Київський національний університет будівництва i архітектури

Кафедра інформаційних технологій проектування та прикладної математики

Спеціальність: Комп’ютерна інженерія

Курс І Група КСМ-22с Семестр 1

**ЗАВДАННЯ**

**на курсову роботу студентки**

1.Тема роботи «Алгоритми на графах. Алгоритм Дейкстри»

2.Термiн здачі студентом закінченої роботи: \_\_\_\_\_\_

3.Вихiднi дані до роботи: кількість шляхів та відстані між ними

4.Змiст пояснювальної записки перелік питань, які належить розробити:

1.Вступна частина;

2.Математичний опис розв’язку задачі;

3.Схема алгоритму;

4.Програмна реалізація алгоритму.

5.Перелiк графічного матеріалу (з точним зазначенням обов’язкових

креслень): Схема алгоритму

6. Дата видачі завдання “15” вересня 2022 р.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Керiвник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпис) (пiдпис)

Київ 2022 рік

# Зміст

 Вступ ……………………………………………………………………………… 3

1. Постановка задачі ……………………………………………………………… 4
2. Теоретичні відомості …………………………………………………………...4
3. Вхідні та вихідні дані …………………………………………………………...8
4. Математична постановка та опис розв’язку задачі …………………………...8
5. Алгоритм розв’язку задачі………………………………………………………9
	1. Схема алгоритму ………………………………………………………..9
	2. Оцінка складності алгоритму …………………………………………12
6. Програмна реалізація алгоритму …………………………………………….. 13
	1. Загальні відомості мови програмування …………………………… 13
	2. Опис процедур і функції програми …………………………………. 14
	3. Контрольний приклад ……………………………………………….. 15
		1. Приклад розрахований вручну …………………………….. 15
		2. Результат роботи програми ………………………………... 20

Висновок …………………………………………………………………………. 20

Рекомендовані джерела інформації ……………………...…………………….. 22

Додаток та код програми ……………………………………………………….. 23

**Вступ**

Теорія графів має широке застосування в моделюванні інформаційних процесів, у програмуванні й у вирішенні економічних завдань.

Перша згадка про неї зустрічається в роботах Ейлера, який у1736 році розв’язав задачу про кенігсберзькі мости.

У Кенігсберзі було два острови, з’єднаних сімома мостами з берегами річки Прегель та один із одним. Суть задачі полягала в пошуку маршруту проходження всіх чотирьох частин суші, який мав починатися на будь-якій частині суші та закінчуватися на ній же за умови проходження кожного моста по одному разу. Проте всі спроби знайти маршрут були невдалими.

Щоб довести не існування такого маршруту, Ейлер позначив кожну частину суші точкою (вершиною або вузлом), а кожен міст – лінією (ребром), що з’єднує відповідні точки, і отримав “граф”. Твердження про не існування маршруту тотожна неможливості деяким чином обійти граф. Виходячи з цього конкретного випадку, Ейлер узагальнив постановку задачі та знайшов критерій існування обходу.

Пізніше елементи теорії графів з’явилися в таких природничих науках, як географія, фізика, хімія, електротехніка. У загальному, графи використовуються в усіх галузях, де є елементи та зв’язки між ними, тому теорія графів є актуальним прикладним розділом математики.

У загальному випадку граф – це множина *V* точок, певним чином з'єднаних між собою лініями, необов'язково прямими. Точки множини *V* називаються *вершинами* графа, а лінії, які їх з’єднують –*ребрами*.

Вершини графа зазвичай нумерують десятковими числами, але можна використовувати і будь-які інші знаки. Якщо вершини пронумеровані, то ребра означають неупорядковані пари номерів вершин. Кожну пару утворюють номери тих вершин, які з'єднані ребром.

* + - 1. **Постановка задачі**

Дана мережа автомобільних доріг, що з'єднують міста Київської області. Знайти найкоротшу відстань від Києва до кожного міста області, якщо рухатись можна тільки по дорогах

* + - 1. **Теоретичні відомості**

**Теорія графів** ***-*** молода сфера дискретної математики. Однак методи теорії графів здобули визнання не лише серед математиків, а й інженерів, економістів, психологів, біологів та хіміків. Використання мови та методів теорії графів пришвидшує вирішення практичних задач, спрощуючи обчислення. Теорія графів є однією з важливих частин математичної системи інформатики та кібернетики. У контексті теорії графів сформульовано велику кількість задач, пов’язаних з дискретними об’єктами.

**Граф** — це група об’єктів із зв’язками між собою.

Об'єкти розглядаються як графіка, або графіка, вузли та з'єднання - як дуги або ребра. Для різних застосувань типи графіків можуть відрізнятися щодо орієнтації, обмежень кількості з'єднань та додаткових даних по кутах або краях.

**Граф** або **неорієнтований граф** G {\displaystyle G} — це [впорядкована пара](https://uk.wikipedia.org/wiki/%D0%92%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BF%D0%B0%D1%80%D0%B0) {\displaystyle G:=(V,E)}G:=(V, E), для якої виконуються такі умови:

* {\displaystyle V}V — [множина](https://uk.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0%22%20%5Co%20%22%D0%9C%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0) **вершин** або **вузлів**,
* {\displaystyle E}E — множина пар (у випадку неорієнтованого графу — невпорядкованих) вершин з {\displaystyle V}V, які називаються **ребрами**.

Граф, який містить тільки ребра називається ***неорієнтованим***, який містить тільки дуги — ***орієнтованим***. Граф, що має як ребра так і дуги, називається ***мішаним*.**



Рисунок 2.1 – Орієнтований граф з трьома вершинами і трьома ребрами



Рисунок 2.2 - Простий не орієнтований граф. Кожна вершина має степінь два, тому це буде регулярний граф.

Іноді є потреба пару вершин з'єднати більше, ніж одним ребром. Ребра чи дуги одного напряму, які з'єднують ту саму пару вершин, називаються [***кратними***](https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%82%D0%BD%D1%96_%D1%80%D0%B5%D0%B1%D1%80%D0%B0) (паралельними) ребрами.

Дуга чи ребро, що сполучає вершину саму зі собою називається ***петлею*.**

Граф без кратних дуг і петель називається ***простим*.**

Орграф, отриманий з [простого графа](http://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA_%D1%82%D0%B5%D1%80%D0%BC%D1%96%D0%BD%D1%96%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2#.D0.9F) орієнтацією ребер, називається **орієнтованим**. На відміну від останнього, у довільного **простого орграфа** дві вершини можуть з'єднуватися двома різноорієнтованими дугами.

Орієнтований [повний граф](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%BD%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) називається **турніром**.

**Цикл** — замкнутий ланцюг, для орграфів цикл називається контур. Цикл в орграфі — це простий шлях довжини не менше 1, котрий починається і закінчується в одній і тій самій вершині.

**Дерево** — зв'язний граф без циклів.

**[Маршрутом](http://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82_%28%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2%29)** орграфа називають послідовність вершин і *дуг*, виду

 (вершини можуть повторюватися).

**Довжина маршруту** — кількість дуг у ньому.

**Шлях** — *маршрут* орграфа без повторюваних дуг, **простий шлях** — без повторюваних вершин. Якщо існує шлях з однієї вершини в іншу, то друга вершина досяжна з першої.

**Контур** — замкнений *шлях*.

Для **напівмаршруту** знімається обмеження на напрямок дуг, аналогічно визначаються **напівшлях** і **напівконтур**.

Орграф **сильно зв'язний**, або просто **сильний**, якщо всі його вершини взаємно *досяжні*;

**Односторонньо зв'язний**, або просто односторонній якщо для будь- яких двох вершин, принаймні одна досяжна з іншою;

**Слабо зв'язний**, або просто **слабкий**, якщо при ігноруванні напрямів дуг виходить зв'язний граф;

**Конденсацією** орграфа D називають орграф D\*, вершинами якого служать сильні компоненти D, а дуга в D\* показує наявність хоча б однієї дуги між вершинами, що входять у відповідні компоненти.

**Матриця суміжності** — один із способів представлення [графа](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) у вигляді матріці.

Матриця суміжності графа G зі скінченною кількістю вершин n (пронумерованих числами від 1 до n) — це квадратна матриця A розміру n в якій значення елементу aij рівне числу ребер з i-ї вершини графа в j-у вершину.

Матриця суміжності простого графу (що не містить петель і кратних ребер) є бінарною матрицею і містить нулі на головній діагоналі.



Рисунок 2.3 – Матриця суміжності

Іноді, особливо у разі неорієнтованого графа, петля (ребро з i-ї вершини в саму себе) вважається за два ребра, тобто значення діагонального елементу *aii*в цьому випадку рівне подвоєному числу петель навколо i-ї вершини.

Матриця суміжності простого графа є [бінарною матрицею](http://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BD%D0%B0%D1%80%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F) і містить нулі на головній діагоналі.

**Орієнтований (направлений) ациклічний граф** — випадок [орієнтованого](http://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA_%D1%82%D0%B5%D1%80%D0%BC%D1%96%D0%BD%D1%96%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2%22%20%5Cl%20%22.D0.9E) графа, в якому відсутні *орієнтовані [цикли](http://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA_%D1%82%D0%B5%D1%80%D0%BC%D1%96%D0%BD%D1%96%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2%22%20%5Cl%20%22.D0.A6)*, тобто [шляхи](http://uk.wikipedia.org/wiki/%D0%A8%D0%BB%D1%8F%D1%85_%28%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2%29), що починаються і закінчуються в одній і тій самій вершині. Орієнтований ациклічний граф є узагальненням [дерева](http://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2%29) (точніше, їх об'єднання — *лісу*).

**Алгоритм Дейкстри**

Найефективнішим алгоритмом знаходження довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої є алгоритм, запропонований 1959 р. нідерландським математиком Е. Дейкстрою (Е. Dijkstra).

Цей алгоритм застосовується лише у випадку, коли **вага кожної дуги додатня.** Нехай G=(V,E) – орієнтований граф, w(vi ,vj ) – вага дуги (vi ,vj ) .

Пошук мінімального шляху здійснюється за допомогою присвоювання вершинам міток. Мітки є двох типів - тимчасові й постійні. Вершини з постійними мітками групують у множину М, яку називають множиною позначених вершин. Решта вершин має тимчасові мітки, і множину таких вершин позначають через T (T=V\M). Величина постійної мітки вершини l(v) дорівнює довжині найкоротшого шляху від вершини *a* до вершини *v*. Якщо ж мітка тимчасова, то вона дорівнює довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.

* + - 1. **Вхідні та вихідні дані.**

В якості **вхідних даних** до даної задачі виступає матриця суміжності.

**Вихідними даними** матриця суміжності, на якій буде зображено найкоротші маршрути між точками, визначеними за алгоритмом Дейкстри.

* + - 1. **Математична постановка та опис розв’язку задачі**

**Пошук маршрутів:**

1. Дана матриця суміжності **а[N][N]**
2. За допомогою алгоритму Дейкстри перевіряємо найкоротшу довжину маршрутів.
3. Результат записуємо до **d [N][N]**

Кожну вершину з V можна порівняти з відомою мінімальною відстанню між A і цією вершиною. Алгоритм працює поетапно - на кожному кроці він виконує піковий "візит" і намагається зменшити тег. Алгоритм закінчується, коли всі вершини відвідані.

Ініціалізація: Мітка вершини приймається рівною 0, мітка інших вершин – нескінченності. Це відображає той факт, що відстань від А до інших вершин досі невідома. Усі вершини на графіку позначені як недоступні.

Кроки алгоритму: Якщо всі вершини відвідані, алгоритм завершується. В іншому випадку, з ще не відвіданих вершин вибирається вершина u, що має мінімальну мітку.

Ми розглядаємо всілякі маршрути, в яких u є передостаннім пунктом. Вершини, в які ведуть ребра з u, назвемо сусідами цієї вершини. Для кожного сусіда вершини u, крім зазначених як відвідані, розглянемо нову довжину шляху, рівну сумі значень поточної мітки u і довжини ребра, що з'єднує u з цим сусідом.

Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, позначимо вершину u як відвідану і повторимо крок алгоритму.

* + - 1. **Алгоритм розв’язку задачі**

 **5.1 Схема алгоритму**

Структура програми наведена на рис. 5.1. Схема алгоритму визначення min шляху наведена на рис. 5.2.

Початок

Вивід матриці

Вивід результату

Кінець

Ініціалізація матриці суміжності

Перевірка за алгоритмом Дейстри

Рисунок 5.1 - Структура програми

Початок

Зупинка циклу

Вивід min шляху

Кінець

Ініціалізація масиву

Найкоротший уже знайдений

Зрівняли вершини

While

i=1; i<; i++

d=0

i=1; i<; i++

Вивід найкоротших шляхів до вершини

i=k; i>=0; i--

 Новий шлях і вершина

min шляху не знайдено

так

так

ні

ні

Рисунок 5.2 – Алгоритм Дейкстри.

Початок

I<SIZE

a[i][i] = 0;

int j = i + 1

I<SIZE

a[i][j] = temp;

a[j][i] = temp;

int i = 0

I<SIZE

ні

 так

так

ні

 так

a[SIZE][SIZE]; d[SIZE]; v[SIZE]; int temp, int I, SIZE=8;

d[i] = 10000;

v[i] = 1;

ні

minindex = 10000;

min = 10000;

int i = 0

I<SIZE

v[i] == 1) && (d[i] < min

min = d[i];

minindex = i;

ні

так

ні

так

minindex != 10000

int i = 0

I<SIZE

ні

так

ні

так



Рисунок 5.3 - Схема алгоритму визначення min шляху

**5.2 Оцінка складності алгоритму**

Складність алгоритму Дейкстри залежить від методу пошуку вершини V, а також способу зберігання безлічі невідвіданих вершин і способи оновлення міток. Позначимо через n кількість вершин, а через m - кількість ребер в графі G. У найпростішому випадку, коли для пошуку вершини з мінімальним d [v] проглядається все безліч вершин, а для зберігання величин d використовується масив, час роботи алгоритму є O (n ^ 2).

Основний цикл виконується порядку n разів, в кожному з них на знаходження мінімуму витрачається близько n операцій. На цикли по сусідах кожної відвідуваною вершини витрачається кількість операцій, пропорційне кількості ребер m (оскільки кожне ребро зустрічається в цих циклах рівно двічі і вимагає константне число операцій). Таким чином, загальний час роботи алгоритму O (n ^ 2 + m), але, так як m \ le n (n-1), воно складає O (n ^ 2). Для розріджених графів (тобто таких, для яких m багато менше n²) невідвідані вершини можна зберігати в двійковій купі, а в якості ключа використовувати значення d [i], тоді час видалення вершини з \ overline U стане \ log n, при тому, що час модифікації d [i] зросте до \ log n. Так як цикл виконується порядку n разів, а кількість релаксацій (змін міток) не більше m, швидкість роботи такої реалізації O (n \ log n + m \ log n). Якщо для зберігання невідвіданих вершин використовувати Фібоначчієва купу, для якої видалення відбувається в середньому за O (\ log n), а зменшення значення в середньому за O (1), то час роботи алгоритму складе O (n \ log n + m).

* + - 1. **Програмна реалізація алгоритму**

**6.1 Загальні відомості мови програмування**

Для розробки програми обрала середовище програмування Visual Studio 2019.

Visual Studio 2019 – перша версія нової Visual Studio (Preview) була випущена у грудні 2018 року. Вона мала 5 етапів. У березні був вже визначений RC (Release Candidate). Тому через місяць, а саме 2 квітня була випущена стабільна версія Visual Studio 2019 на Windows та Visual Studio on MAC. Як і у минулих версіях є три типи (Community, Professional, Enterprise).

Серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Включає в себе: редактор початкового програмного коду, С / С++, підсвічування синтаксису, розумне введення, автодоповнення, сучасну систему навігації проекту.

Для початку роботи в середовищі Visual Studio 2019 треба, слідуючи наступному алгоритму:

1. Запустити програму за допомогою файлу .exe, або ярлику на робочому столі.
2. У відкритому вікні вибрати файл чи проект, що нещодавно відкривався, або створити новий файл, чи проект. Також можна відкрити приклад готового проекту.
3. У разі вибору «Відкрити нещодавній файл/проект», вказати місце розташування.
4. У разі вибору «Створити новий файл», відкриється чистий аркуш з якого можна розпочати роботу.
5. У разі вибору «Створити новий проект», у вікні що відкрилося, треба вибрати тип проекту (Console Application, Windows Application…), і вказати назву.
6. У вікні, що відкрилося вибрати тип застосунку, а в наступному вікні мову програмування.
7. Можна починати розробку проекту.

**6.2 Опис процедур і функцій програми**

При запуску програми з’являється консольне вікно, де нам потрібно вводити поступово шлях між двома містами, після того як ми закінчили введення, програма порахує найкоротші відстані Від 1 міста до всіх інших за Алгоритмом Дейстри.

Підпрограма int main() – визначає мінімальний шлях між двома заданими вершинами.

**6.3 Контрольний приклад**

**6.3.1 Розрахований вручну приклад**

Потрібно знайти відстань від першої вершини, до всіх інший



**Крок 1**

Ініціалізація. Відстань до всіх вершин графа V = ∞. Відстань до а =0. Жодна вершина графа ще не опрацьована.

****

**Крок 2**

Находимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.

****

**Крок 3**

Перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1- у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.

****

**Кроки 4 - 5**

Аналогічну операцію проробляєм з двома іншими сусідами 1-ї вершини — 3-ю та 6-ю.

****

**Крок 6**

Всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів Дейкстра). Тому викреслимо її з графа, щоб відмітити цей факт.

****

**Крок 7**

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7. І знову намагаємося зменшити відстань до всіх сусідів 2-ої вершини, намагаючись пройти в них через 2-у. Сусідами 2-ої вершини є 1, 3, 4.

****

**Крок 8**

Перший (по порядку) сусід вершини № 2 — 1-а вершина. Але вона вже оброблена (або викреслена — див. крок 6). Тому з 1-ою вершиною нічого не робимо. (З іншими вхідними даними) Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ої + відстань між 2-ою і 4-ою вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22.

****

**Крок 9**

Ще один сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ої вершини не міняємо.

****

**Крок 10**

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з графа.

****

**Кроки 11 — 15**

По вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» виявляється вершина № 3. Після її «обробки» отримаємо такі результати:

****

Наступні кроки Проробляємо те саме з вершинами, що залишилися (№ по порядку: 6, 4 і 5)

****

Алгоритм закінчує роботу, коли викреслені всі вершини. Результат його роботи видно на останньому малюнку: найкоротший шлях від 1-ої вершини до 2-ої становить 7, до 3-ої — 9, до 4-ої — 20, до 5-ої — 20, до 6-ої — 11 умовних одиниць.

**Реалізація алгоритму Дейкстри**

Для зберігання ваги графа використовується квадратна матриця. У заголовках рядків і стовпців знаходяться вершини графа. А ваги дуг графа розміщуються у внутрішніх осередках таблиці. Граф не містить петель, тому на головній діагоналі матриці містяться нульові значення.

Таблиця 1 – Реалізація алгоритму

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 |  ∞ |  ∞ |  ∞ |  ∞ |  ∞ |
| 2 |  | 7 | 9 |  ∞ |  ∞ | 14 |
| 3 |  |  | 9 | 22 |  ∞ | 14 |
| 4 |  |  |  | 20 |  ∞ | 11 |
| 5 |  |  |  | 20 | 20 |  |
| 6 |  |  |  |  | 20 |  |

**Результат роботи програми**



**Висновок:** в процесі виконання курсової роботи продемонструвала вміння застосовувати на практиці теоретичних знань, які були отримані під час вивчення дисципліни, також оволоділа математичним апаратом, який є характерним для сучасної теорії проєктування систем управління. Покращила навикі самостійного синтезу логічних схем алгоритмів.

Детальніше вивчила алгоритми на графах та алгоритм Дейкстри. Продемонструвала завдання в практичній частині яку розраховувала вручну та за допомогою Visual Studio 2019. Два способи реалізації співпали, отже, поставлена задача була виконана.

**Рекомендовані джерела інформації**

1. Нікольський Ю.В. Дискретна математика: Підручник.\_Львів: Магнолія Плюс, 2005. - 608 с.

2. Акимов О.Е. Дискретная математика: логика, группы, графы. 2-е узд., дополн. - М: Лабораторія Базових Знаний, 2001. - 376 с.

3. Андрійчук В. І., Комарницький М.Я. Вступ до дискретної математики: Навч. Посіб., - К.:Центр, 2004. - 254 с.

4. Бардачов Ю.М. та ін. Дискретна математика.: Підручник / Ю.М. Бардачов, І. А. Соколова, В. Є. Ходакова. - К.:Вища школа, 2002. - 287 с.

5. Капітонова Ю.В. та ін. Основи дискретної математики, - К.:Наукова думка, 2002. - 578 с.

6. Р. Хаггарти. Дискретная математика для программистов, — «Техносфера» 2012. — 317с.

7. Новиков Ф.А. Дискретная математика для программистов — СПб.: «Питер», 2000. — 301с.

8. Інтернет джерело: <https://blog.skillfactory.ru/glossary/algoritm-dejkstry/>

9. Інтернет джерело: <https://sites.google.com/site/algorithmsandmath/algoritmi-na-grafah/algoritm>

10. Інтернет джерело: <https://prog-cpp.ru/deikstra/>

11. Інтернет джерело: <https://habr.com/ru/post/63347/>

**Додаток 1 .Код програми**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define SIZE 6

int main()

{

 int a[SIZE][SIZE];

 int d[SIZE];

 int v[SIZE];

 int temp, minindex, min;

 int begin\_index = 0;

 system("chcp 1251");

 system("cls");

 for (int i = 0; i < SIZE; i++)

 {

 a[i][i] = 0;

 for (int j = i + 1; j < SIZE; j++) {

 printf("Введіть відстань %d - %d: ", i + 1, j + 1);

 scanf("%d", &temp);

 a[i][j] = temp;

 a[j][i] = temp;

 }

 }

 for (int i = 0; i < SIZE; i++)

 {

 for (int j = 0; j < SIZE; j++)

 printf("%5d ", a[i][j]);

 printf("\n");

 }

 for (int i = 0; i < SIZE; i++)

 {

 d[i] = 10000;

 v[i] = 1;

 }

 d[begin\_index] = 0;

 do {

 minindex = 10000;

 min = 10000;

 for (int i = 0; i < SIZE; i++)

 {

 if ((v[i] == 1) && (d[i] < min))

 {

 min = d[i];

 minindex = i;

 }

 }

 if (minindex != 10000)

 {

 for (int i = 0; i < SIZE; i++)

 {

 if (a[minindex][i] > 0)

 {

 temp = min + a[minindex][i];

 if (temp < d[i])

 {

 d[i] = temp;

 }

 }

 }

 v[minindex] = 0;

 }

 } while (minindex < 10000);

 printf("\nНайкоротшиі відстані до вершин \n");

 for (int i = 0; i < SIZE; i++)

 printf("%5d ", d[i]);

 getchar(); getchar();

 return 0;

}