимо одне завдання на частини і розподілимо їх обробку між різними потоками. При цьому продуктивність зростає пропорційно кількості ядер в системі. В системі 8 ядер? Відмінно! В системі 64 ядра? Ще краще! Функціональне розпаралелювання дозволяє позначати фрагменти коду як багатопотокові, після чого ці фрагменти працюють самостійно. При розпаралелюванні даних потрібна додаткова робота для узгодження. Наприклад, можна використовувати один потік ядра (головний потік), щоб відстежувати роботу всіх інших потоків. Підлеглі потоки будуть запитувати «роботу» в головному потоці, щоб виключити дублювання виконання однієї і тієї ж роботи.

Використання одного потоку ядра для управління розпаралелюванням даних є насправді гібридним підходом. Виходить, що головний потік використовує функціональне розпаралелювання, а потім розділяє дані між ядрами для розпаралелювання даних.

### Реалізація

У прикладі з системою пошуку шляхів ця система зберігає список запитаних шляхів. Потім система циклічно переглядає цей список і запускає функції пошуку шляхів для окремих запитів, зберігаючи їх у списку шляхів. Цей цикл можна розподілити по потоках так, щоб кожна ітерація циклу поділялася на різні потоки. Ці потоки будуть запускатися на першому ж доступному ядрі, що дозволяє використовувати всю доступну обчислювальну потужність. Ядро процесора має діяти тільки при відсутності роботи.

У таких системах є можливість отримання декількох запитів на одне і те ж завдання. Якщо ці запити розділені за часом, функція пошуку шляхів автоматично перевіряє, чи був вже оброблений такий запит раніше. При роботі з розпаралелюванням даних кілька запитів одного і того ж шляху може виникнути одночасно. Це може привести до дублювання роботи, в разі чого втрачається весь сенс багатопоточних обчислень.

Для виключення таких (та інших) випадків дублювання роботи система повинна від-

стежувати, як виконуються завдання, і видаляти їх з черги запитів тільки після завершення. Якщо ж надходить запит на шлях, який вже запитано, система повинна перевірити це і повернути існуючий шлях, призначений цій квитанції.

На утворення нових потоків витрачаються ресурси. Цей процес включає системні виклики до операційної системи (ОС). Коли у ОС до цього «доходять руки», вона виділяє необхідний код і створює потік. Це може зайняти багато часу (по відношенню до швидкості роботи ЦП). Тому немає сенсу створювати занадто багато потоків. Якщо запитана робота вже обробляється, то не потрібно запускати завдання. Крім того, якщо завдання нескладне (наприклад, пошук шляхів між двома точками, що знаходяться поруч одна з одною), то може не мати сенсу розділяти таку задачу на кілька потоків.

Ось як функціональний потік пошуку шляхів буде працювати і розділяти дані на потоки.

RequestPath(start, goal). Ця функція викликається зовні системи пошуку шляхів для отримання потоку. Ця функція виконує наступні завдання:

- о переглядає список виконаних запитів і визначає, чи був вже знайдений такий шлях (або близький шлях), потім повертає квитанцію на цей шлях;
- о переглядає список активних запитів (якщо шлях не був знайдений) для пошуку цього шляху;
- якщо шлях в ньому є, функція повертає квитанцію на що розраховується шлях;
- о створює новий запит і повертає нову квитанцію (якщо пошук по обом зазначеним вище списками не дав результату).
- CheckPath («ticket»). Використовуючи квитанцію, ця функція переглядає список виконаних запитів і знаходить шлях, для якого діє дана квитанція. Функція повертає дані про те, чи був знайдений такий шлях.
- UpdatePathFinde().Це керуюча функція, обробна витрати для потоків пошуку шляхів. Ця функція виконує наступні завдання.

Аналіз нових запитів. Можливо одночасне створення різними ядрами кількох запитів на один і той же шлях. Ця секція видаляє дубльовані запити і призначає кілька квитанцій (з різних запитів) одному і тому ж запиту.

Циклічний перегляд активних запитів. Ця функція переглядає всі активні запити і розподіляє їх по потокам. На початку і в кінці кожного циклу код позначається як потік. кожен потік:

- знаходить заппитанй шлях;
- зберігає його в списку готових шляхів разом з призначеними цим шляхом квитанціями;
- видаляє завдання зі списку активних завдань.

### Усунення конфліктів

При такому розподілі робіт можуть виникнути проблеми. Різним потокам потрібно проводити запис в чергу запитів; потокам даних потрібно додавати результати в список

готових шляхів. Все це може привести до конфліктів записів, коли один потік записує що-небудь в комірку пам'яті A в той же самий час, коли інший потік записує в цю ж комірку щось інше. Такий потік може привести до добре відомого «станом змагання».

Щоб уникнути конфліктів, можна помітити деякі частини коду як особливо важливі. При виконанні особливо важливого коду тільки один потік зможе отримати доступ до цієї секції коду в один момент часу. Всім іншим потокам, які збираються зробити те ж саме (отримати доступ до тієї ж області пам'яті), доведеться почекати. Така поведінка може призвести до серйозних проблем, таких як взаємне блокування, що виникає, коли кілька потоків блокують один одного, перешкоджаючи в отриманні доступу до пам'яті. Це рішення дозволяє уникнути взаємного блокування. Коли фактична робота потоку буде завершена, можна надавати доступ до важливої області пам'яті відразу ж по доступності без блокування інших секцій, які можуть бути необхідні іншим потокам.

### Синхронізація

Ми домоглися автономності всіх окремих підсистем ШІ і надали їм всі обчислювальні ресурси нашої системи. Все працює швидко, але чи все працює правильно?

Дії різних елементів гри повинні бути узгодженими. Ігровий движок повинен синхронізувати елементи гри. Не можна, щоб половина елементів гри працювала на пару кадрів швидше за інших елементів. Не можна, щоб юніти демонстрували бездіяльність, чекаючи обчислення шляху, тоді як юніти супротивника вже прийшли в рух.

Основний цикл ігрового движка займається двома класами дій: відрисовка і оновлення. При послідовному програмуванні синхронізація таких дій не становить труднощів. Спочатку все оновлюється, а потім те, що було оновлено, заново промальовується. При паралельних обчисленнях ситуація ускладнюється.

Оновлення руху (які часто виконуються на основі траєкторій) можуть бути оброблені на кілька кадрів швидше, ніж відрисовка. В результаті вийде «засмикана» анімація: об'єкти ігрового світу будуть не переміщатися плавно, а «перестрибувати» з одного місця на інше швидше, ніж слід. При пошуку шляхів, коли аналізується знімок положень різних об'єктів в світі, це може привести до обробки невірних вхідних даних.

Вирішення цієї проблеми полягає в синхронізації різних елементів і відрізняється витонченістю і простотою. Більш того, потрібні функції можуть вже бути вбудовані в більшість ігрових движків. При оновленні головного циклу гри відстежується глобальний індекс часу. Всі різноманітні потоки повинні обробляти запити тільки для поточного (і для минулого, але не для майбутнього) індексу часу.

Коли вся робота по отриманій задачі завершена для поточного індексу часу, потік може перейти в стан сну до нового індексу часу. Такий алгоритм не тільки гарантує синхронізацію різних ігрових елементів, а й вивільняє ресурси системи: потоки не завантажують ядра, коли в цьому немає необхідності. Тому завдання по переміщенню, здатне усувати зіткнення, обчислювати траєкторії і т.д., надасть свої обчислювальні

ресурси інших завдань, якщо впорається з роботою раніше. При цьому всі доступні ядра використовуються в повній мірі.

Завдання штучного інтелекту для ігор полягає в імітації поведінки об'єктів реального світу. І це зовсім не складно, якщо почати розгляд штучного інтелекту з базових компонентів – від низькорівневих правил і алгоритмів пошуку шляхів до більш високого рівня, на якому працює тактичний і стратегічний ШІ. При цьому, слід домогтися високої ефективності роботи системи ШІ, оптимізувати її для використання на комп'ютерах з великою кількістю обчислювальних ядер. Можливості ШІ в системі повинні обмежуватися тільки фактично наявними ресурсами обладнання, а не нездатністю використовувати ці ресурси. Тільки тоді ми зможемо створити більш цікавих і складних противників для гравців, які будуть з нетерпінням чекати продовження гри.

# ЛЕКЦІЯ 2. СПОСОБИ ПОДАННЯ ІНТЕЛЕКТУАЛЬНОЇ ЗАДАЧІ ТА МЕТОДИ ПОШУКУ РІШЕНЬ

Об'єкт, який може виконувати дії, діяти, називається агентом. Штучний інтелект, реалізований згідно з програмно-прагматичним напрямом у вигляді комп'ютерних програм, є комп'ютерним агентом. Раціональним агентом є агент, який діє на досягнення найкращого результату, а в умовах невизначеності – найкращого очікуваного результату. Для досягнення результатів агенту необхідно виконувати пошук рішень задач. Більшість підходів до пошуку рішень, які розглядаються дослідниками штучного інтелекту, базуються на методі спроб та помилок, тобто задачі вирішуються шляхом пошуку рішення серед множини можливих рішень.

Процес рішення задачі, як правило, складається з двох етапів: представлення задачі і пошуку рішення. Успіх рішення задачі в значній мірі визначається формою її подання. Форми подання (представлення) задачі можуть бути різними і залежать як від природи самої задачі, так і від її вирішувача.

Особливістю людського мислення є те, що етап рішення і форма подання задачі, які використовує людина, не завжди нею усвідомлюються. Тому етап представлення задачі часто випадає з поля зору людини, оскільки виконується на підсвідомому, інтуїтивному рівні. Але при спробі побудувати алгоритм рішення задачі та виконати його програмну реалізацію відразу усвідомлюється важливість та складність цього етапу. Слід особливо наголосити на важливості процесу формального представлення задачі, оскільки від нього залежить подальший хід її рішення та отримані результати.

Пошук форми представлення задачі, зручної для її машинного рішення, є творчим процесом, що важко формалізується. Найчастіше використовуються такі форми:

- представлення в просторі станів;
- представлення шляхом зведення задачі до підзадач;
- представлення у вигляді теореми.

## 2.1 СПОСОБИ ПОДАННЯ ІНТЕЛЕКТУАЛЬНОЇ ЗАДАЧІ

### 2.1.1 ПОДАННЯ ЗАДАЧІ В ПРОСТОРІ СТАНІВ

Повне подання задачі в просторі станів передбачає опис всіх можливих станів, визначення операторів, що відображають перетворення одних станів в інші, визначення множин початкових та цільових станів, визначення цільових критеріїв.

Опис стану в певній мірі відображає фізичні властивості вирішуваної задачі. Для задання станів можна використовувати різні форми опису: рядки символів, списки, масиви, вектори, графи. Форму опису станів слід вибирати такою, щоб застосування оператора перетворення одного стану в інший було досить простим.

Оператори перетворення можуть бути задані у вигляді таблиці, що зв'язує кожен вхідний стан з певними вихідним. Але для великих задач таке задання операторів є незручним у реалізації. Якщо для опису станів використовуються рядки (вектори), то опе-

ратор перетворення зручно задати у вигляді правила переписування одного рядка в інший.

Процедура пошуку рішення в просторі станів полягає в тому, щоб знайти послідовність операторів, яка перетворює початковий стан в цільовий. Рішенням задачі буде знайдена послідовність операторів перетворення.

Формально подання задачі визначається сукупністю трьох складових:

$$(S, Г, T),$$

де S – множина початкових станів;

Г – множина операторів перетворення, що відображають один стан в інший;

Т – множина цільових станів.

Простір станів доцільно представляти у вигляді графа, вершинами якого є стани, а дуги відповідають операторам. Оператори переводять один стан в інший. При представленні задач у просторі станів оператори є функціями опису станів. Значеннями функцій є нові описи станів.

Наведемо приклади представлення задачі у просторі станів [8]. Необхідно перетворити алгебраїчний вираз (A*B+C*D)/(B*C) у вираз A/C+D/B.

Стани можна представити як алгебраїчні вирази у формі двійкових дерев (рис. 2.1). Листові вершини такого дерева є змінними (A,B,C,D), а не листові – знаками арифметичних операцій (+,–,*,/).

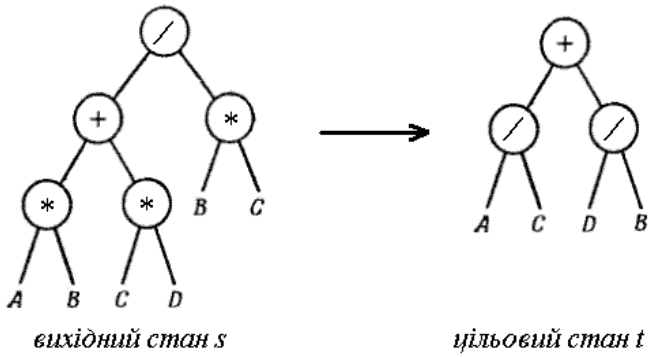



Рисунок 2.1 – Приклад подання задачі у вигляді графа

Задача полягає у перетворенні вихідного стану s у цільовий стан t шляхом застосування законів алгебраїчних перетворень (операторів перетворення).

Іншим способом є опис станів у вигляді рядків символів:

( / ( + ( * A B) ( * C D)) (* B C)) ⟶ ( + ( / A C) ( / D B)).

У даному випадку арифметичні оператори (+,–,*,/) використано як префіксні оператори, які виконують дію над операндами. Префіксна нотація операторів використовується у мовах програмування LISP, CLIPS.

При описі станів у вигляді рядків символів оператори є правилами переписування (продукціями). Правила переписування задаються у формі $S_i$ -> $S_j$, що означає можливість перетворення рядка $S_i$ у рядок $S_j$.

Оскільки задачі доцільно та зручно представляти у формі графів, розглянемо деякі поняття теорії графів [8]. Граф складається з множини N вершин. Деякі пари вершин є зв'язаними одна з одною лініями – ребрами. Якщо на всіх ребрах задано напрямок, то граф називають орієнтованим графом, а його ребра – дугами. Якщо дуга направлена від вершини $n_i$ до вершини $n_j$, то вершина $n_j$ називається дочірньою для вершини $n_i$, а вершина $n_i$ є батьківською для вершини $n_j$ . Якщо дві вершини є дочірніми одна для одної, то пара дуг об'єднується та стає ребром.

Якщо граф є формою представлення задачі у просторі станів, то його вершинами є описи станів, а дугами – оператори перетворення станів.

Шляхом довжиною k від вершини $n_{i,1}$ до вершини $n_{i,k}$ називається послідовність вершин $n_i$,1, $n_i$,2, ... , $n_i$,k, у якій кожна вершина $n_{i,j}$ є дочірньою для $n_{i,j}$-1, j=2, ... , k .

Якщо існує шлях від вершини $n_i$ до вершини $n_j$, то вершина $n_j$ є досяжною з вершини $n_i$ або є потомком $n_i$, а вершина $n_i$ є предком $n_j$.

Пошук послідовності операторів перетворення станів є еквівалентним задачі пошуку шляху на графі.

Якщо вершина ni графа пов'язана з вершиною $n_j$ дугою певної вартості, то це записується як $c(n_i, n_j)$, і означає вартість застосування відповідного оператора перетворення. Вартість шляху між двома вершинами визначається як сума вартостей усіх дуг цього шляху.

Проста задача формулюється як необхідність знаходження шляху (можливо з мінімальною або максимальною вартістю) від вершини s (початковий стан) до вершини t (кінцевий стан). У складних задачах необхідно знайти шлях від вершин з множини $S=\{s_i\}$ початкових станів до вершин множини $T=\{t_j\}$ цільових станів, застосовуючи оператори $Г=\{г_k\}$.

Граф може бути заданий неявно: задається кінцева множина вершин $\{s_i\}$ (початкових станів) та оператор Г, застосування якого до будь-якої вершини дає усі її дочірні вершини та вартості відповідних дуг. В процесі пошуку у просторі станів тієї послідовності операторів, яка дає рішення задачі, виконується перетворення в явну форму достатньо великої частини неявно заданого графа, такої, щоб у неї входила цільова вершина.

### 2.1.2 ЗВЕДЕННЯ ЗАДАЧІ ДО ПІДЗАДАЧ

Подання задачі, що зводить задачу до підзадач, передбачає розбиття вихідної задачі на множину підзадач, рішення певної підмножини яких дає рішення вихідної задачі [8]. У цьому випадку маємо справу з описами задачі та описами її підзадач. Оператори переводять опис задач до описів підзадач. Для опису задачі можна використовувати такі ж структури даних, як і для опису станів – рядки символів, списки, дерева, масиви, вектори та ін.

Для рішення кожної з підзадач можуть бути застосовані методи пошуку в просторі станів або, в свою чергу, підзадача може бути також розбита на свої підзадачі. Кількість рівнів розбиття теоретично не обмежена. На практиці розбиття триває до отримання на нижньому рівні множини задач (підзадач), спосіб вирішення яких відомий. Такі за-

дачі називають елементарними, а їх рішення є тривіальним. Таким чином, виконується пошук рішень у просторі множини задач.

Підхід зведення задач до підзадач називається редукцією задачі. При такому підході, як і при пошуку рішень в просторі станів, значну роль відіграє метод спроб та помилок.

Формально опис задачі можна виконати за допомогою трійки:

(S,F,G),

де S – множина початкових описів задач;

F – множина операторів перетворення, що відображають одну задачу в іншу;

G – множина цільових описів задач.

Якщо задача має тільки одну підзадачу, маємо найпростіший випадок заміни однієї задачі іншою, їй еквівалентною.

При розбитті задач на підзадачі може існувати багато операторів, застосування кожного з яких породжує альтернативні множини підзадач.

Деякі підзадачі можуть не мати рішення, у таких випадках пробують застосувати інший оператор перетворення. Метою цього процесу перебору операторів є побудова такої множини елементарних підзадач, усі члени якого є розв'язуваними.

Розбиття задач на підзадачі зручно представляти у вигляді графу – графу редукції задачі. Вершини графа відповідають задачам, а дуги – операторам редукції задач. Кореню відповідає вихідна задача, вершинам 1-го рівня – задачі, породжені вихідною задачею, вершини 2-го рівня – підзадачі вершин першого рівня і т.д.

Існують два типи структур взаємозв'язку підзадач: І-структури та І-АБО-структури. У структурах типу І для вирішення основної задачі потрібно вирішити всі підзадачі. У структурах І-АБО підзадачі розбиваються на групи, всередині яких вони пов'язані відношенням І, а між групами – відношенням АБО. У цьому випадку для рішення вихідної задачі досить вирішити всі підзадачі тільки якої-небудь однієї групи.

Вершини, що відповідають елементарним задачам і мають рішення, називаються заключними.

Вершини, які не мають дочірніх вершин і не є заключними, називаються тупиковими. Тупиковим вершин відповідають задачі, які в рамках даного подання неможливо розв'язати.

Таким чином, заключні вершини є розв'язуваними, а тупикові – нерозв'язуваними.

Вершина, що не є ні заключною, ні тупиковою, буде розв'язуваною тоді і тільки тоді, коли всі її дочірні вершини можна розв'язати, якщо вони є пов'язаними або хоча б одна з дочірніх непов'язаних вершин є розв'язуваною.

Очевидно, завдання А є вирішуваною в тому і тільки в тому випадку, якщо вершини Н і І є заключними або заключними є вершини G і F або G і Е.

Граф редукції завдання може бути заданий у явному вигляді (рис. 2.2). Але частіше він, як і граф стану, задається в неявному вигляді за допомогою опису вихідної завдання і операторів редукції.

Наприклад, на графі редукції задачі (рис. 2.2, а) задача А може бути вирішена, або якщо буде вирішена задача В, або якщо будуть вирішені задачі С і D, або задача Е. Задачі В, С і D та Е є підмножинами елементарних задач. Дуга об'єднує задачі, що належать до однієї підмножини. На основі графу редукції задачі був побудований І-АБО граф (рис. 2.2, б). При побудові І-АБО графу вводять допоміжні вершини, які стають батьківськими для однієї множини підзадач. У нашому прикладі це вершина F.

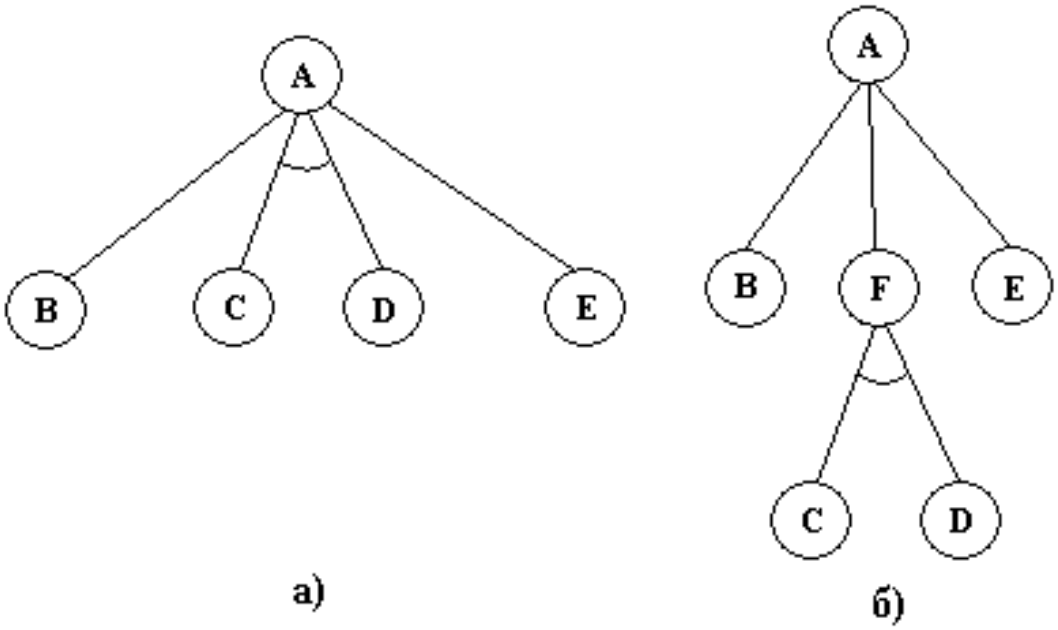



Рисунок 2.2 – Граф редукції задачі та І-АБО граф

Дочірні вершини певної даної вершини повинні бути або усі І-вершинами або усі АБО-вершинами.

Структурно граф редукції задачі відрізняється від графа простору станів тим, що в ньому є вершини, пов'язані дугами. Пов'язані вершини називають І-вершинами, непов'язані – АБО-вершинами, а граф називається графом типу І-АБО.

Одна з вершин І-АБО графу є початковою та являє собою опис вихідної задачі. Вершини, що є кінцевими (заключними), представляють описи елементарних задач. Метою пошуку на І-АБО графі є показати, що початкова вершина є вирішуваною.

Вершина І-АБО графу є вирішуваною, коли:

1) заключні вершини (елементарні задачі) є вирішуваними;

2) якщо вершина не є заключною, а її дочірні вершини пов'язані зв'язкою АБО, то дана вершина є вирішуваною, коли є вирішуваною хоча б одна з її дочірніх вершин;

3) якщо вершина не є заключною, а її дочірні вершини пов'язані зв'язкою І, то дана вершина є вирішуваною, коли є вирішуваною кожна з її дочірніх вершин.

Граф рішення складається з пов'язаних вирішуваних вершин і є підграфом вихідного І-АБО графу. Побудований граф рішення показує, що вихідна задача є вирішуваною (рис. 2.3).

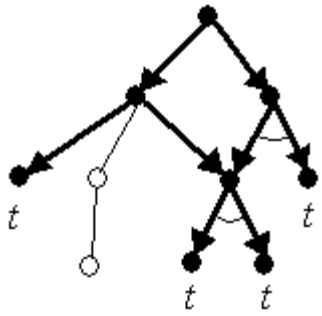



Рисунок 2.3 – Приклад І-АБО графу та графів рішення

На рисунку 2.3 дуги графу рішення зображено жирними лініями зі стрілками. Вершини, що мають рішення, зафарбовано чорним. Не зафарбованими є вершини, що відповідають описам задач, рішення яких неможливе. Літерою t позначено вершини, що є елементарними задачами. З наведеного графу видно, що вихідна задача має більше одного рішення, тобто більше одного шляху до заключних вершин.

Граф, зображений на рис. 2.3, задано у явній формі, що трапляється рідко. Найчастіше доводиться мати справу з неявним описом, тобто заданням опису вихідної задачі, а для побудови І-АБО графу застосовувати оператор Г, за допомогою якого рекурсивно виконується побудова поточних дочірніх вершин, тобто отримання описів підзадач. Оператор F являє собою множину усіх операторів, які застосовуються для зведення задач до підзадач.

### 2.1.3 ПОДАННЯ ЗАДАЧІ У ВИГЛЯДІ ТЕОРЕМИ

Багато задач (логічного типу та інші) можуть бути сформульовані як теореми, що підлягають доведенню. До доказу теорем можуть бути зведені головоломки, ігрові задачі, а також ряд практичних задач, таких як прийняття рішень, планування дій. Для рішення таких задач може бути використано логічний аналіз та логічні міркування. Для автоматизації логічних міркувань доцільно застосовувати логіку предикатів, тобто формальний апарат, який дозволяє формулювати посилання та робити логічні висновки за допомого логічного виведення.

Представляючи задачу у вигляді теореми необхідно розділити знання на чотири частини:

- множину висловів, в якій визначаються важливі факти про проблемну область даної задачі;
- множину рівнянь, відомих як правила перезапису;
- множину аксіом, які надають фонові знання про проблемну область; визначення межі між тим, що має увійти до складу задачі, і тим, що належить до фонових знань (і тому має увійти в число корисних аксіом) та передається на розсуд аналітика;
- множину параметрів і виразів, які визначають стратегію управління; аналітик може задавати евристичну функцію для управління пошуком і функцію фільтрації для усунення деяких підцілей які не представляють інтересу.

Числення предикатів першого порядку – це формальна система, яка дозволяє формувати правильно побудовані формули та виконувати пошук нові вірні логічні ствердження.

Формальну систему можна представити так:

(T,P,A,R),

де T – множина базових елементів;

P – множина синтаксичних правил;

A – множина аксіом;

R – множина правил виведення.

Множина T – це кінцева або нескінченна множина елементів різної природи. Елементами множини T є алфавіт формальної системи, на основі якого виконується побудова усіх інших частин формальної системи. На множину T не накладається ніяких обмежень. Для T повинна існувати процедура перевірки приналежності деякого елемента множині T.

Множина P дозволяє будувати з елементів T синтаксично правильні сукупності базових елементів. На множину P також не накладається ніяких обмежень. Для P повинна існувати конструктивна процедура, яка дозволяє за кінцеву кількість кроків дати однозначну відповідь на питання, чи є синтаксично правильною дана сукупність елементів множини T. Такі сукупності називаються правильно побудованими формулами (ППФ).

Серед усіх ППФ виділяють підмножину аксіом A. Для A повинна існувати процедура,-що дозволяє для будь-якої ППФ визначити, чи є вона аксіомою формальної системи.

R – це кінцева множина відношень між ППФ, що є правилами виведення. Правило виведення – це відношення на множині формул. Якщо з формул $F_1,F_2,...,F_n$ безпосередньо виводиться формула F, то це можна записати у вигляді:

$$F_1,F_2,...,F_n => F$$

Передумова Наслідок

Застосовуючи нові правила виведення до множини аксіом A, можна отримувати нові ППФ, до яких можна знову застосовувати правила з множини R. Це дозволяє виконувати виведення нових ППФ.

Множину R називають множиною семантичних правил.

Множина ППФ, отриманих після застосування правил до аксіом, називається множиною семантично правильних сукупностей.

Висновком формули B з формул $A_1,A_2,...,A_n$ називається послідовність ППФ $F_1,F_2,...,F_n$ така, що $F_m$ = B, а для будь-якого i (1<=i<=m) Fi є або аксіомою формальної системи, або однією з початкових формул $A_1,A_2,...,A_n$, або безпосереднім висновком $F_1,F_2,...,F_{i-1}$, отриманим за допомогою правил виведення.

Деяка ППФ F є виводимою у формальній системі, тобто є теоремою формальної системи, якщо існує виведення, у якому останньою формулою є F. Це записується так:

$$F_1,F_2,...,F_n \vdash F.$$

Якщо існує ефективна процедура, яка дозволяє за даною ППФ F встановлювати, чи існує її виведення у формальній системі, то така формальна система називається розв'язуваною, інакше – нерозв'язуваною.

Для того, щоб правильно побудованій формулі надати зміст, її необхідно інтерпретувати як певне ствердження, що стосується даної проблемної області. На цій області доцільно мати функції, які пов'язують з кожним константним символом в ППФ – певний символ з непустої множини предметної області; з кожною функціональною буквою в ППФ – певну конкретну функцію предметної області; з кожною предикатною буквою в ППФ – певне відношення між елементами предметної області. Конкретизація області та названих відповідностей дає інтерпретацію (модель) ППФ.

При заданій ППФ та її інтерпретації формулі приписується значення true або false.

Процес рішення задачі виконується у такій послідовності:

формується множина вхідних істинних тверджень (аксіом) відносно умови задачі; твердження записуються у вигляді формул мови числення предикатів;

будується гіпотеза відносно результату рішення задачі, яка також записується на мові числення предикатів та називається цільовим твердженням;

аксіоми комбінуються між собою на основі допустимих правил виведення для отримання множини нових тверджень;

виконується перевірка, чи не містить множина нових тверджень цільове твердження або його заперечення; якщо містить, то теорема вважається доведеною, і процес рішення задачі вважається завершеним; якщо цільове твердження або його заперечення не входять до множини нових сформованих тверджень, то отримана множина тверджень об'єднується з множиною вихідних аксіом, і процес повторюється.

Такий метод гарантує знаходження рішення, якщо воно існує, оскільки виконується повний перебір усіх варіантів. Застосування цього методу обмежується розмірами ресурсів часу та пам'яті. Для управління пошуком рішення та обмеження кількості можливих варіантів застосовують стратегії вибору пар аксіом або проміжних тверджень в процесі виведення.

Формальну систему можна розглядати як генератор нових знань. У цьому випадку з множини аксіом A (початкові знання, що зберігаються у системі штучного інтелекту) виводяться за допомогою правил виведення нові, похідні знання. Ланцюжки символів, які виділяються при цьому у пам'яті комп'ютера, являють собою конкретні знання про предметну область, і називаються образами. Ланцюжки символів, які є передумовами правил, називаються зразками. Якщо образ та зразок є порівнянними, то в базі знань з'являється новий елемент – висновок правила.

Засоби автоматичного доведення теорем можуть застосовуватися для вирішення задач, пов'язаних з перевіркою та синтезом як апаратних, так і програмних засобів, оскільки для обох цих проблемних областей можуть бути передбачені правильні варіанти аксіоматизації. Тому дослідження на основі доведення теорем проводяться не тільки в штучному інтелекті, але й в таких областях, як проектування апаратних засобів, мови програмування та розробка програмного забезпечення. У випадку програмного забезпечення аксіоми визначають властивості кожного синтаксичного елементу мови програмування.

## 2.2 СТРАТЕГІЇ ТА МЕТОДИ ПОШУКУ РІШЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ЗАДАЧІ

Функціонування багатьох інтелектуальних систем носить цілеспрямований характер. Типовим актом такого функціонування є рішення задачі планування шляху досягнення потрібної мети з деякої фіксованої початкової ситуації. Результатом виконання задачі повинен бути план дій –частково упорядкована сукупність дій. Такий план нагадує сценарій, в якому як між вершинами існують відношення типу: «ціль-підцілі» «ціль-дія», «дія-результат» і т.п. Будь-який шлях у цьому сценарії, який веде від вершини, відповідної поточній ситуації , в будь-яку з цільових вершин, визначає план дій.

Стратегія управління інтелектуальної системи визначає характер пошуку необхідних знань в інтелектуальній системі. Стратегія управління є засобом міркування або засобом виведення на основі знань, що містяться в базі знань ІС.

Механізм виведення реалізує загальну настроювану схему пошуку рішень. Стратегія управління реалізує різноманітні види управління ходом виконання виведення, тобто визначає послідовність і зміст дій при реалізації механізму виведення (рис. 2.4). Стратегія управління може являти собою частину метазнань.

У стратегії ***пошуку, керованого зразком***, для виявлення знань в ході виведення використовують зразки – формати, що визначають умови активізації відповідних модулів структурованих знань. Такі системи зазвичай грунтуються на правилах, але можуть базуватися і на знаннях у вигляді семантичних або фреймових мереж. В якості модулів, керованих зразками, використовуються відповідно блоки правил або підмережі.

Пошук за зразком (PS-пошук) об'єднує сімейство методів чисельної оптимізації, які не вимагають, щоб градієнт пошуку був оптимізованим. PS-пошук може бути застосований до функцій, які не є безперервними або диференційовними. Такі методи оптимізації, також відомі як методи прямого пошуку, вільних похідних, або чорного ящику.

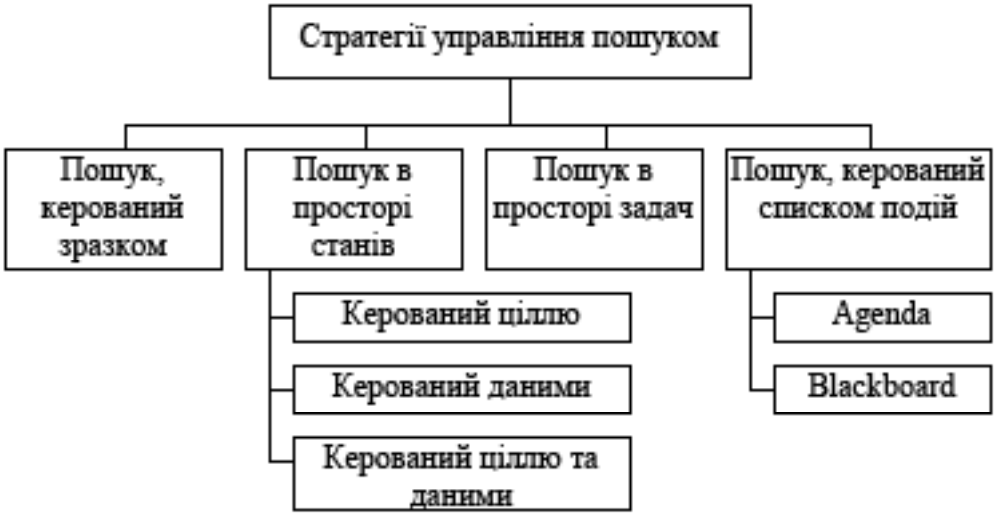



Рисунок 2.4 – Стратегії управління пошуком рішень інтелектуальної задачі

Процес реалізації стратегії пошуку за зразком проходить через такі стадії:

1) **вибір** – з робочої області БЗ вибираються модулі і дані, які можуть мати відношення до даної ситуації, це скорочує простір пошуку;

2) **зіставлення** – з набору модулів і даних, відібраних на етапі 1, шляхом зіставлення зі зразками вибираються такі, для яких задовольняються умови виконання, тобто формується конфліктний набір;

3) **розв'язання конфліктів** – приймається рішення про те, які саме модулі з конфліктного набору будуть виконуватися згідно з чинною в системі стратегією вирішення конфліктів;

4) **виконання** – запускаються модулі, вибрані на стадії 3; в результаті виконання модифікуються елементи і структури даних робочої пам'яті, виводяться рекомендації або рішення користувачеві, можлива зміна бази знань.

Відомо два методи управління виведенням:
- встановлення обмежень на генерацію конфліктного набору;
- визначення алгоритму вирішення конфлікту.

Спосіб встановлення обмежень на генерацію конфліктного набору залежить від змісту відповідних модулів структурованих знань. В якості цього способу можуть використовуватися:
- метод застосування метазнань, тобто знань про зміст модулів структурованих знань; метазнання можуть містити відомості про те, які модулі не слід включати в конфліктний набір при рішенні конкретної проблеми та про умови застосування модулів;
- метод з попередньою розбивкою модулів структурованих знань на окремі категорії і впевних ситуаціях досліджується можливість застосування модулів, що належать лише до певної категорії; досліджується можливість застосування модулів тільки з тієї групи, в межах якої виконується конкретна умова застосування.

У пошуку за зразком використовуються такі групи методів та алгоритмів:
- пошук за зразком (оптимізація);
- розпізнавання образів за зразком;
- видобування зразків даних (належить до технології Data Mining) – використовує асоціативні правила;
- пошук за зразком у тексті; виконує пошук підрядків у рядках тексту за зразками;
- нечіткий пошук за зразком у тексті;
- Bitap-пошук – пошук у тексті компонентів, приблизно схожих на зразок;
- k-оптимальне знаходження зразків (належить до технології Data Mining) – пошук k-зразків, що достатньо часто наявні у даних, параметр k-визначає міру інтересу до пошуку;
- пошук найближчого сусіда – пошук об'єктів, що найбільш схожі на зразок, схожість визначається оцінною функцією;
- «очне яблуко» – пошук специфічних компонентів за допомогою візуальних пристроїв; застосовуються методи vgrep (пошук схожих характеристик) та vdiff (пошук відмінностей між порівнюваними об'єктами).

У разі **пошуку в просторі станів** (SS-проблема) вважається заданим деякий простір ситуацій. Опис ситуацій включає стан зовнішнього світу і стан інтелектуальної системи, що характеризуються рядом параметрів. Ситуації утворюють деякі узагальнені

стани, а дії інтелектуальної системи або зміни у зовнішньому середовищі призводять до зміни актуалізованих в даний момент станів. Серед узагальнених станів виділені початкові стани (зазвичай один) і кінцеві (цільові) стани. SS-проблема полягає у пошуку шляху, що веде з початкового стану в один з кінцевих.

Якщо, наприклад, інтелектуальна система призначена для гри в шахи, то узагальненими станами будуть позиції, що складаютьсяна шаховій дошці. В якості початкового стану може розглядатися позиція, яка зафіксована в даний момент гри, а в якості цільових позицій – множина нічийних позицій. Відзначимо, що в разі шахів прямий перелік цільових позицій є неможливим. Матові та нічийні позиції описані мовою, що відрізняється від мови опису станів, що характеризуються розташуванням фігур на полях дошки. Саме це ускладнює пошук плану дій в шаховій грі.

При **пошуку в просторі задач** (PR-проблема) ситуація дещо інша. Простір утворюється в результаті введення на безлічі завданьвідносини типу: «частина – ціле», «завдання – підзадача»,»загальний випадок – окремий випадок» і т. п. Іншими словами, простір завдань відображає декомпозицію задач на підзадачі(цілі на підцілі) . PR-проблема полягає в пошуку декомпозиціївихідної задачі на підзадачі, що приводить до завдань, вирішення яких системі відомо. Наприклад, ІС відомо, як обчислюються значення sin(x) і cos(x) для будь-якого значення аргументу і як проводиться операція ділення. Якщо інтелектуальній системі необхідно обчислити tg(x), то рішенням PR-проблеми буде подання цієї задачі у вигляді декомпозиції tgx=sin x/cos x (крім x=  /2+k  ).

Пошук, **керований списком подій**, реалізується методами дошки оголошень (blackboard) та порядку денного (agenda).

У разі використання пошуку, **керованого списком подій**, розробник не може заздалегідь передбачити послідовність викликів правил, методів та процедур, оскільки ця послідовність визначається на етапі виконання пошуку.

Методи пошуку рішень, керовані подіями, характеризуються великою гнучкістю в сенсі вибору порядку виконання операцій. Характерно те, що послідовність дій часто визначається конкретною ситуацією і залежить від потоку повідомлень про події в системі. Система, що використовує методи пошуку, керовані подіями, знаходиться в стані очікування подій, точніше повідомлень про них. Повідомлення можуть надходити від різних джерел, але всі вони потрапляють в одну чергу системних повідомлень. Система постійно виконує цикл очікування повідомлень.

Стратегія дошки оголошень (blackboard) – спосіб управління асинхронними паралельними процесами рішення задач, при яких інформація про процеси, що закінчилися, та отримані результати «вивішується» на дошці оголошень, до якої мають незалежний доступ всі процеси, які очікують потрібних результатів. Дошка оголошень часто використовується в експертних системах, інтелектуальних роботах та інших інтелектуальних системах.

Системи з дошкою оголошень організують взаємодію джерел знань через загальну область пам'яті – так звану дошку оголошень. У неї поміщаються активні джерела знань, поточний план рішення, проміжні результати і поточні дані для вирішення завдання.

Дошка оголошень містить зону для предметної області і зону для планування. Зона для предметної області призначена для експертних знань з розв'язуваної проблеми в межах предметної області. Зона для планування містить міркування про функціонування.

Процес пошуку рішень у blackboard-системах, практично, являє собою процес планування формування на дошці оголошень гіпотез і їх перевірки.

Стратегія порядку денного (agenda) – передбачає формування упорядкованого списку правил, методів та процедур, готових до виконання.

Агенда-система працює з таким поняттям, як джерело знань. Джерела знання – це модулі, що містять в правилах переходу, окрім умови і дії, ім'я і набір параметрів планування, які визначають умови входження джерела знання в план. План являє собою послідовність або список модулів (джерел знань), які виконуються виходячи зі стану системи та дисципліни планування.

Агенда-система функціонує наступним чином. На початковій стадії вибираються джерела знань, які готові до застосування, тобто їх зразки співставні з поточними даними або умовні частини задоволені. Ці джерела знань вносяться до Агенда (тобто в список заявок). Далі з цього списку Агенда-система планує активізацію певного джерела знаннь. Планування здійснюється на основі відповідності значень параметрів планування дисципліні планування. По завершенні процесу визначають, виходячи зі стану системи, активізацію якого наступного джерела знань слід виконати. Реалізація виведення на основі Агенда-системи здійснюється доти, поки вся Агенда не буде вичерпана.

### 2.2.1 МЕТОДИ РІШЕННЯ ЗАДАЧ ЗА СТРАТЕГІЄЮ ПОШУКУ У ПРОСТОРІ СТАНІВ

Теорія графів – один з фундаментальних розділів дискретної математики. Графи є дуже продуктивним засобом інформаційного (математичного) моделювання структур систем і процесів, уявлення завдань інформаційного характеру. Інтелектуальна задача може бути подана у вигляді шрафу.

Вершини графа з'єднуються ребрами. Вершинами можуть бути об'єкти будь-якої природи: населені пункти, об'єкти на місцевості, стани об'ктів та ін.

Орієнтований граф (орграф) – граф, в якому зв'язки між вершинами зображені дугами. Дуга з'єднує пару вершин (v, w), де вершина в називається початком, а w – кінцем дуги.

Граф є повним, якщо він має n вершин, а кількість рèбер n*(n-1)/2.

Маршрут графа – це чергування послідовності вершин і ребер, в якій кожне ребро інцидентне двом вершинам.

Шляхом (маршрутом) в орграфі називається кінцева послідовність суміжних вершин і дуг, що з'єднують ці вершини. Довжина шляху (маршруту) – кількість дуг, що складають шлях.

#### Пошук на графі

При формулюванні задачі в просторі станів рішення отримують в результаті застосу-

вання операторів до описів станів до тих пір, поки не буде отримано вираз, який описує стан, що відповідає досягненню мети. Простори станів можуть зображуватися у вигляді графів. Мова графів застосовується для опису ефективних стратегій перебору (пошуку) в просторі станів. Всі методи перебору, які будуть розглянуті далі, можуть бути змодельовані за допомогою наступного теоретико-графового процесу:

- початкова вершина відповідає опису початкового стану;
- вершини, безпосередньо наступні за початковою, отримують в результаті використання операторів, які застосовні до опису стану, асоційованого з цією вершиною;

Нехай Г – деякий спеціальний оператор, який будує всі вершини, безпосередньо наступні за даною. Процес застосування оператора Г до вершини називають розкриттям вершини.

- від кожної наступної вершини до вершини, що її породила, йдуть вказівники; вказівники дозволяють знайти шлях назад до початкової вершини, вже після того, як виявлена цільова вершина.
- для вершин, наступних за даною, робиться перевірка, чи не є вони цільовими вершинами. Якщо цільова вершина ще не знайдена, то процес розкриття вершин (і установки вказівників)триває. При знаходженні цільової вершини вказівники переглядаються в зворотному порядку – від цілі до початку, що дає шлях рішення задачі в цілому.

Етапи, описані вище, представляють основні елементи перебору. При повному описі процесу слід задати порядок розкриття вершин.

Якщо вершини розкриваються в тому ж порядку, що і породжуються, то маємо повний перебір (breadth-first process). Якщо першою розкривається та вершина, яка була отримана останньою, то маємо перебір в глибину (depth-first process). Повний перебір і перебір в глибину називаються процедурами сліпого перебору, оскільки розташування цілі не впливає напорядок розкриття вершин.

Часто може існувати евристична інформація про глобальний характер графа і загальне розташування цілі пошуку. Використання такої інформації може підвищити ефективність пошуку, тобто направити пошук в бік цілі та розкривати насамперед найбільш перспективні вершини.

Сутність методів розглянемо на деревах. Деревом називається граф, кожна вершина якого має тільки одну батьківську вершину, за винятком вершини-кореня дерева, яка не має батьківської вершини. На дереві шлях від кореня до даної вершини є єдиним,оскільки при побудові нової вершини ми впевнені в тому, що така вершина ніколи не будувалася. Пошук на дереві можна надалі модифіковані для довільних графів.

### Методи повного перебору

У методі повного перебору вершини розкриваються в тому порядку, в якому вони будуються (рис. 2.5). Передбачається, що початкова вершина не задовольняє поставленої мети. Таку перевірку можна додати на початку алгоритму.

Використовуються два списки – ВІДКРИТО для породжених вершин і ЗАКРИТО для

вершин-батьків.

Вершини і вказівники, побудовані в процесі перебору, утворюють піддерево всього неявно визначеного дерева простору станів.

Очевидно, що в результаті повного перебору неодмінно буде знайдений найкоротший шлях до цільової вершині за умови, що такий шлях існує.

Можуть зустрітися задачі, в яких до рішення пред'являються інші вимоги, ніж вимога знаходження найкоротшого шляху. Присвоєння дугам дерева певних цін (з подальшим знаходженням шляху, що має мінімальну вартість) відповідає багатьом з узагальнених критеріїв.

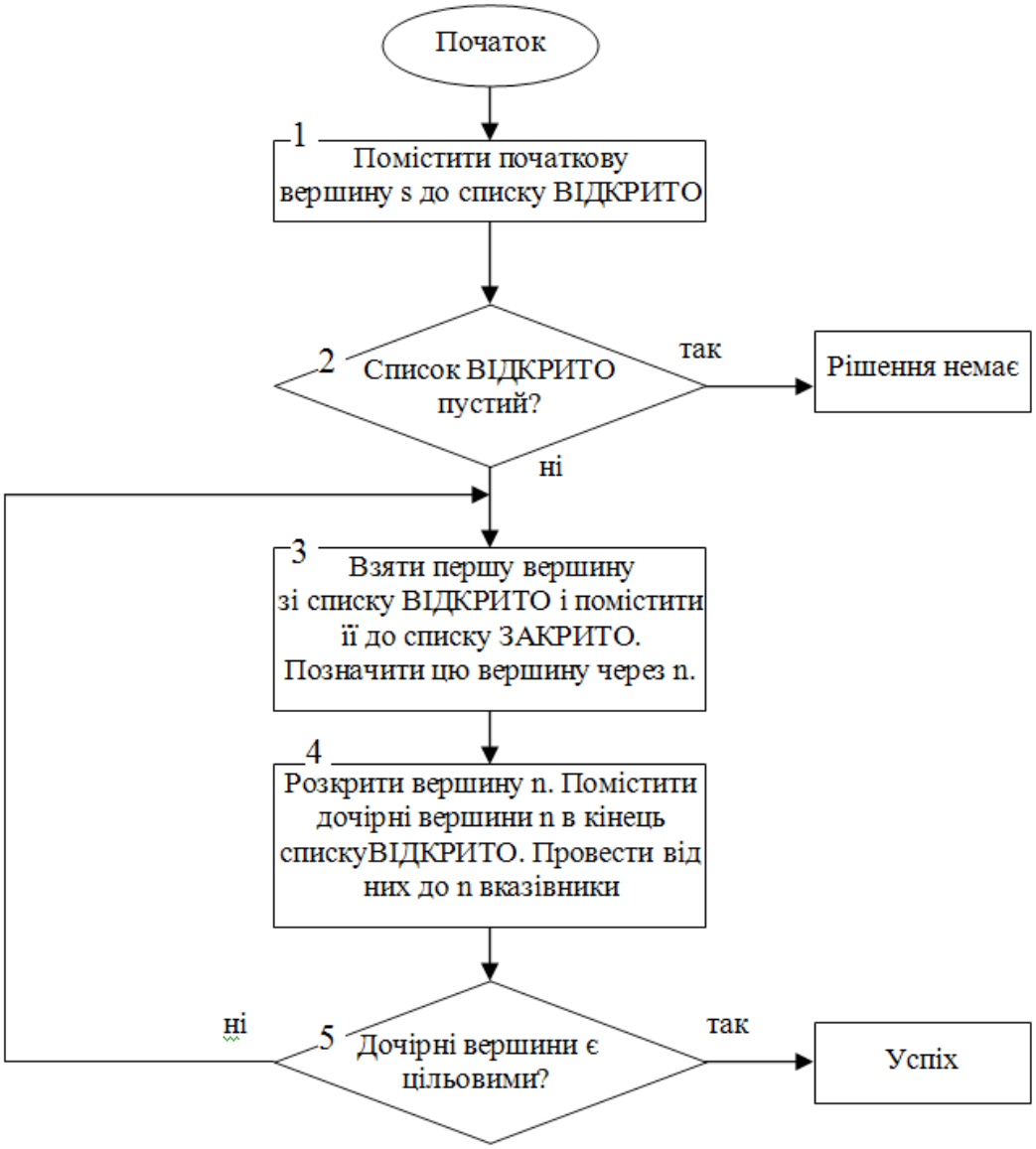



Рисунок 2.5 – Схема алгоритму повного перебору для дерева

Більш загальний варіант методу повного перебору називається методом рівних цін. Він дозволяє в усіх випадках знайти певний шлях від початкової вершини до цільової з мінімальною вартістю (рис.2.6). Передбачається, що задана функція вартості c(ni,nj), що дає вартість переходу від вершини ni до деякої наступнійвершині nj. У методі рів-

них цін для кожної вершини n в дереві перебору потрібно пам'ятати вартість шляху, побудованого від початкової вершини s до вершини n.

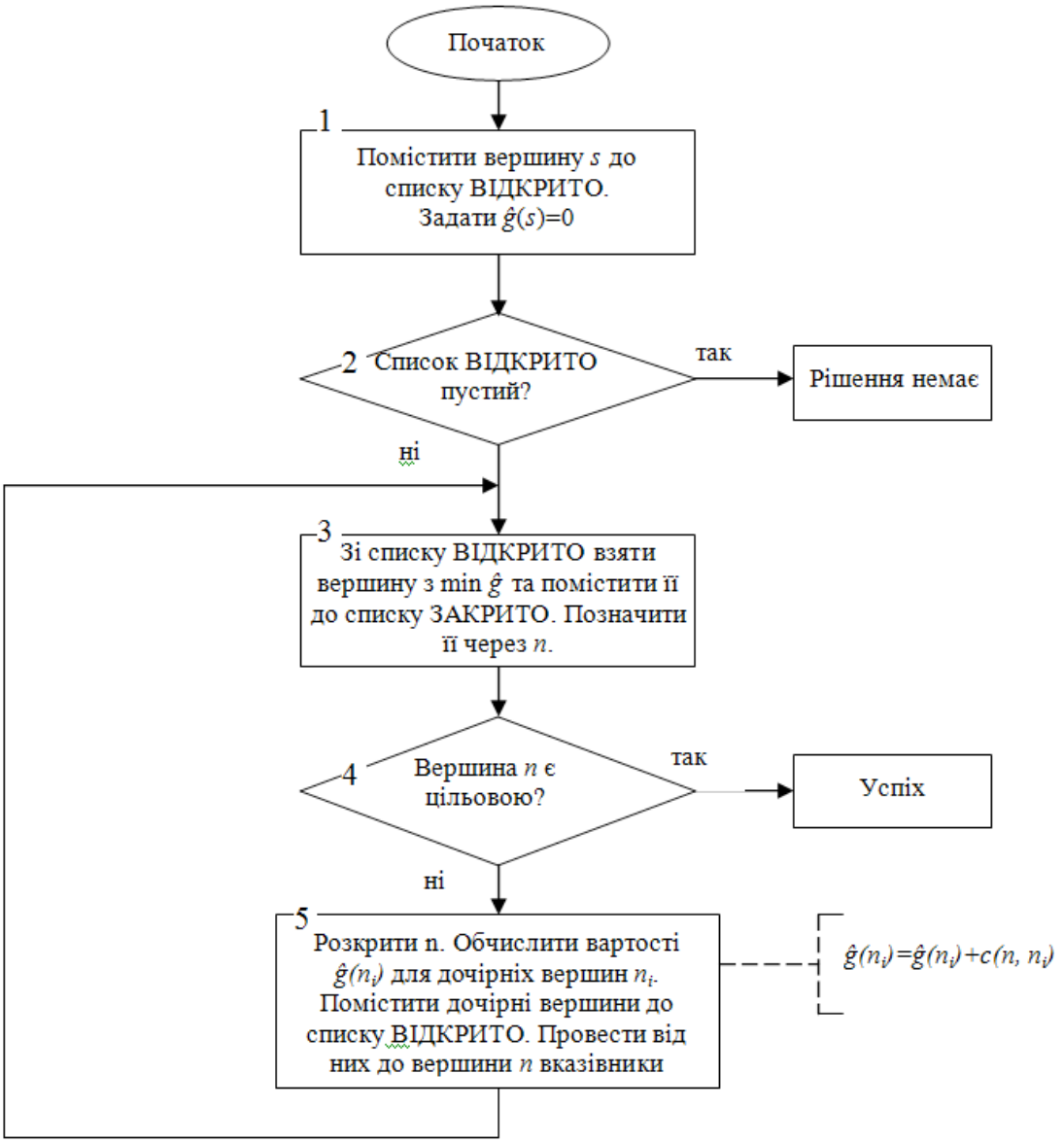



Рисунок 2.6 – Схема алгоритму рівних цін для дерев

Нехай ĝ(n) – вартість шляху від вершини s до вершини n на дереві перебору. У разі дерев перебору ми можемо бути впевнені, що ĝ(n) є до того ж вартістю того шляху, який має мінімальну вартість, тому що цей шлях є єдиним. У методі рівних цін вершини розкриваються в порядку зростання вартості ĝ(n).

Очевидно, що алгоритм рівних цін може бути використаний для пошуку шляхів мінімальної довжини, якщо покласти вартість кожного ребра дорівнює одиниці.

Якщо є кілька початкових вершин, то алгоритм модифікується: на кроці (1) усі початкові вершини поміщаються в список ВІДКРИТО.

Якщо стану, що відповідають поставленої мети, можуть бути описані явно, то процес перебору можна пустити в зворотномунапрямку, прийнявши цільові вершини в якості початкових і використовуючи звернення оператора Г.

Застосування алгоритму гарантує виявлення шляхів мінімальної вартості.

Алгоритм рівних цін може бути використаний для пошуку шляхів мінімальної довжини, якщо задати вартість кожного ребра рівною одиниці. Якщо є більше однієї початкової вершини, то алгоритм модифікується так, що на початку усі початкові вершини поміщаються у список ВІДКРИТО.

Якщо стани, які відповідають поставленій цілі, можуть бути описані явно, то процес перебору можна виконати у зворотному напрямку, прийнявши цільові вершини за початкові та використовуючи звернення оператора Г.

### Метод перебору в глибину

У методах перебору в глибину (рис.2.7) перш за все розкриваються вершини, які були побудовані останніми.

Глибина вершини в дереві визначається так:

глибина кореня дерева дорівнює нулю;

глибина будь-якої подальшої вершини дорівнює одиниці плюс глибина вершини, яка їй безпосередньо передує.

Таким чином, вершиною, яка має найбільшу глибину в дереві перебору, в поточний момент є та, яка повинна в цей момент бути розкрита. Такий підхід може призвести до проходження неперспективного шляху, тому потрібно ввести деяку процедуру повернення. Після того, як в ході обробки будується вершина з глибиною, що перевищує деяку граничну глибину, розкривають вершину найбільшої глибини, яка неперевищує цієї межі, і т.д.

В алгоритмі пошуку в глибину спочатку йде перебір вздовж одного шляху, поки не буде досягнута максимальна глибина. Потім розглядаються альтернативні шляхи тієї ж або меншої глибини, які відрізняються від нього лише останнім кроком. Після цього розглядаються шляхи, що відрізняються останніми двома кроками, і т.д.

При пошуку в глибину кожна альтернатива досліджується до кінця, без урахування інших альтернатив. Метод є поганим для «високих» дерев, тому що легко можна пройти повз потрібну гілку та витратити багато зусиль на дослідження «порожніх» альтернатив. При пошуку в ширину на фіксованому рівні досліджуються всі альтернативи, і тільки після цього здійснюється перехід нанаступний рівень.

Метод може виявитися гірше методу пошуку вглиб, якщо в графі всі шляхи, що ведуть до цільової вершини, розташовані приблизно на одній і тій же глибині. Обидва сліпих методу вимагають великої витрати часу і тому доцільно використовувати спрямовані методи пошуку.

### Використання евристичної інформації та оціночних функцій

Методи сліпого перебору, повного перебору і пошуку в глибину часто неможливо використовувати, оскільки при переборі доведеться розкрити занадто багато вершин, перш ніж буде знайдено потрібний шлях.

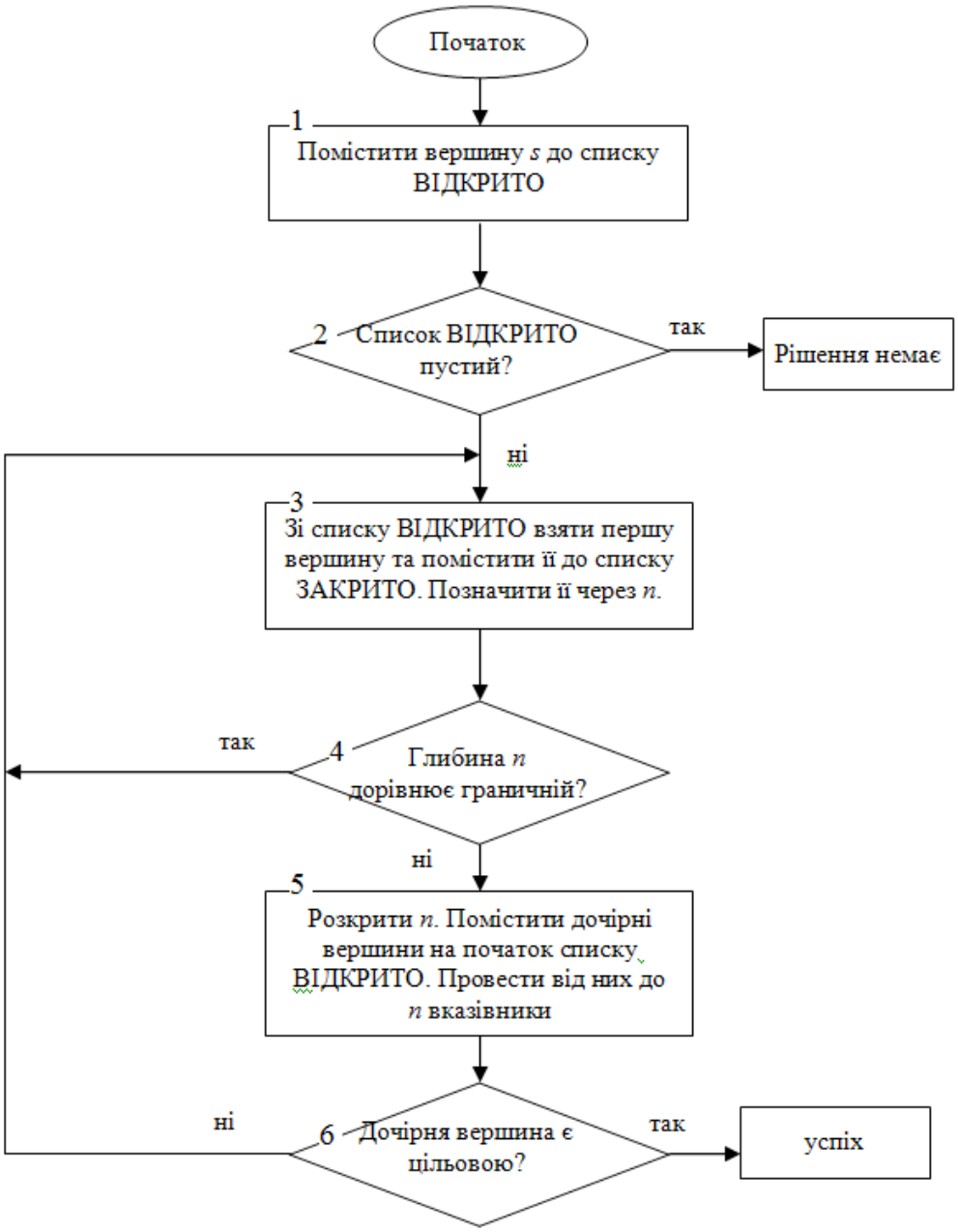



Рисунок 2.7 – Схема алгоритму пошуку в глибину для дерева

Для багатьох задач можна сформулювати чисто емпіричні правила, що дозволяють зменшити обсяг перебору. Такі правила залежать від специфіки наявної інформації про розв'язувану задачу. Така інформація називається евристичної інформацією, тобто інформацією, що допомагає знайти рішення. Методи пошуку рішень на основі евристичної інформації називаються евристичними методами пошуку.

Один із шляхів зменшення перебору полягає у виборі більш «інформованого» оператора Г, який не будує так багато «зайвих» вершин, які не відносяться до справи. Цей спосіб застосовується як в методі повного перебору, так і в методі перебору в глибину.

Інший шлях полягає у використанні евристичної інформації для модифікації кроку 5 алгоритму перебору в глибину. Список ВІДКРИТО слід відсортувати з урахуванням еврис-

тичної інформації, що дозволить відкривати найбільш підходящі вершини в першу чергу.

Більш гнучкий (і дорожчий) спосіб використання евристичної інформації полягає в тому, щоб, відповідно до деякого критерію, на кожному кроці змінювати порядок вершини списку ВІДКРИТО. Міра, що дозволяє оцінити «перспективність» вершини називається оціночною функцією.

У більшості практичних задач необхідно мінімізувати певну комбінацію вартості шляху і вартості перебору, необхідного для знаходження цього шляху.

Цікавими є методи перебору, при яких мінімізується така комбінація, усереднена по всіх задачах, які можуть зустрітися. При обчисленні цієї середньої вартості необхідно використовувати ваги, пропорційні частоті появи кожної задачі, тобто допускаються великі вартості для задач, які нечасто зустрічаються, якщо вони компенсуються меншими вартостями для задач, що зустрічаються часто. Якщо усереднена комбінаційна вартість одного методу перебору нижче відповідної вартості іншого методу, то говорять, що перший метод перебору має більшу евристичну силу, ніж інший.

Визначення евристичної вартості пошуку є складною, трудомісткою і не завжди розв'язуваною задачею. Тому вибір методу перебору визначається інтуїцією інженера по знаннях, підкріпленою експериментами з методами.

Оціночна функція повинна забезпечувати можливість ранжирування вершин-кандидатів на розкриття для виділення тієї вершини, яка з найбільшою ймовірністю знаходиться на кращому шляху до цілі.

Для визначення оціночної функції можна:
- визначити імовірність того, що вершина розташована накращому шляху;
- використовувати відстань або інші заходи відмінності між довільною вершиною і множиною цільових вершині т.д.

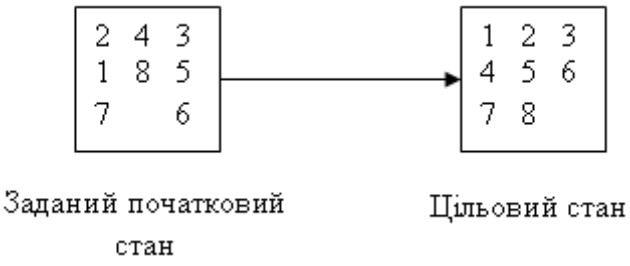
Одним із прикладів використання оціночної функції є алгоритм А*.

### Алгоритм А*

Більшість пошукових задач формулюються як задачі пошуку в просторі станів шляху від початкового стану задачі до цільового стану шляхом повторення певних перетворень (за допомогою операторів). Для організації пошуку зручно використовувати дерево пошуку (або його загальну форму – граф). Основні методи систематичного пошуку – вертикальний і горизонтальний пошук або пошук в глибину і пошук в ширину. В алгоритмі А* використані апріорні оцінки вартості шляху до цільового стану (різновид евристичних знань), що забезпечує високу ефективність пошуку.

Опис цього алгоритму зручно розглянути на прикладі гри у вісім. Ця гра – мініваріант гри в 15. На полі розміром 3х3 розміщено вісім пронумерованих шашок, мета гри – від заданого початкового стану перейти до цільового стану, як показано нижче.

На полі є один порожній квадрат: стан можна змінити, пересуваючи шашку зверху, знизу, праворуч або ліворуч на порожній квадрат. Отже, в цій грі є чотири оператори перетворення стану, відповідні одному з пересувань шашки напорожній квадрат. А раз так, то не потрібно стежити за пересуванням деякої конкретної шашки.




Тому змінимо точку зору, і будемо переміщувати порожній квадрат. Тепер просто визначити чотири оператори:

1) переміщення порожнього квадрата вліво (при цьому ліворуч є порожній квадрат);

2) переміщення порожнього квадрата вгору (при цьому вгорі є порожній квадрат);

3) переміщення порожнього квадрата вправо (при цьому справа є порожній квадрат);

4) переміщення порожнього квадрата вниз (при цьому внизу є порожній квадрат).

Використовуючи ці оператори, будемо здійснювати пошук в просторі станів. Розглянемо також наступну оціночну функцію як функцію, що регламентує вибір ефективного напряму пошуку. Нехай f(n) – вартість оптимального шляху до цілі від першої вершини (початкового стану) через n вершин дерева пошуку. Розділимо f(n) і представимо її наступним чином:

$$f(n) = g(n) + h(n),$$

де g(n) – вартість оптимального шляху від першої вершини до n-ї;

h(n) – вартість оптимального шляху від n-ї вершини до цілі.

Будемо вважати, що вартість переміщення однієї шашки дорівнює 1, а до цілі веде оптимальний шлях з мінімальною вартістю.

В процесі пошуку в загальному випадку не можна точно знати f(n), тому розглянемо апріорне значення f(n)_. У момент проходження n-ї вершини g(n) відомо, тому можна записати:

f(n)_ = g(n) + h(n)_,

де h(n)_ – апріорне значення h(n). Якщо уявити простір пошуку вгрі 8 у вигляді дерева, то g(n) – це глибина від першої вершинидо n-й вершини. В якості h(n)_, наприклад, виберемо число шашок, що знаходяться не на своїх місцях. Визначивши такимчином оціночну функцію, виберемо стратегію проходження вершин (застосування операторів), в яких значення функціїмінімальні. На рис. 2.8 показаний результат пошуку.

Вибираємо вершину з найменшим із значень оцінночної функції, зазначених цифрами збоку, застосовуємо оператор і розкриваємо вершину.

Потім створюємо дочірні вершини (при цьому не повертаємося до вже розкритих вершин). Повторюємо цю процедуру до цільового стану. Цифри зліва вгорі кожної вершини вказують на порядок розкриття вершин.

Тут важливо, що h(n)_ <= h(n), а саме, якщо апріорне значення вартості n-ї вершини

до цілі (h(n)_) менше або дорівнює дійсній вартості оптимального шляху до цілі (h(n)), то це гарантує, що обов'язково буде знайдений оптимальний шлях. Пошук з використанням функції

$$f(n)\_ = g(n) + h(n)\_$$

називається алгоритмом A*.

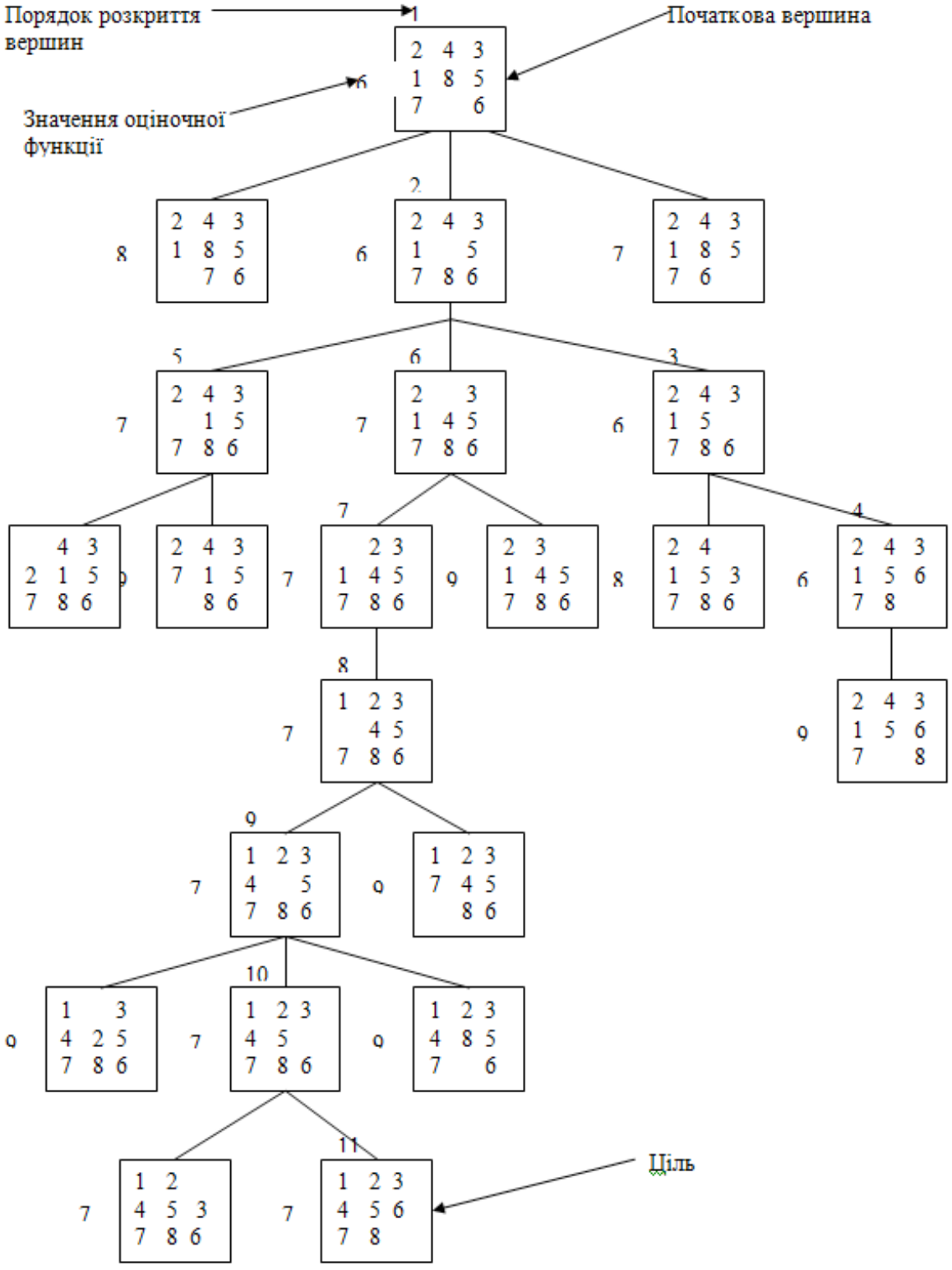



Рисунок 2.8 – Дерево перебору при застосуванні алгоритму A*

У цій грі в якості h(n)_ вибрано число шашок, що знаходяться не на своїх місцях, причому h(n)_ <= h(n) (не досягаємо цілі, поки число переміщень менше числа шашок, що знаходяться не на своїх місцях).

Якщо ж вибрати h(n)_ == 0, то маємо горизонтальний пошук, при якому насамперед розкриваються довколишні вершини (число таких вершин зростає), оптимальний шлях при цьому в кінці кінців знаходиться. Якщо

h1(n)_ < h2(n)_ <= h(n),

то можна довести, що число вершин, що розкриваються при пошуку з використанням функції f2(n) = g(n) + h2(n), завжди менше , ніж при пошуку з використанням функції f1(n) = g(n) + h1(n).А саме, h(n) можна розглядати як обсяг евристичних знань, що передбачають вартість шляху до цілі, отже, обсяг евристичних знань h2(n) більше, ніж h1(n).

### Метод гілок і меж

З незакінчених шляхів, які формуються в процесі пошуку, вибирається найкоротший і продовжується на один крок. Отримані нові незакінчені шляхи (їх стільки, скільки гілок в даної вершини) розглядаються поряд зі старими, і знову продовжується на один крок найкоротший з них. Процес повторюється до першого досягнення цільової вершини, рішення запам'ятовується. Потім з решти незакінчених шляхів виключаються довші, ніж закінчений шлях, або рівні йому, а що залишилися продовжуються за таким же алгоритмом до тих пір, поки їх довжина менше закінченого шляху. У результаті або всі незакінчені шляху виключаються, або серед них формується закінчений шлях, коротший, ніж раніше отриманий. Останній шлях починає грати роль еталону і т. д.

### Алгоритм найкоротших шляхів Мура

Вихідна вершина X0 позначається числом 0. Нехай в ході роботи алгоритму на поточному кроці отримано множину дочірніх вершин X(xi) вершини xi. Тоді з неї викреслюються всі раніше отримані вершини, а решта позначаються міткою, збільшеною на одиницю в порівнянні з міткою вершини xi, і від них проводяться вказівники до Xi. Далі на множині помічених вершин, ще не фігурують як адреси вказівників, вибирається вершина з найменшою позначкою, для неї будуються дочірні вершини. Розмітка вершин повторюється до тих пір, поки не буде отримана цільова вершина.

### Алгоритм Дейкстри

Алгоритм Дейкстри визначення шляхів з мінімальною вартістю є узагальненням алгоритму Мура за рахунок введення дуг змінної довжини.

### Алгоритм Дорана і Мічі

Алгоритм Дорана і Мічі є алгоритмом пошуку з низькою вартістю. Використовується, коли вартість пошуку велика в порівнянні з вартістю оптимального рішення. В цьому випадку замість вибору вершин, найменш віддалених від початку, як в алгоритмах Мура і Дійкстри, вибирається вершина, для якої евристична оцінка відстані до цілі є найменшою. При хорошій оцінці можна швидко отримати рішення, але немає гарантії, що шлях буде мінімальним.

### Алгоритм Харта, Нільсона і Рафаеля

В алгоритмі об'єднані обидва критерії: вартість шляху до вершини g (x) і вартість шляху від вершини h (x) – у вигляді аддитивної оціночної функції f{x)=g(x}-h(x). За умови h(x) <hp(x), де hp(x) – дійсна відстань до цілі. Алгоритм гарантує знаходження оптимального шляху.

Алгоритми пошуку шляху на графі розрізняються також напрямком пошуку. Існують прямі, зворотні і двонаправлені методи пошуку.

Прямий пошук йде від початкового стану і, як правило, використовується тоді, коли цільовий стан задано неявно.

Зворотний пошук йде від цільового стану і використовується тоді, коли початковий стан задано неявно, а цільовий – явно.

Двонаправлений пошук вимагає одночасного вирішення двох проблем: зміни напрямку пошуку і оптимізації «точки зустрічі». Одним з критеріїв для вирішення першої проблеми є порівняння»ширини» пошуку в обох напрямках – вибирається той напрям, який звужує пошук. Друга проблема викликана тим, що прямий і зворотний шляхи можуть розійтися і чим вужчий пошук, тим це є більшй імовірним.

## 2.2.2 МЕТОДИ РІШЕННЯ ЗАДАЧ ЗА СТРАТЕГІЄЮ ПОШУКУ В ПРОСТОРІ ЗАДАЧ

Методи пошуку рішень в просторі задач (рішення задач методом редукції, тобто зведення задач до підзадач) дає добрі результати, тому що часто хід рішення задач має ієрархічну структуру. Проте не обов'язково вимагати, щоб основна задача і всі її підзадачі вирішувалися однаковими методами. Редукція корисна для подання глобальних аспектів задачі, а при рішенні більш специфічних задач кращим є метод пошуку в просторі станів. Метод пошуку рішення в просторі станів можна розглядати як окремий випадок методу пошуку за допомогою редукцій, бо кожне застосування оператора в просторі станів означає зведення початкової задачі до двох більш простих, з яких одна є елементарною. У загальному випадку редукція початкової задачі не зводиться до формування таких двох підзадач, з яких хоча б одна була елементарною.

Пошук рішення в просторі задач полягає в послідовному зведенні вихідної задачі до все більш простих до тих пір, поки не будуть отримані тільки елементарні задачі. Частково впорядкована сукупність таких задач складе рішення вихідної задачі.

Розчленування задачі на альтернативні множини підзадач зручно представляти у вигляді І/АБО-графу. Граф-рішення складається з вирішуваних вершин і вказує спосіб розв'язання початкової вершини. Наявність тупикових вершин призводить до нерозв'язних вершин. Нерозв'язними є тупикові вершини, І-вершини, у яких нерозв'язна хоча б одна дочірня вершина, та АБО-вершини, у яких нерозв'язна кожна дочірня вершина.

Етапи перебору на І/АБО-графах:

1) початкова вершина асоціюється з початковим описом задачі;

2) будуються множини дочірніх вершин для початкової вершини за допомогою тих операторів зведення задач до підзадач, які є застосовними. Позначимо як γ такий

комбінований оператор, застосування якого дає всі вершини, дочірні для даної, а сам процес застосування γ до вершини назвемо розкриттям вершини;

3) від кожної дочірньої вершині назад до її батьківських вершин проводяться вказівники;

Ці вказівники використовуються при спробі розмітити розв'язні і нерозв'язні вершини, а після закінчення вони дають вирішальний граф.

4) Процес розкриття вершин і встановлення вказівників продовжується до тих пір, поки початкова вершина не може бути позначена як розв'язна або нерозв'язна.

У процесі перебору породжується структура з вершин і вказівників, що представляє собою І/АБО граф. Цей граф будемо називати графом перебору.

Основні особливості процесу розкриття вершин:
- складність контролю закінчення пошуку і методів упорядкування вершин;
- замість пошуку цільової вершини будується вирішувальний граф;
- зменшується обсяг обчислень за рахунок запам'ятовування розв'язних серед раніше розкритих вершин;
- при відсутності необхідності пошуку більш ніж одного рішення задачі має сенс відкинути на пошуковому графі всі нерозв'язні вершини (включаючи наступні за ними), дочірні для вирішуваних.

При пошуку рішення на І/АБО графі застосовують такі методи:
- повний перебір – вершини розкриваються в порядку побудови (рис. 2.9);
- перебір в глибину – в першу чергу розкриваються вершини, побудовані останніми (рис. 2.10);
- впорядкований перебіру – для впорядкування вершин при їх розкритті використовуються оціночні функції (рис. 2.11).

Основною причиною наявності у вершин І/АБО графа більш ніж однієї батьківської вершини є наявність ідентичних підзадач серед тих, на які розбивається вихідна задача. Якщо є можливість розпізнавання еквівалентності описів завдач, то задачу пошуку можна в загальному випадку істотно спростити за рахунок одноразового рішення ідентичних підзадач.

Проблеми методів перебору:
- складність, яка полягає у виключенні з розгляду деяких вірних нециклічних рішень;
- організація коректного аналізу структури вказівників графа перебору при наявності вказівників, що йдуть від даної вершини до більш ніж однієї батьківської, деякі з цих вказівників можуть не знадобитися в рішенні;
- при побудові дочірніх вершин слід перевірити наявність цих вершин в списку ЗАКРИТО і чи не були вони попередньо помічені як розв'язні або як нерозв'язні.

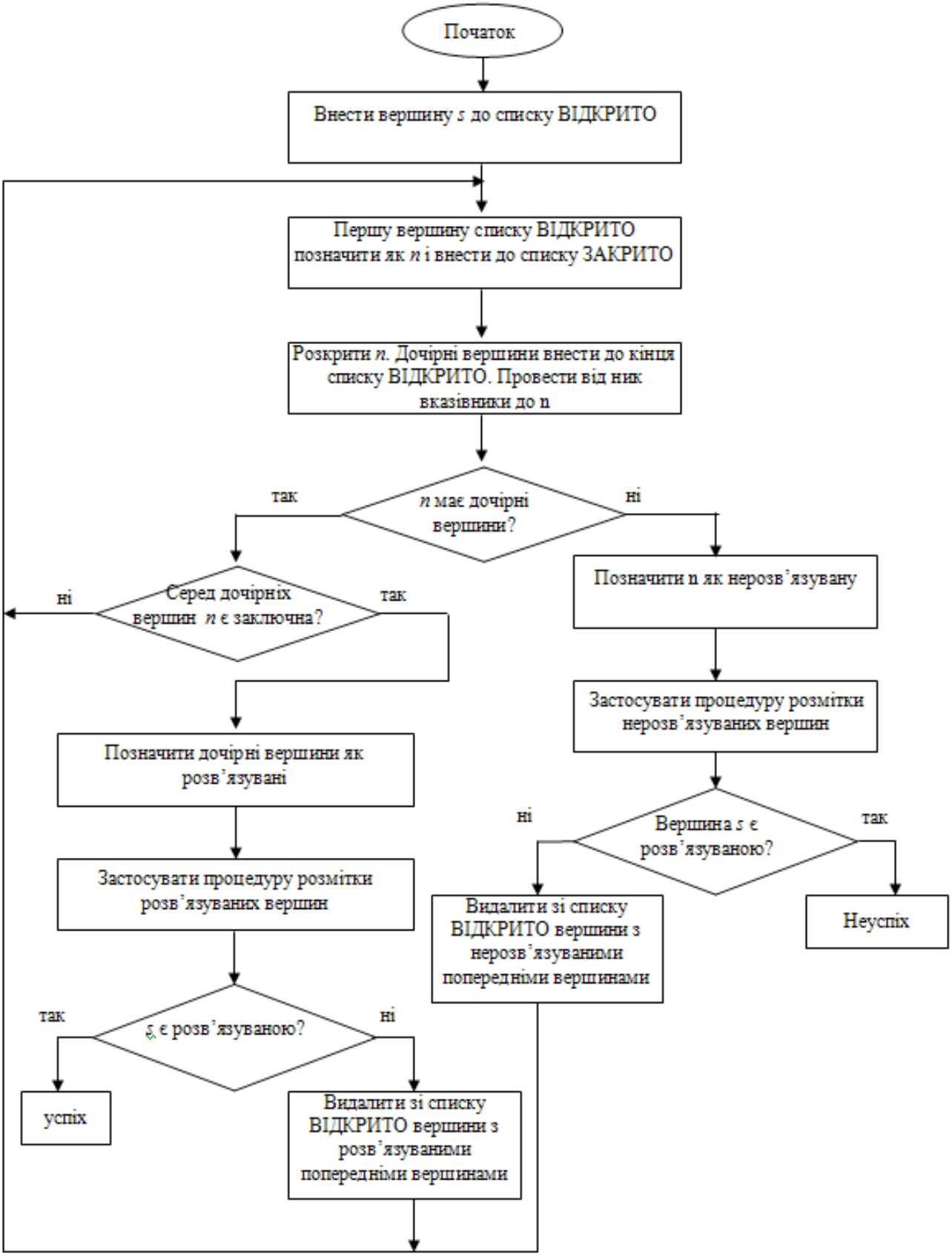



Рисунок 2.9 – Алгоритм повного перебору

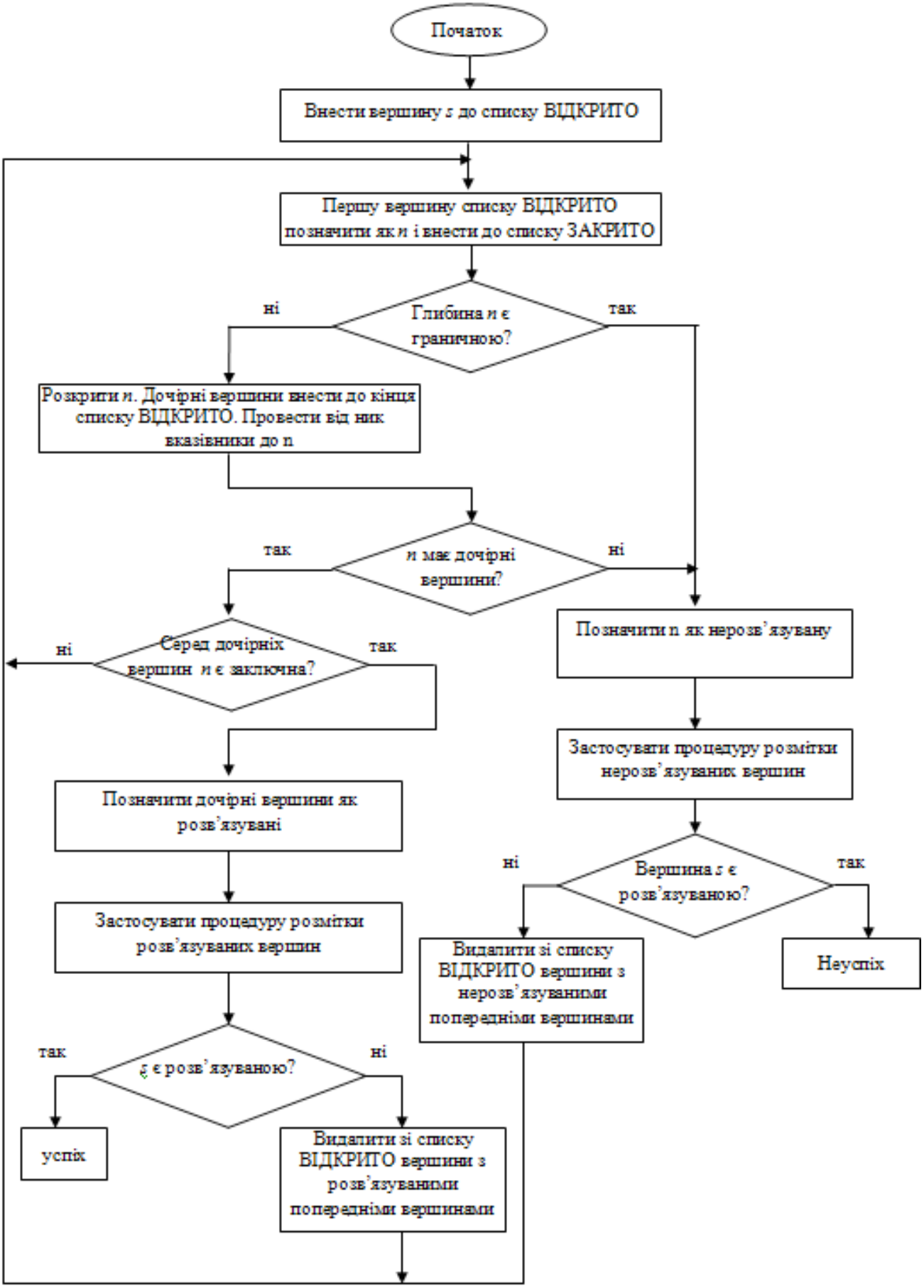



Рисунок 2.10 – Алгоритм пошуку в глибину

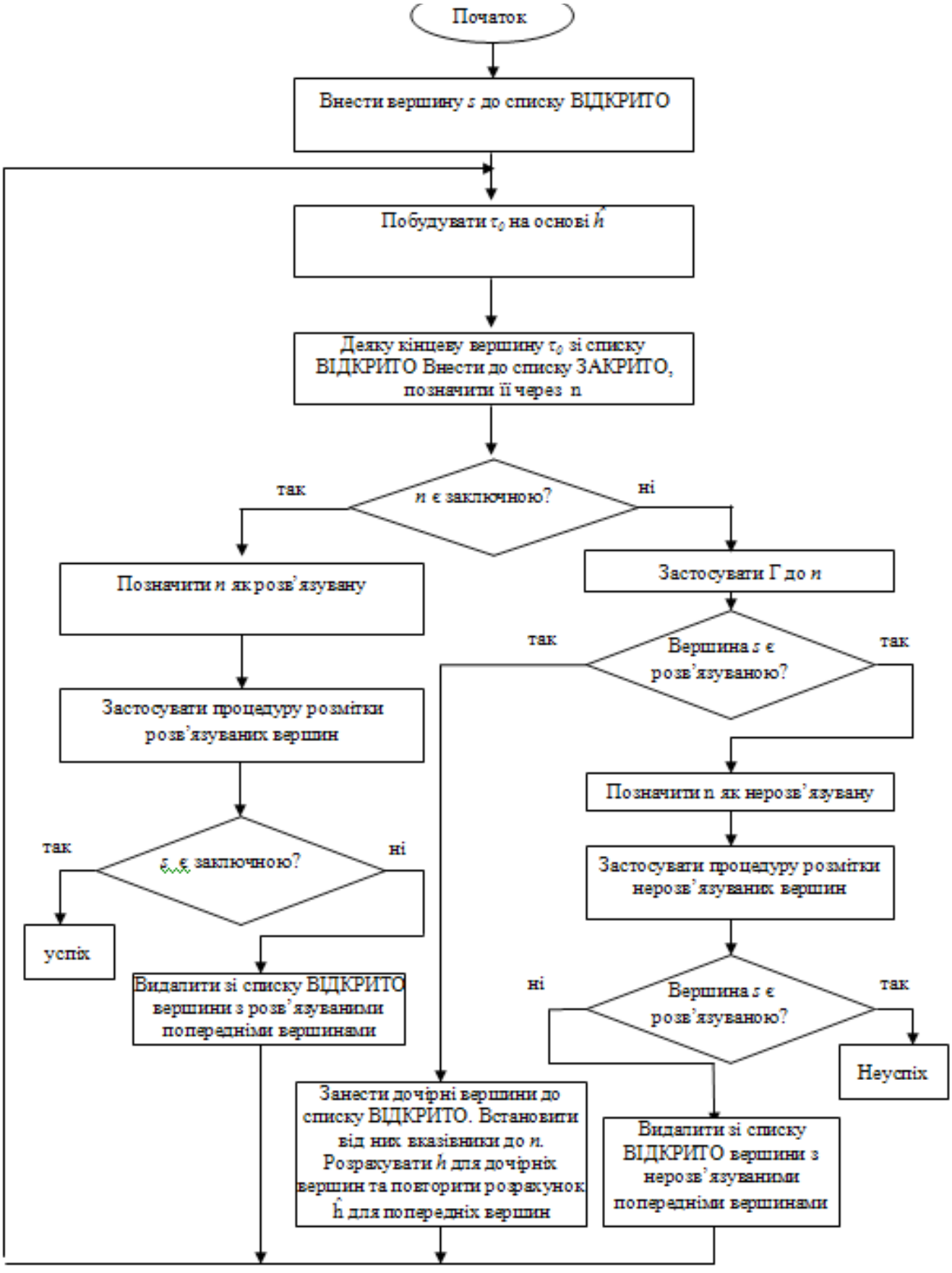



Рисунок 2.11 – Алгоритм впорядкованого перебору для І/АБО дерева

## Вартість дерев рішень

Якщо при цьому ми прийшли до вже розв'язуваної (або нерозв'язуваної) дочірньої вершини по новому шляху, то необхідно скористатися процедурою розмітки в цілях перевірки можливості розв'язуваності/нерозв'язуваності початкової вершини.

Для І/АБО дерев будемо розглядати сумарну і максимальну вартості.

Сумарна вартість являє собою суму вартостей всіх дуг в дереві рішення.

Шляхом по дереву з коренем у вершині n будемо називати послідовність вершин ($n_1$, $n_2$, ..., $n_k$) дерева, де $n_1$ = n, $n_k$-заключна вершина, а вершина $n_j$ – дочірня для $n_{j-1}$ при j = 2, ..., k .

Вартість шляху є сумою вартостей дуг, що зв'язують вершини цього шляху.

Максимальна вартість – вартість шляху по дереву рішення, що має максимальну вартість.

На розміченому дереві рішень (рис. 2.12) числа, позначені біля дуг, визначають вартість дуг. Маємо дві гілки рішень – Рішення А та В. Сумарна вартість рішення А дорівнює 20, а максимальна 15. Для рішення В сумарна вартіть дорівнює 18, максимальна 17. Вибір рішення залежить від того, що мінімізується.

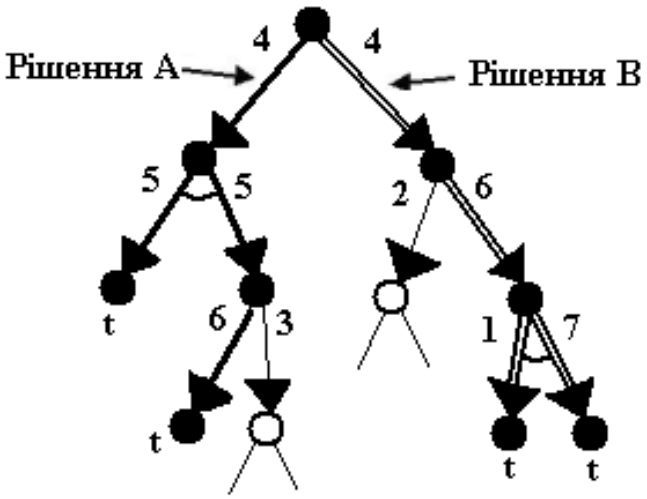



Рисунок 2.12 – Вартості вирішувальних дерев

Оптимальним для І/АБО дерева будемо називати його складове дерево рішення з мінімальною вартістю.

Позначимо вартість оптимального дерева з коренем в початковій вершині s як h(s). Дамо рекурсивне визначення мінімальної вартості h(s) через через мінімальну вартість h(n) дерева рішення з коренем в будь-якій вершині n. Позначимо вартість дуги між вершиною ni та її дочірньої вершиною $n_j$ через c($n_i$, $n_j$).

Якщо n є заключною вершиною (відповідає елементарній задачі), то

$$h(n)=0.$$

Якщо n не є заключною вершиною і має дочірні вершини $n_1$, ..., $n_k$ типу АБО, то

$$h(n) = \min_i [c(n, n_i) + h(n_i)]$$

Якщо n не є заключною вершиною і має дочірні вершини $n_1$, ..., $n_k$ типу І, то для сумарних вартостей

$$h(n) = \sum_{i=1}^{k} [c(n, n_i) + h(n_i)],$$

а для максимальної вартості

$$h(n) = \max_i [c(n, n_i) + h(n_i)]$$

Функція h(n) не визначена для нерозв'язних n.

Оцінки вартості використовуються у методах прямого перебору при пошуку рішень дерев з мінімальною (або сумарною, або максимальною) вартістю. Для цього використаємо оцінену вартість дерева рішення. Введемо функцію $\hat{h}$, яка для кожної вершини n, що не є нерозв'язуваною, служить оцінкою h(n), тобто вартістю оптимального дерева рішення з коренем у вершині n.

Вершину дерева перебору будемо вважати кінцевою в одному з трьох випадків:

- вершина визнана заключною;
- вершина не визнана заключною і не має дочірніх вершин;
- у процесі перебору для вершини ще не були побудовані дочірні вершини.

Для різних типів вершин дерева перебору (кінцевих і некінцевих) евристична функція $\hat{h}$(n) повинна будуватися по-різному:

1) n – кінцева вершина.

якщо вершина n є заключною, то h(n) = 0;

якщо n не є заключною і у неї немає дочірніх вершин, то функція h(n) не визначена;

якщо для вершини n ще не були побудовані дочірні вершини, то за оціночну функцію $\hat{h}$(n) приймається деяка евристична оцінка вартості величини h(n) оптимального дерева рішення з коренем у вершині n. Тут використовується евристична інформація, визначена характером розв'язуваної задачі;

2) n не є кінцевою вершиною і має дочірні вершини $n_1$, ..., $n_k$ типу АБО

$$h(n) = \min_i \left[ c(n, n_i) + \hat{h}(n_i) \right];$$

3) n не є кінцевою вершиною і має дочірні вершини $n_1$, ..., $n_k$ типу І

$$h(n) = \sum_{i=1}^{k} \left[ c(n, n_i) + \hat{h}(n_i) \right].$$

Для максимальних вартостей:

$$h(n) = \max_i \left[ c(n, n_i) + \hat{h}(n_i) \right]$$

На кожному етапі пошуку на дереві є множина піддерев з коренем в s. Кожне з цих піддерев може бути частиною повного дерева рішення і називається потенційним деревом рішення з вершиною в s. На кожному етапі перебору можна знайти потенційне дерево рішення т0 з коренем s, яке за оцінкою є верхньою частиною оптимального дерева рішення з вершиною в s. Процедура пошуку визначається так:

1. Початкова вершина входить в т0.

2. Якщо у вершини n, яка входить до т0, у дереві перебору є

а) дочірні вершини n1, ... , nk типу АБО, то та з них, для якої значення [c(n,ni)+ĥ(ni)] є мінімальним, також входить в т0;

б) дочірні вершини типу І, то усі вони входять в т0.

Процедура впорядкування повинна дати відповідь, яке з потенційних дерев є найбільш перспективним для розкриття. При виборі дерева т0 необхідно у кожній вершині дерева пошуку враховувати оцінку ĥ. Після знаходження дерева т0 розкривають його ще не вибрані кінцеві вершини і знову розраховуються значення ĥ для розкритої вершини.

### Мінімаксний метод

Підхід зведення задач до сукупності підзадач може бути використаний для пошуку ігрових стратегій деяких ігор.

Будемо розглядати клас ігор двох осіб з повною інформацією. У таких іграх беруть участь два гравці, які по черзі роблять свої ходи. У будь-який момент кожному гравцеві відомо все, що сталося в грі до цього моменту і що може бути зроблено в даний момент. Гра закінчується або виграшем одного гравця (і програшем іншого), або нічиєю [Нильсон].

Нехай іменами гравців будуть ПЛЮС і МІНУС. Будемо використовувати наступні позначення:

$X^S$ або $Y^S$ – деяка конфігурація гри, причому індекс S приймає значення + або – , вказуючи, кому належить наступний хід (тобто в конфігурації $X^+$ наступний хід повинен робити гравець ПЛЮС, а в $X^-$ □ – гравець МІНУС);

$W(X^S)$ - задача доказу того, що гравець ПЛЮС може виграти, виходячи з конфігурації $X^S$;

$V(X^S)$ – задача доказу того, що гравець МІНУС може виграти, виходячи від конфігурації $X^S$.

Розглянемо ігрову задачу $W(X^S)$. Оператори зведення (редукції) цієї задачі до підзадач визначаються з урахуванням ходів, допустимих у проведеній грі:

– якщо в деякій конфігурації $X^+$ черговий хід робить гравець ПЛЮС, і є N допустимих

ходів, що призводять відповідно до конфігурацій $X1^-$ ,$X2^-$ , ..., $X_N^-$, то для вирішення завдання $W(X^+)$ необхідно вирішити принаймні одну з підзадач $W(X_i^-)$. Дійсно, так як хід вибирає гравець ПЛЮС, то він виграє гру, якщо хоча б один з ходів веде до виграшу (рис. 2.13,а);

- якщо ж в деякій конфігурації $Y^-$ хід повинен зробити гравець МІНУС і є K допустимих ходів, що призводять до конфігурацій $Y_1^+,Y_2^+,...,Y_K^+$, то для рішення задачі $W(Y^-)$ потрібно вирішити кожну з виникаючих підзадач $W(Y_i^+)$. Дійсно, оскільки хід вибирає МІНУС, то ПЛЮС виграє гру, якщо виграш гарантований йому після будь-якого ходу супротивника (рис. 2.13,б).

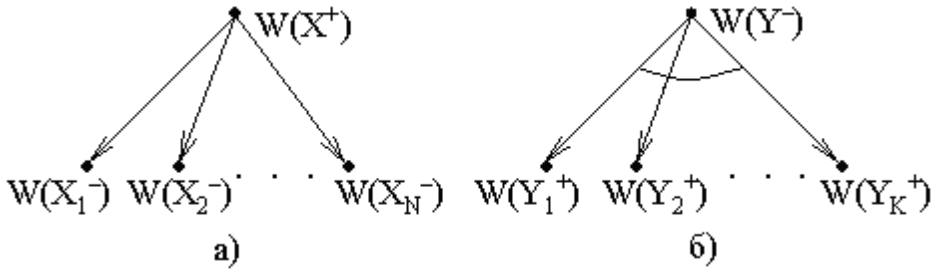



Рисунок 2.13 – Редукція ігрової задачі

Послідовне застосування до початкової конфігурації гри даної схеми редукції породжує І/АБО-дерево (І/АБО-граф), яке називають деревом (графом) гри. Дуги ігрового дерева відповідають ходам гравців, вершини конфігураціям гри, причому листи дерева це позиції, в яких гра завершується виграшем, програшем або нічиєю. Деякі листи є заключними вершинами, що відповідають елементарним задачам позиціям, виграшним для гравця ПЛЮС. Для конфігурації, де хід належить гравцю ПЛЮС, в ігровому дереві зображається АБО-вершиною, а для позицій, в яких ходить МІНУС, – І-вершиною.

Метою побудови ігрового дерева або графа є одержання розв'язувального піддерева для задачі $W(X^S)$, що показує, як гравець ПЛЮС може виграти гру з позиції $X^S$ незалежно від відповідей супротивника. Для цього можуть бути застосовані різні алгоритми пошуку на І/АБО-графах. Розв'язувальне дерево закінчується на позиціях, виграшних для ПЛЮСа, і містить повну стратегію досягнення ним виграшу: для кожного можливого продовження гри, обраного супротивником, у дереві є відповідний хід, який веде до перемоги.

Для задачі $V(X^S)$ схема зведення ігрових задач до підзадач аналогічна: ходам гравця ПЛЮС будуть відповідати І-вершини, а ходам МІНУСа – АБО-вершини, заключні ж вершини будуть відповідати позиціям, виграшним для гравця МІНУС.

При переборі на ігрових деревах кожен супротивник прагне знайти «хороший» хід, робить його, потім чекає на хід супротивника (який робить свій «хороший» хід), і знову прагне знайти свій «хороший» хід з нової позиції і т.д. Оцінку ходу виконують за допомогою статичної оціночної функції.

Хороший перший хід можна знайти за допомогою мінімаксного методу.

Процедура знаходження стратегії гри виконується таким чином:

1) дерево гри будується (проглядається) одним з відомих алгоритмів перебору (як правило, алгоритмом пошуку вглиб) від початкової позиції S до глибини N;

2) кінцеві вершини отриманого дерева, тобто вершини, які знаходяться на глибині N, оцінюються за допомогою статичної оціночної функції;

3) відповідно до мінімаксного принципу обчислюються оцінки всіх інших вершин: спочатку обчислюються оцінки вершин, батьківських для кінцевих, потім – батьківських для цих батьківських вершин і т.д.; таке оцінювання вершин триває при русі знизу вгору по дереву пошуку до тих пір, поки не будуть оцінені вершини, дочірні для початкової вершини, тобто для початкової конфігурації S;

4) серед вершин, дочірніх для початкової, вибирається вершина з найбільшою оцінкою: хід, який до неї веде, і є найкращим ходомд в ігровій конфігурації S.

Приклад І/АБО дерева гри наведено на рис. 2.14. Рішення для гравця ПЛЮС виділено жирними лініями.

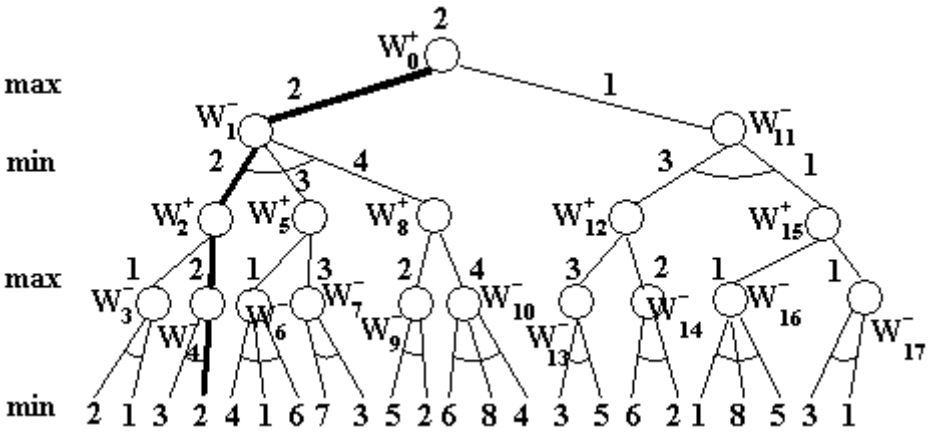



Рисунок 2.14 – Дерево гри для двох гравців

Гравець ПЛЮС робить перший хід, і вершина, що цій позиції відповідає, є вершиною АБО на дереві гри. Гравцеві ПЛЮС необхідно вибрати хід, що має максимальну оцінку. На наступній позиції гравець МІНУС прагне мінімізувати виграш гравця ПЛЮС і вибирає хід з мінімальною оцінкою (мова йде про вершину І дерева).

### Метод альфа-бета відсікання

У мінімаксному методі процес побудови дерева гри відділений від процесу оцінювання вершин, оцінка позицій виконується після побудови дерева гри. Тому в цілому мінімаксна процедура виявляється неефективною стратегією пошуку хорошого першого ходу. Для підвищення ефективності пошуку, тобто скорочення перебору, необхідно обчислювати оцінки вершин (статичні та мінімаксні) одночасно з побудовою ігрового дерева і не розглядати вершини, які є гіршими від вже побудованих вершин.

Метод альфа-бета відсікання базується на такому міркуванні для скорочення пошуку: якщо є два варіанти ходу одного гравця, то найгірший в ряді випадків можна відразу відкинути, не з'ясовуючи, наскільки конкретно він гірше.

Дерево гри будується до глибини N алгоритмом перебору вглиб. Відразу, як це стає можливим, обчислюються не тільки статичні оцінки кінцевих вершин, але й попередні мінімаксні оцінки проміжних вершин. Попередня оцінка визначається відповідно як мінімум або максимум уже відомих до теперішнього моменту оцінок дочірніх вершин. У загальному випадку, ця оцінка може бути отримана при наявності оцінки хоча б однієї дочірньої вершини. У ході подальшої побудови дерева гри і отримання нових оцінок вершин попередні оцінки поступово перераховуються за мінімаксним принципом.

Правила обчислення оцінок вершин дерева гри, в тому числі попередніх оцінок проміжних вершин (альфа- і бета-величин):

- кінцева вершина ігрового дерева оцінюється статичною оціночною функцією відразу, як тільки вона побудована;
- проміжна вершина попередньо оцінюється за мінімаксним принципом, як тільки стала відома оцінка хоча б однієї з її дочірніх вершин; кожна попередня оцінка перераховується (уточнюється) кожного разу, коли отримана оцінка ще однієї дочірньої вершини;
- попередня оцінка АБО-вершини (альфа-величина) дорівнює найбільшій з обчислених до поточного моменту оцінок її дочірніх вершин;
- попередня оцінка І-вершини (бета-величина) дорівнює найменшій з обчислених до поточного моменту оцінок її дочірніх вершин.

Альфа-величини не можуть зменшуватися, а бета-величини не можуть збільшуватися.

Правила переривання перебору, або відсікання гілок ігрового дерева:

а) перебір можна перервати нижче будь-якої І-вершини, бета-величина якої не більше, ніж альфа-величина однієї з попередніх їй АБО-вершин (включаючи кореневу вершину дерева);

б) перебір можна перервати нижче будь-якої АБО-вершини, альфа-величина якої не менша, ніж бета-величина однієї з попередніх їй І вершин.

У випадку (а) кажуть, що має місце альфа-відсікання, оскільки відсікаються гілки дерева, починаючи з АБО-вершин, яким приписана альфа-величина, а в разі (б) – бета-відсікання, оскільки відсікаються гілки, які починаються з бета-величин.

Розглянуті правила працюють у ході побудови ігрового дерева вглиб; це означає, що попередні оцінки проміжних вершин з'являються лише в міру просування від кінцевих вершин дерева вгору до кореня і реально відсікання можуть початися тільки після того, як отримана хоча б одна точна мінімаксна оцінка проміжної вершини.

### Алгоритм Ченга і Слейгла

Цей алгоритм базується на перетворенні довільного І/АБО-графа в спеціальний АБО-граф, кожна АБО-гілка якого має І-вершини тільки в кінці. Перетворення використовує представлення довільного І/АБО-графа як довільної формули логіки висловлювань з подальшим перетворенням цієї довільної формули в діз'юнктівную нормальну форму. Подібне перетворення дозволяє далі використовувати алгоритм Харта,Нільсона і Рафаеля.

Метод ключових операторів. Нехай задана задача <A,B> і відомо, що оператор f обов'язково повинен входити у рішення цієї задачі. Такий оператор називається ключовим. Нехайдля застосування f необхідним є стан C, а результат його застосування є I(c). Тоді I-вершина <A,B> породжує три дочірні вершини: <A,C>, <C,f(c)> та <f(c),B>, з яких середня є елементарною задачею. До задач < A,C > і < f(c),B > також підбираються ключові оператори, і зазначена процедура редукування повторюється до тих пір, поки це можливо. У підсумку початкова задача <A,B> розбивається на упорядковану сукупність підзадач, кожна з яких вирішується методом планування в просторі станів.

Можливі альтернативи щодо вибору ключових операторів, так що в загальному випадку буде мати місце І/АБО-граф. У більшості задач вдається не виділити ключовий оператор, а тільки вказати множину операторів, що його містить. В цьому випадку для задачі <A,B> обчислюється різниця між A і B, якій ставиться у відповідність оператор, що знімає цю відмінність. Останній і є ключовим.

### Метод планування загального вирішувача задач (ЗВЗ)

ЗВЗ був першою найбільш відомою моделлю планувальника. Він використовувався для рішення задач інтегрального числення, логічного виведення, граматичного розбору та ін. ЗВЗ об'єднує два основних принципи пошуку: аналіз цілей і засобів та рекурсивне рішення задач. У кожному циклі пошуку ЗВЗ вирішує в жорсткій послідовності три типи стандартних задач: перетворити об'єкт A в об'єкт B, зменшити різницю D між A і B, застосувати оператор f до об'єкта A. Рішення першої задачі визначає відмінність D, другої – відповідний оператор f, третьої -необхідну умову застосування C.

Якщо C не відрізняється від A, то оператор f застосовується, інакше C представляється як чергова ціль і цикл повторюється, починаючи з задачі «перетворити A на C». В цілому стратегія ЗВЗ здійснює зворотний пошук – від заданої цілі B до необхідного засобу її досягнення C, використовуючи редукцію початкової задачі <A,B> до задач <A,C> і <C,B>.

Зауважимо, що в ЗВЗ за замовчуванням передбачається незалежність відмінностей одна від одної, звідки витікає гарантія, що зменшення одних відмінностей не призведе до збільшення інших.

## 2.2.3 Пошук рішень за допомогою методів логічного підходу

Таке планування пошуку рішень на основі логічного виведення передбачає:

- опис станів у вигляді правильно побудованих формул (ППФ) певного логічного числення;
- опис операторів у вигляді ППФ, або правил перепису одних ППФ в інші.

Подання операторів у вигляді ППФ дозволяє створювати дедуктивні методи планування, представлення операторів у вигляді правил перепису – методи планування з елементами дедуктивного виведення.

### Дедуктивний метод планування системи QA3

ЗВЗ не виправдав сподівань, які на нього покладали, в основному через незадовіль-

не подання задач. Спроба виправити становище привела до створення питально-відповідної системи QA3. Система розрахована на довільну предметну область і здатна шляхом логічного виведення відповісти на питання: чи можливе досягнення стану B з A? В якості методу автоматичного виведення використовується принцип резолюції. Для вибору напрямку логічного виведення QA3 застосовує різні стратегії, в основному синтаксичного характеру, що враховують особливості формалізму принципу резолюції. Експлуатація QA3 показала, що виведення в такій системі є повільним, детальним, що невластиво міркуванням людини.

### Метод продукцій системи STRIPS

У цьому методі оператор представляє продукцію P, A => B, де P, A і B – множини ППФ числення предикатів першого порядку, P задає умови застосування ядра продукції A => B, де B містить список ППФ, що додаються і список ППФ, що видаляються. Метод повторює метод ЗВЗ з тією різницею, що стандартні задачі визначення відмінностей та застосування відповідних операторів вирішуються на основі принципу резолюції. Відповідний оператор вибирається так само, як в ЗВЗ, на основі принципу «аналіз засобів та цілей». Наявність комбінованого методу планування дозволила обмежити процес логічного виведення описом стану світу, а процес породження нових таких описів залишити за евристикою «від цілі до засобу її досягнення».

## 2.2.4 РІШЕННЯ ЗАДАЧ ДЕДУКТИВНОГО ВИБОРУ

У дедуктивних моделях подання та обробки знань розв'язувана проблема записується у вигляді тверджень формальної системи, ціль – у вигляді твердження, справедливість якого слід встановити або спростувати на підставі аксіом (загальних законів) і правил виведення формальної системи. В якості формальної системи використовують числення предикатів першого порядку.

Відповідно до правил, встановлених у формальній системі, заключному твердженню-теоремі, отриманому з початкової системи тверджень (аксіом, посилань), приписується значення ІСТИНА, якщо кожному посиланню, аксіомі також приписано значення ІСТИНА.

Процедура виведення являє собою процедуру, яка із заданої групи виразів виводить вираз, відмінний від заданих.

Зазвичай в логіці предикатів використовується формальний метод доведення теорем, що допускає можливість його машинної реалізації, але існує також можливість доказу неаксіоматичним шляхом: прямим виведенням, зворотним виведенням.

Метод резолюції використовується як повноцінний (формальний) метод доказу теорем.

Для застосування цього методу початкову групу заданих логічних формул потрібно перетворити в деяку нормальну форму. Це перетворення проводиться в кілька стадій, які складають машину виведення.

## 2.2.5 МЕТОДИ ПОШУКУ РІШЕНЬ У ВЕЛИКИХ ПРОСТОРАХ СТАНІВ

Реальні предметні області часто характеризуються великими просторами станів, тому методи перебору практично неможливо застосувати для пошуку рішень, оскільки із збільшенням розміру простору час пошуку експоненціально зростає. При великому розмірі простору пошуку можна спробувати розбити загальний простір на підпростори і здійснювати пошук спочатку в них. У цьому випадку простір пошуку представлено ієрархією просторів.

На вибір методів організації пошуку рішень у великих просторах станів впливають:
- характер простору станів;
- можливості його структурування;
- можливості абстрактного опису області рішень;
- можливість впливати на напрям пошуку рішень та ін.

У великих просторах станів доцільно задіяти **метод породження та перевірки**. Цей метод застосовують, коли простір є таким, що факторизується, тобто його можна розбити на досить незалежні підпростори з характерними неповними рішеннями. Характерне неповне рішення дає змогу зробити висновок про перспективність пошуку повного рішення у даному підпросторі.

Сутність методу породження та перевірки полягає в наступному:
- для даної предметної області генерується множина характерних неповних рішень у відповідності з описами наявних підпросторів;
- виконується перевірка характерних неповних рішень за допомогою оціночних функцій;
- якщо рішення є неприйнятним, то з подальшого розгляду виключається клас породжуваних повних рішень на основі даних неповних; тобто на ранній стадії пошуку виключаються неперспективні гілки без породження повних рішень та їх перевірки;
- якщо неповне рішення оцінюється як перспективне, то на його основі у даному підпросторі генеруються повні рішення з аналізом більш глибоких рівнів ієрархії;
- для отриманих повних рішень виконується перевірка того, чи є вони цільовими.

Метод породження та перевірки застосовується для рішення задач перспективного планування та проектування, але має такі проблеми:
- часто відсутні достовірні способи оцінки характерних неповних рішень;
- існує ризик виключити з пошуку частину простору, яка при подальшому розгляді міститиме шукане рішення;
- не завжди необхідно шукати усі можливі рішення, а достатньо мати деяке прийнятне.

Для рішення названих вище проблем виконують процедуру абстрагування простору рішення. При абстрагуванні пробують знаходити рішення на різних рівнях рівня абстракції шляхом уточнення його з описом простору. Абстракція дозволяє виявити важливі деталі, які додатково характеризують вирішувану задачу, та допомагає редукувати задачу на фіксовану множину описів під задач.

Якщо предметна область містить велику кількість підзадач, які в повному обсязі вирішити неможливо, то абстрагування допомагає виробити структуру плану дій.

Інший метод пошуку рішень у великих просторах – **метод послідовного уточнення згори**. У цьому разі розглядається абстракція для кожної задачі. Спочатку отримують узагальнений опис простору і вирішують задачу у цьому просторі. Потім виконують поетапну деталізацію простору і знаходять деталізовані рішення. Перехід до наступного рівня деталізації виконується тільки по завершенню рішення на поточному рівні.

**Евристичні методи та процедури** також використовуються для пошуку рішень у великих просторах. Часто евристині процедури входять до складу методів пошуку рішень. Наприклад, у методі породження та перевірки за допомогою евристик генеруються гіпотези про характерні неповні рішення; у методі метод послідовного уточнення згори висуваються гіпотези про те, що рішення задачі на абстрактному рівні допоможе знайти її рішення при зменшенні рівня абстрагування.

Якщо неможливо однозначно визначити напрям пошуку на певному етапі рішення, то також висуваються гіпотези про напрями пошуку та досліджуються, при цьому використовується нова інформація, яка отримується у ході рішення. На основі нової інформації гіпотеза або стає більш достовірною, і пошук продовжується у даному напрямі, або гіпотеза відхиляється і виконується дослідження іншої гіпотези. Основними проблемами евристичних методів є формування розумних гіпотез та переоцінка наявних гіпотез при отриманні нової ніформаціїї як внутрішньої, так і зовнішньої.

### 2.2.6 ПОШУК РІШЕННЯ ЗАДАЧ В УМОВАХ НЕВИЗНАЧЕНОСТІ

Дані і знання, з якими доводиться мати справу в ІС, рідко бувають абсолютно точними і достовірними. Властива знанням невизначеність може мати різноманітний характер, і для її опису використовується широкий спектр формалізмів. Розглянемо один з типів невизначеності в даних і знаннях – їх неточність. Будемо називати висловлювання неточним, якщо його істинність (або хибність) не може бути встановлена з певністю. Основними підходами при рішенні задач в умовах невизначеності є

- модель неточного виведення на основі імовірностей;
- модель на основі нечіткої логіки.

Модель використання неточних даних і знань включає дві складові: мову подання неточності і механізм виведення на неточних знаннях. Для побудови мови необхідно вибрати форму подання неточності (наприклад, скаляр, інтервал, розподіл, лінгвістичний вираз, множина) і передбачити можливість приписування мір неточності всім висловлюванням.

Механізми оперування неточними висловлюваннями можна розділити на два типи. До першого належать механізми, що носять «приєднаний» характер: перерахунок мір неточності ніби супроводжує процес виведення, що ведеться на точних висловлюваннях. Для розробки приєднаної моделі неточного виведення в системі, що базується на правилах виведення, необхідно задати функції перерахунку, що дозволяють обчислювати: а) міру неточності антецедента правила (його лівої частини) щодо мір неточності його складових висловлювань; б) міру неточності консеквента правила (його правої частини) відносно мір неточності правила і посилання правила; в) об'єднану міру неточності висловлювання щодо мір, отриманим з правил.

Введення міри неточності дозволяє додати до процесу виведення щось принципово нове – можливість об'єднання сили декількох свідоцтв, що підтверджують або спростовують одну і ту ж гіпотезу. Іншими словами, при використанні мір неточності доцільно виводити одне і те ж твердження різними шляхами (з наступним об'єднанням значень неточності), що абсолютно безглуздо в традиційній дедуктивній логіці. Для об'єднання свідоцтв використовується функція перерахунку, що займає центральне місце в перерахунку. Зауважимо, що, незважаючи на «приєднання» механізмів виведення цього типу, їх реалізація в базах знань впливає на загальну стратегію виведення: з одного боку, необхідно виводити гіпотезу всіма можливими шляхами для того, щоб врахувати всі релевантні цій гіпотезі свідоцтва, з іншого – запобігти багаторазовий вплив сили одних і тих же свідчень.

Для механізмів оперування з неточними висловлюваннями другого типу характерною є наявність схем виведення, спеціально орієнтованих на використовувану мову представлення неточності. Як правило, кожному кроку виведення відповідає перерахунок мір неточності, обумовлений співвідношенням на множині висловлювань (співвідношенням може бути елементарний логічний зв'язок, безвідносно до того, чи є це відношення фрагментом правила). Таким чином, механізми другого типу можуть бути застосовані не тільки до знань, представлених у формі правил. Разом з тим для них, як і для механізмів «приєднаного» типу, однією з головних є проблема об'єднання свідоцтв.

Загальним недоліком числових підходів при пошуку рішень в умовах невизначеності є те, що в них невизначеність представляється за допомогою більш-менш точних значень. Підходи, що використовують більше одного числа для подання невизначеності, дозволяють працювати з неточними, неповними, недостовірними знаннями, використовуючи інтервали для подання невизначеності (довіра, можливість, необхідність і т.д.). Недоліком таких підходів є необхідність отримувати знання від експертів та формувати проекцію нечіткості на числові шкали. Іншим недоліком є необхідність отримання та обробки великих об'ємів даних та пошук рішень у великих просторах.

# Лабораторні роботи

# ЛАБОРАТОРНА РОБОТА №1. ПОДАННЯ ІНТЕЛЕКТУАЛЬНОЇ ЗАДАЧІ

## МЕТА РОБОТИ:

- знайомство зі способами подання інтелектуальної задачі (ІЗ);
- вивчення основних можливостей формалізації ІЗ та подання її у просторі станів та просторі задач.

## ПОРЯДОК РОБОТИ

Вивчити інструкцію та лекційний матеріал.

Сформулювати характеристики предметної області, використавши ігрові ситуації.

Поставити задачі для вирішення у предметній області.

Виконати подання задач у просторі станів та у просторі задач у формі предикатів та у вигляді графів.

Продемонструвати результати розробки викладачеві.

Оформити звіт.

Захистити роботу.

## ПОДАННЯ ЗАДАЧІ У ПРОСТОРІ СТАНІВ

Одним з аспектів формальних систем є пошук усіх можливих варіантів рішення задачі, поставленої перед інтелектуальною системою. Досить часто людина саме так вирішує проблеми, наприклад, шахіст вибирає найкращий хід або інженер розглядає ряд можливих причин несправності пристрою. Така поведінка лежить в основі методу пошуку рішення в просторі станів.

Найбільш зручно простір станів представляти у вигляді графа, вузли якого відповідають різним станам навколишнього світу або станам розв'язуваної задачі, а дуги – допустимим операціями, що переводять світ з одного стану в інший, або крокам, виконуваним в процесі рішення задачі. Практично будь-яку задачу можна представити у вигляді такого графа, вибравши відповідний опис для станів і операцій. У такому випадку для вирішення поставленої задачі необхідно знайти на графі простору станів шлях від вихідного стану до необхідного.

Подання завдання у вигляді графа простору станів дозволяє використовувати теорію графів як для аналізу структури і ступеня складності завдання, так і для розробки процедури її рішення.

Швейцарський математик Леонард Ейлер розробив теорію графів для вирішення задачі про кенігсбергські мости. Місто розташоване на двох берегах річки і двох островах, з'єднаних між собою сімома мостами. Задача – знайти такий маршрут обходу міста, при якому через кожен міст проходять тільки один раз.

Береги річки позначимо ($b_1$,$b_2$), острови ($o_1$,$o_2$) і мости ($m_1$,$m_2$,$m_3$,$m_4$,$m_5$, $m_6$,$m_7$). Введемо предикат, який з'єднує (X,Y,Z), де X, Y – місце, тобто острів або берег; Z – міс. Систему

мостів в термінах предикатів подамо так: з'єднує ($o_1$,$o_2$,$m_1$); з'єднує ($o_1$,$b_1$,$m_2$); з'єднує ($o_1$,$b_1$,$m_3$); з'єднує ($o_2$,$b_1$,$m_4$); з'єднує ($o_1$ $b_2$,$m_5$); з'єднує ($o_1$,$b_2$, $m_6$); з'єднує ($o_2$,$b_2$,$m_7$). Еквівалентну системи мостів можна подати у вигляді графа (рис. 1)

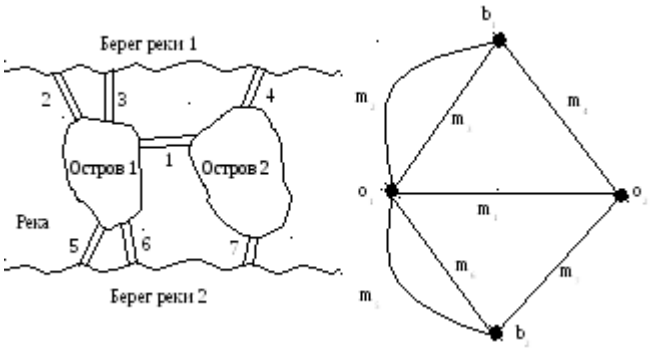



Рисунок 1 – Задача про кенігсбергські мости

Щоб довести неможливість існування шуканого маршруту, Ейлер ввів поняття ступеня вершини графа, яка дорівнює числу дуг, що з'єднують дану вершину з іншими вершинами. Ейлер показав, що якщо число вершин непарного ступеня не дорівнює нулю або двом, то маршрут неможливий, тому що якщо вершин непарного степеня дві, то маршрут можна почати в одній з них, а закінчити в інший. Якщо вершин з непарним ступенем немає, то маршрут повинен починатися і закінчуватися в одній і тій же вершині. З рис. 1 видно, що ця умова не виконується, отже, шуканий маршрут не існує.

Предикатне подання, описуючи відношення між мостами, берегами і островами, не забезпечує однозначного зв'язку кожної вершини з дугами і, як наслідок, не дозволяє ввести поняття ступеня вершини. Отже, подання навколишнього світу у вигляді графа підвищує інформативність в порівнянні з предикатним поданням.

Нагадаємо деякі поняття теорії графів. Граф – це множина вершин або вузлів $N_1$, $N_2$, ... $N_n$ і дуг або ребер, що з'єднують деякі пари вершин. На графі станів кожна вершина іменована, а якщо імена двох вершин збігаються, то вони вважаються однаковими.

Дуги графа зазвичай ідентифікуються іменами вершин, які вони з'єднують, тобто позначаються впорядкованої парою вершин, наприклад, ($N_2$, $N_3$). Дуга може мати власну мітку, що характеризує дану дугу, наприклад, відстань між вузлами транспортної мережі. Якщо кожній дузі приписано напрямок, то граф називається орієнтованим, а пара вершин, що з'єднуються спрямованої дугою, називаються відповідно батьком і нащадком. У однієї вершини-батька може бути кілька вершин-нащадків, які називаються братами. Вершина, яка не має нащадків, називається кінцевою. Вершина, яка не має предків, називається коренем, а граф з такою вершиною – кореневий граф.

Упорядкована послідовність вершин ($N_1$, $N_2$, ... $N_n$), де кожна пара ($N_i$, $N_i$ + 1) є дугою, називається шляхом довжини n-1 від вершини $N_1$, до вершини $N_n$. Якщо шлях включає якусь проміжну вершину більш ніж один раз, то шлях містить петлю. Дві вершини називаються пов'язаними, якщо існує шлях, що містить ці вершини. Якщо в графі для кожної пари вершин існує єдиний шлях, то такий граф називається деревом і, як наслідок, не містить петель. У кореневому дереві кожна вершина має єдиного батька.

Приклад кореневого дерева – каталог файлової системи.

Простір станів задачі представляється чотирма множинами:

- множина вершин графа N, що визначають можливі стану завдання, що виникають в процесі її рішення;
- множина дуг між вершинами A, що відповідають можливим кроків в процесі виконання завдання;
- множина початкових станів завдання S;
- множина цільових станів D.

Цільові стани описуються набором деяких вимірюваних властивостей станів, що зустрічаються в процесі пошуку, наприклад, виграшною комбінацією, або набором деяких властивостей шляхів, наприклад, вартістю переміщення по дугах.

Допустимим вважається будь-який шлях з вершини множини S в вершину множини D. Слід зазначити, що множини S і D є підмножинами множини N.

Одна зі складностей, які виникають при розробці алгоритму пошуку допустимого шляху на графі станів, полягає в тому, що один і той же стан може бути досягнуто різними шляхами, що може привести до виникнення петель. В результаті виникає проблема вибору оптимального в деякому сенсі шляху. Наприклад, граф простору станів гри в «хрестики-нулики» є кореневим, тому що має один початковий стан. Він не є деревом, тому що деякі стани можуть бути досягнуті різними шляхами, і не має петель, тому що всі дуги орієнтовані і спрямовані від кореневих вершин до нащадків. Такі графи часто зустрічаються при пошуку в просторі станів.

Граф простору станів гри «головоломка-8» або гри в «п'ятнашки» (рис. 2), теж кореневий, але, на відміну від попереднього, може мати цикли. Мета гри – для початкової розстановки фішок з номерами від 1 до 8 знайти таку послідовність їх переміщення на порожнє місце, в результаті якої фішки займуть задане положення, наприклад, по зростанню номерів зліва направо і зверху вниз. Повне число можливих станів досить велике (8!=40320), якщо враховувати симетричні відображення. Визначення чергового ходу значно зручніше задавати переміщенням порожньої клітини в одне з допустимих положень (рис. 2).

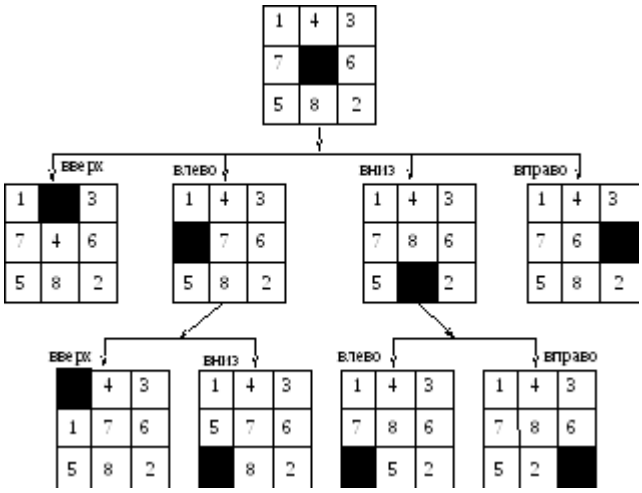



Рисунок 2 – Фрагмент простору станів гри «8»

Повний простір станів цієї гри розпадається на два незв'язаних підграфа однакового розміру, з чого випливає, що половина станів є недосяжними з будь-якої заданої початкової вершини.

Для «світу блоків» теж може бути сформований граф простору станів (рис.3). Вершини графа відповідають можливим станам світу блоків. Дуги графа позначені операціями, які можуть бути виконані в стані, відповідному батьківській вершині. Планування операцій, які необхідно виконати для отримання необхідного розташування блоків, можна уявити як пошук шляху на графі можливих станів від поточного розташування блоків до цільового розташуванню.

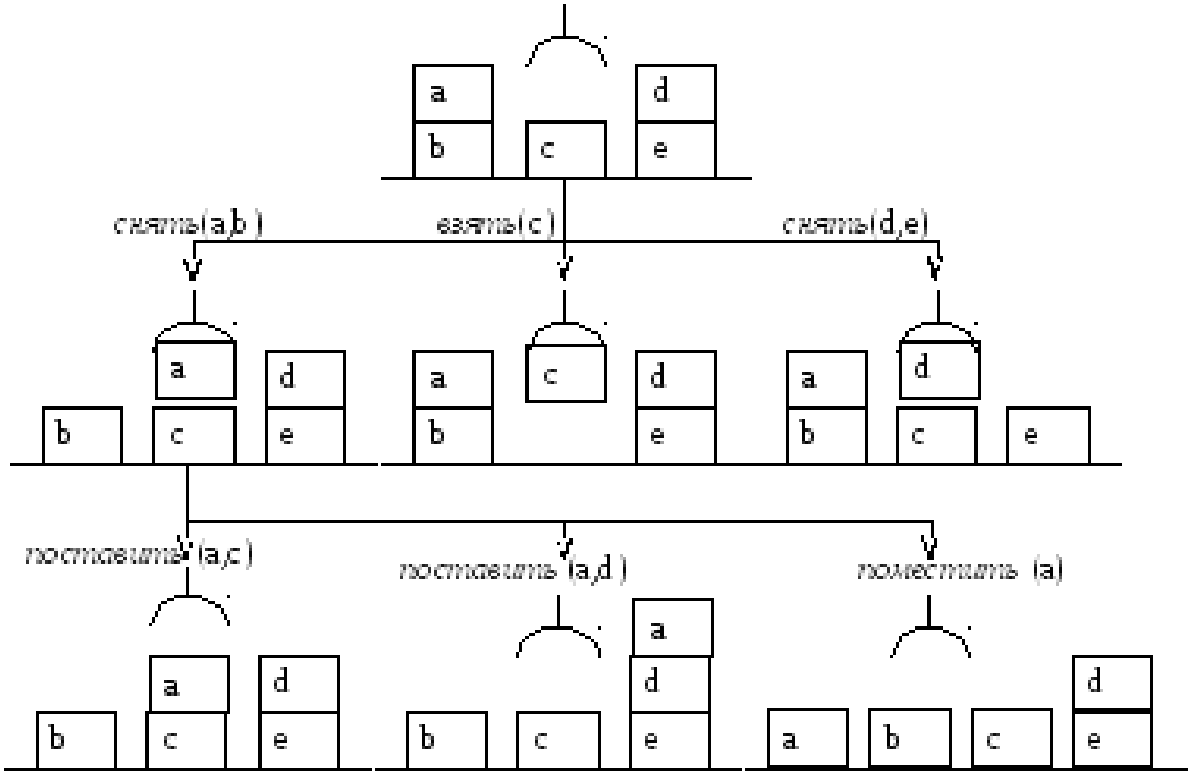



Рисунок 3 – Фрагмент графа простору станів світу блоків

Прикладом задачі, в якій мета задається властивістю шляху, є «задача комівояжера», яка полягає в тому, що необхідно знайти найкоротший шлях, що з'єднує всі вершини заданого графа. Вершини цього графа – пункти, які повинен відвідати комівояжер, наприклад, міста. Помічені дуги – можливість безпосереднього переміщення між цими пунктами, наприклад, довжина шляху між містами в кілометрах. На рис. 4 наведено такий граф для шести міст і фрагмент графа станів задачі за умови, що початковий пунктом є A.

Будь-який маршрут зручно представити послідовністю пунктів призначення, наприклад, (A, B, C, E, F, D) і його довжиною. Повне число вершин графа станів становить N !, де N – число пунктів призначення, (6! = 720), за умови, що комівояжер не повинен повертатися в початковий пункт, і кожен пункт має безпосередній зв'язок з усіма іншими пунктами. В даному випадку метою є не досягнення певного стану в просторі пошуку, а

знаходження на графі станів шляху мінімальної довжини, що в принципі вимагає повного перебору всіх станів. Однак зі зростанням N число вершин графа станів зростає так швидко, що прямий перебір всіх можливих варіантів стає неможливим.

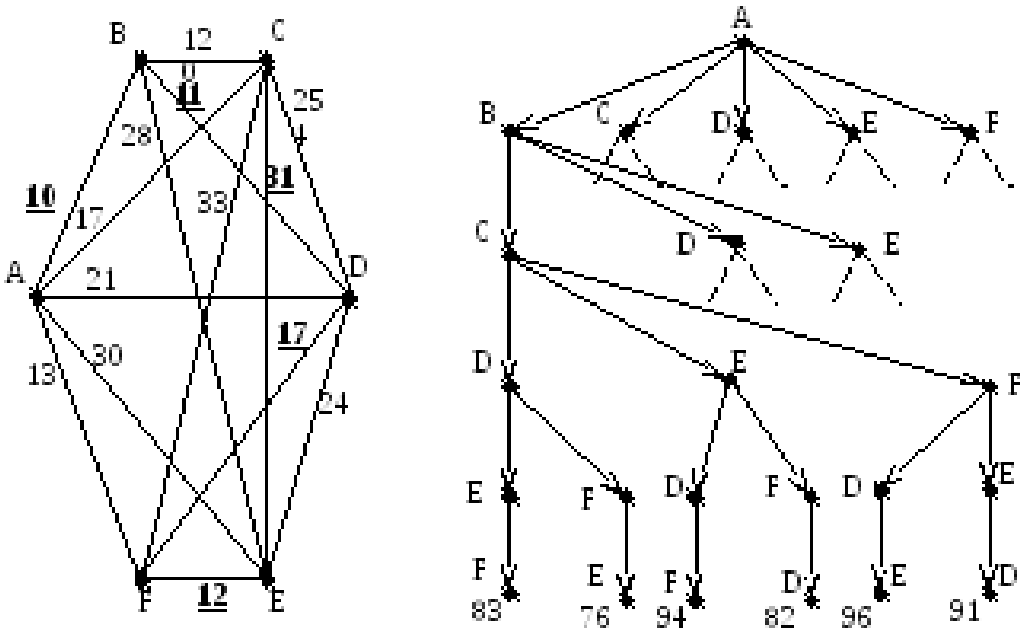



Рисунок 4 – Задача комівояжера

Задача комівояжера використовується на практиці, в тому числі забезпечує вирішення проблеми розводки електронних схем, задачу рентгенівської кристалографії і маршрутизації при комп'ютерних мережах.

## ПОНЯТТЯ ЗАДАЧІ ТА ПІДЗАДАЧІ

При формулюванні будь-якої задачі необхідно визначити вихідні дані, які ми будемо називати параметрами завдання.

Наприклад, якщо ми вирішуємо задачу знаходження коренів квадратного рівняння ax2 + bx + c = 0, то ця задача визначається трьома параметрами – коефіцієнтами a, b і c.

Якщо ж ми хочемо вирішити задачу знаходження середнього арифметичного деякого набору чисел, то параметрами задачі будуть кількість чисел та їх значення.

Припустимо, ми хочемо навчитися вирішувати задачу, зводячи її до вирішення підзадач. При такому підході будь-яка задача може бути формалізована у вигляді деякої функції, аргументами якої можуть бути такі величини, як:

- кількість параметрів;
- значення параметрів.

Тут і далі в якості параметрів будуть розглядатися цілі невід'ємні числа.

Як правило, одним з аргументів задачі є кількість параметрів задачі. У тому випадку, коли за значенням цього параметра можна визначити конкретні значення інших параметрів, ми ці параметри будемо опускати. Це зазвичай робиться в разі, коли параметри задані таблицею. Наприклад, якщо нам необхідно знайти суму перших K елементів таблиці, то для вирішення задачі досить знати один параметр K, а всі інші

параметри можна вибрати з таблиці.

Після того, як задача формалізована (представлена) у вигляді функції з деякими аргументами, визначимо поняття підзадачі. Під підзадачею ми будемо розуміти ту ж задачу, але з меншим числом параметрів або задачу з тим же числом параметрів, але при цьому хоча б один з параметрів має менше значення.

### Приклад 1.

Нехай у ігровій ситуації потрібно вибрати найважчий помідор з 10 наявних, щоб запустити ним у супротивника.

Для формалізації задачі визначимо функцію «Найважчий помідор», аргументами якої є кількість помідорів (10) і маса кожного з помідорів. Поки нас не цікавить конкретний вид цієї функції, для нас найважливішим фактором є те, що вона дає правильне рішення. Для даної задачі можна розглянути 9 підзадач, які мають менше значення аргументів:

- «Найважчий помідор» з 1 помідора,
- « Найважчий помідор» з 2 перших помідорів,
- « Найважчий помідор» з 3 перших помідорів,
- ...
- «Найважча монета» з 9 перших помідорів.

Слід зазначити, що під підзадачею не слід розуміти деякі етапи рішення задачі, такі, як організація введення і виведення даних, їх упорядкування, і т.д.

### ПОДАННЯ ЗАДАЧІ У ПРОСТОРІ ЗАДАЧ

Одним з основних способів рішення задач є їх зведення до рішення такого набору підзадач, щоб, виходячи з рішень підзадач, було можливо отримати рішення вихідної задачі.

При цьому для вирішення вихідної задачі може знадобитися рішення однієї або декількох підзадач.

### Приклад 2.

Є два баки різної ємності з водою. Підібрати відро такого розміру, щоб ним можна було спустошити обидва баки за мінімальне число вичерпувань.

Цю задачу можна формально представити так: знайти найбільший спільний дільник (НСД) двох натуральних чисел N і M.

Розглянемо варіанти рішення задачі:

1) числа рівні, їх НСД дорівнює одному з чисел, тобто НСД(N,M) = N;

2) числа не рівні; НСД(N, M) = НСД(N, M + N) = НСД(N + M, M); крім того, при N>M НСД(N,M)=НСД(N-M,M), а при M>N НСД(N,M)=НСД(N,M-N).

Останні співвідношення і забезпечують основний принцип зведення рішення задачі до підзадач: значення одного з параметрів стало менше, хоча їх кількість і залишилося колишнім.

Таким чином, рішення задачі знаходження НСД(N,M) при різних значеннях N і M зводиться до двох підзадач:

- НСД(N – M, M), якщо N > M;
- НСД(N, M – N), якщо M > N.

**Приклад 3.**

Гравець збирає речі з полиці та складає їх у рюкзак. Визначити вагу рюкзака після того, як збір речей закінчено.

Цю задачу можна представити як задачу знахождення суми N елементів таблиці A.

Нехай функція S(N) відповідає рішенню нашої вихідної задачі. Ця функція має один аргумент N – кількість елементів таблиці A, сума яких обчислюється. Для пошуку суми N елементів досить знати суму перших N-1 елементів і значення N-го елемента. Тому рішення вихідної задачі можна записати у вигляді співвідношення

S(N) = S(N – 1) + aN.

Це співвідношення справедливо для будь-якої кількості елементів N≥1.

Співвідношення можна переписати у вигляді

S(i) = S(i – 1) + ai при i≥1,

S(0) = 0.

Послідовне застосування першого співвідношення при i = 1, 2, ..., N і використовується при обчисленні суми N елементів, при цьому обчислення функції проводиться від менших значень номерів аргументів до великих.

## ПОДАННЯ ЗАДАЧ У ВИГЛЯДІ І/АБО-ГРАФІВ

Для деяких категорій задач подання у вигляді І/АБО-графа є природним. Таке уявлення сформоване на розбитті задачі на підзадачі. Розбиття на підзадачі дає переваги в тому випадку, коли підзадачі є взаємно незалежними, а, отже, і вирішувати їх можна незалежно одну від одної.

І/АБО-граф – це спрямований граф, вершини якого відповідають задачам, а дуги – відношення між задачами. Між дугами також існують свої відношення. Це відношення І/АБО, в залежності від того, чи можемо ми вирішити тільки одну з задач-наступників або ж повинні вирішити усі з них (рис. 5). У І/АБО графі вирішити P (рис. 5, а) — значить вірі шити P1 або P2 або Pn ; вірі шити Q (рис. 5, б) — значить вірі шити все: Q1 і Q2 і Qn.

В принципі з вершини можуть виходити дуги, що перебувають у відношенні І разом з дугами, що знаходяться в відношенні АБО. Тим не менш, ми будемо припускати, що кожна вершина має або тільки І-наступників, або тільки АБО-наступників; справа в тому, що в таку форму можна перетворити будь-який І/АБО граф, вводячи в нього при необхідності допоміжні АБО-вершини. Вершину, з якої виходять тільки І-дуги, називають І-вершиною; вершину, з якої виходять тільки АБО-дуги, – АБО-вершиною.

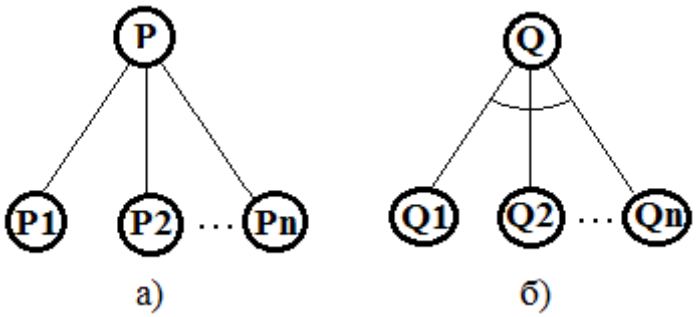



Рисунок 5 – Варіанти з'єднання вершин у І/АБО графі

Рішення в разі І/АБО-подання має включати в себе всі підзадачі І-вершини. Отже, це вже не шлях, а дерево. Таке дерево рішення Т визначається наступним чином:

- вихідна задача Р – це корінь дерева Т;
- якщо Р є АБО-вершиною, то в Т міститься тільки один з її наступників (з І/АБО-графа) разом зі своїм власним деревом рішення;
- якщо Р – це І-вершина, то все її наступники (з І/АБО-графа) разом зі своїми деревами рішень містяться в Т.

Дуги І.АБО графа можна «навантажувати» вартостями (рис. 4): d, g і h є цільовими вершинами; а – вихідна задача. На рис. 4(б) і (в) зображені два вирішальних дерева, вартості яких дорівнюють 9 і 8 відповідно. Тут вартість вирішального дерева визначена як сума вартостей всіх вхідних в нього дуг.

**Приклад 4.**

Потрібно відшукати на карті доріг маршруту між двома заданими містами (рис. 6). Не будемо поки враховувати довжину шляхів.

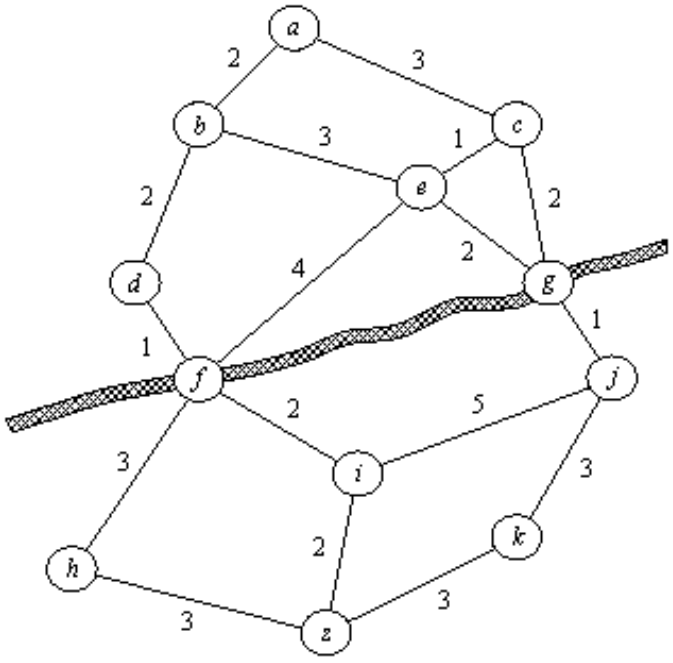



Рисунок 6 – Пошук маршруту з а в z на карті доріг. Через річку можна переправитися в містах f і g.

Цю задачу можна сформулювати як пошук шляху в просторі станів. Простір станів буде виглядати, як карта (рис. 6): вершини відповідають містам, дуги – дорогами між містами.

При формулюванні задачі, заснованому на природному розбитті на підзадачі врахуємо також річку. Припустимо, що переправитися через неї можна тільки по двох мостах: один розташований в місті f, інший – в місті g. Очевидно, що шуканий маршрут обов'язково повинен проходити через один із мостів, а значить, він повинен пройти або через f, або через g. Таким чином, для того, щоб знайти шлях з a в z ми маємо дві головні альтернативи:

Припустимо, що переправитися через неї можна тільки по двох мостах:

(1) шлях з a в z через f;

(2) шлях з a в z через g.

Виконаємо подання задачі у вигляді І/АБО графа (рис. 7). Вершини відповідають задачам або підзадачам, напівкруглі дуги означають, що всі (точніше, обидві) підзадачі повинні бути вирішені.

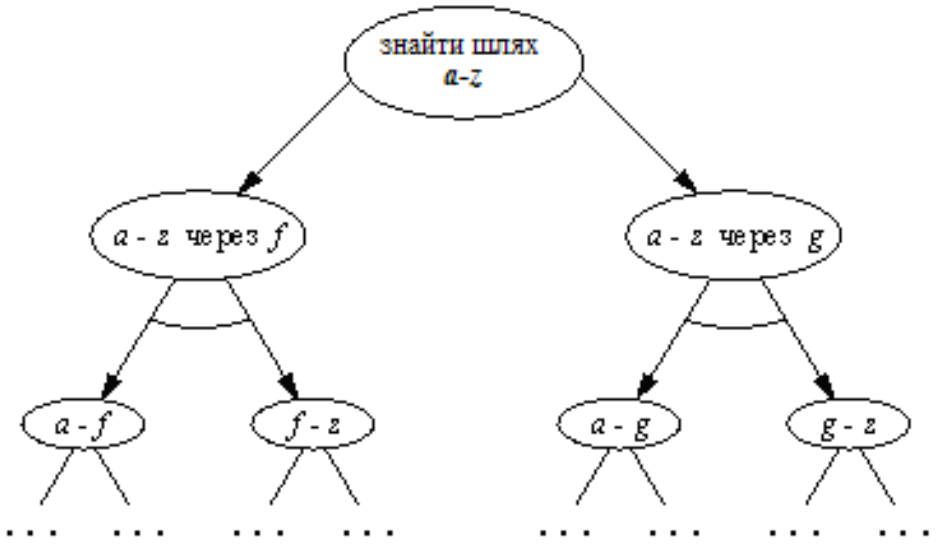



Рисунок 7 – Подання задачі пошуку маршруту у вигляді І/АБО графа

Тепер кожну з цих двох альтернативних задач можна, в свою чергу, розбити наступним чином:

(1) для того, щоб знайти шлях з a в z через f, необхідно:

1.1 знайти шлях з a в f;

1.2 знайти шлях з f в z;

(2) для того, щоб знайти шлях з a в z через g, необхідно:

2.1 знайти шлях з a в g;

2.2 знайти шлях з g в z.

Важливою є обставина, що кожну з підзадач в обох варіантах можна вирішувати неза-

лежно від іншої. Граф, показаний на рис. 2, – це лише верхня частина всього І/АБО-дерева. Подальше розбиття підзадач можна було б будувати на основі введення додаткових проміжних міст.

Цільові вершини І/АБО-графа – це тривіальні, або «примітивні» завдання. У нашому прикладі такою підзадачею можна було б вважати підзадачу «знайти шлях з а в с», оскільки між містами а й с на мапі є безпосередній зв'язок.

Використовуючи вартості, ми можемо формулювати критерії оптимальності рішення. Наприклад, можна визначити вартість вирішального графа як суму вартостей всіх дуг, що входять до нього. Тоді, якщо шукаємо рішення з мінімальною вартістю, ми віддамо перевагу вирішального графу, зображеному на рис. 8(в)

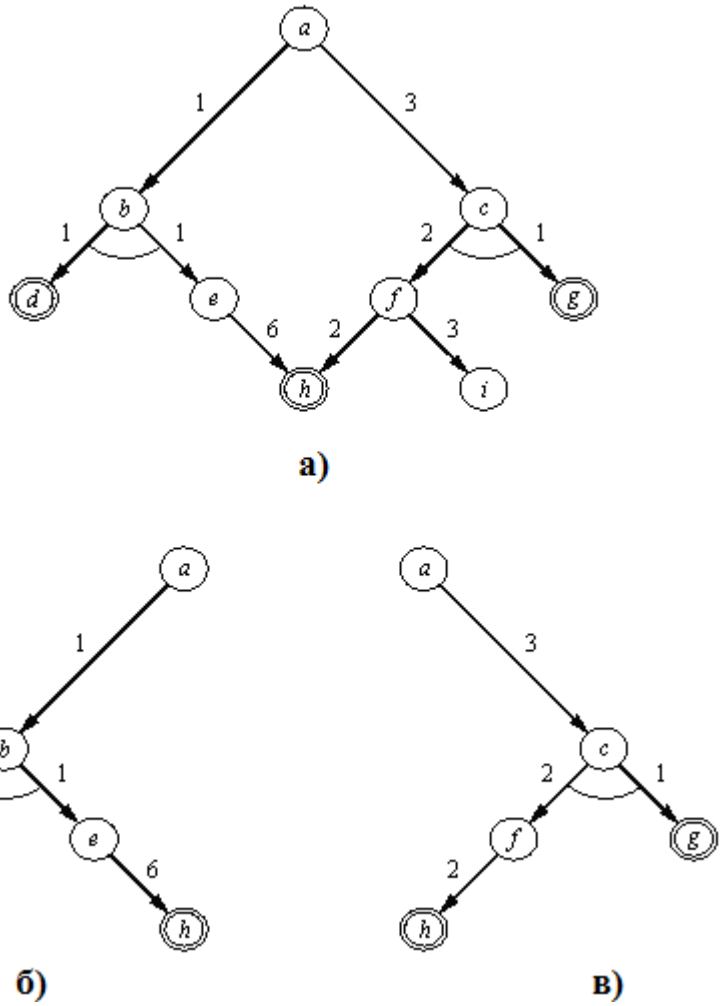



Рисунок 8 – Зважений граф

Ступінь оптимальності рішення можна вимірювати, базуючись не тільки на цінах дуг. Іноді більш природним виявиться приписувати ціна не дуг, а вершин, або ж і тим, і іншим одночасно.

Таким чином:

- І/АБО-подання сформоване на філософії зведення задачі до під задач;
- вершини І/АБО-графа відповідають задачам; зв'язки між вершинами – відношенням між задачами;

- вершина, з якої виходять АБО-зв'язки, називається АБО-вершиною; для того, щоб вирішити відповідну задачу, потрібно вирішити одну з її задач-наступників;
- вершина, з якої виходять І-зв'язки, називається І-вершиною; для того, щоб вирішити відповідну задачу, потрібно вирішити всі її задачі-наступники;
- при заданому І/АБО-графі конкретна задача визначається заданням стартової вершини та цільової умови для розпізнавання цільових вершин;
- цільові вершини (або «термінальні вершини») відповідають тривіальним (або «примітивним») задачам;
- рішення представляється у вигляді графа рішення, який є підграфом всього І/АБО-графа;
- подання задач в формі простору станів можна розглядати як спеціальний окремий випадок І/АБО-подання, коли всі вершини І/АБО-графа є АБО-вершинами.
- І/АБО-подання має перевагу в тому випадку, коли вершинами, що знаходяться в відношенні І, представлені підзадачі, які можна вирішувати незалежно одну від одної; критерій незалежності можна дещо послабити, а саме вимагати, щоб існував такий порядок вирішення І-задач, при якому рішення більш «ранніх» підзадач не руйнувалося б при вирішенні більш «пізніх» підзадач.
- дугам або вершинам, або і тим, і іншим можна приписати вартості з метою отримати можливість сформулювати критерій оптимальності рішення.

# ЛАБОРАТОРНА РОБОТА №2. ПОДАННЯ ІЗ У ПРОГРАМНІЙ СИСТЕМІ. ВИКОНАННЯ ПОШУКУ РІШЕНЬ У ПРОСТОРІ СТАНІВ

## МЕТА РОБОТИ:

- знайомство зі способами та алгоритмами пошуку рішень у просторі станів;
- вивчення основних можливостей застосування та програмної реалізації алгоритмів пошуку у просторі станів.

## ПОРЯДОК РОБОТИ

Вивчити інструкцію та лекційний матеріал.

Для предметних областей, поданих у просторі станів у лабораторній роботі №1, розробити програмний додаток з функціями:

- подання та редагування станів;
- додавання та видалення станів;
- подання функцій перетворення станів;
- подання графу станів;
- редагування графу станів.

Виконати програмну реалізацію алгоритмів пошуку рішення у просторі станів, заданих викладачем.

Порівняти результати роботи алгоритмів за ефективністю пошуку.

Продемонструвати результати розробки викладачеві.

Оформити звіт.

Захистити роботу.

## ПОШУК ШЛЯХУ

Одним з аспектів формальних систем є пошук усіх можливих варіантів рішення задачі, поставленої перед інтелектуальною системою. Досить часто людина саме так вирішує проблеми, наприклад, шахіст вибирає найкращий хід або інженер розглядає ряд можливих причин несправності пристрою. Така поведінка лежить в основі методу пошуку рішення в просторі станів.

Пошук шляху (pathfinding) – термін в інформатиці та штучному інтелекті, який означає визначення комп'ютерною програмою найкращого, оптимального маршруту між двома точками.

Приклади.

1. Місто має мережу доріг, частина яких має односторонній рух. Знайти найкоротші шляхи з

2. Є деяка кількість автобусних маршрутів між містами країни, які виконуються різними перевізниками. Для кожного маршрута відома вартість квитка. Вартість проїзду з міста A до міста B залежить від фірми-перевізника та може від не дорівнювати вартості проїзду з B в A. Знайти маршрут мінімальної вартості (можливо, з пересадками) з міста A до міста B.

Пошук шляху в контексті комп'ютерних ігор стосується шляху, на якому об'єкт, що рухається шукає шлях навколо перешкод. Найбільш часто завдання пошуку шляху виникає в стратегіях реального часу, в яких гравець дає завдання ігровим юнитам (одиницям) рухатися через ігровий рівень, який містить перешкоди. Крім стратегій, завдання пошуку шляху, так чи інакше, в тій чи іншій мірі зустрічається в більшості сучасних ігрових жанрів. Так як ігри стають все складніше, то пошук шляху також еволюціонує і розвивається разом з ними.

Стратегії реального часу зазвичай містять великі території з відкритим ландшафтом, в яких пошук шляху зазвичай є простим завданням. Однак в більшості випадків по карті переміщується не один юніт, а кілька, що створює потребу в різних і набагато більш складних алгоритмах пошуку шляху для уникнення заторів у вузьких областях ігрового ландшафту. У стратегіях ігровий рівень ділиться на тайли (tiles), які діють як вузли (nodes) в алгоритмі пошуку шляху.

У жанрі 3D-шутерів використовуються набагато більш обмежені простори, які не так легко розділити на вузли. Тут замість вузлів використовуються так звані «точки шляху» (waypoints). Waypoints – це нерегулярні і вручну встановлені вузли, які містять інформацію про те, до яких інших вузлів можливо дістатися від даного.

## СТРАТЕГІЇ ПОШУКУ В ПРОСТОРІ СТАНІВ

Подання задач у просторі станів (ПС) передбачає задання описів станів, множини операторів і їх впливів на переходи між станами, цільових станів. Описи станів можуть являти собою рядки символів, вектори, двомірні масиви, дерева, списки та ін. Оператори переводять один стан в інший. Простір станів можна зобразити як граф, вершини якого позначені станами, а дуги – операторами.

Таким чином, проблему пошуку розв'язку задачі <A,B> в разі планування за станами можна розглядати як проблему пошуку на графі шляху з A до B. Звичайно графи не задають, а генерують у міру потреби.

Пошук в ПС можна вести в таких напрямках: від вихідних даних задачі до мети, в зворотному напрямку від мети до вихідних даних, або застосовувати комбінацію пошуку у прямому та зворотному напрямках.

При пошуку на основі даних (прямий ланцюжок) дослідник починає процес вирішення задачі, аналізуючи її умову, а потім застосовує допустимі ходи або правила для переходу до іншого стану.

Можливий альтернативний підхід (зворотний ланцюжок). Розглядається мета, яку потрібно досягти. Аналізують допустимі ходи, що ведуть до мети, і визнають умови їх застосування. Ці умови стають новими цілями, або підцілями, пошуку. Пошук триває в зворотному напрямку від досягнутих цілей до тих пір, поки не досягнуто вихідних даних завдання.

В обох випадках працюють з одним і тим же графом ПС, однак порядок і число станів у процесі пошуку можуть відрізнятися. Яку стратегію пошуку віддати перевагу залежить від поставленої задачі. При цьому слід враховувати складність правил перетворення

станів, природу і доступність даних задачі. Вибір залежить також від структури розв'язуваної задачі.

Процес пошуку від мети рекомендований в наступних випадках:

1. Мета пошуку явно присутня в постановці задачі або може бути легко сформульована.

2. Є велика кількість правил, які на основі отриманих даних дозволяють генерувати зростаюче число цілей. Своєчасний відбір цілей дозволяє відсіяти безліч можливих гілок, що робить процес пошуку в ПС більш ефективним.

3. Вихідні дані не наводяться в задачі, але мають бути відомі досліднику. У цьому випадку пошук від мети може служити керівництвом для правильної постановки завдання.

Пошук на основі даних застосується до вирішення задачі в наступних випадках:

1. Всі або більшість вихідних даних задані в постановці задачі.

2. Існує велика кількість потенційних цілей, але всього лише кілька способів застосування даних і конкретних прикладів рішення задачі.

3. Сформувати мету або гіпотези рішення задачі дуже складно.

2. Можлива реалізація пошуку на графах.

3. Можлива реалізація пошуку з поверненням.

## АЛГОРИТМИ ПОШУКУ У ПРОСТОРІ СТАНІВ

### Алгоритм Беллмана-Форда

Постановка задачі.

Дано орієнтований або неорієнтований граф G зі зваженими ребрами. Довжиною шляху назвемо суму ваг ребер, що входять в цей шлях. Потрібно знайти найкоротші шляхи від виділеної вершини s до всіх вершин графа. Найкоротших шляхів може не існувати. Так, в графі, що містить цикл з негативною сумарною вагою, існує як завгодно короткий шлях від однієї вершини цього циклу до іншої (кожен обхід циклу зменшує довжину шляху). Цикл, сума ваг ребер якого негативна, називається негативним циклом.

Рішення для орієнтованого графа, що не містить контурів негативною довжини.

Основними обчислюваними величинами цього алгоритму є $\lambda_i(k)$, де i=1,2,..., n (n – число вершин графа); k = 1, 2, ..., n – 1. Для фіксованих i та k величина $\lambda_i(k)$ дорівнює довжині мінімального шляху, що веде з заданої початкової вершини $x_1$ у вершину xi та містить не більше k дуг.

Крок 1. Встановлення початкових умов.

Ввести число вершин графа n та матрицю ваг C = (cij).

Крок 2. k = 0; $\lambda_i(0) = \infty$ для всіх вершин, крім xi; $\lambda_i(0) = 0$.

Крок 3. У циклі по k, k = 1, .., n – 1, кожній вершині xi на k-му кроці приписати індекс $\lambda_i(k)$ за таким правилом:

$$\lambda_i(k) = \min_{1 \le j \le n} \{\lambda_j(k-1) + c_{ij}\} \quad (1)$$

для всіх вершин, крім x1, встановити $\lambda_1(k) = 0$.

В результаті роботи алгоритму формується таблиця індексів

$$\lambda_1(k),\ i = 1, 2, \ldots, n,\ k=0,1,2,\ldots,n-1.$$

При цьому $\lambda_1(k)$ визначає довжину мінімального шляху з першої вершини в i-у, що містить не більше k дуг.

Крок 4. Відновлення мінімального шляху.

Для будь-якої вершини xs попередня їй вершина xr визначається зі співвідношення:

$$\lambda_r(n-2)+crs= \lambda_s(n-1),\ xr \in G^{-1}(xs), \textbf{(2)}$$

де $G^{-1}(xs)$ – прообраз вершини xs.

Для знайденої вершини xr попередня їй вершина xq визначається зі співвідношення:

$$\lambda q(n-3)+cqr= \lambda r(n-2),\ xq \in G^{-1}(xr),$$

де $G^{-1}(xr)$ – прообраз вершини xr і т.д.

Послідовно застосовуючи це співвідношення, починаючи від останньої вершини xi, знайдемо мінімальний шлях.

**Алгоритм Левіта**

Знаходить найкоротшу відстань від однієї з вершин графа до всіх інших, працює для і для графів з ребрами негативного ваги.

Постановка задачі.

Маємо зважений орієнтований граф G(V,E) без петель. Знайти найкоротші шляхи від деякої вершини a графа до всіх інших вершин цього графа.

Позначення:

N — множина вершин графа;

M — множина ребер графа;

g[ij] — вага (довжина) ребра ij;

s — стартова вершина, від якої шукають відстань;

d[i] — поточна довжина найкоротшого шляху від вершини s до вершини i;

p[i] —вершина, попередня вершині i у найкоротшому шляху від вершини s до i;.

M0 — вершини, відстань до яких вже обчислено (але, можливо, не остаточно);

M1 — вершини, відстань до яких обчислюється;

M2 — вершини, відстань до яких ще не обчислено;

state[i] — зберігає номер множини, до якої належить вершина i;

Рішення.

Нехай масив D[1..N] буде містити поточні найкоротші довжини шляхів. Спочатку масив D заповнений значеннями «нескінченність», крім D[s] = 0. Після закінчення роботи алгоритму цей масив буде містити остаточні найкоротші відстані.

Нехай масив P[1..N] містить поточних предків. Так само як і масив D, масив P змінюється поступово по ходу алгоритму і до кінця його приймає остаточні значення.

Спочатку всі вершини поміщаються в множину M2, крім вершини V0, яка поміщається в множину M1.

На кожному кроці алгоритму ми беремо вершину з множини M1 (дістаємо верхній елемент з черги). Нехай V – це вибрана вершина. Переводимо цю вершину в множину M0. Потім переглядаємо всі ребра, що виходять з цієї вершини. Нехай T – це другий кінець поточного ребра (тобто не рівний V), а L – це довжина поточного ребра.

- Якщо T належить M2, то T переносимо в множину M1 в кінець черги. DT вважаємо рівним DV + L.
- Якщо T належить M1, то намагаємося поліпшити значення DT: DT = min (DT, DV + L). Сама вершина T ніяк не пересувається в черзі.
- Якщо T належить M0, і якщо DT можна поліпшити (DT> DV + L), то покращуємо DT, а вершину T повертаємо в множину M1, поміщаючи її в початок черги.

При кожному оновленні масиву D слід оновлювати і значення в масиві P.

**Метод перебору в глибину** *(ДИВ. ЛЕКЦІЙНИЙ МАТЕРІАЛ).*

**Метод перебору в ширину** *(ДИВ. ЛЕКЦІЙНИЙ МАТЕРІАЛ).*

**Двонаправлений пошук**

Двонаправлений пошук шляху в ширину (або глибину) – ускладнений алгоритм пошуку в ширину (або глибину). Ідея пошуку полягає у формуванні процесу пошуку від початкової (прямий пошук) і від кінцевої вершини (зворотний пошук) графа.

Знаходження шуканого шляху зводиться до визначення шляхів від початкової до певної проміжної, а від неї до кінцевої вершини. Реалізується перевіркою в одному або обох процесах, коли лист одного дерева пошуку співпаде з листом іншого, після чого виділяються шляху до цього елемента. Поєднавши шляхи, отримуємо шуканий шлях. Якщо два пошуки здійснюються паралельно – це ще більше економить час на отримання шуканого шляху в порівнянні з односпрямованим пошуком.

**Алгоритм A*** *(ДИВ. ЛЕКЦІЙНИЙ МАТЕРІАЛ).*

**Метод гілок і меж**

З незакінчених шляхів, що формуються в процесі пошуку, вибирається найкоротший і продовжується на один крок. Отримані нові незакінчені шляхи (їх стільки, скільки гілок в даній вершині) розглядаються поряд зі старими, і знову продовжується на один крок найкоротший з них. Процес повторюється до першого досягнення цільової вершини, рішення запам'ятовується.

Потім з решти незакінчених шляхів виключаються довші, ніж закінчений шлях, або рівні йому, а ті, що залишилися, продовжуються за таким же алгоритмом до тих пір, поки їх довжина менше закінченого шляху. У результаті або всі незакінчені шляхи виключаються, або серед них формується закінчений шлях, коротший, ніж раніше отриманий. Останній шлях починає грати роль еталону і т.д.

### Алгоритм найкоротших шляхів Мура

Вихідну вершину X0 позначають числом „0". Нехай у ході роботи алгоритму на поточному кроці отримано множину дочірніх вершин X(xi) вершини xi. Тоді з нього викреслюють усі раніше отримані вершини, решту позначають міткою, збільшеною на одиницю порівняно з міткою вершини xi, і від них проводять покажчики до Xi. Далі на множині позначених вершин, які ще не фігурують як адреси покажчиків, вибирають вершину з найменшою міткою і для неї будують дочірні вершини. Розмітку вершин повторюють, доки не буде отримана цільова вершина.

### Алгоритм Дейкстри

Є узагальненням алгоритму Мура внаслідок уведення дуг змінної довжини.

### Алгоритм Дорана і Мічі

Виконує пошук з низькою вартістю. Застосовуються, коли вартість пошуку велика порівняно з вартістю оптимального рішення. У цьому випадку замість вибору вершин, найменш віддалених від початку, як в алгоритмах Мура і Дейкстри, вибирають вершину, для якої евристична оцінка відстані до мети найменша. У разі добре сформованої оцінки можна швидко одержати рішення, але немає гарантії, що шлях буде мінімальним.

### Алгоритм Харта, Нільсона і Рафаеля

Поєднує критерії вартості шляху до вершини g(x) і вартості шляху від вершини h(x) в адитивній оціночній функції f(x) =g(x)+h(x). За умови h(x)<hp(x), де hp(x) – дійсна відстань до мети, алгоритм гарантує знаходження оптимального шляху.

# ЛАБОРАТОРНА РОБОТА №3. ПОДАННЯ ІЗ У ПРОГРАМНІЙ СИСТЕМІ. ВИКОНАННЯ ПОШУКУ РІШЕНЬ У ПРОСТОРІ ЗАДАЧ

## МЕТА РОБОТИ:

- знайомство зі способами та алгоритмами пошуку рішень у просторі задач;
- вивчення основних можливостей застосування та програмної реалізації алгоритмів пошуку у просторі задач.

## ПОРЯДОК РОБОТИ

Вивчити інструкцію та лекційний матеріал.

Для предметних областей, поданих у просторі задач у лабораторній роботі №1, розробити програмний додаток з функціями:

- подання та редагування задач;
- додавання та видалення задач;
- подання І-АБО графа;
- редагування І-АБО графа.

Оцінити результат роботи алгоритму за ефективністю пошуку.

Продемонструвати результати розробки викладачеві.

Оформити звіт.

Захистити роботу.

## РОЗВ'ЯЗАННЯ ЗАДАЧ МЕТОДОМ РЕДУКЦІЇ

Якщо розв'язання задач має ієрархічну структуру, доцільно застосувати метод редукції. Основну задачу і всі її підзадачі не обов'язково розв'язувати однаковими методами. Редукція ефективна для подання глобальних аспектів задачі, а в процесі розв'язання більш специфічних задач краще застосовувати метод планування за станами. Метод планування за станами можна розглядати як окремий випадок методу планування за допомогою редукцій, тому що кожне застосування оператора в просторі станів означає зведення вхідної задачі до двох більш простих, одна з яких є елементарною.

Пошук планування в просторі задач полягає в послідовному зведенні вхідної задачі до все більш простої, доки не будуть отримані тільки елементарні задачі. Частково впорядкована сукупність таких задач складе розв'язок вихідної задачі.

Розчленування задачі на альтернативні множини підзадач зручно подавати у вигляді І/АБО-графа. У такому графі будь-яка вершина, крім кінцевої, має або кон'юнктивно зв'язані дочірні вершини (І-вершина), або диз'юнктивно зв'язані (АБО-вершина). За відсутності І-вершин має місце граф простору станів. Кінцеві вершини є або заключні (їм відповідають елементарні задачі), або тупикові. Початкова вершина (корінь І/АБО-графа) відображає вхідну задачу.

Мета пошуку на І/АБО-графі – показати, що початкова вершина є розв'язуваною. Розв'язуваними є заключні вершини (І-вершини), у яких розв'язувані всі дочірні вер-

шини, і АБО-вершини, у яких розв'язувана хоча б одна дочірня вершина. Розв'язувальний граф складається з розв'язуваних вершин і вказує спосіб розв'язання початкової вершини. Наявність тупикових вершин приводить до нерозв'язуваних вершин. Нерозв'язувані є тупикові вершини, І-вершини, у яких нерозв'язувана хоча б одна дочірня вершина, і АБО-вершини, у яких нерозв'язувана кожна дочірня вершина.

Алгоритм Ченга і Слейгла грунтується на перетворенні довільного І/АБО-графа на спеціальний АБО-граф, кожна АБО-гілка якого має І-вершини тільки в кінці. Перетворення використовує подання довільного І/АБО-графа як довільної формули логіки висловлювань із подальшим перетворенням цієї довільної формули на диз'юнктивну нормальну форму. Таке перетворення дозволяє далі застосовувати алгоритм Харта, Нільсона і Рафаеля.

Опис основних алгоритмів пошуку у просторі задач дано у лекційному матеріалі.

Навчальне видання

НІКІТІНА Людмила Олексіївна
НІКІТІН Сергій Олександрович

# МОДЕЛІ ТА МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ У КОМП'ЮТЕРНИХ ІГРАХ

Роботу до видання рекомендував *О.А. Сєрков*

Редактор *Л.П. Гобельовська*
Комп'ютерна верстка *Ю.Ф. Власова*
Коректор *Г.В. Сільченко*

Підписано до друку 07.02.2018 р.
Формат 60x84 1/16. Папір офсетний.
Цифровий друк.
Гарнітура DINPro. Ум. друк. арк. 5,93.
Наклад 50 прим. Зам. № ГЛ00-000476.

Видавець і виготовлювач:
ТОВ «Друкарня Мадрид»
61024, м. Харків, вул. Максиміліанівська, 11
Тел.:(057)756-53-25
www.madrid.in.ua info@madrid.in.ua

Свідоцтво суб'єкта видавничої справи:
ДК № 4399 від 27.08.2012 р.


9 786177 683024