Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Єрастова Едуарда Юрійовича*
(ПІБ)

академічної групи *122-18ск-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка навчальної інформаційної системи для пошуку оптимальних шляхів на зв'язних неорієнтованих ациклічних графах*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18ск-1* *Єрастова Едуарда Юрійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка навчальної інформаційної системи для пошуку оптимальних шляхів на зв'язних неорієнтованих ациклічних графах*

затверджена наказом ректора НТУ «ДП» від « » 2021 р. №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Єрастов Е.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 65с., 14 рис., 3 дод., 22 джерела.

Об'єкт розробки: навчальна інформаційна система для пошуку оптимальних шляхів на зв'язних неорієнтованих ациклічних графах.

Мета дипломного проекту: розробка алгоритму і програмна реалізація методу пошуку найкоротшого шляху між двома заданими вершинами у зв’язному неорієнтованому ациклічному графі та підвищення ефективності методів самостійної роботи з графами.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування підсистеми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню системи та розраховано час на її створення.

Практичне значення полягає у створенні інформаційного додатку для роботи з графами.

Актуальність програмного продукту визначається великим попитом на подібні роботи.

Список ключових слів: ГРАФ, МАРШРУТ, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note: 65p., 14 fig., 3 appendices, 22 sources.

Object of development: educational information system for finding optimal paths on connected undirected acyclic graphs.

The purpose of the diploma project: development of algorithm and software implementation of the method of finding the shortest path between two given vertices in a coherent undirected acyclic graph and increase the efficiency of methods of independent work with graphs.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the subsystem, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and application load, describes the program.

In the economic section, the complexity of the developed information subsystem is determined, the cost of work on the creation of the system is calculated and the time for its creation is calculated.

The practical significance lies in the creation of an information application for working with graphs.

The relevance of the software product is determined by the high demand for such work.

Keywords: GRAPH, ROUTE, INFORMATION SYSTEM.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ - електронно-обчислювальна машина;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

ОС - операційна система;

ІТ - інформаційні технології.

ВСТУП

Теорія графів – розділ математики, в якому вивчаються графи та їх властивості. Графи звичайно зображують у вигляді точок (вершин), з’єднаних відрізками (ребрами). Графи можна задавати графічно, у вигляді списків ребер, за допомогою матриць інцидентності та суміжності, що значно спрощує створення математичних моделей на основі візуального представлення. За допомогою графів можна візуально представити об’єкти та відношення, або зв’язки, між ними. Все в реальному світі пов'язано: міста можна представити як мережу вулиць, залізничну та польотну мережу.

Теорію графів широко застосовують у логістиці. Інтелектуальні транспортні системи можуть працювати, збираючи дані про місцезнаходження від навігаторів автомобілів і передавати інформацію водіям, де і як швидко їздити, щоб зменшити загальне перевантаження. Теорія графів вже використовується авіакомпаніями, які хочуть з'єднати велику кількість міст найефективнішим чином, створити систему переміщення великої кількості пасажирів з найменшою кількістю можливих поїздок. Дана проблема схожа за своєю суттю на задачу про комівояжера. У той же час, авіадиспетчери повинні переконатися, що сотні літаків знаходяться в потрібному місці в потрібний час і запобігти можливим аваріям. Вирішення цього завдання не було б можливим без комп'ютерів і теорії графів.

Теорія графів широко застосовується в задачах планування, архітектури, електроенергетики, електротехніки, інженерії транспорту, а особливо в інформатиці та телекомунікаціях. Графи пов’язані з комбінаторикою, дискретною математикою, топологією, теорією алгоритмів та іншими розділами математики.

Завдання пошуку найкоротшого шляху між двома вузлами в мережі – одна з класичних проблем теорії графів. Одною з вимог пошуку є мінімізація часу обчислення найкоротших шляхів. Алгоритм повинен знаходити оптимальне або близьке до оптимального рішення за мінімальний час.

Проблема пошуку найкоротших шляхів у графі є загальновідомою і важливою для різних галузей. Існує ряд алгоритмів для вирішення цього завдання. Останнім часом ця проблема інтенсивно вивчається для графів складної багаторівневої структури. У даній роботі розглядається задача пошуку найкоротших шляхів між двома заданими вершинами у зв’язних неорієнтованих ациклічних графах.

Пошук оптимального шляху нині широко використовується: від систем глобального позиціонування для знаходження найкоротшого маршруту серед міських вулиць і шляхів між містами, у військових і цивільних системах автопілотування, у транспортно-експедиційному обслуговуванні, при комутації інформаційного пакету Internet – і аж до комп’ютерних ігор. Велике значення вирішення такого завдання має при організації маршрутизації в мережах зв’язку і є важливою складовою процесу розробки штучних нейронних мереж, який втілює визначальну тенденцію структурного підходу по дослідженню можливості створення природного інтелекту за допомогою комп’ютерних алгоритмів.

Тому тематика даної роботи актуальна.

Мета роботи – розробка алгоритму і програмна реалізація методу пошуку найкоротшого шляху між двома заданими вершинами у зв’язному неорієнтованому ациклічному графі.

Завдання:

1. Узагальнити основні поняття стосовно теорії графів, які потрібні для розв’язання задачі знаходження найкоротшого шляху.
2. Розробити алгоритм пошуку найкоротшого шляху між двома заданими вершинами в зв’язному неорієнтованому ациклічному графі.
3. Проаналізувати алгоритм пошуку найкоротшого шляху на довільному неорієнтованому ациклічному графі (дереві) та узагальнити цей алгоритм для довільних ациклічних зв’язних неорієнтованих графів.
4. Розробити програму для знаходження оптимальних шляхів у зв’язних неорієнтованих ациклічних графах.

5. Проаналізувати розроблений алгоритм та зробити висновки щодо його оптимальності.

Практична цінність розробки – можливість застосування алгоритму пошуку найкоротшої відстані при розробці програмного забезпечення при вирішенні практичних задач пошуку найкоротших шляхів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Граф - це сукупність об'єктів із зв'язками між ними. Об'єкти розглядаються як вершини, або вузли графу, а зв'язки – як дуги, або ребра. Для різних областей використання види графів можуть відрізнятися орієнтовністю, обмеженнями на кількість зв'язків і додатковими даними про вершини або ребра.

Ребра графа можуть бути направленими або ненаправленими. Наприклад, якщо вершини будуть представляти людей на вечірці, і існуватиме ребро між двома людьми, якщо вони потиснули руки, тоді цей граф є ненаправлений, оскільки будь-яка особа A може потиснути руки із особою B лише, якщо B також потисне руки із A. На противагу цьому, якщо будь-яке ребро від особи A до персони B означатиме, що персоні A' подобається B, тоді цей граф буде направленим, оскільки таке вподобання не обов'язково буде взаємним. Перший тип графів називатиметься неспрямованим графом, а ребра в свою чергу називатимуться неспрямованими ребрами в той час як другий тим графів називатиметься спрямованим графом і ребра називатимуться спрямованими ребрами.

Орієнтований граф з трьома вершинами і трьома ребрами (рис. 1.1.)

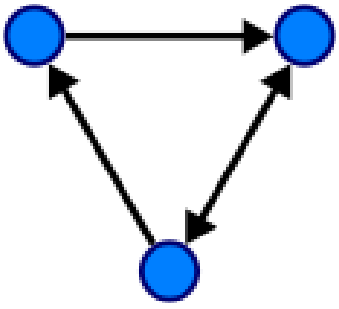

Рис. 1.1. Орієнтований граф з трьома вершинами і трьома ребрами

Велика кількість структур, які мають практичну цінність в математиці та інформатиці, можуть бути представлені графами.

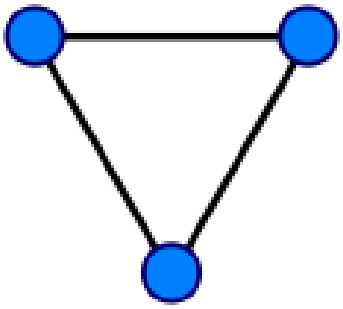

Рис. 1.2. Простий не орієнтований граф

Простий не орієнтований граф (рис. 1.2.). Кожна вершина має ступінь два, тому це буде регулярний граф.

Граф або неорієнтований граф G - це впорядкована пара G:=(V,E), для якої виконуються наступні умови:

- V - множина вершин або вузлів,
- E - множина пар (у випадку неорієнтованого графу - невпорядкованих) вершин з V, які називають ребрами.

V (і так само E) зазвичай вважаються скінченними множинами. Велика кількість результатів, отриманих для скінченних графів, невірна (або інша) для нескінченних графів. Це пов'язано з тим, що певний набір ідей стає хибним у випадку нескінченних множин.

Граф (геометричний граф) - це фігура на площині, яка складається з непорожньої скінченної множини V точок (вершин) і скінченної множини E орієнтованих чи не орієнтованих ліній (ребер), що з'єднують деякі пари вершин.

Орієнтований граф, це граф який містить тільки ребра називається неорієнтованим, який містить тільки дуги - орієнтованим. Граф, що має як ребра так і дуги, називається мішаним. Якщо пара вершин сполучається кількома ребрами чи дугами одного напрямку, то ребра (дуги) називають

кратними (паралельними). Дуга чи ребро, що сполучає вершину саму із собою називається петлею. Граф без кратних дуг і петель називається простим.

Вершини сполучені ребром чи дугою називають суміжними, також називають суміжними ребра, що мають спільну вершину. Ребро (чи дуга) і її вершина називаються інцидентними. Ребро (u, v) з'єднує вершини u і v, дуга (u, v) починається у вершині u і закінчується у вершині v.

Кожен граф можна відобразити в евклідовому просторі множиною точок, які відповідають вершинам, сполучених лініями, що відповідають ребрам (дугам).

Іноді є потреба пару вершин з'єднати більше, ніж одним ребром. Мультиграфом називають пару G=(V,E), де V - множина, елементи якої називають вершинами. E - сім'я ребер, кожне з яких - це пара вершин із V.

Ребра, які з'єднують одну й ту саму пару вершин, називають кратними (паралельними) ребрами.

Мультиграф, який може мати петлі, іноді називають псевдографом.

Граф – скінченна кількість точок, які називаються вершинами, та ліній, що попарно з’єднують деякі з цих вершин і називаються ребрами (в неорієнтованому графі), або дугами (в орієнтованому). Вершини, сполучені ребром, називаються суміжними, а ребро, яке їх з’єднує - інцидентним цим вершинам.

Повний граф: у ньому кожна вершина з’єднана з усіма іншими. Кількість ребер у повному неорієнтованому графі з N вершинами дорівнює: $\frac{N*(N-1)}{2}$.

Ступінь вершини графа - кількість ребер, інцидентних даній вершині.

Шлях у графі - послідовність вершин, де кожна вершина з'єднана з наступною ребром або дугою.

Список ребер – один зі способів зберігати граф у пам’яті комп’ютера. Це список, де в кожному рядку - два числа: номери вершин, сполучених даним ребром.

Імітація відпалу - у кожного металу є кристалічна решітка, в її вузлах атоми металу. Сукупність позицій всіх атомів є станом системи, кожному стану відповідає певний рівень енергії. Мета відпалу - привести систему до стану з найменшою енергією. Чим нижче рівень енергії, тим досконаліше кристалічна решітка (тим менше у ній дефектів і міцніше метал).

При відпалі метал спочатку нагрівають до певної температури, що змушує атоми кристалічної решітки покинути свої позиції. Потім починається повільне і контрольоване охолоджування. Атоми прагнуть потрапити до стану з меншою енергією, однак, із певною ймовірністю, можуть перейти і до стану з більшою. Ця ймовірність зменшується із температурою. Перехід до стану з більшою енергією, як не дивно, допомагає в підсумку відшукати стан з енергією меншою, ніж початкова. Процес завершується, коли температура падає до заданого значення.

Мурашиний алгоритм - один з ефективних поліноміальних алгоритмів для знаходження наближених рішень задачі комівояжера, а також вирішення аналогічних завдань пошуку маршрутів на графах. Суть підходу полягає в аналізі та використанні моделі поведінки мурах, що шукають шляхи від колонії до джерела живлення.

Теорія графів - розділ математики, що вивчає властивості графів. Наочно граф можна уявити як геометричну конфігурацію, яка складається з точок (вершини) сполучених лініями (ребрами). У строгому визначенні графом називається така пара множин G = (V, E), де V є підмножина будь-якої зліченної множини, а E - підмножина V × V.

Визначення графу є настільки загальним, що цим терміном можна описувати безліч подій та об'єктів повсякденного життя. Високий рівень абстракції та узагальнення дозволяє використовувати типові алгоритми теорії графів для вирішення зовнішньо несхожих задач у транспортних і комп'ютерних мережах, будівельному проектуванні, молекулярному моделюванні тощо. Теорія графів знаходить застосування, наприклад, в геоінформаційних системах (ГІС). Існуючі або запроектовані будинки,

споруди, квартали тощо розглядаються як вершини, а дороги, інженерні мережі, лінії електропередачі, що їй з'єднують, тощо - як ребра. Застосування різних обчислень, вироблених на такому графі, дозволяє, наприклад, знайти найкоротший об'їзний шлях або найближчий продуктовий магазин, спланувати оптимальний маршрут.

Теорія графів містить велику кількість невирішених проблем і поки не доведених гіпотез.

Останнім часом зростає увага до вивчення і аналізу пошуку найкоротших шляхів між заданими пунктами. Поряд з удосконаленням існуючих методів розробляються принципово нові.

Математичні моделі великої кількості задач можуть бути описані в термінах теорії графів, тому алгоритми дослідження обробки графів, а також способи їх подання дуже важливі. Але перш ніж розглядати задачу пошуку найкоротшого шляху і описувати алгоритм її рішення, слід дати визначення використовуваним в науково-дослідницькій роботі термінам.

Граф (неорієнтований) G = (V, E) – сукупність двох множин, де V – кінцева множина елементів, яка називається вершинами, а E – множина пар різних елементів множини V (ці пари називаються ребрами). Граф, що містить одну вершину, називається тривіальним.

Граф простий, якщо будь-яка пара вершин з’єднана не більше ніж одним ребром і граф не має петель – ребр, що повертаються до вершини, з якої виходять.

Маршрут (шлях) – це послідовність вершин і ребр графа:

$$a = v_0, e_1, v_1, e_2, \dots, e_{n-1}, e_n, v_n = b \tag{1.1}$$

Ця послідовність така, що $e_i = (v_{i-1}, v_i)$, $1 \le i \le n$. Кажуть, що маршрут з'єднує вершини a і b – кінці маршруту. У простому графі маршрут можна задати перерахуванням лише його вершин $a = v_0, v_1, \dots, v_n = b$ або його ребр $e_1, e_2, \dots, e_n$.

Деревом називається зв'язний граф без циклів.

Неорієнтований граф можна представити матрицею суміжності a[n][n], де n – кількість вершин у графі. Для будь-якого її елементу a[i][j] де i та $j \in V$:

$$a[i][j] = \begin{cases} 1, \text{якщо вершина i суміжна з вершиною j,} \\ 0, \text{якщо вершина i не суміжна з вершиною j.} \end{cases} \quad (1.2)$$

Матриця суміжності однозначно описує граф (рис 1.3). Для неорієнтованого графа вона симетрична відносно головної діагоналі. Число одиниць у рядку дорівнює ступеню відповідної вершини.

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	0
2	1	0	0	0	0
3	0	1	0	0	1
4	0	0	0	1	0

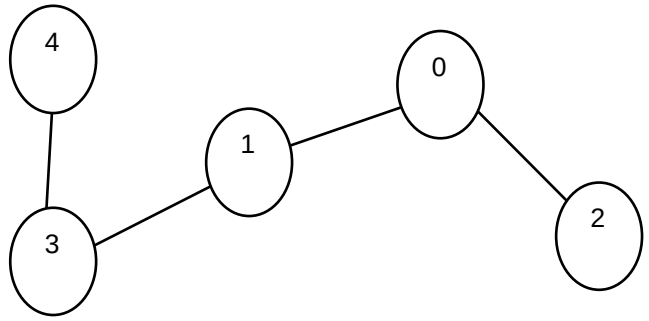

Рис. 1.3. Неорієнтований зв’язний ациклічний граф, поданий у вигляді матриці суміжності a[n][n] та у графічному вигляді

1.2. Призначення розробки та галузь застосування

Іінформаційна система, що розробляється, в першу чергу призначена для підвищення ефективності методів самостійної роботи з графами.

Практична цінність розробки полягає у можливості застосування алгоритму пошуку найкоротшої відстані при розробці програмного забезпечення при вирішенні практичних задач пошуку найкоротших шляхів з міста на одному притоку великої річки до міста на іншому її притоку та інших аналогічних задач.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки кваліфікаційної роботи є:

- Освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчальногопроцесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07. 06. 2021р;
- завдання на кваліфікаційну роботу на тему «Розробка навчальної інформаційної системи для пошуку оптимальних шляхів на зв'язних неорієнтованих ациклічних графах».

1.4. Постановка завдання

У квіліфікаційній роботі за мету ставиться підвищення ефективності методів самостійної роботи з графами при вирішенні математичних та логістичних задач.

Завдання:

1. Узагальнити основні поняття стосовно теорії графів, які потрібні для розв’язання задачі знаходження найкоротшого шляху.
2. Розробити алгоритм пошуку найкоротшого шляху між двома заданими вершинами в зв’язному неорієнтованому ациклічному графі.
3. Проаналізувати алгоритм пошуку найкоротшого шляху на довільному неорієнтованому ациклічному графі (дереві) та узагальнити цей алгоритм для довільних ациклічних зв’язних неорієнтованих графів.
4. Розробити програму для знаходження оптимальних шляхів у зв’язних неорієнтованих ациклічних графах.

5. Проаналізувати розроблений алгоритм та зробити висновки щодо його оптимальності.
6. Розробка алгоритму роботи програмного додатку.
7. Розробка інформаційного додатку, що відповідає сучасним вимогам.
8. Тестування розробленоъ системи.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

До додатку, що розобляється в роботі, за мету ставляться наступні вимоги до функціональних характеристик:

- зручний інтуітвно-зрозумілий інтерфейс;
- невимогливість до складу програмно-апаратного комплексу.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

– контроль вхідних даних;

– обробку виняткових ситуацій;

– виведення повідомлень про помилки;

– можливість повторного введення даних;

– можливість безперервної роботи протягом не менше 120 годин (5 діб);

– платформну незалежність;

– вірогідність виникнення не більше 2 логічних помилок на 1000 операторів за 1 рік експлуатації.

При роботі з програмою достатньо встановлених в системі вимог до інформаційної безпеки. Особливі вимоги до інформаційної безпеки не встановлюються.

Виключень з боку антивірусних програм та програм захисту не потребує.

Програма не потребує прав адміністратора чи спеціальної кваліфікації.

1.5.3. Вимоги до складу та параметрів технічних засобів

Апаратні вимоги:

- Двоядерний процесор частотою 2 GHz або вище.
- 4096 MB RAM.
- 100 MB вільного місця на жорсткому диску.
- Відеокарта — 1024 Mb.
- Монітор.
- Клавіатура та миша.

1.5.4. Вимоги до інформаційної та програмної сумісності

Вимоги до інформаційних структур, вихідним кодам, мов програмування та програмним засобам, використаним для розробки та функціонування розробленої системи не встановлюються за винятком того що програма повинна бути невимогливою до програмно-апаратного комплексу та встановлено наступні:

- Версія ОС Windows: 7 та новіші.
- NET.Framework 4.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Іінформаційна система, що розробляється, в першу чергу призначена для підвищення ефективності самостійної роботи з графами.

Також результати роботи можна використовувати для пошуку оптимального шляху нині широко використовується: від систем глобального позиціонування для знаходження найкоротшого маршруту серед міських вулиць і шляхів між містами, у військових і цивільних системах автопілотування, у транспортно-експедиційному обслуговуванні, при комутації інформаційного пакету Internet – і аж до комп’ютерних ігор.

Велике значення вирішення такого завдання має при організації маршрутизації в мережах зв’язку і є важливою складовою процесу розробки штучних нейронних мереж, який втілює визначальну тенденцію структурного підходу по дослідженню можливості створення природного інтелекту за допомогою комп’ютерних алгоритмів.

2.2. Опис застосованих математичних методів

Застосовані математичні методи, що було використано в ході виконання роботи описано в п.1.1 та п. 2.4. Додаткові математичні методи

2.3. Опис використаних технологій та мов програмування

Програму створено в середовищі програмування Microsoft Visual Studio 2019 мовою C#. Було використано технології об'єктно-оріентованого програмування та компонентного програмування.

Об'єктно-оріентоване програмування (ООП) - головним елементом

алгоритму є класи - нові типи даних, що розширюють можливості мови. Використання класів дозволяє сховати від програміста більшість чорнової роботи, тому що змінні й підпрограми ховаються в класах і викликаються невидимо для програміста. Це дозволило писати великі й складні програми, тому що в процедурних мовах з ростом програми різко збільшувалося число помилок. Проте, усередині класи пишуться, як і звичайні процедурні програми.

C# відноситься до сім'ї мов із C-подібним синтаксисом, з них її синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, винятки, коментарі у форматі XML.

У принципі, явного лідера немає, тому при виборі мови потрібно враховувати те, що важливіше - швидкість розробки або швидкість роботи готового додатку. Можна, звичайно, зробити і багатомовний проект, але поєднувати такі мови, як Java, C++ і C# - технічно не найпростіше завдання. Хоча в деяких програмах воно і вирішене, варто взяти до уваги, що такий підхід збільшує витрати на розробку.

Ще один важливий фактор - кількість підтримуваних платформ. Для Java і C# це не особливо критично, а ось на C++ весь платформно-залежний код потрібно буде писати окремо для кожної ОС. Відповідно, чим менше підтримуваних ОС, тим менший код.

C# – об’єктно-орієнтована мова програмування. Розроблена в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберга в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework і згодом був стандартизований як ECMA-334 та ISO/IEC 23270.

Перейнявши багато чого від своїх попередників – мов C++, Java, Delphi і Smalltalk – C#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних

систем, наприклад, C# не підтримує множинне успадкування класів (на відміну від C++).

C# розроблявся як мова програмування прикладного рівня для CLR і, як такий, який залежить, насамперед, від можливостей самої CLR. Це стосується, перш за все, системи типів C#, яка відображає BCL. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність була порушена з виходом C# 3.0, що представляє собою розширення мови, не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а виробляється CLR для програм, написаних на C# точно так само, як це робиться для програм на VB.NET, J# і інше.

У роботі використано наступні елементи управління.

- TextBox – для зчитування даних.
- ListBox – для зображення вхідних даних.
- Button – для запуску алгоритму.
- Label – як засіб підпису елементів управління (для кращого орієнтування на формі).

2.4. Опис структури системи та алгоритмів її функціонування

Розглянемо довільне неорієнтоване зв’язне незважене ациклічне дерево (рис. 2.1), яке задовольняє умові задачі. На ньому будемо аналізувати алгоритм пошуку оптимальної відстані між двома вершинами.

Використаємо алгоритм пошуку в глибину для знаходження відстаней від кореня дерева root до усіх інших вершин дерева (рис. 2.2).

```
// пошук в глибину з вершини v (корінь дерева, тому v = 0)
    void dfs (int v = 0, int len = 0) {
    used[v] = 1; // відзначаємо вершину v пройденою
    dist[v] = len; // відстань від кореня дерева до вершини v дорівнює len
// шукаємо ребро, по якому можна перейти до не відвіданої вершини
з номером i
    for (int i = 0; i < n; i++)
// якщо вершина i ще непройдена та має ребро з v, переходимо до неї
    if ((g[v][i] == 1) && (used[i] == 0))
// рекурсивний запуск пошуку в глибину з вибраної вершини i, додаємо
одиницю до відстані від кореня
    dfs(i, len + 1); }
```

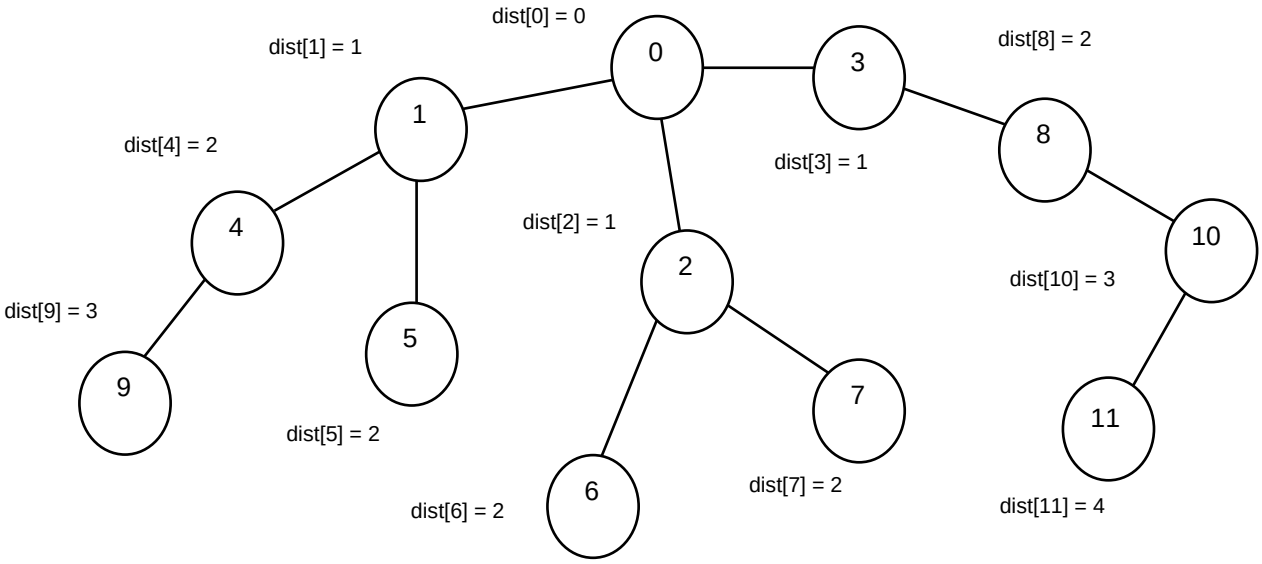

Рис. 2.2. Неорієнтоване зв’язне ациклічне дерево. Результат пошуку в глибину – знайдено відстані від кореня дерева до усіх вершин графа

Далі слід запам’ятати послідовність вершин при пошуку в глибину (рис. 2.3).

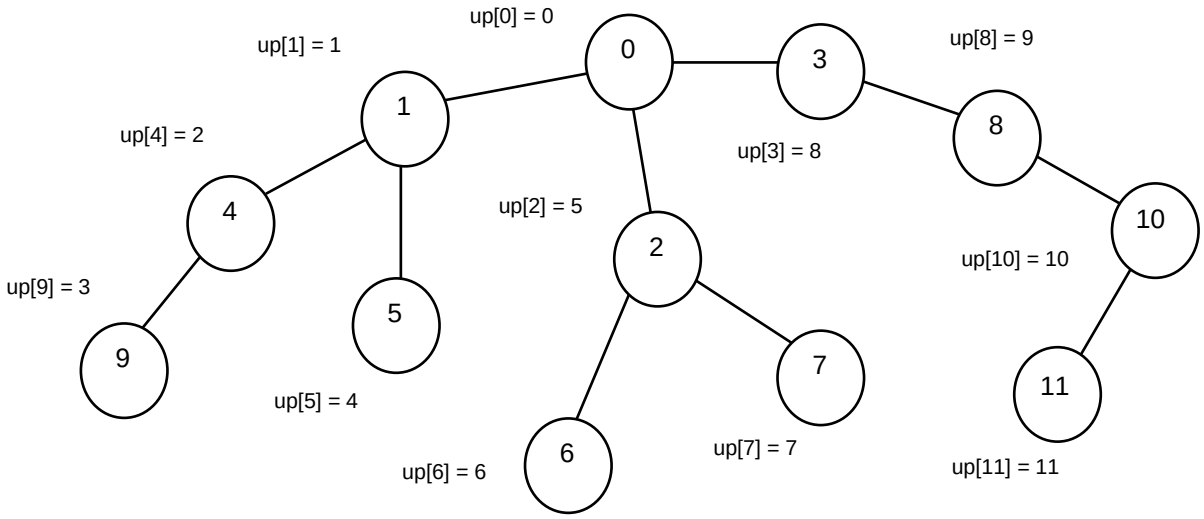

Рис. 2.3. Неорієнтоване зв’язне ациклічне дерево. Зображено номери вершин при пошуку в глибину.

```
// пошук у глибину з вершини v (корінь дерева, тому v = 0)
    void dfs(int v, int len = 0) {
// помічаємо вершину v як пройдену та вказуємо шлях до цієї вершини від кореня дерева
    used[v] = 1; dist[v] = len;
    up[j] = v;
// Запам’ятовуємо номери вершин, між якими потрібно знайти оптимальний шлях, при застосуванні //пошуку в глибину.
    if (v == a) ja = j;
    if (v == b) jb = j;
    j++; // лічильник.
// Шукаємо вершину, від якої можна перейти до не пройденої вершини з номером i.
    for (int i = 0; i < n; i++)
    if ((g[v][i] == 1) && (used[i] == 0))
    fs(i, len + 1); } // Рекурсивний пошук у глибину з даної вершини
```

Після знаходження dist[n] і up[n] можна визначити рівні, де знаходяться вершини, між якими шукаємо оптимальний шлях – dist[a] та dist[b].

Нехай потрібно знайти оптимальний шлях між вершинами 9 та 11 (рис. 2.4).

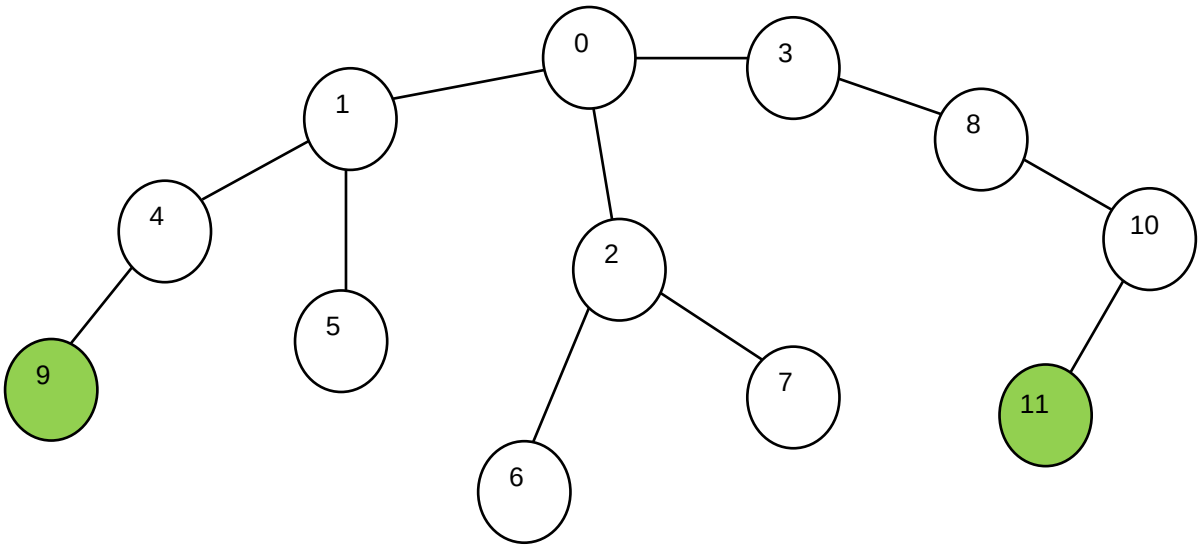

Рис. 2.4. Неорієнтоване зв’язне ациклічне дерево. Зображено вершини, між якими будемо шукати оптимальний шлях (вершини 9 та 11)

Для визначення довжини оптимального шляху потрібно між двома заданими вершинам (вершини 9 та 11) слід визначити рівні, на яких вони знаходяться. З (рис. 2.2) бачимо, що вершина 9 заходиться на третьому рівні (dist[9] = 3), а вершина 11 - на четвертому (dist[11] = 4).

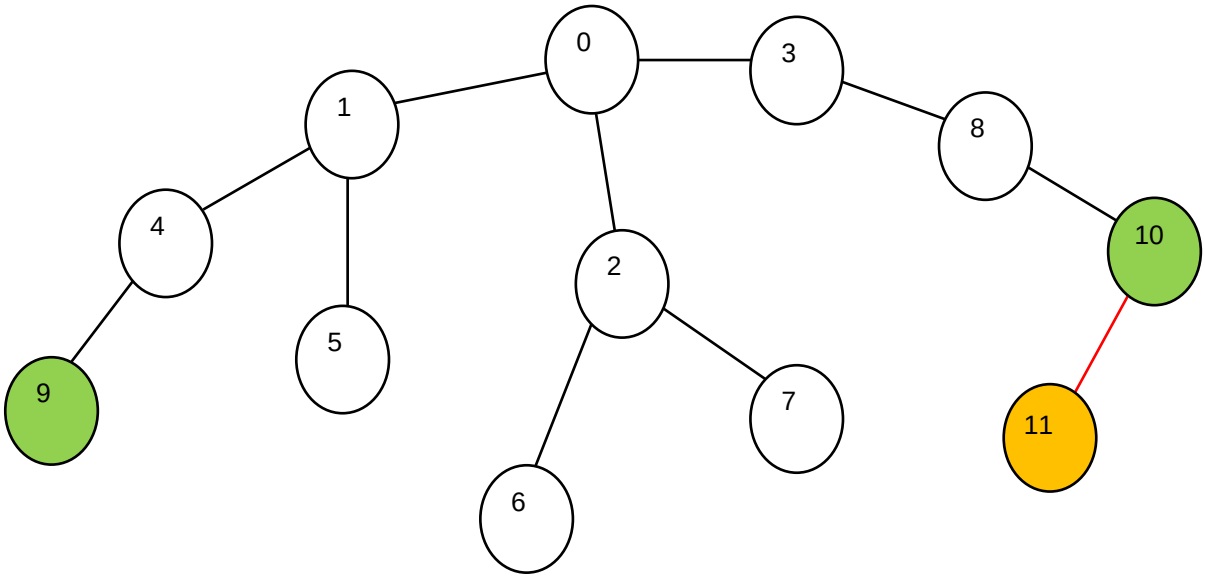

Рис. 2.5. Неорієнтоване зв’язне ациклічне дерево. Вирівнюємо відстані до кореня

Після визначення рівнів вершин слід перейти від рівня нижчої до рівня вищої, додаючи кількість пройдених ребр до загального шляху (рис 2.5). У нашому випадку нижча вершина має номер 11. Тому підіймаємось від вершини 11 до вершини 10. Загальна пройдена відстань дорівнює 1 (пройдено одне ребро). Далі будемо шукати шлях між вершинами 9 та 10.

Опис цього пересування вищої робимо в циклі: поки обидві вершини не будуть на однаковій відстані від кореня, підіймаємо нижчу на одиницю вище у бік кореня:

```
// Застосовуємо допоміжні змінні, щоб не втратити номери шуканих вершин при пошуку в глибину.
    int i = ja, j = jb;
// Доки не перейдемо від нижчої вершини до рівня вищої, виконуємо наступні дії.
    while (dist[a] != dist[b]) {
    if (dist[a] < dist[b]) // Визначаємо, яка з вершин нижча.
// Якщо нижча - b, порівнюємо її рівень з рівнем вершини, що передувала ній при пошуку в глибину (вища або лівіша).
    if (dist[b] > dist[up[j - 1]]) {
// Якщо вища, переходимо до неї.
    b = up[j - 1];
// номер при пошуку в глибину нової вершини (до якої перейшли).
    jb = j - 1;
// Зменшуємо допоміжну змінну на одиницю (щоб вона дорівнювала jb).
    j--;
// Збільшуємо довжину пройденого шляху на одиницю (пройдено одне ребро.)
    l++; }
    else
// Якщо вершина, що передувала поточній при пошуку в глибину, лівіша, то беремо попередню вершину, при пошуку в глибину
```

```
        j--;
// Якщо нижча вершина a, виконуємо аналогічні дії для вершини a.
        else
// Якщо нижча a, порівнюємо її рівень з рівнем вершини, яка передувала ній при пошуку в глибину (вища або лівіша).
        if (dist[a] > dist[up[i - 1]])
        {
// Якщо вища, переходимо до неї
        a = up[i - 1];
// Це номер при пошуку в глибину нової вершини (до якої перейшли).
        ja = i - 1;
// Зменшуємо допоміжну змінну на одиницю (щоб вона дорівнювала ja).
        i--;
// Збільшуємо довжину пройденого шляху на одиницю (пройдено одне ребро).
        l++;
        }
        else
// Якщо вершина, що передувала поточній при пошуку в глибину, лівіша, то беремо попередню вершину, при пошуку в глибину
        i--;
        }
```

Далі на кожній ітерації циклу робимо кроки від кожної вершини у бік кореня дерева (рис. 2.6, 2.7, 2.8). На кожному кроці пройдений шлях збільшуємо на 2. Коли ці шляхи зійдуться – кінець пошуку (оптимальний шлях знайдено). Жовтим кольором позначено пройдені вершини, зеленим – вершини, на яких ми знаходимося у даний момент, червоним – ребра, додані до загального пройденого шляху:

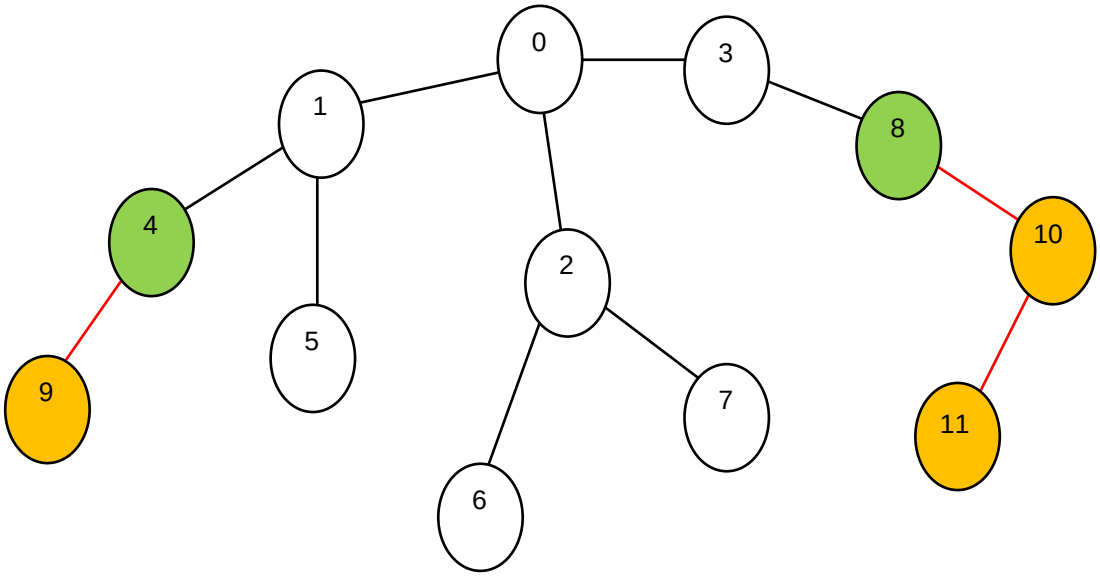

Рис. 2.6. Неорієнтоване зв’язне ациклічне дерево. Одночасно підіймаємось від вершин 9 та 10 (у бік кореня дерева) до вершин 4 та 8

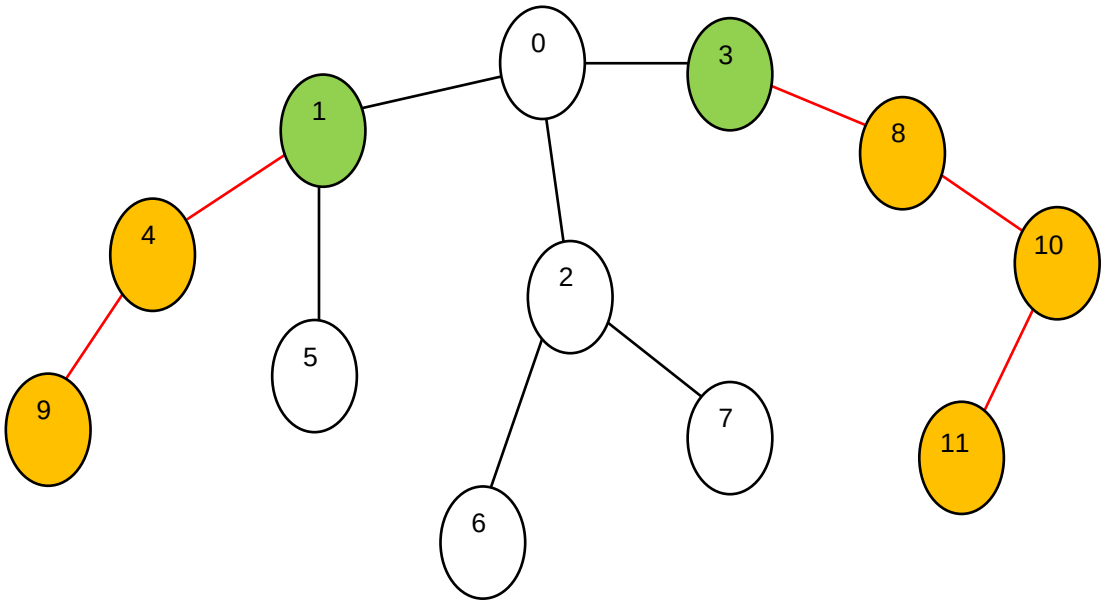

Рис. 2.7. Неорієнтоване зв’язне ациклічне дерево. Одночасно підіймаємось від вершин 1 та 3 (у бік кореня дерева) до вершини 0 (корінь дерева)

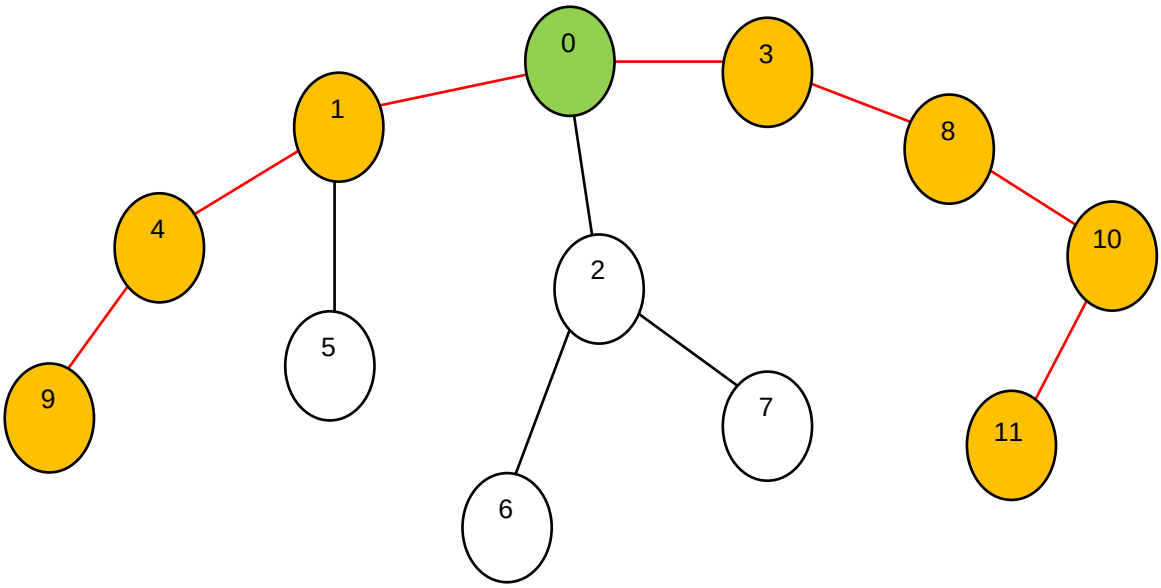

Рис. 2.8. Неорієнтоване зв’язне ациклічне дерево. Кінець підйому: шляхи зійшлись в одній вершині (у цьому прикладі – в корені дерева).

Наступний крок є останнім (рис. 2.8): після нього шляхи зійдуться в одній вершині (у нашому випадку – в корені дерева). Оптимальний шлях знайдено – це загальний пройдений шлях.

// Застосовуємо допоміжні змінні, щоб не втратити номери шуканих вершин при пошуку в глибину.

int i = ja, j = jb;

// Поки шляхи не зійдуться в одній вершині, виконуємо команди циклу.

while (a != b)

{

// Порівнюємо рівні заданої вершини та вершини, що передувала їй при пошуку в глибину (вона може бути або лівіша, або вища).

if (dist[a] > dist[up[i - 1]])

{

// Якщо вища, то переходимо до неї.

a = up[i - 1];

// Це номер нової вершини, до якої перейшли при пошуку в глибину.

ja = i - 1;

```
// Зменшуємо допоміжну змінну на одиницю (щоб вона дорівнювала ja).
        i--;
// Збільшуємо загальний пройдений шлях на одиницю (пройдено одне ребро).
        l++;
        }
        else
// Якщо вершина, яка передувала поточній при пошуку в глибину, лівіша, переходимо до цієї вершини.
        i--;
// Порівнюємо рівні другої заданої вершини та вершини, попередньої поточній у пошуку в глибину (вона може буди або лівіша або вища).
// Доки вершини a і b не будуть на одному рівні, виконуємо наступні дії.
        while (dist[a] != dist[b]
        {
        if (dist[b] > dist[up[j - 1]])
        {
// Якщо попередня вершина при пошуку в глибину вища, переходимо до неї.
        b = up[j - 1];
// Це номер при пошуку в глибину нової вершини (до якої перейшли).
        jb = j - 1;
// Зменшуємо допоміжну змінну на одиницю, щоб вона дорівнювала jb.
        j--;
// Збільшуємо довжину пройденого шляху на одиницю (пройдено одне ребро).
        l++;
        }
        else
// Якщо попередня вершина при пошуку в глибину лівіша, переходимо до неї.
        j--;
        }
        }
```

2.5. Обгрунтування та організація вхідних та вихідних даних програми

Вхідними даними є обрані та задані користувачем початкові дані: категорії та точки графа (місця для відвідування).

Вихідними є дані побудованого маршруту та виконані дії додатку.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний вироб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки.

Для роботи системи можуть використовуватись різні типи комп’ютерів чи ноутбуків під управлінням ОС Windows та різних типів і характеристик.

Програма розроблена для робочих станцій під управлінням ОС Windows написана мовою програмуванням C# у середовищі Visual Studio 2019.

Згідно з довідкою розробника середовища, фірмою Microsoft, розміщеною на офіційному сайті для використання Visual Studio 2019 встановлено наступні системні вимоги Visual Studio 2019:

Середа Visual Studio 2019 підтримується на наступних операційних системах:

- Windows 7 з Service Pack 1;
- Windows 8.1;
- Windows 10.

Дану систему було розроблено та протестовано на нутбуці Acer Aspire 5 A515-54G з такими характеристиками:

- Діагональ екрана: 15.6" (1920x1080) Full HD.
- Процесор: Чотириядерний Intel Core i5-8265U (1.6 - 3.9 ГГц).
- Частота оновлення екрана: 60 Гц.

– Обсяг оперативної пам'яті: 8 ГБ.

– Покоління процесора Intel: 8-е Whiskey Lake.

– Кількість слотів M.2: 1.

– Кількість відсіків 2.5" SATA для HDD/SSD: 1.

– Обсяг накопичувача: 256 ГБ SSD.

– Кількість слотів для оперативної пам'яті: 1.

– Тип оперативної пам'яті: DDR4-2400 МГц.

– Додаткові можливості: Вебкамера HD, Вбудований мікрофон, Вбудовані динаміки.

– Графічний адаптер: Дискретний, nVidia GeForce MX250, 2 ГБ виділеної пам'яті GDDR5.

– Мережеві адаптери: Wi-Fi 802.11 ac, Bluetooth 5.0, Gigabit Ethernet.

– Роз'єми та порти введення-виведення: 1 x USB 3.1 Type-C/2 x USB 3.1 Gen1/1 x USB 2.0/HDMI/LAN (RJ-45)/комбінований аудіороз'єм для навушників/мікрофона.

2.6.2. Використані програмні засоби

Програму створено в середовищі програмування Microsoft Visual Studio 2019 мовою програмування C#.

Проект розроблено для робочих станцій на базі персональних комп’ютерів чи ноутбуків під управлінням ОС Windows.

Програма є невимогливою до програмних засобів.

2.6.3. Виклик та завантаження програми

Спосіб виклику програми з відповідного носія даних та умови його завантаження є стандартними для запуску виконуючих файлів при роботі в ОС Windows. Додаткових чи специфічних вимог щодо запуску програми не встановлено, програма не потребує спеціального завантаження та

налаштування і може бути скопійована на інші носії інформації.

Для роботи потрібен ПК чи наутбук з ОС Windows. Щоб установити програму на комп'ютер, треба скопіювати папку «Graph» та відкрити файл graph.exe.

2.6.4. Опис інтерфейсу користувача

Щоб запустити програму треба відкрити відповідний виконуваний файл graph.exe. Після запуску буде відкрито головну форму додатку (рис. 2.9.)

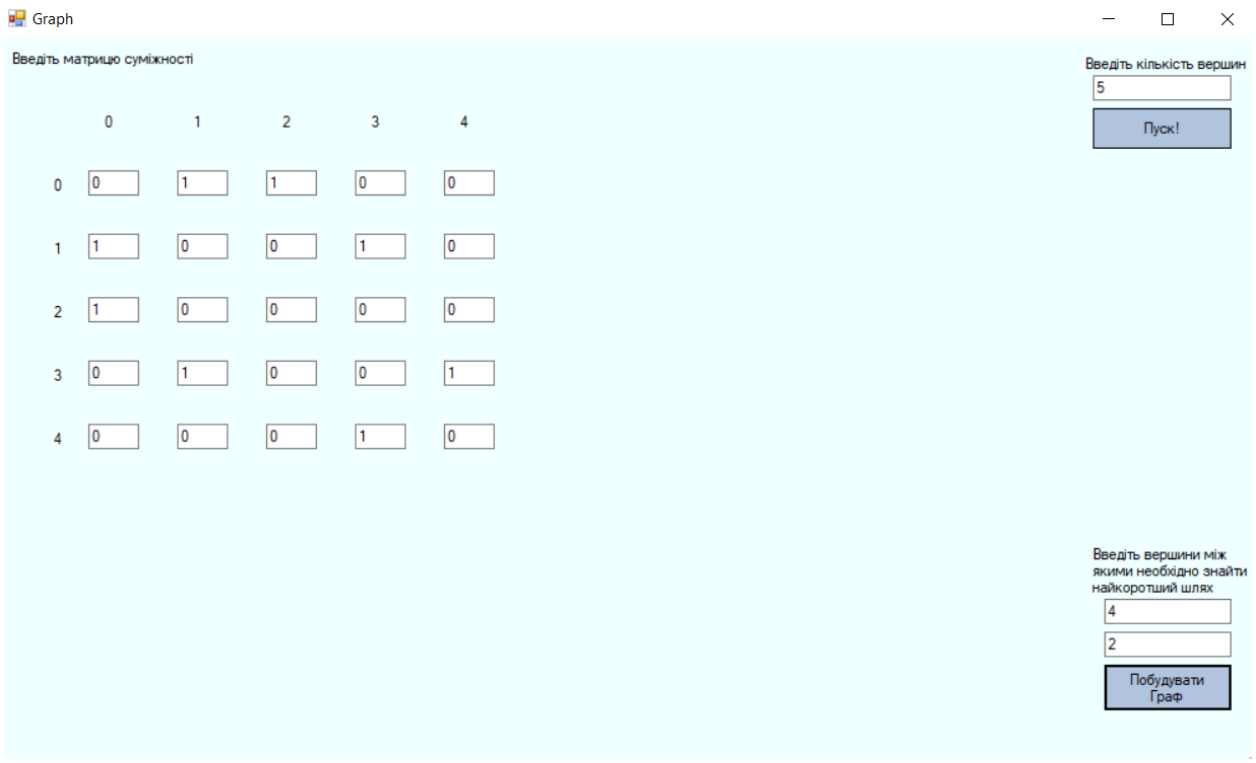

Рис. 2.9. Головна форма системи

Після заповнення відповідних полів, та запуску (кнопка «Пуск»), в окремому вікні будується граф с заданими характеристиками (рис. 2.10).

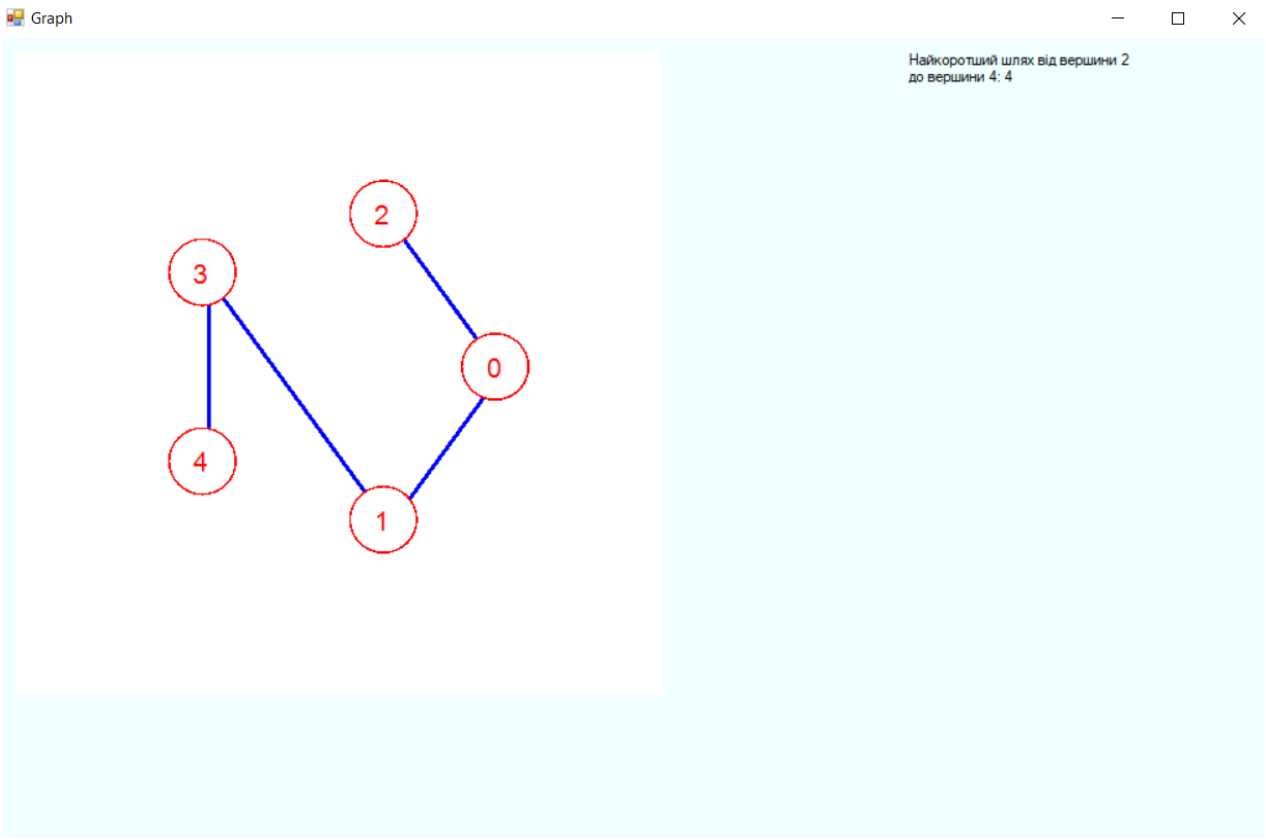

Рис. 2.10. Форма відображення графа

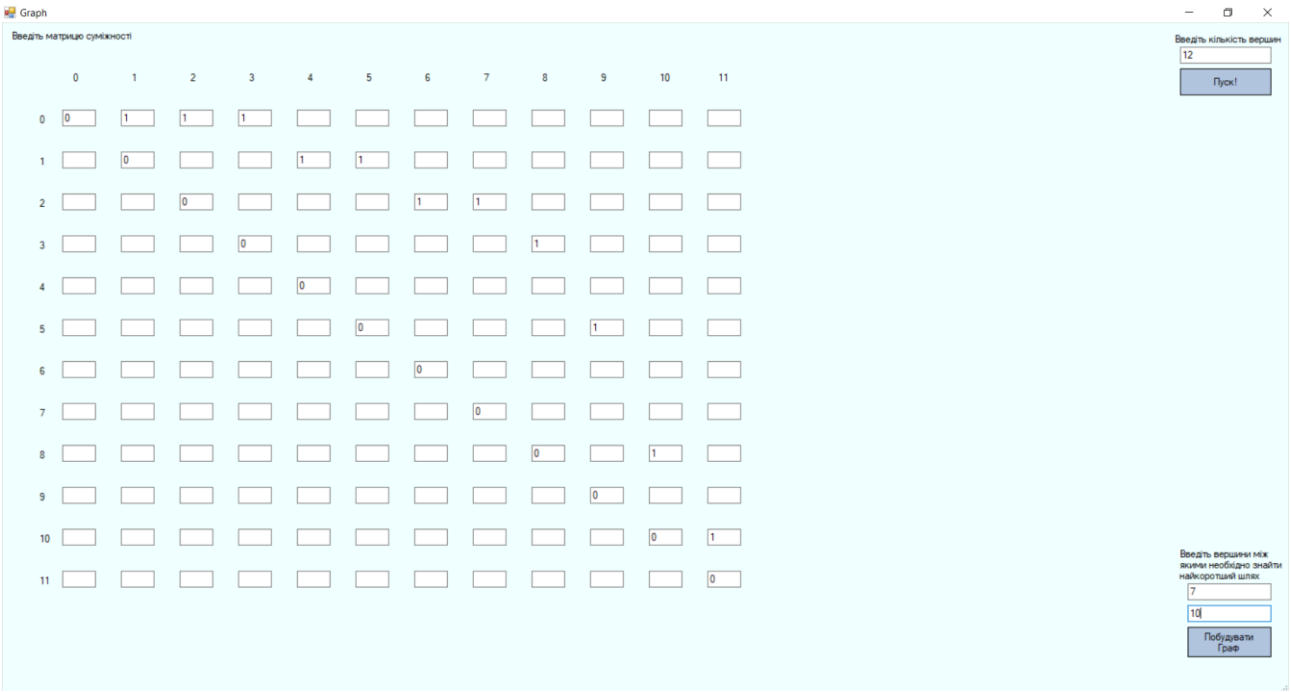

Рис.2.11. Форма введення початкових даних

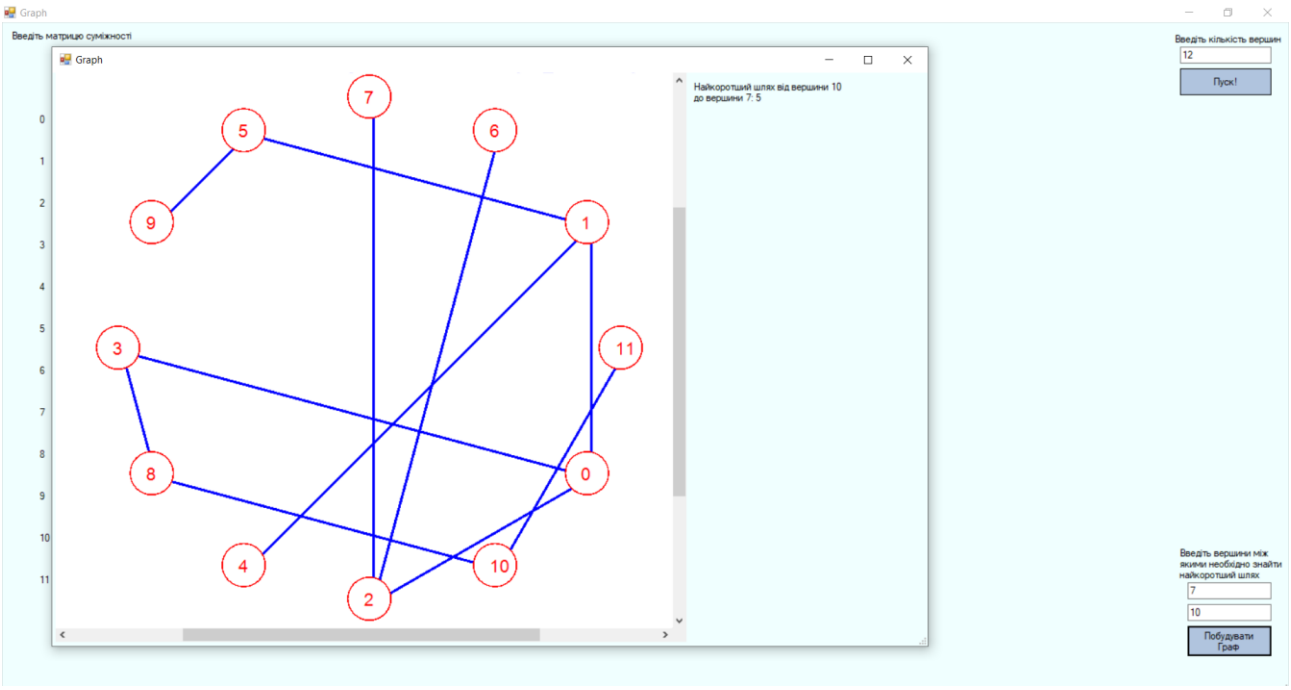

Рис. 2.12. Форма побудови графа

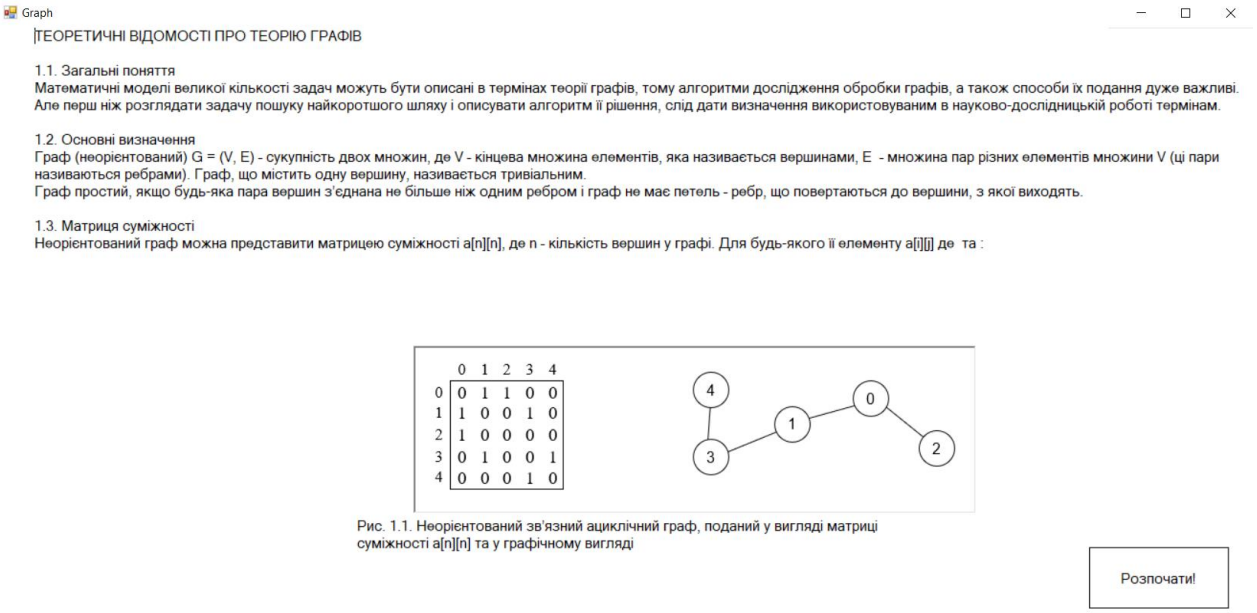

Рис. 2.13. Форма з методичними вказівками

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1300;
2. коефіцієнт складності програми – 1,6;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста– 60грн/год;
5. коефіцієнтзбільшеннявитратпрацівнаслідокнедостатньогоопи сузадачі – 1,2;
6. коефіцієнткваліфікаціїпрограміста, обумовленийвід стажу роботи з даноїспеціальності – 1,2;
7. вартість машино-години ЕОМ –13грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1300);

C - коефіцієнт складності програми (1,6);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}6 \cdot 1300 \cdot (1 + 0{,}05) = 2184$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\ \text{, людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2184 \cdot 1{,}2) / (75 \cdot 1{,}2) = 29{,}12 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин,}\quad (3.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2184 / (20 \cdot 1{,}2) = 91 \text{людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

$$t_n = 2184 / (20 \cdot 1{,}2) = 91 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин.}$$

$$t_{отл} = 2184 / (5 \cdot 1{,}2) = 364 \text{ чел.-ч.}$$

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}\ \text{, людино-годин.}$$

$$t^{k}_{отл} = 1,5 \cdot 364 = 546 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,}$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 \cdot t_{др} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 2184 / (18 \cdot 1,2) = 101,11 \text{ людино-годин.}$$

$$t_{до} = 0,75 \cdot 101,11 = 75,83 \text{ людино-годин.}$$

$$t_{д} = 101,11 + 75,83 = 176,94 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 29,12 + 91 + 91 + 364 + 176,94 = 802,06 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $K_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 60 грн / год, отримуємо:

$$З_{ЗП} = 802{,}06 \cdot 60 = 48123{,}87 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн,} \quad (3.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 364 \cdot 13 = 4732 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 48123{,}87 + 4732 = 52855{,}87 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k- число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$T = 802{,}06 / 1 \cdot 176 \approx 4{,}5 \text{ міс.}$$

Висновок: в роботі розроблена програма з метою підвищення ефективності методів самостійної роботи з графами та побудови оптимального маршруту у ньому. Вартість даного програмного забезпечення становить 52 тис. грн. і не вимагає додаткових витрат як при розробці програми. Очікуваний час розробки становить 4,5 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

Теорія графів використовується в різних галузях знань та має широке прикладне застосування від візуалізації, наочного представлення об’єктів та зв’язків між ними, до створення, формалізації, аналізу та перетворення моделей. З розвитком різних сфер діяльності окремих людей та суспільства в цілому, для аналізу і вирішення різних задач доцільно використовувати вже існуючий опрацьований математичний апарат теорії графів.

У роботі запропоновано алгоритм обчислення оптимального шляху між двома заданими вершинами в неорієнтованому ациклічному зв’язному незваженому графі. Алгоритм базується на порівнянні рівнів, на яких знаходяться вершини, між якими відбувається пошук оптимального шляху.

Запропонований алгоритм працює для усіх зв’язних неорієнтованих ациклічних незважених графів та виглядає наступним чином:

- Визначаємо вершини, між якими шукаємо шлях.
- Якщо вони на різних рівнях дерева і не суміжні, переходимо від тої, що належить до нижчого рівня (назвемо її другою з пари) до найближчої вершини рівня, до якого належить перша. При цьому кількість пройдених ребр додаємо до шляху.
- Далі від кожної з вершин робимо в кожній ітерації крок у бік кореневої, до найближчої вершини рівня з меншим номером. На кожному такому кроці додаємо одиниці до пройденого шляху для обох вершин.
- Цей рух припиниться, якщо шляхи у бік кореня від обох вершин зустрінуться в одній вершині.

Систематизовано теоретичні відомості теорії графів, розуміння яких вимагає розв’язання запропонованої задачі оптимального шляху на ациклічних неорієнтованих зв’язних незважених графах.

Представлений опис алгоритмів на мові C++, що містить досить деталей, щоб забезпечити майже пряму реалізацію на будь-якій мові програмування. Запропонований метод пошуку оптимального шляху на графі G = (V, E),

представленому у вигляді матриці суміжності, реалізовано у програмному коді та розібрано для можливості подальших досліджень та вдосконалення алгоритму.

Програму створено в середовищі програмування Microsoft Visual Studio 2019 мовою C#. Для пошуку інформації використано Google Chrome.

Під час написання програми були використані методи об’єктно-орієнтованого підходу.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту (802 люд-год), проведений підрахунок вартості роботи по створенню програми (52855грн) та розраховано час на його створення (4,5 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rakip Ismail Karas and Umit Atila. A Genetic Algorithm Approach for Finding the Shortest Driving Time on Mobile Devices‖, Scientific Research and Essays. 2011. Vol. 6(2), pp. 394-405. URL: http://www.academicjournals.org/SRE

2. Алексеев В.Е., Таланов В.А. Глава 3.4. Нахождения кратчайших путей в графе // Графы. Модели вычислений. Структуры данных – Нижний Новгород: Изд-во ННГУ. 2005. С. 236 – 237. – 307 с.

3. Галкина В.А. Глава 4. Построение кратчайших путей в ориентированном графе // [www.biblion.ru/product/257960/ Дискретная математика. Комбинаторная оптимизация на графах]. – Москва: Издательство Гелиос АРВ, 2003. С. 75 –94. – 232 с.

4. Xi C., Qi F., and Wei L. A New Shortest Path Algorithm based on Heuristic Strategy, Proc. of the 6th World Congress on Intelligent Control and Automation. 2006. Vol. 1, pp. 2531 – 2536. URL: https://ieeexplore.ieee.org/document/1712818

5. Hasan B.S., Khamees M.A., and Mahmoud A.S.H. A Heuristic Genetic Algorithm for the Single Source Shortest Path Problem, // Proc. of International Conference on Computer Systems and Applications. 2007. pp.187 – 194. URL: https://ieeexplore. ieee.org/document/4230957

6. Li T., Qi L., and Ruan D. An Efficient Algorithm for the Single-Source Shortest Path Problem in Graph Theory. / Proc. of 3rd International Conference on Intelligent System and Knowledge Engineering. 2008. Vol. 1, pp. 152 – 157. URL: https:// ieeexplore.ieee.org/document/4730916

7. Cherkassky B. V., Goldberg A. V., Radzik T. Shortest paths algorithms: Theory and experimental evaluation // Math. Prog. Springer-Verlag, 1996. Vol. 73, Iss. 2. pp. 129 – 174. URL: https://link.springer.com/article/10.1007/BF02592101

8. Окулов С. М. Программирование в алгоритмах. М.: БИНОМ. Лаборатория знаний. 2002. 341 с.

9. Алгоритмы: построение и анализ / Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. / Москва, Санкт-Петербург, Киев. 2005. 1292 с.

10. Практика и теория программирования. Книга 2. / Винокуров Н.А., Ворожцов А.В. М.: Физматкнига. 2008. 288 с.

11. Евстигнеев В.А. Применение теории графов в программировании. / Под ред. Ершова А.П. М.: Главная редакция физ.-мат. лит-ры. 1985. 350 с.

12. Дискретна математика. Комплекс навчально-методичного забезпечення для студентів за напрямом 125 «Кібербезпека» / Укл.: Ротаньова Н.Ю. – Маріуполь: МДУ, 2016. – 19 с.

13. Дискретна математика. Навчальна програма дисципліни для студентів галузі знань 0501 Інформатика та обчислювальна техніка напряму підготовки 6.050101 Комп'ютерні науки факультету електроніки. – Львів: ЛНУ імені Івана Франка, 2012. – 6 с.

14. Иванов Б.Н. Дискретная математика. Алгоритмы и программы. Полный курс. − М.: Физматлит, 2007. – 408 с.

15. Кривий С.Л. Курс дискретної математики. Навчальний посібник.– К:Наукова думка,2007. – 432 с.

16. Кушнерьов О. Про деякі застосування теорії графів [Текст] / О. Кушнерьов // Фізико-математична освіта : збірник наукових праць. – Суми : Вид-во фізико-математичного факультету СумДПУ імені А.С. Макаренка, 2015. – № 1 (7). – С. 50–56.

17. Матвієнко М. П. Дискретна математика Навч. посібник.– К.: «ВидавництвоЛіра-К», 2013. – 324 с.

18. Нікольський Ю. В., Пасічник В., Щербина Ю. М. Дискретна математика: Підручник. – Львів: «Магнолія – 2006», 2009. – 432 с.

19. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов. − 2-е изд. − СПб.: Питер, 2007. − 364 с.

20. Трохимчук Р.М. Теорія графів. Навчальний посібник для студентів факультету кібернетики / Трохимчук Р.М. К.: РВЦ «Київський університет», 1998. – 43 с.

21. Агуров Павел. C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.

22. Албахари Джозеф. C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2013. - 944 с.

23. Альфред В. Ахо. Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.

24. Бишоп Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.

25. Вагнер Билл. C# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 с.

26. Зиборов Виктор. Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.

27. Ишкова Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.

28. Лотка Рокфорд. C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

29. Мак-Дональд Мэтью. Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

30. Подбельский В. В. Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

31. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 с.

32. Троелсен Эндрю. Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 с.

ДОДАТОК А

КОД ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Graph
{
    class Class1
    {
            public static int[][] g =
RectangularArrays.RectangularIntArray(DefineConstants.MAX,
DefineConstants.MAX);
            private  int[] used = new int[DefineConstants.MAX];
            private  int[] dist = new int[DefineConstants.MAX];
            private  int[] up = new int[DefineConstants.MAX];
            private  int n;
            private  int a;
            private  int b;
            private  int i;
            private  int ja;
            private  int jb;
            private  int l;
            private  int j = 0;
            // пошук у глибину з вершини v (корінь дерева, тому v = 0)
            private  void dfs(int v, int len = 0)
            {
                  // помічаємо вершину v як пройдену та вказуємо шлях до цієї
вершини від кореня дерева
                  used[v] = 1;
                  dist[v] = len;
                  up[j] = v;
                  // запам'ятовуємо номери вершин, між якими потрібно знайти
оптимальний шлях, при застосуванні пошуку в глибину
                  if (v == a)
                  {
                        ja = j;
                  }
                  if (v == b)
                  {
                        jb = j;
                  }
                  j++; // лічильник
```

```
                        // шукаємо ребро, по якому можна потрапити в ще
непройдену вершину i
                for (int i = 0; i < n; i++)
                {
                        if ((g[v][i] == 1) && (used[i] == 0))
                        {
                                dfs(i, len + 1); // рекурсивний запуск пошуку в
глибину з даної вершини
                        }
                }
        }
        // процедура кроку від обох вершин у бік кореня
        private  void uplevel(int a, int b, int ja, int jb) //вхідні дані з
процедури tolevel
        {
                // Застосовуємо допоміжні змінні, щоб не втратити номери
шуканих вершин при пошуку в глибину.
                int i = ja;
                int j = jb;
                // Поки шляхи не зійдуться в одній вершині, виконуємо команди
циклу.
                while (a != b)
                {
                        // Порівнюємо рівні заданої вершини та вершини, що
передувала їй при пошуку в глибину (вона може бути або лівіша, або вища).
                        if (dist[a] > dist[up[i - 1]])
                        {
                                // Якщо вища, то переходимо до неї.
                                a = up[i - 1];
                                // Це номер нової вершини, до якої перейшли при
пошуку в глибину.
                                ja = i - 1;
                                // Зменшуємо допоміжну змінну на одиницю (щоб
вона дорівнювала ja).
                                i--;
                                // Збільшуємо загальний пройдений шлях на одиницю
(пройдено одне ребро).
                                l++;
                        }
                        else
                        {
                                // Якщо вершина, яка передувала поточній при
пошуку в глибину, лівіша, переходимо до цієї вершини.
                                i--;
                        }
                        // Порівнюємо рівні другої заданої вершини та вершини,
попередньої поточній у пошуку в глибину (вона може буди або лівіша або вища).
                        // Доки вершини a і b не будуть на одному рівні,
виконуємо наступні дії.
```

```
                        while (dist[a] != dist[b])
                        {
                              if (dist[b] > dist[up[j - 1]])
                              {
                                    // Якщо попередня вершина при пошуку в
глибину вища, переходимо до неї.
                                    b = up[j - 1];
                                    // Це номер при пошуку в глибину нової
вершини (до якої перейшли).
                                    jb = j - 1;
                                    // Зменшуємо допоміжну змінну на одиницю,
щоб вона дорівнювала jb.
                                    j--;
                                    // Збільшуємо довжину пройденого шляху на
одиницю (пройдено одне ребро).
                                    l++;
                              }
                              else
                              {
                                    // Якщо попередня вершина при пошуку в
глибину лівіша, переходимо до неї.
                                    j--;
                              }
                        }
                  }
            }
            // Процедура переходу від нижчої вершини до рівня вищої.
            private  void tolevel(int a, int b, int ja, int jb)
            {
                  // Застосовуємо допоміжні змінні, щоб не втратити номери
шуканих вершин при пошуку в глибину.
                  int i = ja;
                  int j = jb;
                  // Доки не перейдемо від нижчої вершини до рівня вищої,
виконуємо наступні дії.
                  while (dist[a] != dist[b])
                  {
                        if (dist[a] < dist[b]) // Визначаємо яка з вершин
нижча.
                        {
                              // Якщо нижча b, порівнюємо її рівень з рівнем
вершини, що передувала ній при пошуку в глибину (вища або лівіша).
                              if (dist[b] > dist[up[j - 1]])
                              {
                                    // Якщо вища, переходимо до неї.
                                    b = up[j - 1];
                                    // номер при пошуку в глибину нової вершини
(до якої перейшли).
                                    jb = j - 1;
```

```
                                    // Зменшуємо допоміжну змінну на одиницю
(щоб вона дорівнювала jb).
                                    j--;
                                    // Збільшуємо довжину пройденого шляху на
одиницю (пройдено одне ребро.)
                                    l++;
                              }
                              else
                              {
                                    // Якщо вершина, що передувала поточній при
пошуку в глибину, лівіша, то беремо попередню вершину, при пошуку в глибину
                                    j--;
                              }
                        }
                        // Якщо нижча вершина a, виконуємо аналогічні дії для
вершини a.
                        else
                        {
                              // Якщо нижча a, порівнюємо її рівень з рівнем
вершини, яка передувала ній при пошуку в глибину (вища або лівіша).
                              if (dist[a] > dist[up[i - 1]])
                              {
                                    // Якщо вища, переходимо до неї
                                    a = up[i - 1];
                                    // Це номер при пошуку в глибину нової
вершини (до якої перейшли).
                                    ja = i - 1;
                                    // Зменшуємо допоміжну змінну на одиницю
(щоб вона дорівнювала ja).
                                    i--;
                                    // Збільшуємо довжину пройденого шляху на
одиницю (пройдено одне ребро).
                                    l++;
                              }
                              else
                              {
                                    // Якщо вершина, що передувала поточній при
пошуку в глибину, лівіша, то беремо попередню вершину, при пошуку в глибину
                                    i--;
                              }
                        }
                  }
                  // Запускаємо процедуру одночасного пересування від двох
вершин у бік кореня.
                  uplevel(a, b, ja, jb);
            }
            internal  class DefineConstants
            {
                  public const int MAX = 1000;
```

```
            }
            internal  class RectangularArrays
            {
                  public static int[][] RectangularIntArray(int size1, int
size2)
                  {
                        int[][] newArray = new int[size1][];
                        for (int array1 = 0; array1 < size1; array1++)
                        {
                              newArray[array1] = new int[size2];
                        }

                        return newArray;
                  }
            }
            public int ain(int c, int[][] arr)
            {
                  for (int i0 = 0; i0 < 1000; i0++)
                        for (int j0 = 0; j0 < 1000; j0++)
                              g[i0][j0] = 0;
                  n = c;
                  // Описуємо граф матрицею суміжності.
                  for (i = 0; i < c - 1; i++)
                  {
                        a = arr[i][0];
                        b = arr[i][1];
                        g[a][b] = g[b][a] = 1; // між вершинами a і b є ребро
                  }
                  // Вводимо номери вершин, між якими потрібно знайти
оптимальний шлях.
                  a = arr[c - 1][0];
                  b = arr[c - 1][1];
                  // Запускаємо пошук у глибину від кореня дерева.
                  dfs(0);
                  // Запускаємо процедуру переходу від нижчої вершини до рівня
вищої.
                  tolevel(a, b, ja, jb);
                  //Виводимо оптимальний шлях.
                  return 1;
            }
      }
}



namespace Graph
{
    partial class Form1
```

```
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен
быть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.button2 = new System.Windows.Forms.Button();
            this.panel1 = new System.Windows.Forms.Panel();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.textBox3 = new System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.button1.BackColor = System.Drawing.Color.LightSteelBlue;
            this.button1.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.button1.ForeColor = System.Drawing.SystemColors.ControlText;
            this.button1.Location = new System.Drawing.Point(863, 55);
            this.button1.Name = "button1";
```

```
            this.button1.Size = new System.Drawing.Size(109, 32);
            this.button1.TabIndex = 1;
            this.button1.Text = "Пуск!";
            this.button1.UseVisualStyleBackColor = false;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // button2
            //
            this.button2.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom
| System.Windows.Forms.AnchorStyles.Right)));
            this.button2.BackColor = System.Drawing.Color.LightSteelBlue;
            this.button2.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.button2.Location = new System.Drawing.Point(872, 494);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(100, 36);
            this.button2.TabIndex = 2;
            this.button2.Text = "Побудувати Граф";
            this.button2.UseVisualStyleBackColor = false;
            this.button2.Visible = false;
            this.button2.Click += new System.EventHandler(this.button2_Click);
            //
            // panel1
            //
            this.panel1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
            | System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.panel1.AutoScroll = true;
            this.panel1.Location = new System.Drawing.Point(12, 29);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(838, 533);
            this.panel1.TabIndex = 3;
            //
            // textBox1
            //
            this.textBox1.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.textBox1.Location = new System.Drawing.Point(863, 29);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(109, 20);
            this.textBox1.TabIndex = 4;
            //
            // textBox2
            //
```

```
            this.textBox2.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom
| System.Windows.Forms.AnchorStyles.Right)));
            this.textBox2.Location = new System.Drawing.Point(872, 468);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(100, 20);
            this.textBox2.TabIndex = 5;
            this.textBox2.Visible = false;
            //
            // textBox3
            //
            this.textBox3.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom
| System.Windows.Forms.AnchorStyles.Right)));
            this.textBox3.Location = new System.Drawing.Point(872, 442);
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(100, 20);
            this.textBox3.TabIndex = 6;
            this.textBox3.Visible = false;
            //
            // label1
            //
            this.label1.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(854, 13);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(134, 13);
            this.label1.TabIndex = 7;
            this.label1.Text = "Введіть кількість вершин";
            //
            // label2
            //
            this.label2.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom
| System.Windows.Forms.AnchorStyles.Right)));
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(860, 400);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(132, 39);
            this.label2.TabIndex = 8;
            this.label2.Text = "Введіть вершини між \r\nякими необхідно знайти
\r\nнайкоротший шлях";
            this.label2.Visible = false;
            //
            // label3
            //
            this.label3.AutoSize = true;
```

```
            this.label3.Location = new System.Drawing.Point(9, 9);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(153, 13);
            this.label3.TabIndex = 9;
            this.label3.Text = "Введіть матрицю суміжності ";
            this.label3.Visible = false;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.Azure;
            this.ClientSize = new System.Drawing.Size(993, 574);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.panel1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Graph";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox3;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
    }
}



namespace Graph
{
    partial class Form2
    {
        /// <summary>
```

```
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.panel1 = new System.Windows.Forms.Panel();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.label1 = new System.Windows.Forms.Label();
            this.panel1.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
            this.SuspendLayout();
            //
            // panel1
            //
            this.panel1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
            | System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.panel1.AutoScroll = true;
            this.panel1.Controls.Add(this.pictureBox1);
            this.panel1.Location = new System.Drawing.Point(4, 0);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(693, 625);
            this.panel1.TabIndex = 0;
            //
            // pictureBox1
```

```
            //
            this.pictureBox1.Location = new System.Drawing.Point(12, 10);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(667, 599);
            this.pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.AutoSize;
            this.pictureBox1.TabIndex = 0;
            this.pictureBox1.TabStop = false;
            //
            // label1
            //
            this.label1.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(703, 10);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(35, 13);
            this.label1.TabIndex = 1;
            this.label1.Text = "label1";
            //
            // Form2
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.Azure;
            this.ClientSize = new System.Drawing.Size(985, 630);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.panel1);
            this.Name = "Form2";
            this.Text = "Graph";
            this.Load += new System.EventHandler(this.Form2_Load);
            this.panel1.ResumeLayout(false);
            this.panel1.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Label label1;
    }
}
```

```
namespace Graph
{
    partial class Form3
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form3));
            this.richTextBox1 = new System.Windows.Forms.RichTextBox();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.label1 = new System.Windows.Forms.Label();
            this.button1 = new System.Windows.Forms.Button();

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
            this.SuspendLayout();
            //
            // richTextBox1
            //
            this.richTextBox1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
```

```
            | System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.richTextBox1.BackColor = System.Drawing.Color.White;
            this.richTextBox1.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.richTextBox1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 11.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.richTextBox1.Location = new System.Drawing.Point(54, -1);
            this.richTextBox1.Margin = new System.Windows.Forms.Padding(4, 4,
4, 4);
            this.richTextBox1.Name = "richTextBox1";
            this.richTextBox1.ReadOnly = true;
            this.richTextBox1.Size = new System.Drawing.Size(1484, 402);
            this.richTextBox1.TabIndex = 0;
            this.richTextBox1.Text = resources.GetString("richTextBox1.Text");
            this.richTextBox1.TextChanged += new
System.EventHandler(this.richTextBox1_TextChanged);
            //
            // pictureBox1
            //
            this.pictureBox1.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
            this.pictureBox1.Image =
((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(583, 409);
            this.pictureBox1.Margin = new System.Windows.Forms.Padding(4, 4, 4,
4);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(588, 175);
            this.pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.AutoSize;
            this.pictureBox1.TabIndex = 1;
            this.pictureBox1.TabStop = false;
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif",
11.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label1.Location = new System.Drawing.Point(500, 628);
            this.label1.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(722, 48);
            this.label1.TabIndex = 3;
            this.label1.Text = "Рис. 1.1. Неорієнтований зв’язний ациклічний
граф, поданий у вигляді матриці \r\ncy" +
    "міжності a[n][n] та у графічному вигляді\r\n";
```

```
            //
            // button1
            //
            this.button1.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.button1.Font = new System.Drawing.Font("Microsoft Sans Serif",
11.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.button1.Location = new System.Drawing.Point(1524, 667);
            this.button1.Margin = new System.Windows.Forms.Padding(4, 4, 4, 4);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(195, 79);
            this.button1.TabIndex = 4;
            this.button1.Text = "Розпочати!";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // Form3
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.White;
            this.ClientSize = new System.Drawing.Size(1757, 768);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.pictureBox1);
            this.Controls.Add(this.richTextBox1);
            this.Margin = new System.Windows.Forms.Padding(4, 4, 4, 4);
            this.Name = "Form3";
            this.Text = "Graph";
            this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.Form3_FormClosed);

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.RichTextBox richTextBox1;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button1;
    }
}
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему: «Розробка навчальної інформаційної системи для пошуку оптимальних шляхів на зв'язних неорієнтованих ациклічних графах»
студента групи 122-18ск-1 Єрастова Едуарда Юрійовича

Керівник економічного розділу
Зав. каф. ПЕП та ПУ, д.е.н. **О. Г. Вагонова**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Диплом.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом.pdf	Пояснювальна записка до кваліфікаційної роботив форматі PDF.
Програма	
program.zip	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація.ppt	Презентація кваліфікаційної роботи.