Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

МЕТОДИ ПОШУКУ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

Уклав: В. А. Грабовський

Методичні рекомендації щодо виконання лабораторних робіт
з курсу «Методи і системи штучного інтелекту»
для студентів напряму 6.050101 «Комп'ютерні науки»
факультету електроніки та комп'ютерних технологій

Львів 2018

Рекомендовано до друку
Вченою радою факультету електроніки та комп’ютерних технологій
Протокол № 3/9 від 23 квітня 2018 р.

Методи пошуку у системах штучного інтелекту: Методичні рекомендації щодо виконання лабораторних робіт з курсу «Методи і системи штучного інтелекту» (для студентів напряму 6.050101 «Комп’ютерні науки» факультету електроніки та комп’ютерних технологій) / Укл.: В. А. Грабовський – Львів : ЛНУ імені Івана Франка, 2018 р. – 62 с.

Методичні рекомендації містять вказівки щодо виконання лабораторних робіт з курсу «Методи і системи штучного інтелекту» і призначені для студентів напряму підготовки 6.050101 «Комп’ютерні науки» факультету електроніки та комп’ютерних технологій. Дана коротка характеристика систем штучного інтелекту, основних методів пошуку досягнення мети. Основна увага звернута на особливості використання графів для розв’язування задач штучного інтелекту, зокрема – на особливості застосування деяких часто вживаних методів пошуку шляху між заданими вершинами графу. Також представлені завдання щодо виконання лабораторних робіт та представлення їх результатів, вимоги до звітів за виконану роботу та їх захисту.

Зміст

1. СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ТА ОСОБЛИВОСТІ ЇХ ФУНКЦІОНУВАННЯ. 4
2. ПЛАНУВАННЯ РОЗВ'ЯЗКІВ ЗАДАЧ У ШТУЧНОМУ ІНТЕЛЕКТІ 6
3. КЛАСИФІКАЦІЯ МЕТОДІВ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ ВИРІШЕННІ SS- І PR-ПРОБЛЕМ. 8
3.1. Планування за станами. 8
3.2. Планування за задачами. 9
4. ГРАФИ ЯК ІНСТРУМЕНТ ПРЕДСТАВЛЕННЯ ОБ'ЄКТІВ ТА СТРУКТУРИ ВІДНОСИН В СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ. 10
4.1. Деякі визначення і поняття теорії графів. 11
4.2. Основні різновиди графів. 13
4.3. Опис (представлення) графів. 15
5. ПОШУК НА ГРАФІ 20
5.1. І/АБО графи 20
5.2. Основні методи пошуку на графах. 22
5.3. Стратегія пошуку 23
6. АЛГОРИТМИ ПОШУКУ 24
6.1. Метод повного перебору в ширину. 25
6.2. Метод перебору в глибину. 29
6.3. Хвильовий алгоритм 31
6.4. Алгоритм Дейкстри. 36
7. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СТВОРЕННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ 41
8. ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ. 45
8.1. Завдання до роботи 1. 45
8.2. Завдання до роботи 2. 46
8.3. Завдання до роботи 3. 48
8.4. Завдання до роботи 4. 49
8.5. Завдання до роботи 5. 51
Контрольні питання: 53
РЕКОМЕНДОВАНА ЛІТЕРАТУРА. 56
ДОДАТКИ 57
Додаток 1. Схема автомобільних шляхів України 57
Додаток 2. Відстані між містами України. 58
Додаток 3. Теми рефератів 60
Додаток 4. Вимоги до виконання та представлення результатів лабораторних робіт та реферату 61

1. СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ТА ОСОБЛИВОСТІ ЇХ ЗАСТОСУВАННЯ

«Область досліджень, зазвичай звану штучним інтелектом, ймовірно, можна представити як сукупність методів і засобів аналізу і конструювання машин, здатних виконувати завдання, з якими до недавнього часу могла справитися тільки людина. При цьому за швидкістю і ефективністю машини повинні бути порівнянні з людиною.»

Raman S. Banerji. Theory of problem solving. An approach to artificial intelligence. 1969)

Все життя людини – низка задач, які постають перед нею в процесі життєдіяльності і які вона повинна певним чином вирішувати. Причому, при їх вирішенні вона повинна спиратися на свої специфічні властивості, які загалом отримали назву інтелекту – властивості, які притаманні розумним істотам і характеризують їхні розумові здібності, здатність набувати знання, маніпулювати ними, аналізувати й вирішувати проблеми, що виникають в процесі життєдіяльності, адаптуватися до динамічно змінної обстановки та пристосовувати її згідно зі своїми потребами.

Щодо визначення поняття інтелекту є багато підходів, але чи не найчастіше використовується такий: інтелект – це здатність самостійно, ефективно знаходити якісні рішення різноманітних складних завдань, що постають, у тому числі й раніше невідомих. Виходячи з вищесказаного, зрозуміло, що наявність інтелекту передбачає здатність його носія осмислено набувати, відтворювати і використовувати знання для досягнення поставленої мети, розуміти конкретні й абстрактні ідеї, осягати співвідношення між ідеями й об’єктами, тощо.

Задачі, які вирішуються людьми, у багатьох випадках є складними, вимагають використання певної кількості спеціальних знань і навичок, іноді мають багато розв'язків; причому, не всі з цих розв'язків можуть задовольняти вимогам, які до них ставляться – і тому поставлені задачі не завжди можуть вирішуватися якісно і в потрібні часові терміни.

Потреби у вирішенні постійно зростаючої кількості задач, які постають перед людьми в процесі життєдіяльності, та їх складності, а також нові умови, що пред'являються до результатів, що повинні отримуватися в результаті їх розв'язання, зумовили пошуки нових методів їх вирішення. В результаті, ці пошуки в середині XX століття привели до створення нових напрямків досліджень, і, в кінцевому випадку, до створення нового напрямку науково-практичної діяльності, яку назвали штучним інтелектом.

Як і у визначенні поняття природного інтелекту, єдиного визначення поняття штучного інтелекту також нема. Сьогодні під штучним інтелектом (ШІ) загалом найчастіше розуміють область досліджень і прикладних розробок, спрямованих на створення програмно-апаратних засобів, здатних до вирішення таких завдань, вирішення яких людиною передбачає застосування нею своїх інтелектуальних здібностей. Метою ж діяльності, що проводиться у галузі штучного інтелекту, стала розробка комп'ютерних програм, здатних вирішувати завдання, які раніше могла вирішувати лише людина.

Загалом, можна сказати, що система штучного інтелекту – це програмно-апаратний комплекс, здатний вирішувати такі завдання, для розв'язання яких людина використовує свої інтелектуальні здібності. Тому такі системи ще називають інтелектуальними системами (ІС). Застосування інтелектуальних систем доцільне тоді, коли вирішувана задача не має простого рішення з відомим алгоритмом (або ж алгоритм її рішення наперед не відомий) і не може бути вирішена шляхом застосування звичайних комп'ютерних програм, або ж тоді, коли умови задачі неповні, містять неточності чи невизначеності, або ж сама задача недостатньо формалізована.

2. ПЛАНУВАННЯ РОЗВ'ЯЗКІВ ЗАДАЧ У ШТУЧНОМУ ІНТЕЛЕКТІ

Функціонування багатьох інтелектуальних систем носить цілеспрямований характер. Типовим актом такого функціонування є вирішення задачі планування шляху досягнення потрібної мети, виходячи з деякої фіксованої початкової ситуації (*під плануванням тут розуміється знаходження послідовності операцій, за допомогою яких можна розв'язати поставлену задачу*). Результатом вирішення задачі повинен бути *план дій – така частково впорядкована сукупність дій, яка приводить до вирішення проблеми*. Такий план нагадує сценарій, в якому в якості відносин між станами (ситуаціями) наочної (предметної) області, які мають місце при вирішення задачі, виступають відносини типу: «мета – підмета», «мета – дія», «дія – результат», тощо. Будь-який *шлях* у цьому сценарії, *що веде від стану, що відповідає початковій ситуації, до будь-якого з цільових станів, і визначає шуканий план дій*.

Пошук плану дій виникає в інтелектуальній системі лише тоді, коли вона зіштовхується з нестандартною ситуацією, для якої немає заздалегідь відомого набору операцій, що приводять до потрібної мети. *В залежності від особливості і способу представлення задачі, планування дій з її вирішення здійснюється різними методами – плануванням в просторі станів, плануванням шляхом декомпозиції в просторі задач, плануванням за допомогою логічного виведення, плануванням за допомогою правил-продукцій, тощо*.

Оскільки майже будь-яку задачу можна представити у вигляді сукупності певних станів предметної області, якої стосується ця задача, та їх зміни під дією певних чинників, то всі завдання побудови плану дій щодо рішення таких задач можна розбити на два типи, яким відповідають різні підходи: планування в просторі станів (т. з. SS-проблема) і планування в просторі задач (т. з. PR-проблема).

У випадку планування у просторі станів вважається заданим деякий

простір ситуацій. Опис ситуацій включає опис стану предметної області задачі і стану самої інтелектуальної системи, які характеризуються низками параметрів. Ситуації утворюють деякі узагальнені стани, а дії інтелектуальної системи або зміни в зовнішньому середовищі призводять до зміни актуалізованих в даний момент станів. Серед узагальнених станів виділяються початкові (зазвичай один) і кінцеві (цільові) стани. *Вирішення SS-проблеми полягає у пошуку шляху, що веде з початкового стану в кінцевий (або в один з кінцевих)*.

При плануванні у просторі задач ситуація дещо інша. Простір утворюється в результаті введення на множині завдань відносини типу: «загальний випадок – окремий випадок», «частина – ціле», «задача – підзадача», тощо. Іншими словами, простір задач відображає декомпозицію (розбиття) задач на підзадачі (цілі на підцілі). *Вирішення PR-проблеми полягає у пошуку такої декомпозиції вихідної задачі на підзадачі, що приводить до елементарних задач, вирішення яких системі відоме*.

Якщо задачі достатньо добре формалізовані і їх опис може бути здійснений у рамках деякої формальної теорії, то методом планування, який часто використовується в таких випадках, є планування за допомогою логічного виведення. Таке планування передбачає опис станів у вигляді т. з. правильно побудованих формул (ППФ; під ППФ розуміють формули, що складаються з символів алфавіту певної теорії *логічної моделі представлення знань* і конструюються за допомогою синтаксичних правил побудови формул цієї теорії) логічного числення, а опис операторів, які приводять до зміни цих станів – у вигляді або ППФ, або правил перетворення одних ППФ в інші ППФ. Подання операторів у вигляді ППФ дозволяє створювати дедуктивні методи планування, а представлення операторів у вигляді правил перетворення цих станів – методи планування з елементами дедуктивного виведення.

Якщо задачу можна представити у вигляді сукупності фактів та правил їх перетворення (т. з. *продукцій*), то рішення такої задачі у штучному інтелек-

ті можна здійснити в рамках *продукційної системи представлення знань*. Зазвичай, при вирішенні такої задачі не потребуються глибинні знання про проблему, яка вирішується, а сам процес пошуку шляху її розв'язання полягає у знаходженні сукупності наявних у системі правил, ліві (порівняльні, або ж антецедентні) частини яких задовольняються наявними робочій пам'яті системи фактами, та виконанні відповідних дій, передбачених правими (дієвими, або консеквентними) частинами вибраних таким чином правил. Результатами пошуку рішення задачі у цьому випадку буде низка впорядкованих правил, послідовне використання яких приводить до поставленої в ній мети.

3. КЛАСИФІКАЦІЯ МЕТОДІВ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ ВИРІШЕННІ SS- І PR-ПРОБЛЕМ

3.1. Планування за станами

Подання завдань у просторі станів припускає задання ряду описів: множини станів (у т. ч. початкових і цільових) розглядуваної системи, множини операторів переходів та їх впливів на переходи між станами. Описи станів можуть являти собою рядки символів, вектори, двомірні масиви, дерева, списки, тощо. Оператори переводять один стан в інший і їх представлення може мати різний вигляд. Часто вони представляються у вигляді т. з. продукцій виду $A \Rightarrow B$, що означає: за умови задоволення певних умов (A) над системою буде здійснена певна процедура (B), внаслідок чого система з поточного стану перейде в новий стан.

У термінах станів і операторів рішенням задачі є певна послідовність операторів, що перетворює початковий стан в цільовий у множині всіх станів, досяжних з початкового стану за допомогою заданих операторів.

Простір станів зручно представити у вигляді графу, вершинами якого є

стани, а ребрами (дугами) – оператори переходів між цими станами. У цьому випадку проблема пошуку рішення задачі при плануванні за станами (задачі <*A, B*>) представляється як проблема пошуку на графі шляху з вершини *A* у вершину *B*, а її вирішення полягає в знаходженні шляху від стану *A* до стану *B*. Зазвичай графи не задаються, а генеруються у міру потреби.

3.2. Планування за задачами

Цей метод приводить до гарних результатів у випадку, якщо саму задачу та її рішення можна представити у вигляді певної ієрархічної структури. Причому, потрібно мати на увазі, що в процесі пошуку рішення не обов'язково вимагати, щоб основна задача та всі її підзадачі вирішувалися однаковими методами.

Пошук рішення в просторі задач полягає у послідовному зведенні початкової задачі до все більш простих до тих пір, поки не будуть отримані тільки елементарні задачі, рішення яких тривіальне (або ж наперед відоме). Частково упорядкована сукупність таких задач і складе рішення вихідної задачі.

Розчленування задачі на альтернативні множини підзадач зручно представляти у вигляді І/АБО-графа. У такому графі всяка вершина, крім кінцевої, має кон'юнктивно чи диз'юнктивно пов'язані дочірні вершини (відповідно, розглядають І- та АБО-вершини графу). В окремому випадку, за відсутності І-вершин, має місце граф простору станів.

Початкова вершина (корінь І/АБО-графа) представляє вихідні умови задачі. Кінцеві вершини є або заключними (їм відповідають елементарні завдання), або тупиковими (нерозв'язними – такими, що не мають розв'язків). Розв'язуваними є заключні І-вершини, у яких вирішувані всі дочірні вершини, та АБО-вершини, у яких вирішувана хоча б одна дочірня вершина.

Мета пошуку на І/АБО-графі – показати, що початкова вершина розв'язувана, тобто для цієї вершини існує т. з. вирішальний граф. (*Під вирі-*

шальним графом розуміється підграф графу, який складається тільки з розв'язуваних вершин і який, таким чином, вказує спосіб розв'язування початкової вершини.) Наявність тупикових вершин призводить до нерозв'язних вершин. Нерозв'язними є тупикові І-вершини, у яких нерозв'язна хоча б одна дочірня вершина, та АБО-вершини, у яких нерозв'язні всі дочірні вершини.

Загалом, редукція (у даному випадку – розбиття задачі на підзадачі) корисна для представлення глобальних аспектів завдання; при вирішенні ж більш специфічних завдань перевага надається методу планування за станами.

Метод планування за станами можна розглядати як окремий випадок методу планування за допомогою редукцій, оскільки кожне застосування оператора в просторі станів означає за своєю суттю зведення вихідної задачі до кількох більш простих. У загальному випадку редукція вихідної задачі зводиться до низки послідовних перетворень, які призводять до формування в кінцевому випадку таких підзадач, з яких хоча б одна була елементарною або рішення якої відоме.

Таким чином, у системах штучного інтелекту в багатьох випадках при вирішенні поставленої задачі її зручно представити у вигляді графу з зазначеними початковою та цільовою вершинами. Розв'язком цієї задачі зазвичай є знаходження шляху на цьому графі між вказаними вершинами. Деякі методи пошуку цього шляху на графах розглядаються нижче.

4. ГРАФИ ЯК ІНСТРУМЕНТ ПРЕДСТАВЛЕННЯ ОБ'ЄКТІВ ТА СТРУКТУРИ ВІДНОСИН В СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

Теорія графів є одним з найкращих інструментів дослідження структури об'єктів та відносин між ними. Разом з тим, теорія графів представляє потужний апарат для вирішення завдань різного ступеня складності в найріз-

номанітніших галузях наукової і практичної діяльності, серед яких – аналіз та синтез електричних кіл та систем, проектування мереж зв'язку та дослідження скінченних автоматів, мережне планування й управління, вибір оптимальних маршрутів та потоків у мережах, моделювання процесів життєдіяльності й функціонування нервової системи в живих організмах, тощо. Завдяки винятковим можливостям представлення різноманітних предметних областей та їх станів, а також наявності розроблених алгоритмів знаходження шляху до мети, графи знайшли широке застосування при вирішенні різноманітних задач в галузі штучного інтелекту.

Загалом, під графом розуміють множину однотипних об'єктів (вершин), деякі з яких пов'язані один з одним певними зв'язками (*ребрами* або *дугами*; під *дугою* розуміють орієнтоване ребро графу). Один зв'язок завжди з'єднує тільки дві вершини (іноді – вершину саму з собою; тоді ребра мають збіжні кінці і їх називають *петлями*).

4.1. Деякі визначення і поняття теорії графів

Графом $G=(V,E)$ називається об'єкт, який заданий парою множин (V, E), де V – множина його вершин, а $E \subseteq V \times V$ – множина ребер (дуг) графу.

Для задання графу необхідно задати такі два об'єкти:

1) не порожню множину $V=\{x_1, x_2, \ldots, x_n\}$; елементи цієї множини називають *вершинами графу*;

2) деяку множину невпорядкованих пар елементів з V; ця множина позначається E, а її елементи називають *ребрами графу.*

Множину вершин графу G часто позначають $V(G)$, а множину ребер – $E(G)$. Тоді кількість вершин графу визначається як $n(G)=|V(G)|$, а кількість ребер – як $m(G)=|E(G)|$. Кількість вершин $n(G)$ графу називають його *порядком,* а кількість його ребер $m(G)$ – *розміром графу*.

За наочного подавання графу його вершини зображуються у вигляді

певних геометричних об'єктів (зокрема, *точок*), а ребра – *лініями*, які їх з'єднують.

Граф називається *скінченним*, якщо кількість його елементів скінченна, тобто множини його вершин і ребер є скінченними.

Граф, в якому кожна пара вершин з'єднана ребром, називається *повним.*

Граф називається *простим*, якщо кожну пару вершин з'єднує не більше, ніж одне ребро. Іншими словами, простим є граф без кратних ребер і петель.

Граф, що містить тільки ребра, називається *неорієнтованим*; граф, у складі якого є тільки дуги – *орієнтованим*; граф, що містить як дуги, так і ребра – *змішаним*.

Граф, який має кратні ребра – тобто одну й ту саму пару вершин з'єднує декілька ребер – називається *мультиграфом*, а граф, який має і петлі, і кратні ребра – *псевдографом*.

Кореневий граф містить єдину вершину, від якої існує шлях до будь-якої іншої вершини графу; ця вершина називається коренем графу. При зображенні кореневого графу корінь зазвичай поміщають у верхній частині рисунка над усіма іншими вершинами.

Дерево – це граф, на якому для будь-яких двох вершин існує не більше одного шляху між ними. Дерева зазвичай мають корені, які зображуються у верхній частині малюнка, як і для кореневих графів. Оскільки для кожної вершини дерева існує не більше одного шляху в цю вершину з будь-якої іншої вершини, то в ньому не існує шляхів, що містять петлі або цикли.

Для кореневих дерев або графів відносини між вершинами описуються поняттями *батька*, *нащадка* і *вершин-братів(сестер)* – вершин дерева, що мають спільну вершину-батька. Вони використовуються в звичайному сенсі успадкування – при проході уздовж спрямованої дуги батько завжди передує нащадку. Вершини, які розміщені на кінцях усіх спрямованих дуг, що виходять з однієї вершини-батька, називаються вершинами-братами. Аналогічно,

на шляху орієнтованого графу предок завжди передує нащадку.

Шлях у графі G з вершини v_0 у вершину v_n – послідовність вершин v_0, v_1, v_2, ... v_{n-1}, v_n така, що $v_0 \rightarrow v_1$, $v_1 \rightarrow v_2$, ..., $v_{n-1} \rightarrow v_n$.

Довжина шляху у графі – число (для зважених графів – сума ваг) ребер (дуг), з яких складається шлях.

4.2. Основні різновиди графів

- ✓ *неорієнтовані* (звичайні), в яких важливий лише сам факт зв'язку двох вершин без виділеного напрямку цього зв'язку (рис. 1,*а*); вершини тут з'єднуються за допомогою ребра;
- ✓ *орієнтовані* (орграфи), для яких важливим є ще й напрямок зв'язку вершин (рис. 1,*б*); вершини у таких графах з'єднуються дугами;
- ✓ *зважені*, в яких важливою інформацією є ще й величина, яка характеризує ступінь (величина, вага) зв'язку вершин (рис. 1,*в*).

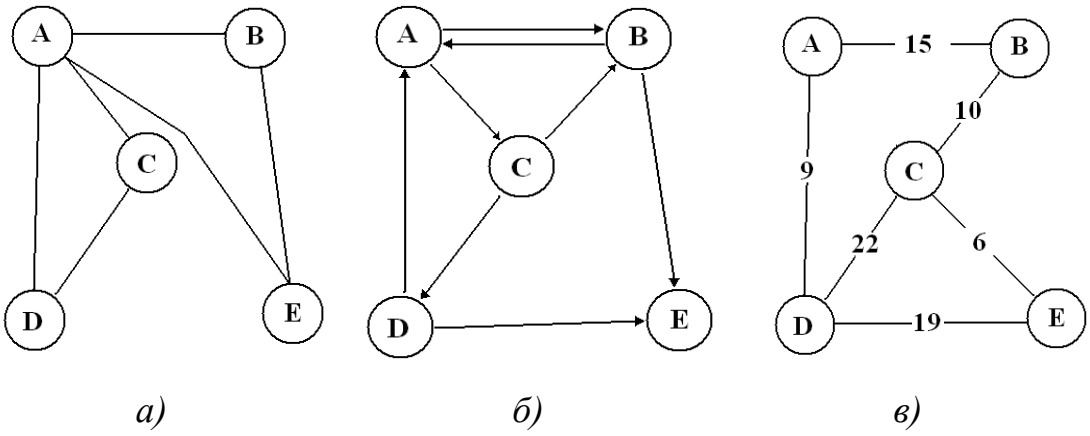


Рис. 1. Графічне представлення деяких графів:
а) звичайний; *б*) орієнтований; *в*) зважений.

Граф називається *орієнтованим* (або *орграфом*), якщо кожній дузі в ньому приписано певний напрямок. Дуги в орієнтованому графі зазвичай містять стрілки, що визначають орієнтацію дуги. Дуги, які можна проходити в

будь-якому з двох напрямків (двонаправлені), можуть містити дві стрілки-покажчики, але частіше зовсім не мають стрілок (тобто двонаправлені дуги аналогічні ребру графу).

На рис. 2 показано два можливих варіанти зображення розміченого орієнтованого графу. По дугах (*a, b*), (*c, d*) і (*e, d*), наприклад, можна рухатися лише в одному напрямку; по дугах (*a, d*), (*b, c*), (*d, e*) можна рухатися в будь-якому з двох напрямків.

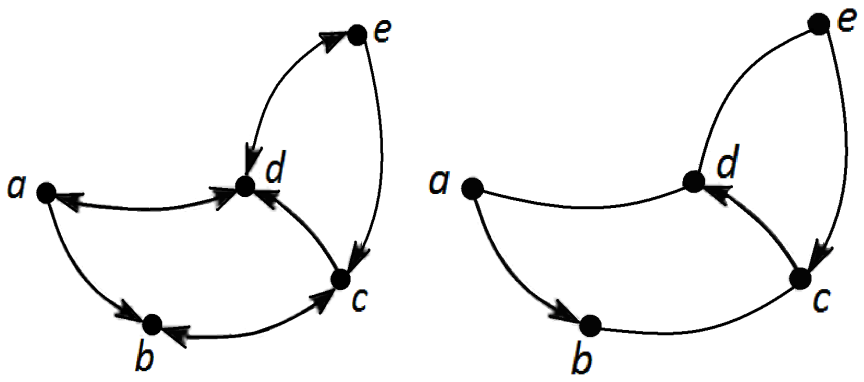


Рис. 2. Вигляд одного і того ж розміченого орієнтованого графу.

Для розрізнення вершин графу використовуються *мітки* (*дескриптори*).

У *розміченому* графі для кожної вершини задається один або декілька *дескрипторів* (міток), які дозволяють відрізнити одну вершину графу від іншої. На графі простору станів ці дескриптори ідентифікують стани в процесі виконання завдання. Якщо дескриптори (мітки) двох вершин не розрізняються, то ці вершини вважаються однаковими.

Дуга між двома вершинами визначається мітками цих вершин: напр., $e_1(v_1,v_2)$ або $e_2(v_2,v_1)$.

Дуги графу також можуть бути розміченими.

Мітка дуги використовується для задання іменованих відносин (як у семантичних мережах) або ваги дуги.

Шлях на графі можна представити і як систематизовану послідовність

відповідних вершин від початкової до цільової, і як послідовність ребер (дуг), що з'єднують сусідні вершини. Шлях тоді визначається списком дуг, систематизованим в порядку їх слідування. На рис. 2 список (*a, b, c, d*) являє шлях, що проходить через вершини *a, b, c, d* в зазначеному порядку.

4.3. Опис (представлення) графів.

Нехай задано граф $G = (V, E)$.

Про ребро $e = \{v_i, v_j\}$ цього графу говорять, що воно з'єднує вершини v_i та v_j.

Дві вершини, з'єднані ребром, називаються *суміжними*, якщо вони є кінцями одного ребра.

Про вершину та ребро, що входить у неї (або виходить з неї) говорять як про *інцидентні*. Отже інцидентними є ребро $e = \{v_i, v_j\}$ та вершина v_i і, аналогічно, ребро $e = \{v_i, v_j\}$ та вершина v_j.

Кількість вершин графу зазвичай позначається літерою n, а кількість ребер графу − літерою m: $|V| = n$, $|E| = m$. Ці числа характеризують основні числові характеристики графу: n – його *порядок*, а m – *розмір,* відповідно.

Кількість ребер, інцидентних до певної вершини x, називається *степенем* цієї вершини і позначається $\delta(x)$, або $\deg(x)$.

Задати граф означає задати множини його вершин і ребер, а також відношення інцидентності. Коли граф G – скінченний, для опису його вершин та ребер досить їх занумерувати. Нехай $v_1, v_2, \ldots, v_n$ – вершини графу G; $e_1, e_2, \ldots, e_m$ – його ребра.

Для опису графу як структури даних для опрацювання на комп'ютері зазвичай використовуються способи представлення у вигляді *матриць суміжності, матриць інцидентності* та *списків суміжності*.

Матриця суміжності простого графу зі скінченною кількістю вершин n – це квадратна матриця $\delta\|\delta_{ij}\|$ розміру n, стовпцям і рядкам якої відповіда-

ють вершини графу:

$$\delta_{ij} = \begin{cases} 1, \ якщо \left(v_i, v_j\right) \in E \\ 0, \ якщо \left(v_i, v_j\right) \notin E \end{cases}.$$

Для неорієнтованого графу значення δ_{ij} дорівнює кількості ребер, інцидентних i- та j-й вершинам, для орієнтованого – цей елемент матриці відповідає кількості ребер з початком в i-й вершині та кінцем у j-й вершині. Таким чином, матриця суміжності неорієнтованого графу є симетричною ($\delta_{ij} = \delta_{ji}$), а орієнтованого – необов'язково. Якщо вона все ж симетрична, то для кожного ребра орієнтованого графу існує ребро, яке з'єднує ті самі вершини, але йде у зворотному напрямку. Очевидно, орієнтований граф із симетричною матрицею суміжності відповідає неорієнтованому графу, який має ту саму матрицю суміжності.

Спосіб представлення графу через матрицю суміжності передбачає використання двовимірного масиву чисел, який для простих графів заповнюється тільки значеннями 0 (немає зв'язку) та 1 (є зв'язок) чи значеннями ваг – для зважених. Для звичайного графу матриця суміжності завжди є симетричною відносно головної діагоналі, а для орграфа найчастіше ця матриця не симетрична, що відображає односторонню спрямованість зв'язків.

Для розглянутих на рис. 1 прикладів матриці суміжності будуть такими (рис. 3):

Недоліки цього способу представлення:

- заздалегідь треба знати хоча б орієнтовне число вершин у графі;
- для графів з великим числом вершин матриця стає занадто великою (квадратично зростає і для графу порядку 100 складає 10000 чисел);
- при малому числі сполучних ребер матриця заповнена в основному нулями.

звичайний граф

	A	**B**	**C**	**D**	**E**
A	0	1	1	1	1
B	1	0	0	0	1
C	1	0	0	1	0
D	1	0	1	0	0
E	1	1	0	0	0

орієнтований граф

	A	**B**	**C**	**D**	**E**
A	0	1	1	0	0
B	1	0	0	0	1
C	0	1	0	1	1
D	1	0	0	0	1
E	0	0	0	0	0

зважений граф

	A	**B**	**C**	**D**	**E**
A	0	15	0	9	0
B	15	0	10	0	0
C	0	10	0	22	6
D	9	0	22	0	19
E	0	0	6	19	0

Рис. 3. Матриці суміжності графів, вказаних на рис. 1.

Цих недоліків багато в чому позбавлений другий спосіб, заснований на використанні *списків суміжних вершин*. Тут списки містять рівно стільки елементів, скільки є ребер в графі, і, крім того, вершини і ребра можуть додаватися динамічно.

Список суміжних вершин являє собою головний список всіх вершин і множину допоміжних списків, що містять перелік вершин, пов’язаних з даною. Для розглянутих вище звичайного і орієнтованого графів списки суміжності показані на рис. 4.

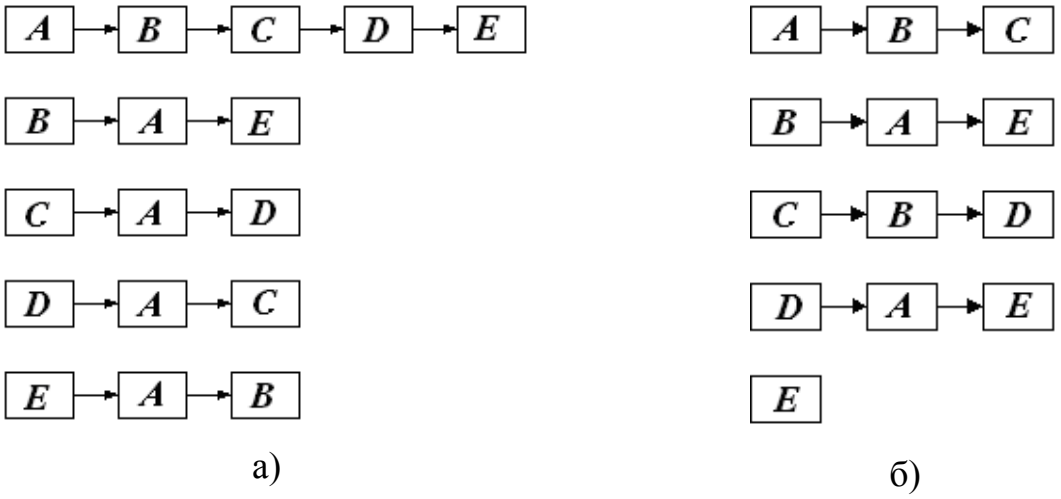


Рис. 4. Списки суміжності для звичайного (див. рис. 1,а) і орієнтованого (див. рис. 1,б) графів.

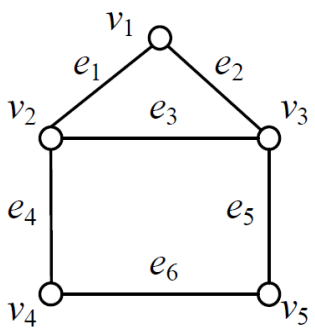
Відношення інцидентності можна означити матрицею $E = \|\varepsilon_{ij}\|$, яка має n рядків та m стовпців. Рядки відповідають вершинам графу, а стовпці – його ребрам. Якщо ребро e_j є інцидентним вершині v_i, то $\varepsilon_{ij} = 1$, в іншому випадку $\varepsilon_{ij} = 0$; у випадку орграфу ε_{ij} набуває значень:

$$\varepsilon_{ij} = \begin{cases} 1, \text{якщо вершина } v_i \text{ є кінцем дуги } e = \{v_i, v_j\}; \\ -1, \text{якщо вершина } v_i \text{ є початком дуги } e = \{v_i, v_j\}; \\ 0, \text{якщо вершина } v_i \text{ не є інцидентна до дуги } e = \{v_i, v_j\}; \end{cases}$$

У кожному рядку матриці інцидентності для неорієнтованого або орієнтованого графу тільки два елементи відмінні від 0 (або один, якщо ребро є петлею). Тому такий спосіб задання графу не досить економний.

Відношення інцидентності можна задати ще *списком ребер* графу. Кожний рядок цього списку відповідає ребру, в ньому записано номери вершин, інцидентних йому. Для неорієнтованого графу порядок цих вершин у рядку довільний, для орієнтованого – першим записується номер або інше найменування початку ребра, а другим – його кінця.

Вигляд матриці інцидентності та список ребер для звичайного графу показаний на рис. 5; аналогічне представлення для орієнтованого графу – на рис. 6.



а)

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	1	1	0	0	0	0
v_2	1	0	1	1	0	0
v_3	0	1	1	0	1	0
v_4	0	0	0	1	0	1
v_5	0	0	0	0	1	1

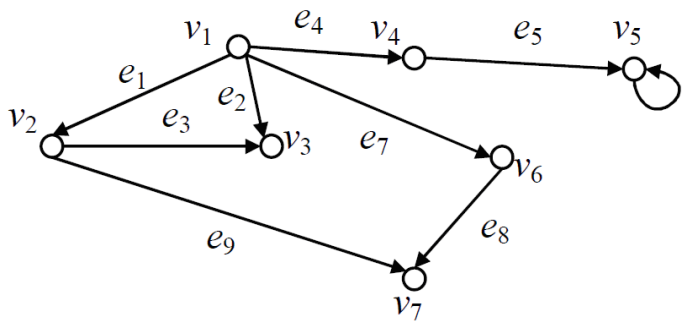
б)

Ребро	Вершини
e_1	v_1 ,v_2
e_2	v_1 ,v_3
e_3	v_1 ,v_3
e_4	v_2 ,v_4
e_5	v_3 ,v_5
e_6	v_4 ,v_5

в)

Рис. 5. Зображення звичайного графу (*а*) і відповідні матриця інцидентності (*б*) та список ребер (*в*).

За списком ребер графу можна легко визначити матрицю інцидентності цього графу. Справді, кожний рядок цього списку відповідає стовпцю матриці з тим самим номером. Для неорієнтованого графу в рядку списку записуються номери елементів стовпця матриці інцидентності, що дорівнюють 1, а для орієнтованого графу в цьому рядку першим зазначається номер елемента стовпця матриці, який дорівнює –1, другим – номер елемента, що дорівнює 1.



а)

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
v_1	-1	-1	0	-1	0	0	-1	0	0
v_2	1	0	-1	0	0	0	0	0	-1
v_3	0	1	1	0	0	0	0	0	0
v_4	0	0	0	1	-1	0	0	0	0
v_5	0	0	0	0	1	2	0	0	0
v_6	0	0	0	0	0	0	1	-1	0
v_7	0	0	0	0	0	0	0	1	1

б)

Дуга	Вершини
e_1	v_1 ,v_2
e_2	v_1 ,v_3
e_3	v_2 ,v_3
e_4	v_1 ,v_4
e_5	v_4 ,v_5
e_6	v_5 ,v_6
e_7	v_1 ,v_6
e_8	v_3 ,v_5
e_9	v_4 ,v_5

в)

Рис. 6. Орієнтований граф (*а*) та відповідні йому матриця інцидентності (*б*) і список ребер (*в*).

Слід зазначити, що за великої кількості вершин графу матриці суміжності та інцидентності стають громіздкими, що загалом обмежує їх викорис-

тання. Використання ж матриць і списків суміжності мають як свої переваги, так і недоліки.

До переваг використання списків суміжності можна віднести те, що, якщо в графі відносно небагато ребер, то списки суміжності часто виявляються кориснішими й ефективнішими. Друга причина використовувати списки суміжності при пошуку на графі – краща ефективність перебору вершин, суміжних із даною, оскільки в матриці суміжності для їх визначення потрібно перебрати весь її відповідний рядок. У той же час, списки суміжності як спосіб представлення програють матриці суміжності – для будь-яких двох вершин за допомогою матриці суміжності можна миттєво визначити, чи наявне між ними ребро, що не можна зробити у випадку зі списками суміжності. Доведеться брати одну з вершин і проходити по списку всіх вершин, суміжних із нею, в пошуках другої вершини.

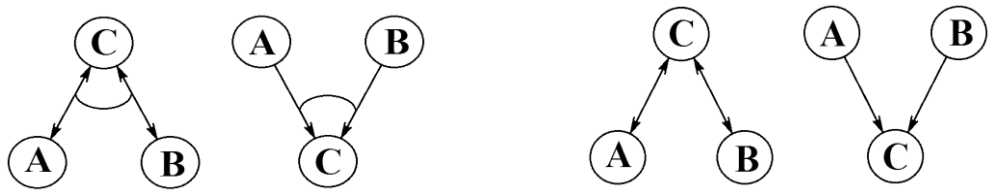
З огляду на це, списки суміжності не можна назвати однозначно «кращою» чи «гіршою» структурою представлення графу, ніж матриця суміжності, і для різних класів задач доцільно застосовувати різні способи подання графа – для деяких задач їх можна навіть комбінувати, використовуючи обидва способи відразу.

5. ПОШУК НА ГРАФІ.

5.1. І/АБО графи

І/АБО граф – орієнтований (направлений) граф, вершини якого відповідають певним завданням, а дуги – відносинам між завданнями.

І-вузли графу (рис. 7,а) – вузли, що подають імплікацію передумов, зв’язаних оператором *І*. Дуги, що ведуть до цих вузлів, з’єднуються кривою лінією. Крива, що з’єднує дуги, означає, що для доказу істинності вузла повинні бути істинними усі його передумови, з’єднані дугою.



а) б)

Рис. 7. *І* (а) та *АБО* (б) графи

АБО-вузли графу (рис. 7,б) – вузли, що подають імплікацію передумов, зв'язаних оператором *АБО*. Дуги, які ведуть до цих вузлів, не з'єднуються кривою, що вказує на те, що істина кожної з передумов є достатньою умовою для істинності висновку.

Схема пошуку на І/АБО-графі – спосіб руху по графу в напрямку, заданому стратегією пошуку. Розрізняють схеми сліпого (неінформованого) і спрямованого (інформованого, або евристичного) пошуків на графі, пов'язані з *перебором альтернативних вершин-підцілей* і *організацією повернення.*

Основна мета пошуку на І/АБО-графі – показати що вершина S_i вирішувана.

Вершина S_i є вирішуваною, якщо виконується одна з умов:

1) вершина S_i є заключною (термінальною);
2) наступні за S_i вершини є вершинами типу «АБО» і при цьому хоча б одну з них можна вирішити;
3) наступні за S_i вершини є вершинами типу «І» і при цьому кожну з них можна вирішити.

Вирішальним графом називається підграф, що складається з вирішуваних вершин з коренем у початковій вершині і є фактичним рішенням задачі.

У разі, якщо у вершини І/АБО-графа, що не є заключною, немає наступних за нею вершин, така вершина називається нерозв'язною.

Породження нових вершин (редукція завдання) виконується шляхом за-

стосування узагальненого оператора g з числа будь-яких операторів з множини допустимих. Застосування оператора g до опису завдання породжує всю структуру І/АБО-графа (графа редукції).

Довжина шляху у графі – це число ребер, які входять у шлях між початковою та кінцевою (цільовою) вершинами (у зважених графах – сума їх ваг).

5.2. Основні методи пошуку на графах.

Пошук на графі базується на послідовній побудові (переборі) вершин графу – до тих пір, поки не буде виявлено цільовий стан.

Для опису різних алгоритмів пошуку використовують такі терміни.

Вершину графу, що відповідає початковому стану, називають *початковою* вершиною, а вершину, що відповідає цільовому стану – *цільовою*.

Вершини, безпосередньо наступні за деякою вершиною, тобто отриманою в результаті застосування допустимих операторів до неї, називаються *дочірними*, а саму початкову вершину – *батьківською*.

Основною операцією, що виконується при пошуку в графі, є розкриття вершини, що означає породження (побудову) всіх її дочірніх вершин шляхом застосування до відповідного опису стану задачі всіх допустимих операторів.

Пошук на графі можна представити як процес поступового розкриття його вершин і перевірки властивостей породжуваних (дочірніх) вершин. В ході цього процесу повинні будуватися (і запам’ятовуватися) покажчики всіх вершин, що виникають – від дочірніх до батьківських. Саме ці покажчики дозволяють відновити шлях назад до початкової вершини після того, як буде побудована цільова вершина. Цей *шлях, узятий у зворотному напрямі, точніше, послідовність операторів, що відповідають дугам цього шляху, і буде шуканим рішенням задачі.*

Вершини і покажчики, побудовані в процесі пошуку, утворюють піддерево всього неявно визначеного при формалізації задачі графу-простору ста-

нів; це піддерево називається *деревом перебору*.

5.3. Стратегія пошуку

Під стратегією пошуку розуміють порядок, у якому відбувається розгортання станів; вона повинна бути виражена у вигляді функції, що певним чином вибирає з периферії наступний вузол, який підлягає розгортанню. Часто колекція вузлів реалізується у вигляді *черги* або *стека*.

Розрізняють стратегії прямого, зворотного та двонаправленого пошуків.

Стратегія прямого пошуку (forward chaining) відповідає руху від вихідних вершин графу до цільової вершини.

Стратегія зворотного пошуку (backward chaining) відповідає руху від цільової вершини до вихідних вершин. Якщо вершин-цілей мало, а вихідних багато, то зворотний пошук є більш природним і ефективним.

Стратегія двонаправленого пошуку (bi-directional chaining) поєднує прямий (рух від вихідних вершин до цільової) і зворотний пошук (рух від цільової вершини до вихідної) та намагається досягти деякого загального для обох пошуків стану, зупиняючись після того, як два процеси пошуку зустрінуться на середині графу. У стратегії двонаправленого пошуку передбачається перевірка в одному чи в обох процесах пошуку кожного вузла перед його розгортанням для визначення того, чи не знаходиться він на периферії іншого, зустрічного дерева пошуку; за позитивного результату такої перевірки вважається, що рішення знайдене.

6. АЛГОРИТМИ ПОШУКУ

Алгоритми пошуку в просторі станів можна *класифікувати* за:

- використанням евристичної інформації;
- порядком розкриття (перебору) вершин;
- повнотою перегляду простору станів;
- напрямом пошуку.

Відповідно до згаданої першою характеристики алгоритми діляться на два класи – *сліпі* та *евристичні* (спрямовані).

У сліпих алгоритмах пошуку місцезнаходження в просторі цільової вершини ніяк не впливає на порядок, в якому розкриваються (перебираються) вершини.

Сліпий метод має два види пошуку – *пошук углиб* і *пошук вшир*.

При пошуку в ширину на фіксованому рівні досліджуються всі альтернативи до кінця і лише після цього здійснюється перехід на наступний рівень.

При пошуку у глибину кожна альтернатива досліджується до кінця, без урахування решти шляхів. Метод поганий для т. з. «високих» дерев, оскільки легко можна прослизнути мимо потрібної гілки і витратити багато зусиль на дослідження «порожніх» альтернатив.

Метод пошуку у ширину може виявитися гіршим за метод пошуку у глибину, якщо в графі всі шляхи, що ведуть до цільової вершини, розташовані приблизно на одній і тій же глибині.

Евристичні алгоритми використовують апріорну, евристичну інформацію про загальний вид графу-простору про те, де в просторі станів розташована мета; тому для розкриття зазвичай вибирається перспективніша саме з цієї точки зору вершина.

Як сліпі, так і евристичні алгоритми пошуку можуть відрізнятися повнотою переглядання простору станів.

Повні алгоритми перебору при необхідності здійснюють повний перегляд графу і гарантують при цьому знаходження рішення, якщо таке існує.

Неповні алгоритми проглядають лише деяку частину графу (простору), і якщо вона не містить цільових вершин, то шукане рішення задачі цим алгоритмом знайдено не буде.

Відповідно до напряму пошуку алгоритми можна розділити на *прямі* (пошук від початкової вершини до цільової), *зворотні* (пошук від цільової вершини у напрямі до початкової) і *двонаправлені* (чергують прямий і зворотний пошук).

Найбільш вживаними (частково, через їх простоту) є алгоритми прямого пошуку. Зворотний пошук можливий у разі оборотності операторів завдання.

6.1. Метод повного перебору в ширину.

Вершини в цьому методі пошуку розкриваються у тому порядку, в якому вони будуються (заносяться в список нерозкритих вершин), тобто за чергою внесення їх в список.

Загалом, програма пошуку на графі передбачає використання в його процесі трьох списків вершин: списку нерозкритих (в якому містяться вершини, які будуть розкриватися в процесі пошуку), списку розкритих (які вже розкриті в процесі пошуку) та списку цільових (які потрібно досягнути в результаті пошуку) вершин. Перед початком пошуку всі ці списки є порожніми. Оскільки пошук на графі передбачає послідовний перебір (тобто розкриття) його вершин, програма пошуку в ширину за своєю суттю містить циклічне повторення операцій, які забезпечують послідовне розкриття кожної окремої вершини (з перевіркою при цьому низки обов’язкових умов) аж до досягнення мети пошуку.

Загальна блок-схема програми пошуку в ширину на графі показана на

рис. 8.

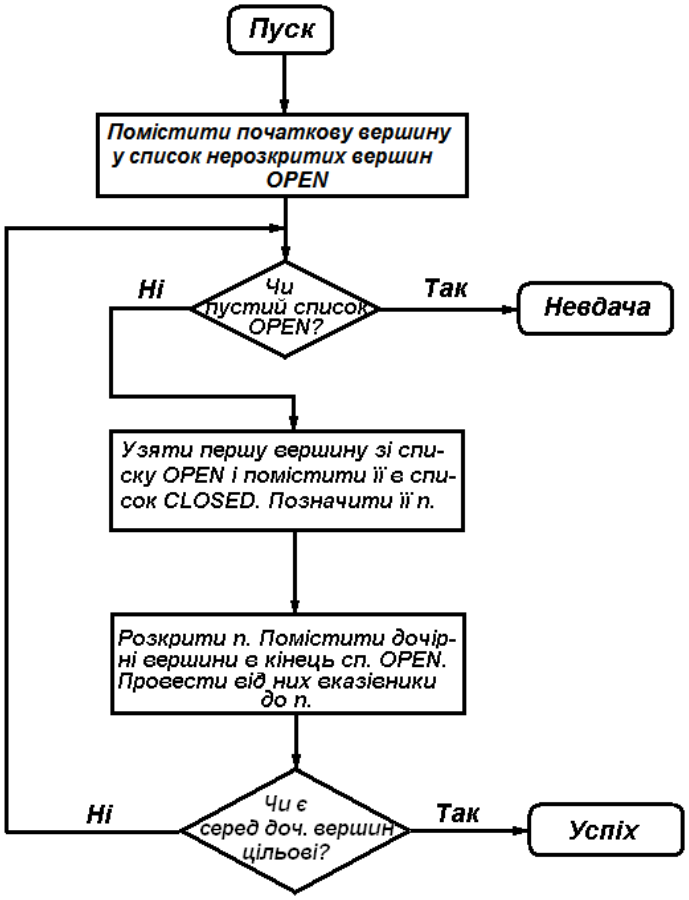


Рис. 8. Блок-схема програми пошуку на графі в ширину.

Послідовність розкриття вершин на графі при пошуку в ширину проілюструємо з використанням графу, показаного на рис. 9 (цільова вершина – 14).

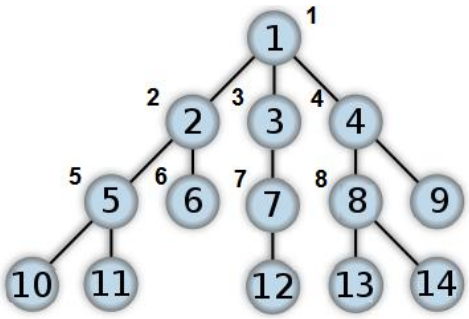


Рис. 9. Порядок розкриття вершин (цифри біля вершин) графу при пошуку у ширину (напрям обходу вершин – від меншої мітки до більшої).

Суть алгоритму пошуку в ширину полягає у виконанні таких дій:

1. На початку пошуку задана початкова вершина (нехай це вершина 1 на рис. 9) вноситься в список нерозкритих вершин, а цільова (нехай це вершина 14 на рис. 9) – в список цільових вершин.
2. Програма переглядає список нерозкритих вершин і вибирає з нього першу вершину (якщо список порожній, програма закінчує роботу).
3. Перша в списку нерозкритих вершин вершина (вершина 1 на рис. 9) розкривається до тих пір, поки її можна розкрити (тобто, доки є суміжні з нею вершини), застосовуючи один і той же (або й різні – дивлячись за умовою задачі) оператор обходу суміжних вершин. При цьому утворюються вершини першого рівня (вершини 2, 3, 4 на рис. 9).
4. Здійснюється перевірка, чи не є кожна з новоутворених вершин (з врахуванням напрямку обходу) цільовою або ж розкритою раніше в процесі пошуку (тобто на відсутність відповідних їм міток у списку розкритих та цільових вершин).
5. Якщо новоутворена вершина є в списку цільових вершин, то формується рішення про досягнення мети пошуку на основі відповідного оператора.
6. Якщо новоутворена вершина не є цільовою, але є в списку розкритих вершин, вона видаляється з подальшого розгляду і розглядається наступна з утворених (при розкритті поточної) вершина.
7. Кожна з новоутворених вершин після підтвердження, що вона не розкривалася до того і що вона не є цільовою, заноситься в кінець списку нерозкритих вершин за чергою, яку визначає заданий порядок обходу при розкритті вершин графа;
8. Розставляються вказівники, що ведуть від щойно занесеної у список нерозкритих вершини до тієї, в результаті розкриття якої вони виникли.
9. Після занесення всіх утворених (в результаті розкриття поточної) вершин у список нерозкритих вершин цикл розкриття поточної вершини

закінчується.

10. Програма перевіряє список нерозкритих вершин і переходить до наступної за чергою вершини в ньому; до неї застосовують дії, описані в п.2-п.8. При цьому в список заносяться почергово (з врахуванням напрямку обходу і відповідних описаних вище в п.3 і п.5 перевірок) нові вершини, отримані при її розкритті.
11. Якщо розкриття і цієї вершини не дозволило досягти цільової вершини, переходять до наступної зі списку нерозкритих і розкривають її, здійснюючи операції п.2-п.8, і т. д.
12. Описані дії продовжуються до тих пір, доки серед отриманих вершин не опиниться цільова (тоді мета пошуку досягнута) або ж вичерпується весь список нерозкритих вершин графу (рішення задачі у цьому випадку немає). Якщо мета досягнута, то, використовуючи занесені в пам'ять (див. п.8) вказівники, визначається шлях від початкової вершини до цільової.

Легко бачити, що пошук в ширину передбачає послідовний пошаровий перебір усіх вершин графу – і з цієї точки зору він є повним пошуком. Повний перебір в ширину гарантує знаходження цільової вершини якраз тому, що перебір у цьому випадку є дійсно повний – якщо шлях від початкової вершини до цільової на графі взагалі існує, то при послідовному переборі всіх вершин на графі він неодмінно буде знайдений.

Потрібно відмітити, що шляхів досягнення мети на розгалужених графах великих розмірів, взагалі кажучи, може бути багато. У цьому випадку є можливість вибрати серед них найліпший (найдешевший/найлегший/найшвидший, тощо). Але може бути і випадок, коли граф пошуку виявиться нескінченним – тоді цей алгоритм ніколи не закінчить роботу. Такий випадок може виникнути, зокрема, тоді, коли пошук ведеться не на дереві, а на графі, тобто коли якась дочірня вершина (або кілька вершин графу) може бути суміжною декільком батьківським вершинам, в результаті чого її розкриття мо-

же привести до зациклення процесу пошуку. Саме для уникнення такої ситуації і проводиться перевірка вершини, що утворюється в процесі розкриття поточної вершини, на присутність її в списку уже раніше розкритих вершин – у випадку наявності мітки цієї вершини в списку розкритих вона не вноситься в список нерозкритих вершин і, завдяки цьому, можливість «зациклення» процесу пошуку повторних розкриттів уже відкритих вершин усувається.

6.2. Метод перебору в глибину.

Суттєвою відмінністю пошуку в глибину на графі від пошуку в ширину є порядок розкриття внесених у список нерозкритих вершин – на відміну від пошуку на графі в ширину, при якому вершини розкриваються за чергою їх внесення у цей список, при переборі в глибину насамперед слід розкривати ті вершини, які були побудовані (тобто внесені у список нерозкритих вершин) останніми. Іншими словами, після завершення розкриття чергової вершини і після занесення відповідних нововідкритих вершин у список нерозкритих вершин для визначення чергової вершини, що розкриватиметься, використовується стек (а не черга, як у випадку пошуку в ширину). Такий метод перебору призводить до того, що (в загальному випадку при т. з «лівому» чи «правому» порядку обходу суміжних вершин) процес розкриття вершин графу завжди буде йти по його крайній лівій (або ж правій – згідно із заданим напрямком обходу) гілці ще не розкритих до цього вершин аж до досягнення їх максимальної глибини, і лише після її досягнення (у випадку, якщо не досягнуто цільової вершини) алгоритм почне відслідковувати іншу альтернативу, тобто іншу, наступну за нею, крайню ще не розкриту гілку графу.

Тому, взагалі кажучи, алгоритм перебору на графі в глибину відрізняється від такого для пошуку в ширину лише зміною вибору черговості вершин, які підлягають наступному розкриттю.

Опис алгоритму аналогічний приведеному у попередньому параграфі і

відрізняється лише п.2, який має бути таким:

2. Розкривається вершина, що є останньою в списку нерозкритих вершин.

Черговість розкриття вершин графу при пошуку в глибину (початкова вершина – 1; цільова – 14; порядок обходу суміжних вершин – від меншої до більшої мітки) на розглянутому вище графі проілюстрована на рис. 10.

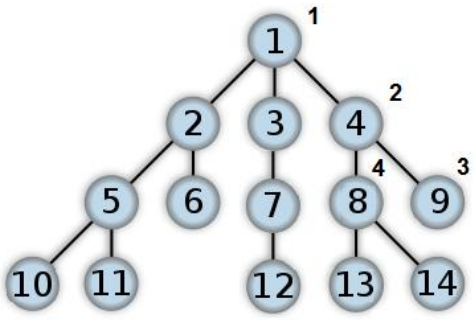


Рис. 10. Порядок розкриття вершин графу (цифри біля вершин графа) при пошуку у глибину (напрям обходу вершин – від меншої мітки до більшої).

У загальному випадку, при пошуку на графах великих порядків і розмірів, розглянутий вище алгоритм пошуку в глибину може привести до суттєвого зниження його ефективності – відомо, що при зростанні глибини шарів графу кількість операцій суттєво зростає з кожним шаром. Тому, при необмеженій глибині пошуку, на таких графах може суттєво зрости кількість непродуктивно розкритих вершин. Щоб якось обмежити перебір, можна задати його граничну глибину. Тоді, як тільки в процесі породження вершин досягається задана гранична глибина, тобто розкривається вершина найбільшої зазначеної глибини, що не перевищує цієї межі, пошук на цій гілці закінчується і далі іде вже по іншій гілці.

Для здійснення такої можливості вводиться поняття глибини вершини на дереві перебору. Глибина кореня дерева дорівнює нулю, а глибина будь-якої наступної вершини дорівнює глибині вершини, що безпосередньо їй пе-

редує, плюс одиниця. Найбільшу глибину завжди буде мати та вершина, яка повинна бути в цей момент розкрита. Якщо шлях, який утворюється при пошуку, не приведе до досягнення мети пошуку (тобто при заданій глибині розкриття цільова вершина не була досягнута), необхідно повернутися в попередню розкриту вершину і спробувати ще раз застосувати до неї операцію розкривання, збільшивши на одиницю глибину розкривання. І так до тих пір, поки не буде отримана цільова вершина.

Повернення здійснюється за допомогою встановлених в процесі розкриття вершин покажчиків.

Блок-схема програми пошуку в глибину з граничною заданою глибиною на графі показана на рис. 11.

Обмежений пошук в глибину найбільш корисний, якщо відомо, що шукана цільова вершина знаходиться в межах певної глибини, або коли можуть бути встановлені обмеження в часі чи простір станів надзвичайно великий (як, наприклад, у шахових задачах).

6.3. Хвильовий алгоритм

Хвильовий алгоритм – алгоритм, що дозволяє знайти мінімальний шлях в графі з ребрами одиничної довжини.

Заснований на алгоритмі пошуку в ширину, хвильовий алгоритм застосовується для знаходження найкоротшого шляху на графі; загалом дозволяє знаходити найкоротший шлях від будь-якої заданої до будь-якої іншої вершини графу, якщо він є.

Зазвичай граф, на якому проводиться пошук, представляється графічно у вигляді «лабіринту» – двовимірної картини (або ж матриці), що складається з «прохідних» (зазвичай позначаються як -1) і «непрохідних» (позначаються як 0) комірок. За своєю суттю, лабіринт, в якому здійснюється пошук, являє собою графічне представлення (візуалізацію) матриці суміжності заданого

одиничного графу, в якому наявні зв'язки між сусідніми вершинами позначаються нулями, а їх відсутність – -1. Зазначаються вихідні умови пошуку – початкова і цільова комірки; задаються також умови порядку перегляду суміжних вершин (оператори переходу – вверх-вниз, вправо-вліво; по діагоналях або ж їх комбінація).

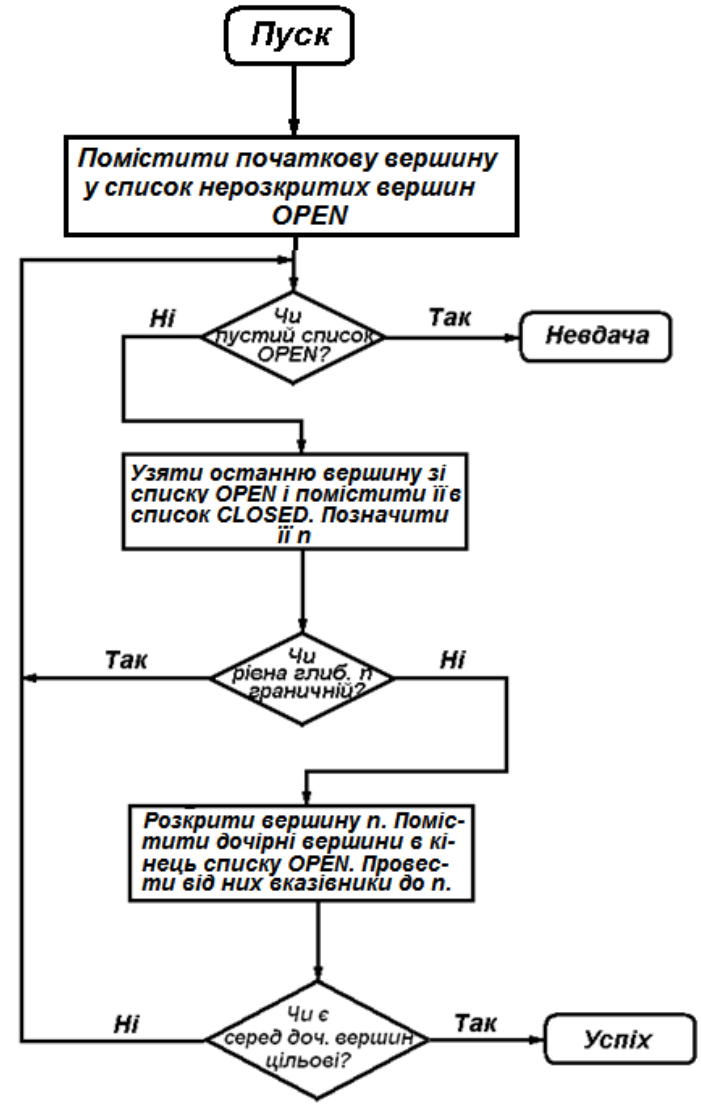


Рис. 11. Блок-схема програми пошуку на графі в глибину.

Оскільки перебір вершин графу здійснюється за принципами пошуку в ширину і полягає в послідовному виконанні циклу почергового розкриття вершин, то у випадку однонаправленого пошуку від початкової точки пошуку в усіх напрямках по лабіринту поширюється хвиля, причому характер її поши-

рення визначається оператором переходу (рис. 12); кожна досягнута хвилею в процесі її руху комірка позначається як «пройдена» (їй присвоюється відповідний номер; номер наступної «пройденої» комірки на одиницю більший, ніж в попередньої).

	■								■		
			■			■	■				
10			■	10	■	10				■	
9	10	■	10	9	8	9	10				
8	9	■	9	8	7	8	9	10	■	■	
7	8	9	8	7	6	7	8	9	10		
6	7	8	7	6	5	6	7	■	■	10	
5	■	■	■	■	4	5	6	7	8	9	10
4	3	2	1	2	3	4	5	6	■	■	
3	2	1	0	1	2	3	4	5	■	■	10
4	3	2	1	2	3	4	5	6	7	8	9

а)

	■								■		
			■			■	■				
			■		■					■	
		■	6		6		6		6		
		■		5		5		5	■	■	
	6		4		4		4		6		
		5		3		3		■	■		
	■	■	■	■	2		4		6		
3		1		1		3		5	■	■	
	2		0		2		4		■	■	
3		1		1		3		5			

б)

	■								■		
			■			■	■				
			■		■					■	
		■	6	6	6	6	6	6	6		
6	6	■	5	5	5	5	5	5	6		
5	5	5	4	4	4	4	4	5	6		
4	4	5	4	3	3	3	4	■	■		
3	■	■	■	■	2	3	4	5	6		
3	2	1	1	1	2	3	4	5	■	■	
3	2	1	0	1	2	3	4	5	■	■	
3	2	1	1	1	2	3	4	5	6		

в)

Рис. 12. Вигляд шляху, знайденого в одному і тому ж лабіринті, від вершини (11, 4) у вершину (5, 4), за допомоги хвильового алгоритму з використанням таких операторів переходу:

(а) – вверх-вниз, вправо-вліво;

(б) – по діагоналях;

(в) – вверх-вниз, вправо-вліво та по діагоналях.

Хвиля не може проходити через комірки, що помічені як «пройдені» або «непрохідні», і поширюється, поки не досягне цільової вершини (вказаної комірки лабіринту) або ж поки в ньому не залишиться непройдених комірок. Якщо хвиля пройшла всі доступні комірки, але так і не досягла цільової точки, значить, шлях від початкової до цільової вершини графу за даних умов пошуку прокласти неможливо.

Після досягнення хвилею фінішу прокладається шлях в зворотному напрямі (від фінішу до старту, за спадними номерами пройдених комірок, з врахуванням оператора переходу), який і є рішенням задачі.

Як видно з рис. 12, використання різних операторів переходу при решті однакових умов пошуку зумовлює різну його ефективність – найбільш ефективним (для визначення ефективності пошуку на графах часто використовують відношення кількості вершин вирішального графу до загальної кількості розкритих в процесі пошуку вершин) може виявитися пошук з використанням діагонального оператора; правда, з іншого боку, недоліком цього методу пошуку може бути ситуація, коли цільова вершина потрапить у зону, через яку «діагональ» розкритих вершин не поширюється (пусті клітинки на рис. 12,б), і пошук в такому випадку не досягне мети. Найменш оптимальним буде пошук з використанням оператора переходу вправо-вліво-вверх-вниз – але він завжди забезпечує пошук шляху, якщо він є.

Пошук може здійснюватися як в одному напрямку (від початкової до цільової вершини – однонапрвлений пошук), так і двох напрямках (від початкової до цільової вершини і навпаки одночасно – двонаправлений пошук).

Варіант опису однонаправленого пошуку за хвильовим алгоритмом:

1. Візуалізуємо граф пошуку – використовуючи матрицю суміжності заданого графу, будуємо “лабіринт”, в якому прохідні точки відмічаємо нулем, а непрохідні – -1.
2. Відмічаємо початкову та цільову точки пошуку. Початкову точку позначаємо 1 і заносимо її в список нерозкритих вершин; оскільки перед

початком пошуку цей список пустий, початкова вершина стає в ньому єдиною і першою.

3. Вибираємо оператор переходу.
4. Розкриваємо, використовуючи обраний оператор переходу, першу вершину з черги в списку нерозкритих вершин; всі вершини, суміжні з нею, помічаємо цифрою на одиницю більшою, ніж та, якою помічена вершина, що зараз розкривається, і перевіряємо кожну з них, чи вона не є цільовою: якщо «так» – пошук закінчено; якщо «ні» – вершина перевіряється на відсутність у списку вже розкритих та вноситься у список нерозкритих вершин; якщо ця вершина є у списку розкритих вершин, вона не вноситься в список нерозкритих. (Потрібно відмітити, що при розкритті вершини за такою схемою в список нерозкритих вершин під однаковим номером може потрапити одночасно від нуля (у випадку тупикової вершини) до восьми вершин – залежно від застосованого оператора переходу та характеристик графу. Відповідно, розкриття всіх вершин, які мають однакову присвоєну при розкритті їх батьківської вершини числову мітку, для забезпечення симетричності поширення хвилі повинне проводитися в одному циклі.)
5. Мітку розкритої вершини видаляємо зі списку нерозкритих і поміщаємо в список розкритих.
6. Процес розкриття вершин (а, отже, і пошук) закінчується, коли досягається цільова вершина або ж список нерозкритих вершин графу вичерпується.
7. У випадку досягнення мети пошуку для знаходження найкоротшого шляху від початкової вершини до цільової здійснюємо пошук у графі в глибину від кінцевої точки до початкової (з врахуванням відповідного оператора переходу).

У випадку двонаправленого пошуку він організовується таким чином, що в одному циклі почергово розкриваються вершини з однаковою цифро-

вою міткою спочатку з ланки пошуку, який іде від початкової вершини, а потім – з цільової. На відміну від однонаправленого пошуку, тут для забезпечення початку реалізації зустрічного пошуку в список нерозкритих вершин вносяться одночасно початкова і цільова вершини під номером 1. Кожен цикл розкриття вершини в цьому випадку передбачає перевірку умови, чи не є нова суміжна вершина наявною в списку вже розкритих вершин альтернативної вітки пошуку (на відміну від однонаправленого, де перевіряється, чи не є вона цільовою); якщо результат такої перевірки позитивний – пошук закінчується, а найкоротший шлях у цьому випадку визначається як сума шляхів від комірки, при розкритті якої хвилі зустрілися, до початкової та цільової. Якщо ж суміжна до тієї, що розкривається, вершина не належить до списку вже розкритих в альтернативній вітці пошуку, вона заноситься в список невідкритих вершин і пошук продовжується або до знаходження точки перетину з альтернативною хвилею, або ж до перебору усіх доступних вершин графу – у цьому випадку рішення поставленої задачі немає.

Можна сказати, що істотною відмінністю цих двох варіантів хвильового пошуку є те, що в кожному циклі розкриття поточної вершини у випадку однонаправленого пошуку здійснюється перевірка кожної нової вершини на предмет того, чи не є вона цільовою, а при двоаправленому – чи не відкрита вже вона в альтернативній гілці пошуку.

Перевага двонаправленого пошуку – зменшення часу знаходження найкоротшого шляху за рахунок вдвічі меншої кількості послідовних циклів розкриття вершин, які необхідно здійснити при досягненні цільової вершини при однакових решті умов пошуку.

6.4. Алгоритм Дейкстри.

Часто (зокрема, в логістиці, а також для вирішення багатьох ігрових задач) зустрічається задача, суть якої зводиться до визначення найкоротшого

шляху між певними вершинами (нехай це вершини *s* і *t*) на заданому орієнтованому зваженому графі. Чи не найбільш поширений алгоритм, який часто використовується для розв'язання такої задачі (і фактично за своєю суттю є модифікованим алгоритмом пошуку в ширину), був запропонований Дейкстрою. Надалі він знайшов застосування і для вирішення деяких технічних завдань – зокрема, модифікований алгоритм Дейкстри використовується при здійсненні «розводки» плат в електро- і радіотехніці.

У загальному випадку запропонований метод ґрунтується на послідовному приписуванні вершинам графу тимчасових числових позначок (числових міток), значення яких відповідає сумі довжин ребер (дуг) від початкової вершини до кожної з суміжних тій вершини, що розкривається в поточному акті розкриття чергової вершини графу. Таким чином, позначка вершини набуває значення довжини шляху за певним маршрутом на графі від заданої початкової вершини *s* до поточної. Значення цих позначок поступово зменшуються за допомогою ітераційної процедури, суть якої – розгляд усіх можливих маршрутів руху по графу і вибір з них мінімальних віддалей (а, відповідно, найкоротших шляхів) від початкової вершини до кожної з суміжних з поточною, яка розкривається. Таким чином, на кожному кроці ітерації знаходяться найменші з усіх можливих тимчасових позначок для кожної вершини, суміжної з тією, що розкривається на цьому кроці – саме ці позначки й дають точну довжину найкоротшого шляху від вершини *s* до кожної з них. Якщо при цьому ще і запам'ятовувати шлях, який дає найкоротшу віддаль між початковою і даною вершинами, то на кожному кроці ітерації будемо мати найкоротший шлях з початкової в кожну досягнуту вершину графу і, відповідно, значення відповідної найкоротшої віддалі між ними.

Роботу алгоритму розглянемо на простому прикладі.

Нехай заданий орієнтований зважений граф, зображений на рис. 13, на якому потрібно знайти найкоротші шляхи з вершини 1 до решти вершин.

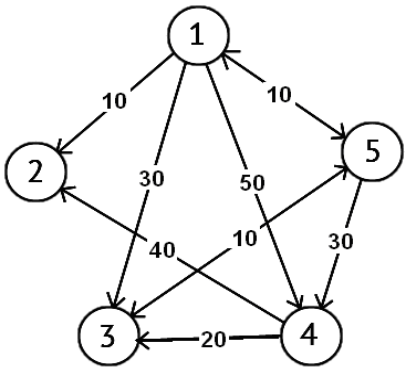
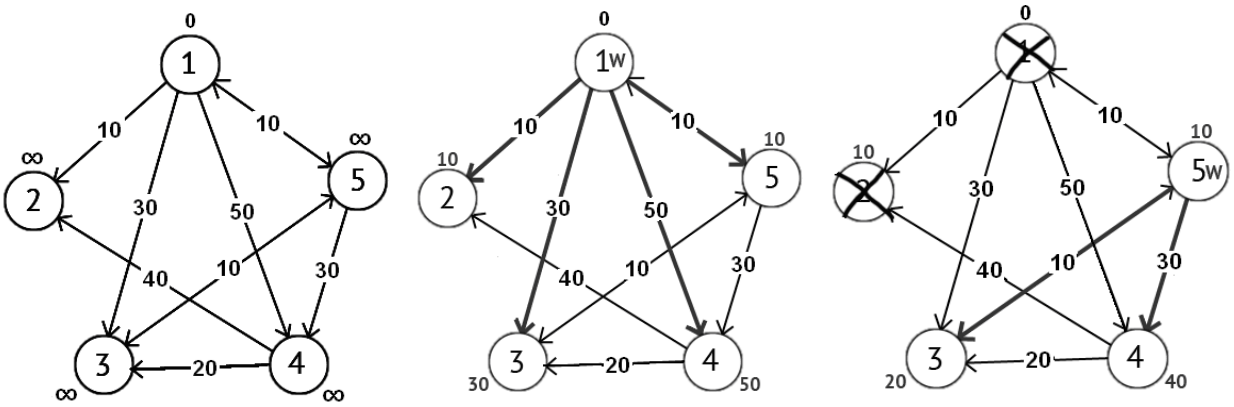


Рис. 13. Орієнтований зважений граф.

На першому кроці роботи алгоритму кожній вершині присвоюється певна числова мітка. У нашому випадку 1-й вершині, оскільки ця вершина – початкова, присвоюється числова мітка, рівна 0; решті ж вершин присвоюємо мітки, рівні нескінченності (рис. 14,*а*; числові мітки – цифри біля вершин). Поміщаємо першу вершину у список нерозкритих вершин.



а) *б)* *в)*

Рис. 14. Кроки алгоритму Дейкстри (жирними лініями позначені дуги, які зв’язують чергову W-вершину з суміжними).

Далі розглянемо (з врахуванням напрямків дуг) всі вершини на графі, суміжні до першої в списку нерозкритих вершин (в нашому випадку – початкової; позначатимемо надалі кожну вершину, що розкривається на поточному

кроці ітерації, через *W*). Кожній з цих вершин (вершини 2, 3, 4 і 5) призначимо мітку, яка дорівнює сумі мітки вершини, що розкривається, та довжини шляху від неї (ваги відповідного ребра графу) до цієї вершини, і порівнюємо його з присвоєним раніше, на попередньому кроці. Якщо це значення менше, то мітці вершини присвоюється нове, менше за попереднє, значення (рис. 14,*б*). Записуємо в пам'ять мітки вершин, які складають відповідні шляхи. Відмічаємо розглянуту вершину *W* як розкриту; всі суміжні до неї вершини поміщаємо список нерозкритих вершин в порядку зростання значення присвоєних їм міток.

Вибираємо першу зі списку ще не розкритих вершин (нею буде та вершина, що отримала на цьому етапі мінімальне значення мітки); вона і буде наступною вершиною, що розкриватиметься (тобто наступною вершиною *W*). В розглядуваному прикладі це вершини 2 або 5. Якщо є кілька вершин з однаковими мітками, то не має значення, яку з них ми виберемо як наступну *W*-вершину.

З вершини 2 (на нашому графі) немає ні одного вихідного шляху, тому ми відразу ж відзначаємо цю вершину як розкриту і переходимо до наступної зі списку нерозкритих вершин – вона і буде наступною *W*-вершиною. Це вершина 5. Розглянемо всі вершини, суміжні до неї, але які ще не помічені як розкриті. Знаходимо суму мітки вершини 5 і ваги ребра з неї в поточну суміжну вершину, і, якщо ця сума буде меншою від мітки цієї вершини, присвоєної їй на попередніх кроках ітерації, то замінюємо значення її мітки на отриману суму. При цьому відмічаємо в пам'яті отриманий таким чином найкоротший шлях з початкової вершини до даної. Якщо ж ця сума не буде меншою, то залишаємо без змін значення попередньої мітки вершини (рис. 14,*в*) і, відповідно, отриманий раніше найкоротший шлях до неї.

Виходячи з рис. 14,*в*, бачимо, що мітки вершин 3 і 4 в результаті останньої ітерації набули менших значень, ніж в попередніх, тобто був знайдений коротший маршрут в ці вершини з початкової вершини графу.

Далі відзначаємо 5-ю вершину як відвідану і вибираємо наступну вершину, яка має мінімальну позначку. Повторюємо всі перераховані вище дії до тих пір, поки є ще невідвідані вершини графу. В результаті отримаємо мінімальні віддалі від вершини 1 до всіх вершин нашого графу (рис. 15).

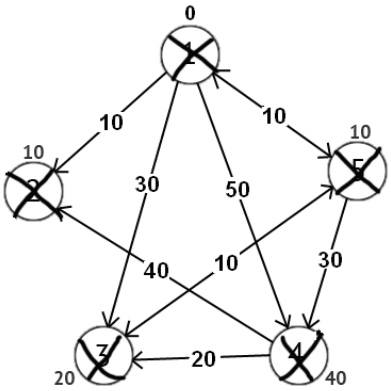


Рис. 15. Граф з отриманими мінімальними віддалями (число біля вершини) кожної вершин від 1-ї вершини.

Якщо в умовах пошуку за алгоритмом Дейкстри задати початкову та цільову вершини, то в кожному ітераційному циклі необхідно перед розкриттям вершини перевірити, чи не є вона цільовою. Якщо «так» – тоді програма сформулює висновок про закінчення пошуку і видасть найкоротший шлях на графі від початкової вершини до цільової та відповідну віддаль. Якщо ж «ні» – тоді здійснюється перехід до наступного циклу ітерації.

Робота алгоритму закінчується або у випадку досягнення цільової вершини, або за умови спустошення списку нерозкритих вершин – в останньому випадку видається повідомлення про відсутність шляху між вказаними вершинами графу.

Отже, загалом роботу алгоритму Дейкстри можна представити як циклічне повторення процедур послідовного розкриття вершин графу, починаючи з початкової. На кожному кроці такої процедури розкриття поточної вершини суміжним до неї вершинам присвоюється числові мітки, які рівні най-

меншій віддалі від початкової вершини до кожної з них. В пам'ять також записуються відповідні найкоротші шляхи. В результаті роботи алгоритму після його закінчення будуть визначені найкоротші шляхи від початкової до всіх інших вершин графу.

Програма пошуку (за умови задання початкової та цільової вершин) може виводити знайдений найкоротший шлях на графі від початкової до цільової вершини, а також значення відповідної віддалі.

7. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СТВОРЕННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Під інструментальними засобами розробки систем штучного інтелекту розуміються мови і середовища програмування, що використовується розробниками (інженерами знань та програмістами) для їх побудови.

При створенні систем ШІ застосовуються як імперативні (процедурні), так і декларативні (функціональні та логічні) мови програмування.

Імперативні (процедурні) *мови* дозволяють створювати послідовності команд, що однаково виконуються в порядку їх розташування при кожному прогоні (*Паскаль, Сі, Форт, Бейсік*, *Ada* та ін.). Програма, написана на процедурній мові, містить алгоритм розв'язання задачі в термінах дій (процедур), які необхідно послідовно виконати. Таким чином, робота програми являє собою послідовний виклик процедур. Цей клас мов є основою всіх мов програмування.

Більш високий рівень програмування надають *об'єктно-орієнтовані мови*, які звільняють програмістів від виконання рутинних процедур, пов'язаних з розподілом пам'яті, процедурами доступу та маніпулюванням даними. Програма, написана на об'єктно-орієнтованій мові, представляє собою набір класів, екземпляри яких взаємодіють між собою за допомогою

надсилання повідомлень. Кожен клас характеризується інформаційною (набір атрибутів) і поведінковою (набір подій і методів) складовими. Робота програми включає послідовний обмін повідомленнями (виклик методів) між об'єктами (екземплярами класів). До об'єктно-орієнтованих мов відносяться *Object Pascal, Visual Basic, C++, C#, Python, Java, Ruby* та ін. Будь-яка об'єктно-орієнтована мова також надає можливість і процедурного програмування.

Декларативні мови (їх ще називають мовами штучного інтелекту) використовуються для написання специфікацій деякої наочної (предметної) області, яка характеризує множину об'єктів цієї області та задані відносини між ними. До особливостей мов цього класу слід віднести насамперед наявність в них вбудованих засобів для організації інтелектуальних процедур, у т. ч. уніфікації, альтернативного пошуку, повернення (backtracking), обробки символьної інформації, можливості реалізації функцій бази даних, тощо. Універсальність цих мов менша, ніж у традиційних мов програмування, але це компенсується багатими можливостями щодо роботи з символьними та логічними даними, що вкрай важливо для задач штучного інтелекту. На основі мов штучного інтелекту створюються спеціалізовані комп'ютери (наприклад, Ліспмашини).

Декларативні мови в свою чергу діляться на *логічні* й *функціональні*.

Логічні мови програмування ґрунтуються на теорії математичної логіки та виведенні нових фактів із заданих за певними логічними правилами. Програми, написані на цих мовах, на відміну від написаних на мовах процедурного програмування, не містять в собі явних алгоритмів – задається опис умов задачі і логічних співвідношень, за якими система програмування будує дерево виведення і знаходить її рішення. Такі програми не відрізняються високою швидкодією, оскільки процес їх виконання зводиться до побудови прямих і зворотних ланцюжків міркувань різноманітними методами пошуку, але зате дуже компактні в написанні.

Найвідомішою мовою логічного програмування є *Prolog*, що реалізує модифіковану логіку предикатів першого порядку. Для пояснення сенсу програми тут застосовуються дві семантичні моделі: декларативна і процедурна. Програма на цій мові будується з послідовності фактів і правил, потім формулюється твердження, яке *Prolog* намагається довести за допомогою правил. Мова сама шукає рішення методами пошуку і зіставлення, які в ній закладені. Відомі також пізніші мови, утворені на основі *Prolog*: *BinProlog, AMZI-Prolog, Arity Prolog, CProlog, Micro Prolog, МПролог, Prolog-2, Quintus Prolog, SICTUS Prolog, Silogic Knowledge Workbench, Strawberry Prolog, SWI-Prolog, Turbo Prolog (PDC Prolog, Visual Prolog), UNSW Prolog та ін.*

Функціональні мови програмування ґрунтуються на використання списків і функцій; змінні тут можуть бути відсутніми взагалі. Чи не найпоширенішою з таких мов є *LISP* (від *List Processing;* в літературі використовуються також назви *Lisp, Ліcп*), яка заснована на функціональному численні – т. з. лямбда-численні, або ж численні Черча – і була створена в 50-х рр.. минулого сторіччя спеціально для вирішення задач в царині штучного інтелекту. Широко поширені версії мови *LISP* (наприклад, *Interlisp*) мають розвинені редактори та засоби відладки.

Від традиційних мов програмування *LISP* відрізняється насамперед тим, що подання програм і оброблюваних даних в ній ідентичні за формою і є обліковими структурами. Програма працює зі списками, що дозволяє не замислюватися про управління пам’яттю – резервування і звільнення осередків пам’яті відбувається динамічно. До переваг мови можна віднести широкі можливості для синтаксичного аналізу виразів, а також можливість використання єдиного спискового контексту для програм і даних. Це надає нові можливості застосування мови, серед яких – обробка інших програм. Важливою є універсальність та масштабованість синтаксису мови, яка дозволяє писати дуже компактні програми. Так, написане на *LISP* ядро *LISP* займає менше 200 рядків, а інтерпретатор *Prolog* – трохи більше 50 рядків. До недоліків можна

віднести використання синтаксисом *LISP* величезної кількості дужок.

На базі *LISP* розроблені могутні інструментальні оболонки експертних систем – *KEE, LOOPS, ART* й ін.

Загалом, за своїм призначенням та функціональними можливостями, інструментальні засоби, які застосовуються при проектуванні систем штучного інтелекту, можна розділити на такі досить великі категорії:

1. *Мови програмування високого рівня*, які звільняють розробника від необхідності заглиблюватися в деталі реалізації системи – способи ефективного розподілу пам'яті, низькорівневі процедури доступу та маніпулювання даними.
2. *Оболонки експертних систем* – створюються, як правило, на основі якої-небудь експертної системи, яка досить добре зарекомендувала себе на практиці, або ж являють собою спеціально створені середовища з засобами створення бази знань та вбудованим механізмом логічного виведення; до чи не найвідоміших і найбільш поширених засобів цього виду, призначених для створення експертних систем продукційного типу, відноситься середовище CLIPS.
3. *Об'єктно-орієнтовані оболонки*, на основі яких можна створювати конкретні застосування.
4. *Середовища програмування, що підтримують декілька парадигм* – програмні модулі, що дозволяють користувачеві комбінувати в процесі розробки різні стилі програмування ;
5. *Додаткові модулі* – призначені для виконання специфічних завдань в рамках обраної архітектури системи рішення проблем.

Тому розробники додатків в галузі штучного інтелекту мають можливість використовувати в своїй роботі будь-який з доступних їм інструментальних засобів, які дозволяють досягти поставленої мети і найкраще відповідають умовам задачі, яка передбачається розв'язуватися з їх допомогою.

8. ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ.

8.1. Завдання до роботи 1.

Тема роботи: Сліпий пошук на графах. Особливості реалізації пошуку в глибину у графах.

Вихідні дані:

Задано:

- звичайний розгалужений граф порядку не менше 30 та розміром не менше 30-40 (не менше 5-и “гілок” у випадку дерева);
- початкова і цільова вершини пошуку;
- порядок обходу вершин графу при здійсненні пошуку.

Завдання:

Створити програму виконання пошуку в глибину на графі, використовуючи одну з мов програмування високого рівня, в якій реалізувати:

- візуальне представлення створеного графу;
- знаходження найкоротшого шляху від вказаної початкової до цільової вершини, використовуючи алгоритм пошуку в глибину;
- виведення результатів пошуку.

У програмі *передбачити*:

- створення графічного інтерфейсу керування роботою програми;
- реалізацію покрокової візуалізації процесу пошуку;
- можливість змінювати розмірність, порядок та вид графу, у т. ч. й заміну частини ребер на дуги і навпаки;
- можливість змінювати напрям обходу вершин графу;
- представлення результатів пошуку (знайдений шлях на графі при заданих вершинах й кількість розкритих вершин) та виведення їх в окремому вікні (вікнах) інтерфейсу;

Дослідити особливості роботи створеної програми при:

- різних напрямках обходу суміжних вершин при розкритті вершини;
- зміні напрямку пошуку (у т. ч. при дзеркальній заміні початкової вершини у і мети пошуку),
- вплив напряму обходу на результат пошуку при решті незмінних умов;
- вплив зміни розмірів та порядку графу, а також виду графу (звичайного та орграфу) на результати пошуку при однакових початковій та цільовій вершинах,

а також:

- вплив на результати пошуку виду графа (дерева та графу того ж розміру (тобто за наявності ребер, які зв'язують різні "гілки" дерева);
- заміни деяких ребер графа на дуги і навпаки.

8.2. Завдання до роботи 2.

Тема роботи: Сліпий пошук на графах. Особливості реалізації пошуку в ширину у графах.

Вихідні дані:

Задано:

- звичайний розгалужений граф (не менше 5-и гілок) порядку не менше 30 та розміром не менше 30-40;
- початкова і цільова вершини графу;
- порядок обходу вершин графу при пошуку.

Завдання:

Створити програму виконання пошуку в ширину на одній з мов програмування високого рівня, в якій:

- візуально представити заданий граф;
- реалізувати знаходження шляху між заданими вершинами графу, ви-

користовуючи алгоритм пошуку в ширину;

- дослідити вплив зміни порядку і розмірів графу, а також заміни звичайного графу на орграф на результати пошуку;
- відмітити переваги і недоліки цього виду пошуку;
- зробити порівняльні висновки про особливості пошуку вглиб та вшир.

У програмі *передбачити*:

- створення графічного інтерфейсу керування роботою програми;
- візуалізацію графу, процесу пошуку і представлення його результатів;
- можливість змінювати розмірність і порядок графу, а також виду графу (звичайного, орграфу та дерева);
- можливість змінювати напрям обходу вершин графу при пошуку;
- результат пошуку (знайдений шлях на графі при заданих вершинах та кількість розкритих вершин) виводити в окремому вікні інтерфейсу.

Дослідити особливості роботи написаної програми при:

- різних напрямках обходу суміжних вершин при розкритті вершини;
- зміні напрямку пошуку (у т. ч. при дзеркальній заміні початкової вершини у і мети пошуку),

а також вплив на результати пошуку виду графа (дослідити особливості пошуку на дереві, звичайному та орієнтованому графах однакового порядку та розміру і різних розмірів), а також заміни деяких ребер графа на однонапрямлені дуги.

Порівняти результати роботи алгоритмів пошуку вшир та вглиб в різних умовах пошуку, використавши для цього один і той самий граф та однакові умови пошуку. Пояснити, в чому полягає основна відмінність організації пошуку на графах в ширину та глибину.

8.3. Завдання до роботи 3.

<u>Тема роботи</u>: Одно-направлений пошук на графі. Дослідження особливостей використання хвильового алгоритму для знаходження найкоротшого шляху у одиничних графах методом одно-направленого пошуку.

Вихідні дані:

Задано:

- одиничний граф порядку 10-20 розміром 20-30;
- початкова і цільова вершини графу;
- оператор переходу.

Завдання:

- представити заданий одиничний граф візуально у вигляді матриці суміжності відповідного розміру (тобто представити задачу у вигляді матриці-лабіринту, де -1 позначена непрохідна ділянка лабіринту, а 0 – прохідна);
- використовуючи одну з мов об’єктно-орієнтованого програмування, створити програму виконання одно-направленого пошуку на основі хвильового алгоритму з використанням різних операторів переходу:
  - «вверх-вниз-вправо-вліво»;
  - «перехід по діагоналях»;
  - комбінація двох вищеназваних операторів.

У програмі *передбачити*:

- створення графічного інтерфейсу;
- можливість змінювати розмірність і порядок графу (відповідної матриці лабіринту та її прохідної і непрохідної частин);
- зміну початкової та цільової вершин графу;
- реалізувати візуалізацію процесу пошуку і представлення його результатів;
- виведення результатів пошуку (координат вершин знайденого шляху,

та кількості циклів розкриття вершин при його визначенні та час, витрачений на здійснення пошуку) в окремому вікні.

Дослідити особливості роботи пошуку за використання різних операторів переходу як при однакових решті умов пошуку (виду графу, однакових початкової та кінцевої вершин), так і зміні розміру графу. Для цього:

- за допомогою створеної програми знайти найкоротший шлях з початкової до цільової точки створеного графу (лабіринту), використовуючи заданий оператор переходу;
- дослідити вплив на результат пошуку та його ефективність використання різних операторів переходу;
- відмітити переваги і недоліки використання кожного з операторів переходу;
- дослідити вплив на результат пошуку зміни розміру графу, на якому проводиться пошук.

Пояснити особливості модифікації пошуку в ширину при одно-направленому хвильовому пошуці. Намалювати блок-схему організації одно-направленого хвильового пошуку.

8.4. Завдання до роботи 4.

Тема роботи: Двонаправлений пошук на графі. Реалізація двонаправленого пошуку у хвильовому алгоритмі для знаходження найкоротшого шляху у одиничних графах.

Вихідні дані:

Задано:

- одиничний граф порядку 10-20 розміром 20-30;
- початкова і цільова вершини графу;
- оператор переходу.

Завдання:

- представити заданий граф у вигляді матриці суміжності відповідного розміру (тобто представити задачу у вигляді матриці-лабіринту, де -1 – непрохідна вершина графу (клітина лабіринту), а 0 – прохідна);
- на одній з мов об'єктно-орієнтованого програмування створити програму виконання двонаправленого пошуку шляху на графі з використанням хвильового алгоритму з використанням операторів переходу:
  - «вверх-вниз-вправо-вліво»;
  - «перехід по діагоналях»;
  - комбінація двох вищеназваних операторів.

У програмі *передбачити:*

- створення графічного інтерфейсу керуванням роботою програми;
- можливість змінювати розмірність і порядок графу (порядок і вигляд відповідної матриці лабіринту);
- зреалізувати візуалізацію процесу пошуку і виведення його результатів;
- результати пошуку – координати вершин найкоротшого шляху між початковою і кінцевою вершинами, віддаль між ними (кількість вершин), а також кількість циклів роботи та час здійснення пошуку вивести в окремому вікні.

При *дослідженні роботи* програми*:*

- знайти найкоротший шлях між заданими вершинами графу, використовуючи двонаправлений пошук (тобто одночасний початок зустрічного пошуку з початкової і цільової вершин) з використанням різних операторів переходу:
- результати пошуку вивести в окремому вікні інтерфейсу;
- відмітити переваги і недоліки використання кожного з операторів переходу;
- відмітити переваги використання двонаправленого пошуку в порів-

нянні з однонаправленим; пояснити, як і за рахунок чого вони отримуються.

Пояснити особливості різницю в організації двонаправленого пошуку в порівнянні з одно-направленим хвильовим пошуком. Навести блок-схему організації двонаправленого хвильового пошуку.

8.5. Завдання до роботи 5.

Тема роботи: Знаходження найкоротшого шляху на зважених графах за допомогою алгоритму Дейкстри. Знаходження найкоротшого шляху автомобільного сполучення між містами України.

Вихідні дані:

Задано:

- схема автомобільних маршрутів України та віддалі між найбільшими містами (див. Додаток 1 та Додаток 2);
- відправна і цільова точки маршруту.

Завдання:

- використовуючи Додаток 1 та Додаток 2, побудувати зважений граф, який відображає схему автомобільних шляхів України (вершини графу – міста, а ваги дуг – відстані на автомобільних шляхах між ними);
- використовуючи алгоритм Дейкстри, на одній з мов об’єктно-орієнтованого програмування написати програму пошуку найкоротшого шляху на зважених графах.

У програмі *передбачити:*

- створення графічного інтерфейсу керуванням роботою програми;
- можливість змінювати розмірність і порядок графу (додавати та видаляти з графу вершини та ребра (дуги));
- візуалізацію процесу пошуку;

- виведення найкоротшого шляху (переліку назв міст на шляху від початкової до цільової вершини) та відповідної йому віддалі в окремому вікні.

Дослідити роботу програми, для чого:

- знайти найкоротший шлях від заданих викладачем початкової до цільової вершини на графі;
- дослідити вплив зміни умов пошуку: розміру (видалення і додавання дуг чи ребер графу зі знайденого шляху) і порядку (видалення та додавання вершин графу) – на його результат.

Пояснити, в чому полягає суть модифікації схеми пошуку в ширину, яку застосував Дейкстра при реалізації свого пошуку найкоротшого шляху між вершинами на графі. Як це відображається на організації пошуку? Намалювати блок-схему реалізації пошуку за алгоритмом Дейкстри.

Контрольні питання:

1. Що являє собою граф?
2. Чому графи знайшли широке застосування при вирішенні задач у штучному інтелекті?
3. Які види графів ви знаєте?
4. Яка відмінність між ребром і дугою графу?
5. Яка є відмінність між поняттями «граф» та «дерево»?
6. Що розуміється рід розміром і порядком графу?
7. Що потрібно задати, щоб представити граф у комп'ютерній програмі?
8. Які способи представлення графів при написанні програм пошуку ви знаєте?
9. Що таке матриця суміжності?
10. Чим відрізняється матриця суміжності від матриці інцидентності?
11. Що таке списки суміжності?
12. В чому полягає суть пошуку на графі?
13. Як представляються результати пошуку на графі?
14. Які види пошуку на графі ви знаєте? Чим вони відрізняються?
15. У яких випадках шлях на графі не може бути знайдений?
16. Яка основна операція пошуку на графі?
17. Що таке стратегія пошуку на графі?
18. Які основні види сліпого пошуку ви знаєте?
19. Чим відрізняється сліпий пошук від евристичного?
20. У які списки зазвичай заносяться вершини під час здійснення пошуку?
21. Чим відрізняється процес організації пошуку на графі в ширину від аналогічного процесу у пошуку в глибину?
22. Які операції включає цикл розкриття вершини при пошуку в глибину?
23. Яку роль при організації пошуку відіграє оператор напрямку обходу вершин і коли він застосовується?
24. Яка відмінність у формуванні та використанні списку нерозкритих вер-

шин у пошуку в глибину та в ширину?

25. Чим суттєво відрізняється організація пошуку в глибину від пошуку в ширину на графах?
26. Як здійснюється знаходження результату пошуку в ширину на графі?
27. У якому виді пошуку (в глибину чи в ширину) розмір і порядок графа може більш вплинути на результат пошуку?
28. Який з двох пошуків (в глибину і в ширину) є повним і чому?
29. Які переваги та недоліки пошуків у глибину та в ширину?
30. Модифікацією якого виду пошуку є хвильовий пошук?
31. У чому суть хвильового пошуку на графах?
32. На яких графах використовується пошук за хвильовим алгоритмом?
33. Який вид представлення графа зазвичай використовується у хвильовому пошуці?
34. Що являє собою лабіринт, який використовується для візуального представлення поля пошуку у хвильовому алгоритмі?
35. Яким чином знімається можливість «зациклення» в пошуці на графі?
36. В чому полягає різниця між однонаправленим та двонаправленим пошуком?
37. Зі списками яких вершин оперує програма однонаправленого хвильового пошуку?
38. Як формується список нерозкритих вершин у одно направленому та двонаправленому пошуках?
39. Які операції входять у цикл розкриття вершин однонаправленого хвильового пошуку?
40. Які операції входять у цикл розкриття вершин двонаправленого хвильового пошуку?
41. Яка відмінність в організації циклу в одно-направленому та двонаправленому хвильових пошуках?
42. Як формується знайдений найкоротший шлях у одно-направленому по-

шуку?

43. В чому полягає відмінність формування найкоротшого шляху у випадку двонаправленого пошуку в порівнянні з однонаправленим?
44. Які переваги двонаправленого пошуку в порівнянні з одно направленим?
45. За рахунок чого досягаються переваги двонаправленого хвильового пошуку у порівнянні з одно-направленим?
46. Модифікацією якого виду пошуку є пошук за алгоритмом Дейкстри?
47. На яких графах можна здійснювати пошук найкоротшого шляху за алгоритмом Дейкстри?
48. Чи є пошук за алгоритмом Дейкстри повним?
49. На яких графах в принципі неможливо застосовувати пошук за алгоритмом Дейкстри?
50. Як можна організувати заповнення спису нерозкритих вершин у пошуку за алгоритмом Дейкстри?
51. Які дані можна отримати після закінчення пошуку на графі за алгоритмом Дейкстри?
52. Для чого на початковій стадії пошуку за алгоритмом Дейкстри всім вершинам присвоюються числові мітки зі значенням «безмежність»?
53. Який «механізм» присвоєння поточного значення числової мітки суміжним з нею вершинам при розкритті чергової вершини при пошуку за алгоритмом Дейкстри?
54. У якому випадку змінюється найкоротший шлях до поточної з розглядуваних вершин у випадку хвильового пошуку?
55. У якому випадку при розкритті вершини у випадку хвильового пошуку шлях до суміжних з нею вершин залишається незмінним?
56. Як може вплинути на результати пошуку зміна розміру і порядку графа, на якому здійснюється пошук з використанням за алгоритмом Дейкстри?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА.

1. Искусственный интеллект. Стратегии и методы решения сложных проблем. / Дж. Ф. Люгер – Изд-во Вильямс, 2003. – 864 с.
2. Нильсон Н. Искусственный интеллект. Методы поиска решений. / Н. Нильсон. – М.: Мир, 1973. — 273 с.
3. Рассел Стюарт. Искусственный интеллект: современный подход, 2-е изд. : Пер. с англ. / Стюарт Рассел, Питер Норвиг. – М. : Издательский дом «Вильямс», 2006. – 1408 с.
4. Бенерджи Р. Теория решения задач. Подход к созданию искусственного интеллекта» / Р. Бенерджи –Москва: Мир, 1972. – 224. с.
5. Кормен, Томас Х. Алгоритмы: построение и анализ, 2-е издание. : Пер. с англ. / Томас Х. Кормен, Чарльз И. Лейзерсон. Рональд Л. Ривест, Клифорд Штайн. – М. : Издательский дом «Вильяме», 2005. – 1296 с.
6. Макаренко С. И. Интеллектуальные информационные системы: уч. пособие. / С. Макаренко. – Ставрополь: СФ МГГУ им. М. А. Шолохова, 2009. – 206 с.
7. Павлов С. Н. Системы искусственного интеллекта: учеб. пособие. В 2-х частях. / С.Н. Павлов. – Томск: Эль Контент, 2011. – Ч. 1. – 176 с.
8. Стрелковська І.В. Дискретна математика: навч. посіб. / І. В. Стрелковська, А. Г. Буслаєв, О. М. Харсун [та ін..] − Одеса: ОНАЗ ім. О. С. Попова, 2010. − 196 с.

ДОДАТКИ

Додаток 1. Схема автомобільних шляхів України

Рис. 1. Схема автомобільних шляхів України

Додаток 2. Відстані між містами України.

Таблиця Д1. Відстані між містами України.

№	Маршрут	Віддаль, км
1.	Вінниця – Житомир	127
2.	Вінниця – Київ	267
3.	Вінниця – Кропивницький	320
4.	Вінниця – Миколаїв	428
5.	Вінниця – Одеса	423
6.	Вінниця – Черкаси	341
7.	Вінниця – Чернівці	276
8.	Дніпро – Донецьк	248
9.	Дніпро – Запоріжжя	85
10.	Дніпро – Кропивницький	244
11.	Дніпро – Кривий Ріг	146
12.	Дніпро – Полтава	160
13.	Дніпро – Харків	218
14.	Донецьк – Луганськ	156
15.	Донецьк – Маріуполь	113
16.	Донецьк – Харків	294
17.	Запоріжжя – Мелітополь	125
18.	Івано-Франківськ – Тернопіль	133
19.	Івано-Франківськ – Ужгород	293
20.	Київ – Житомир	140
21.	Київ – Полтава	340
22.	Київ – Суми	333
23.	Київ – Черкаси	189
24.	Київ – Чернігів	148
25.	Кропивницький – Полтава	246
26.	Луганськ – Харків	325

27.	Львів – Івано-Франківськ	134
28.	Львів – Луцьк	151
29.	Львів – Рівне	211
30.	Львів – Тернопіль	127
31.	Львів – Ужгород	268
32.	Львів – Чернівці	268
33.	Мелітополь – Маріуполь	192
34.	Мелітополь – Сімферополь	250
35.	Миколаїв – Херсон	69
36.	Рівне – Житомир	188
37.	Рівне – Луцьк	73
38.	Рівне – Тернопіль	155
39.	Харків – Полтава	143
40.	Херсон – Мелітополь	231
41.	Хмельницький – Вінниця	119
42.	Хмельницький – Житомир	183
43.	Хмельницький – Рівне	194
44.	Хмельницький – Тернопіль	112
45.	Хмельницький – Чернівці	187
46.	Суми – Харків	184
47.	Черкаси – Полтава	233
48.	Полтава – Суми	175
49.	Чернігів – Суми	306
50.	Сімферополь – Севастополь	81

Додаток 3. Теми рефератів

Таблиця Д.2. Теми рефератів

№ п/п	Назва теми
1.	Основні напрями досліджень у сфері штучного інтелекту.
2.	Сучасні підходи до створення технологій штучного інтелекту.
3.	Нейромережі, їх види та особливості їх застосування у системах ШІ.
4.	Машинний переклад. Особливості сучасного підходу до машинного перекладу.
5.	Робототехніка – напрям діяльності в сфері штучного інтелекту.
6.	Розпізнавання образів та його роль у системах штучного інтелекту.
7.	Експертні системи та їх роль у ШІ.
8.	Інструменти створення експертних систем. Оболонки ЕС.
9.	CLIPS – середовище побудови експертних систем.
10.	Числення предикатів у штучному інтелекті. Мова ПРОЛОГ.
11.	Мови функціонального програмування у ШІ. ЛІСП.
12.	Мови логічного програмування у ШІ.
13.	Онтології та їх роль у поданні знань.
14.	Проект Semantic WEB.
15.	Нечітка логіка у системах штучного інтелекту.
16.	Машинне навчання. Глибоке машинне навчання.
17.	Сучасний стан та перспективи розвитку штучного інтелекту.

Додаток 4. Вимоги до виконання та представлення результатів лабораторних робіт та реферату

При виконанні лабораторної роботи студенти повинні

1) уважно вивчити теоретичні відомості, що відносяться до даної лабораторної роботи;

2) незрозумілі питання з'ясувати за допомогою додаткової літератури;

3) продемонструвати роботу написаної програми на комп'ютері згідно з вимогами, вказаними у завданні до лабораторної роботи;

4) оформити звіт.

5) захистити звіт.

При захисті лабораторної роботи студенту необхідно представити звіт з лабораторної роботи, продемонструвати роботу програми та відповісти на контрольні питання.

Звіт складається з:

- титульної сторінки;
- змісту;
- теми роботи;
- мети мети;
- завдання до роботи;
- теоретичних відомостей, необхідних для виконання роботи;
- тексту написаної програми (у додатку);
- опису інтерфейсу та особливостей роботи програми;
- результатів дослідження роботи програми з врахування представлених у завданні до роботи вимог;
- висновків.

У представленому звіті повинні бути представлені усі потрібні відомості, які потрібні для виконання роботи та її захисту, у т. ч. короткий опис інтерфейсу з докладним описом операцій, що виконуються з його допомогою, до-

кладний опис ходу дослідження особливостей роботи програми з використанням необхідного графічного матеріалу (зокрема, скріншотів вузлових моментів виконання роботи), які пояснюють та ілюструють результати роботи написаної програми, у т. ч. при забезпеченні виконання поставлених у завдання вимог; результати пошуку повинні бути виведені в окремому (стаціонарному чи випадному) вікні інтерфейсу.

Оформлення звіту виконувати з дотриманням прийнятих в університеті нормативних документів щодо оформлення звітної документації.

Терміни захисту виконаних робіт показані в табл. Д3. У випадку невчасного виконання та захисту лабораторної роботи знижуються підсумкові бали за її виконання на 50%.

Таблиця Д3. Терміни захисту виконаних робіт:

Робота, №	1	2	3	4	5	Реферат
Термін захисту (тиждень семестру)	4	6	8	10	12	13-16
Оцінка (бали)	5	5	5	5	5	10

Окрім виконання та захисту лабораторних робіт, при оцінюванні роботи студента протягом семестру будуть оцінюватися також і результати написання та захисту реферату (список рекомендованих тем рефератів – у Додатку 3). При оцінці враховуватимуться зміст, оформлення реферату, його усне представлення, відповіді на запитання.

Реферат складається з титульної сторінки, змісту, вступу, основної частини, висновків і списку використаних джерел. Оформлення реферату – згідно нормативних документів, що діють в ЛНУ ім. І. Франка. Розмір – 15-20 с.

Захист рефератів відбувається на 13-16 лабораторних заняттях. На захист реферату студентові надається до 15 хв, включаючи і відповіді на запитання, у т. ч. представлення реферату – до 7 хв.