Київський національний університет будівництва і архітектури

Кафедра інформаційних технологій проектування та прикладної математики

Спеціальність: Комп'ютерні науки

Курс 2 Група ІУСТ-21 Семестр 3

ЗАВДАННЯ

на курсову роботу студентки

_____

1.Тема роботи «Алгоритми на графах. Орієнтований граф»

2.Термін здачі студентом закінченої роботи: 01.грудня 2014р

3.Вихідні дані до роботи Кількість шляхів заданої довжини. Загальна кількість шляхів між двома заданими вершинами.

4.Зміст пояснювальної записки перелік питань, які належить розробити: 1.Вступна частина; 2.Математичний опис розв'язку задачі; 3.Схема алгоритму; 4.Програмна реалізація алгоритму

5.Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Схема алгоритму

6. Дата видачі завдання "01" жовтня 2014 р.

Студент _____ Керівник _____

(підпис) (підпис)

Зміст

1. Постановка задачі

За заданою матрицею суміжності і двома вершинами орграфа визначити:

1) кількість шляхів заданої довжини між двома заданими вершинами;
2) загальну кількість шляхів між двома заданими вершинами;
3) довжину найкоротшого шляху між заданими вершинами.

2. Теоретичні відомості

Теорія графів - молода область дискретної математики. Проте методи теорії графів завоювали визнання не тільки математиків, а й інженерів, економістів, психологів, біологів, хіміків. Використання мови і методів теорії графів прискорює розв'язання практичних задач, спрощує розрахунки. Теорія графів є однією з важливих частин математичного апарату інформатики і кібернетики. У термінах теорії графів формулюють велику кількість задач, пов'язаних з дискретними об'єктами.

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 орієнтованістю, обмеженнями на кількість зв'язків і додатковими даними про вершини або ребра.

Орієнтований граф (коротко **орграф**) — (мульти) граф, ребрам якого присвоєно напрямок. Орієнтовані ребра називаються також *дугами*, а в деяких джерелах (Ope) і просто ребрами.

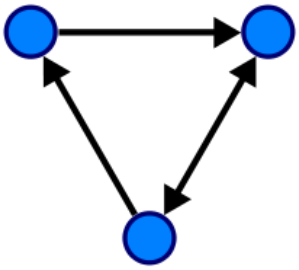

Рис.2.1.

Формально, орграф D = (V, E) є множина E впорядкованих пар вершин $v \in V$

.

Дуга {u, v} **інцидентна** до вершин u і v. При цьому говорять, що u — **початкова вершина** дуги, а v — **кінцева вершина**.

Орграф, отриманий з простого графа орієнтацією ребер, називається **орієнтованим**. На відміну від останнього, у довільного **простого орграфа** дві вершини можуть з'єднуватися двома різноорієнтованими дугами.

Орієнтований повний граф 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 орграфа називають послідовність вершин і *дуг*, виду $v_0\{v_0, v_1\}v_1\{v_1, v_2\}v_2...v_n$ (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 (мульти)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 у вигляді матриці.

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 a_{ij} рівне числу ребер з i-ї вершини графа в j-у вершину.

Іноді, особливо у разі неорієнтованого графа, петля (ребро з і-ї вершини в саму себе) вважається за два ребра, тобто значення діагонального елементу a_{ii}в цьому випадку рівне подвоєному числу петель навколо і-ї вершини.
Матриця суміжності простого графа (що не містить петель і кратних ребер) є бінарною матрицею і містить нулі на головній діагоналі.

Орієнтований (направлений) ациклічний граф — випадок орієнтованого графа, в якому відсутні *орієнтовані цикли*, тобто шляхи, що починаються і закінчуються в одній і тій самій вершині. Орієнтований ациклічний граф є узагальненням дерева (точніше, їх об'єднання — *лісу*).
Алгоритм Дейкстри – алгоритм на графах, що знаходить шлях від однієї вершини графа до всіх інших. Класичний алгоритм працює тільки на графах без від’ємних дуг.

Пошук в глибину (англ. Depth-first search, DFS) – один з методів обходу графа. Алгоритм пошуку описується таким чином: для кожної не пройденої вершини необхідно знайти всі не пройдені суміжні вершини і повторити пошук для них.

В представленій курсовій роботі алгоритм реалізовано за допомогою матриці суміжності (ваг) і рекурсивної функції.

3. Вхідні та вихідні дані

В якості **вхідних даних** до даної задачі виступає матриця суміжності, початкова та кінцева вершина, довжина шуканого шляху.

Вихідними даними Кількість шляхів заданої довжини, загальна кількість шляхів між двома заданими вершинами, довжина мінімального шляху між двома заданими вершинами.

4. Математична постановка та опис розв’язку задачі

Пошук маршрутів:

1. Дана матриця суміжності **map[N][N]** і дві вершини **start** і **finish**: початок маршруту і кінець. Також є масив **road[N]**, що зберігає проміжні вершини і масив **used[N]** (тип **bool**),

2. Якщо початок і кінець співпали (start = finish), тоді у циклі виводимо масив road[N].

Інакше, у циклі з кроком **i** перебираємо масив **map[start][i]** і відшукуємо всі не пройдені вершини суміжні з поточною;

1) відвідані вершини позначаються «прапорцем» – тобто в масиві **used[N]** вершині у відповідність ставиться значення булевої змінної **true**;
2) рекурсивно повторюємо пошук для щойно знайденої не відвіданої вершини.

Рекурсія повторюється поки початок маршруту не співпаде з кінцем (**start** = **finish**).

Для підрахунку кількості шляхів заданої довжини використаємо лічильник (змінна типу **int**).

Пошук мінімального:

1. Множина вершин U, до яких відстань відома, встановлюється рівною {u}.

2. Якщо U = V, алгоритм завершено.

3. Потенційні відстані Di до вершин з U\V встановлюються нескінченними.

4. Для всіх ребер (i, j), де i∈U та j∈V\U, якщо Dj>di+w(i, j), то Dj присвоюється di+w(i, j).

5. Шукається i∈V\U, при якому Di мінімальне.

6. Якщо Di дорівнює нескінченності, алгоритм завершено. В іншому випадку di присвоюється значення Di, U присвоюється U∪{i} і виконується перехід до кроку

5. Алгоритм розв’язку задачі

5.1. Схема алгоритму

Структура програми наведена на рис. 5.1. Схема алгоритму пошуку в глибину представлена на рис. 5.2. Схема алгоритму визначення min шляху наведена на рис. 5.3.

Рисунок 5.1. Структура програми

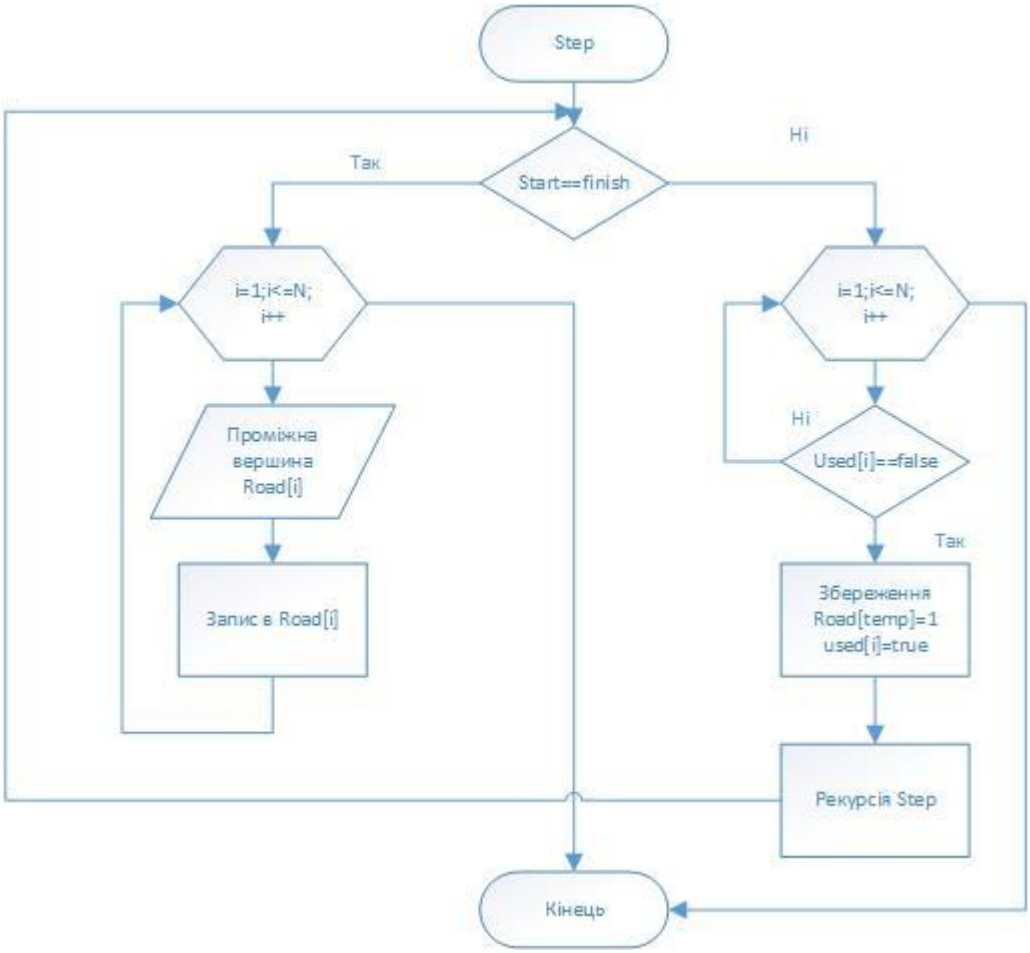

Рисунок 5.2. Схема алгоритму пошуку в глибину

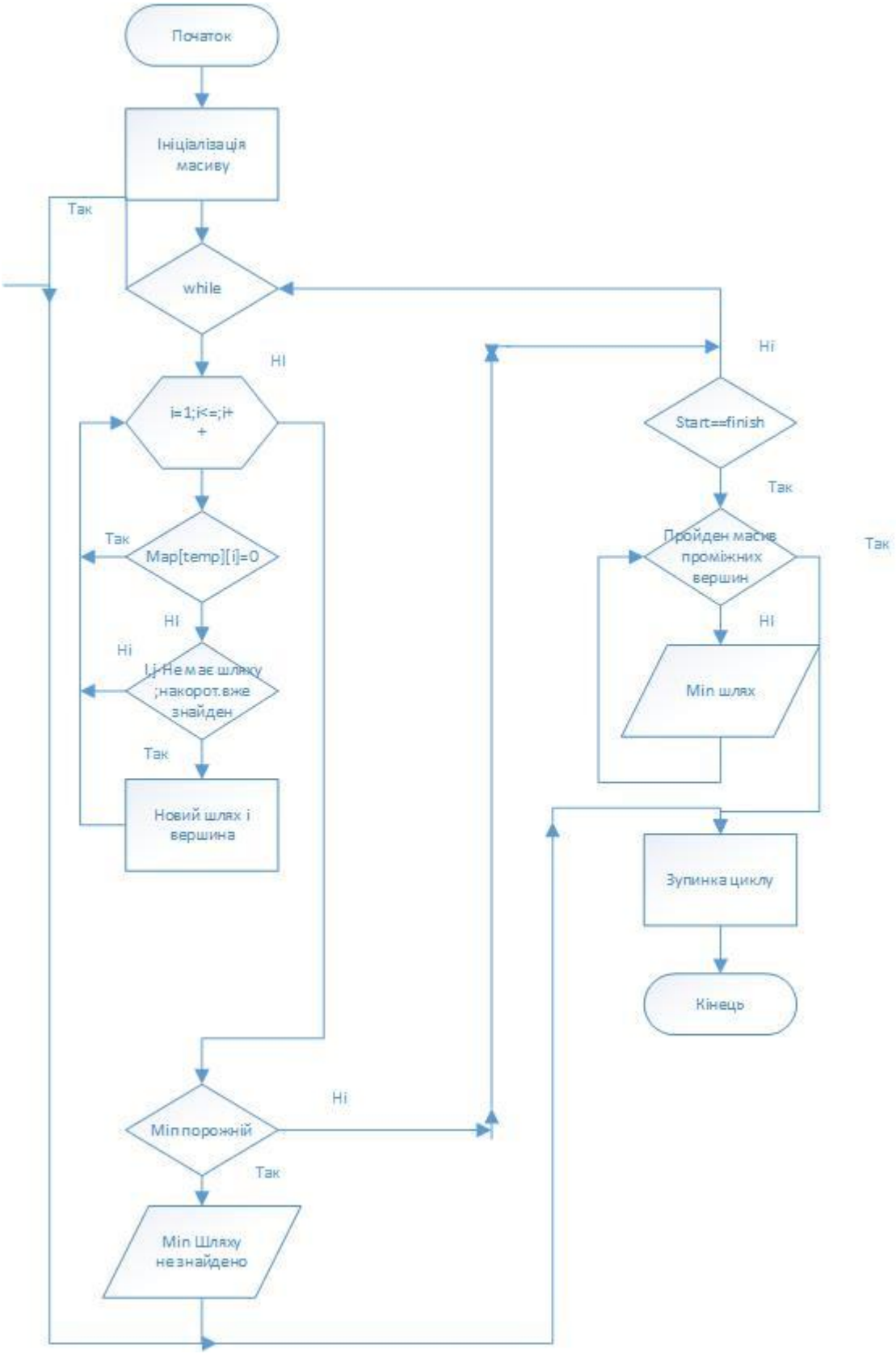

Рисунок 5.3 Схема алгоритму визначення min шляху

5.2. Оцінка складності алгоритму

Складність алгоритму Дейкстри залежить від способу знаходження 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 n^2) 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).

6. Програмна реалізація алгоритму

6.1 Загальні відомості мови програмування

Для розробки програми було використано середовище програмування C-Free 5.

C-Free 5 - це професійне середовище розробки, яка підтримує безліч компіляторів. Використовуючи її програміст може легко редагувати, конструювати, запускати і налагоджувати програми.

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

Для початку роботи в середовищі C-Free 5 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

Підпрограма **void Step** (int s, int f, int p) – пошуком у глибину визначає всі можливі шляхи між двома заданими вершинами, а також виводить кількість шляхів заданої довжини.

Підпрограма **void Deykstra** () – визначає мінімальний шлях між двома заданими вершинами.

6.3 Контрольний приклад

6.3.1 Приклад розрахований вручну

ПРИКЛАД (АБО ПРИКЛАДИ) ДЛЯ ПЕРЕВІРКИ ВСІХ МОЖЛИВИХ ВАРІАНТІВ РЕЗУЛЬТАТУ ВИКОНАННЯ АГГОРИТМУ

6.3.2 Приклад роботи програми

Знайдемо всі шляхи з 5 в 3 і кількість шляхів довжиною 5 (рис. 6.1).

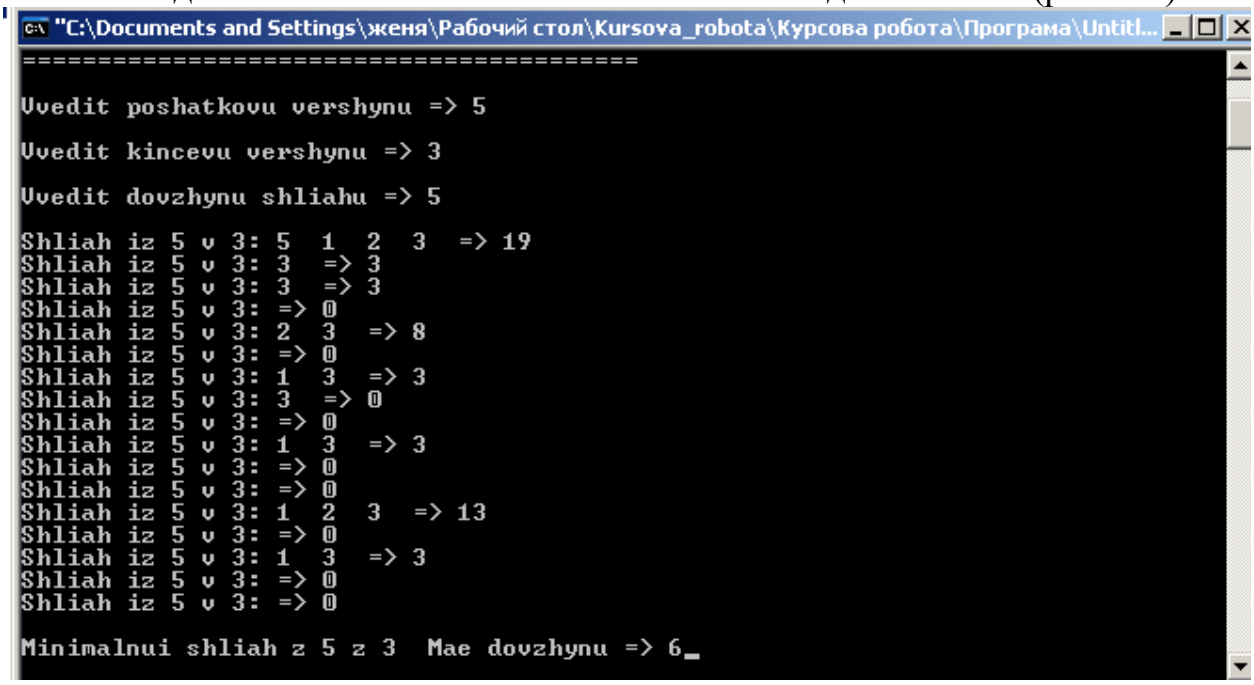

Рисунок 6.1. Результат програми

Якщо у програмі задані не правильно значення (рис. 6.2).

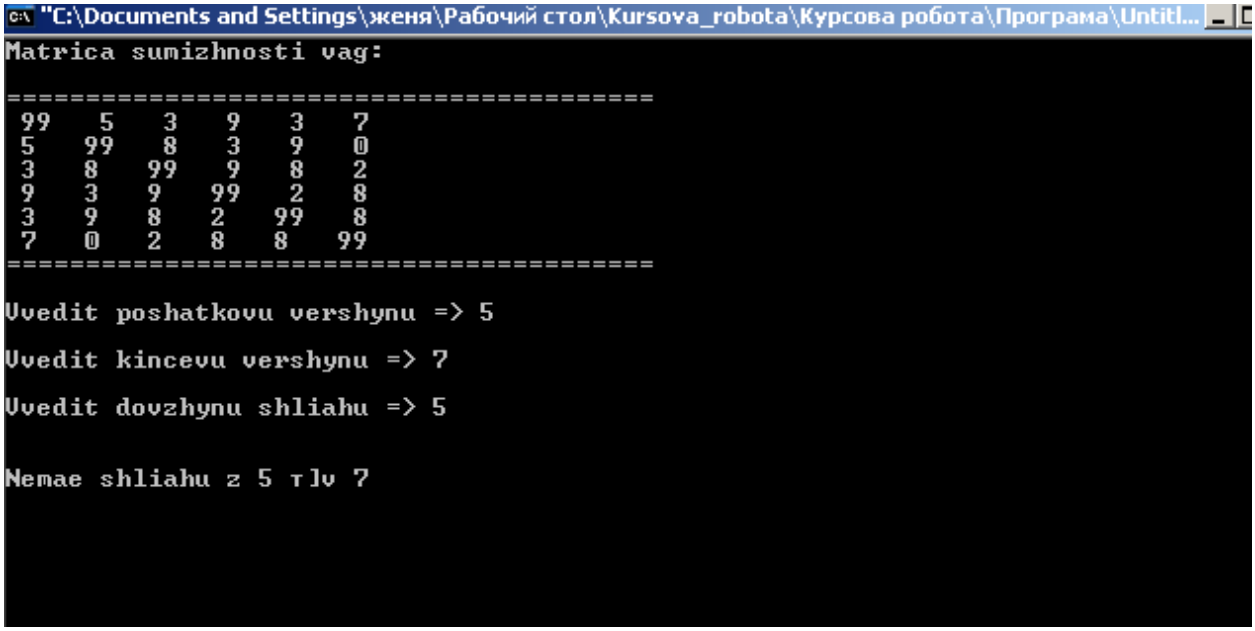

Рисунок 6.2. Помилка

Висновок

В даній курсовій роботі було досліджено алгоритми на графах ,а саме орієнтований граф, розроблено схему алгоритму та написано програму, що визначає кількість шляхів заданої довжини, загальна кількість шляхів між двома заданими вершинами, довжина мінімального шляху між двома заданими вершинами.

Розроблено алгоритм в якому присутні:пошук в глибину та алгоритм Дейкстри. Алгоритм було протестовану на прикладі. Після усіх досліджень можна відзначити, що розроблений алгоритм відповідає таким властивостям:

- результативність- алгоритм закінчується певним результатом (інформація про кількість шляхів між двома заданими вершинами, довжина мінімального шляху між двома заданими вершинами).
- зрозумілість- алгоритм є простим і зрозумілим всім виконавцям.
- дискретність- алгоритм розподілений на певну кількість етапів.

- детермінованість- після кожного кроку, зазначено, який крок робити далі.
- масовість- алгоритм обробляє вхідні(орієнтований граф та матриця суміжності) та вихідні дані(кількість шляхів між двома заданими вершинами, довжина мінімального шляху між двома заданими вершинами).

Отже, поставлена задача була виконана.

Список літератури

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.

Додаток 1 Код програми

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <vector>
#include <clocale>
#include <fstream>
using namespace std;

//=========================================================================

#define N 7
#define INF99
//=========================================================================

int map[N][N];
int road[N];
int start, finish;
int i, j;
int S = 0;
int in_S;
int kaka = 0;
FILE *f_input;
FILE *f_output;
int f_temp;
файла

//=========================================================================

void step (int s, int f, int p)                    {
        int c;
if (s == f)                                                                 {
                printf ("\nШЛях із %d в %d: ", start, finish);
                S = 0;
                for (i = 1; i <= p-1; i++)
                {
                        printf (" %d ", road[i]);
                        fprintf (f_output, " %d - ", road[i]);
                        int t = i+1;
                        S = S + map[road[i]][road[t]];
                }
                fprintf (f_output, "\n");
                printf (" => %d", S);
                if (S == in_S)
                        kaka++;
        }
        else
        {
                for (c = 1; c <= N; c++)
                        if ((map[s][c] != 0) && (!bool_temp[c]))
                        {
                                road[p] = c;
                                bool_temp[c] = true;
                                step(c, f, p + 1);
                                bool_temp[c] = false;
                                road[p] = 0;
                        }
        }
}
```

```
//============================================================================

int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_CTYPE, "");

	f_input = fopen ("Data.txt", "r");
f_output = fopen ("output.txt", "w");

	for (i = 1; i <= N; i++)

	for (i = 1; i <= N; i++)
bool_temp[i] = false;

	printf ("Матриця суміжності ваг:\n\n");
	printf ("=======================================\n");
	for (i = 1; i <= N; i++)
	{
		for (j = 1; j <= N; j++)
		{
			fscanf (f_input, "%d", &f_temp);
			map[i][j] = f_temp;
			printf (" %d  ", map[i][j]);
		}
		printf ("\n");
	}
	printf ("=======================================\n");

	printf ("\nВведіть початкову вершину => ");
	scanf ("%d", &start);
	printf ("\nВведіть кінцеву вершину => ");
	scanf ("%d", &finish);

	printf ("\nВведіть довжину шляху => ");
	scanf ("%d", &in_S);

	road[1] = start;
	bool_temp[start] = true;
step (start, finish, 2);
	getch();

//============================================================================

	int s;
	int g;

	int x[N+1];                              ,

	int t[N+1];
	int h[N+1];
	bool kol[N+1];

	int u;
	for (u = 1; u <= N; u++)
		t[u] = INF;
	x[u] = 0;
	kol[u] = false;
	}

	h[start] = 0;
	t[start] = 0;
	x[start] = 1;
```

```
	int v = start;

	while (1)                                {
		for (u = 1; u <= N; u++)
		{
			if (map[v][u] == 0) continue;                                        //
			if (x[u] == 0 && t[u] > t[v] + map[v][u])
			{
				t[u] = t[v] + map[v][u];
		}

		int w = INF;
		v = -1;

		for (u = 1; u <= N; u++)                          .
		{
			if (x[u] == 0 && t[u] < w)
                                                                        {
				v = u;
				w = t[u];
			}
		}

		if (v == -1)
		{
			printf ("\n\nНемає шляху з %d в %d", start, finish);
			break;
		}

		if (v == finish)
		{
			printf ("\n\nМінімальний шлях з %d в %d", start, finish);
			u = finish;

			while (u != start)
			{
				printf (" ");
				u = h[u];
			}

			printf ("має довжину => %d", t[finish]);
			break;
		}
		x[v] = 1;
	}

	getch();

	printf ("\n\n%d", kaka-1);

	getch();
	return 0;
}
```