МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА КОМП’ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

### КУРСОВА РОБОТА

|  |
| --- |
| З дисципліни „ Системне програмування ”  на тему:  **„ Програмна реалізація алгоритму Дейкстри”** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  | | Студента 2-го курсу  напряму підготовки 6.050102 –  «Комп’ютерна інженерія»  Зверєва Олексія Андрійовича  Керівник :ст.викл. Самусь Є.І.  Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кількість балів: \_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_ | | | Члени комісії | | \_\_\_\_\_\_\_\_\_\_\_\_ | | ст. викл. Самусь Є.І | | ­\_\_\_\_\_\_\_\_\_\_\_\_ | | ст.викл. Далекорей А.В. | |  | | | | |

м. Ужгород – 2016 рік

**ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ**

**Тема:** Програмна реалізація алгоритму Дейкстри.

**Мета:** Реалізувати алгоритм Дейкстри на Ассемблері.

Практична реалізація повинна забезпечити наступні вимоги:

* Введення даних з клавіатури
* Введення даних з файлу
* Перевірка коректності введених даних
* Вибір початкової вершини обходу
* Вибір кінцевої вершини обходу

**Календарний план виконання курсової роботи**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсової роботи | Строк виконання роботи | Примітка |
| 1 | Вивчення предмету дослідження | 15.03.2016 |  |
| 2 | Пошук та вивчення літератури з питань курсової роботи | 23.03.2016 |  |
| 3 | Формування макету курсової роботи | 29.03.2016 |  |
| 4 | Розробка алгоритму розв'язання задачі, вибір структур та методів | 06.04.2016 |  |
| 5 | Розробка програми | 11.04.2016 |  |
| 6 | Тестування та налагодження | 25.04.2016 |  |
| 7 | Написання пояснювальної записки | 01.05.2016 |  |
| 8 | Попередній захист | 23.05.2016 |  |
| 9 | Захист | 03.06.2016 |  |

ЗМІСТ

[ВСТУП 4](#_Toc295401808)

1[ОСНОВИ ТЕОРІЇ ГРАФІВ 5](#_Toc295401809)

[1.1. Основні поняття теорії графів 5](#_Toc295401810)

[1.2. Застосування графів](#_Toc295401811) 6

[1.3. Представлення графів у пам’яті ЕОМ](#_Toc295401812) 7

2 [АЛГОРИТМ ДЕЙКСТРИ](#_Toc295401820) 11

[2.1. Опис алгоритму](#_Toc295401822) 11

3 [ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА](#_Toc295401820) 12

[3.1. Введення даних з клавіатури 12](#_Toc295401810)

[3.2. Введення даних з файлу 1](#_Toc295401811)4

4 [ПРИКЛАДИ РОБОТИ ПРОГРАМИ 16](#_Toc295401820)

[ВИСНОВКИ](#_Toc295401824) 19

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 2](#_Toc295401825)0

[ДОДАТКИ](#_Toc295401826) 21

[Додаток А Лістинг програми](#_Toc295401827) 21

[Додаток Б Блок-схема алгоритму](#_Toc295401827) 29

**ВСТУП**

На даний момент існує велика кількість мов програмування для вирішення великого кола задач. Особливе місце серед усіх мов програмування займає ассемблерна мова програмування. Ця мова є однією із перших мов програмування, яка дозволила перейти на більш високий рівень, який був до цього. Програми, написані на Ассемблері мають переваги над ішими через економію пам’яті та точність роботи програм, а відповідно і швидкодію роботи цих програм. На жаль, написання коду на даній мові займає велику кількість часу програміста через увагу до деталей та код виглядає нагромадженим, тому Ассемблер втрачає свою актуальність.

Алгоритм Дейкстри був розроблений у 1959 році, але і досі зберігає свою актуальність. Через розвиток та поширення транспортних систем, ліній електропередач, комп’ютерних мереж алгоритми на графах набрали актуальність. Існує велика кількість алгоритмів знаходження найкоротших шляхів, але саме алгоритм Дейкстри зберігає свою актуальність через коректну роботу з великою кількістю вершин, економією часу та ресурсів ЕОМ.

**1 ОСНОВИ ТЕОРІЇ ГРАФІВ**

**1.1 Основні поняття теорії графів**

**Теорія графів** — розділ математики, що вивчає властивості графів.

Родоначальником теорії графів вважається Леонард Ейлер. У 1736 році в одному зі своїх листів він формулює і пропонує рішення завдання про сім Кенігсберзьких мостів, що стала згодом однією з класичних задач теорії графів.

Поштовх до розвитку теорія графів отримала на рубежі XIX і ХХ століть, коли різко зросла кількість робіт в сфері топології та комбінаторики, з якими її пов'язують тісні узи спорідненості. Графи стали використовуватися при побудові схем електричних кіл і молекулярних схем. Як окрема математична дисципліна теорія графів булла вперше представлена в роботі угорського математика Кеніга в 30-ті роки XXстоліття.

**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.

**Ребро** — це неорієнтовний зв’язок між двома вершинами графу.

**Дуга** — це орієнтовний зв’язок між двома вершинами графу.

Графи поділяються на такі основні категорії:

1. **Неорієнтовний граф** — граф, у якому всі зв’язки між вершинами представлені ребрами.
2. **Орієнтовний граф** — це граф, у якому всі зв’язки між вершинами представлені дугами.
3. **Мішаний граф** — граф, у якому зв’язки між вершинами представлені ребрами та дугами.
4. **Зважений граф** — граф, кожне ребро, або дуга якого має певну вагу, тобто визначені відстані між точками графа.
5. **Зв’язний граф** — це граф, всі точки(вершини) якого зв’язані.
6. **Дерево** — повністю зв’язний граф, який не містить циклів.

**Граф або неорієнтований граф *G***— це впорядкована пара *G=(V,E)* , для якої виконуються наступніу мови:

*V* — множина вершин або вузлів

*E* — множинаребер (дуг)

V та E зазвичай вважаються скінченними множинами.

**1.2 Застосування графів**

Останнім часом, графи набрали популярність у вирішенні різних задач не тільки з математики, але й з інформатики, фізики, біохімії тощо.

Математична модель графа широко застосовується у вигляді певної математичної абстрактної моделі для опису сукупності пов’язаних між собою об’єктів, наприклад:

* Лінії електропередач
* Залізнодоріжні шляхи
* Карти автомобільних доріг
* Лабіринти
* Ігрова індустрія
* Комп’ютерні системи і мережі
* Соціальні мережі
* Генеологічні дерева
* Пошукові системи
* Картографія

Отже, можна побачити те, що графи досить поширені у сучасному світі, але це лише математична модель для представлення вищевказаних об’єктів, які використовують, щоб на основі теорії графів вирішити такі проблеми:

* Знаходження найкоротшого(“найдешевшого”) шляху з точки A до точкиB
* Пошук зв’язних компонентів у цих мережах
* Розфарбування точок графів
* Побудова остовних дерев

Як можна помітити, графи застосовуються у багатьох галузях людської діяльності. Знаходження найкоротших шляхів, остовних дерев і т.д. займають велику кількість часу, особливо, якщо граф має велику кількість вершин та ребер, тому в еру розквіту інформаційних технологій для роботи з графами широко використовуються електронно-обчислювальні машини.

**1.3 Представлення графів у пам’яті еом**

Якщо потрібно обробляти граф у ЕОМ, то постає питання представлення цього графу у її пам’яті. Вибір структури буде мати вирішальне значення у процесі обробки даного графу. Зазвичай для цього використовують уже готові структури у мовах програмування, а саме масиви та списки.

Основні способи представлення графів у пам’яті ЕОМ:

1. **Матриця суміжності** — це двовимірний масив (N\*N), де N-кількість вершин кожен елемент якого містить певне значення:

0 – вершини не мають спільних ребер(дуг)

1 – вершини мають спільне ребро(дугу), для незважених графів

M – вершини мають спільнеребро(дугу) з вагою M, де M – деяке натуральне число

В цьому випадку індекси елементів слугують ідентифікаторами вершин, які зв’язані певним ребром, чи дугою. До прикладу, якщо G[1][2]=5, де G – матриця суміжності графа, то це означає, що між 1 та 2 вершиною існує дуга, вага якої рівна 5, у випадку неорієнтовного графа G[1][2]=G[2][1], тобто елементи цієї матриці дзеркально відображені відносно головної діагоналі.

**Приклад:**

Неорієнтований зважений граф, зображений на рис.1.1 заданий матрицею суміжності (1).

G=  (1)

2.**Матриця інцидентності** — це двовимірний масив (N\*M), де N – кількість стовпців(ребер), M – кількість рядків(вершин).

Кожна комірка матриці для неорієнтовного графу може набувати трьох значень:

-1 – ребро kj виходить з вершини vi

1–ребро kjвходить у вершинуvi

0 –ребро kj не має стосунку до вершини vi

**Приклад:**

Неорієнтований зважений граф, зображений на рис.1.1 заданий матрицею інцидентності (2)

G= (2)

3.**Список суміжності** — це послідовність розміром **N**, кожен елемент якої є списком вершин суміжних з даною вершиною.

Приклад:

Неорієнтований зважений граф, зображений на рис.1.1 заданий списком суміжності у табл. 1.1.

Таблиця 1.1 Список суміжності графа

|  |  |  |
| --- | --- | --- |
| Вершина виходу | Вершини входу | Ваги ребер |
| 1 | 2,3 | 1,3 |
| 2 | 1,3 | 1,3 |
| 3 | 1,2 | 2,3 |

4.**Список ребер** — це список, де кожному ребру графа відповідає рядок, в якому зберігаються дві вершини, інцидентні ребру.

**Приклад:**

Неорієнтований зважений граф, зображений на рис.1.1 заданий списком ребер у табл. 1.2.

Таблиця 1.2. Список ребер графа

|  |  |
| --- | --- |
| Вага ребра | Інцидентні вершини |
| 1 | 1,2 |
| 2 | 1,3 |
| 3 | 2,3 |

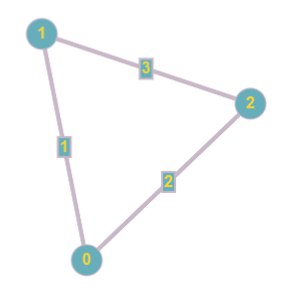


Рисунок 1.1 Тестовий граф

2 **АЛГОРИТМ ДЕЙКСТРИ**

**2.1 Опис алгоритму**

Алгоритм Дейкстри був заснований Едсгером Дейкстрою у 1959 році. Класична версія цього алгоритму знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин у випадку, якщо граф не містить ребра від’ємної довжини.

Даний алгоритм широко застосовується в обчислювальній техніці, а особливо для побудови найкоротших шляхів у навігаційних системах.

На початку роботи алгоритму задається зважений граф G=(V,E) та обирається початкова точка маршруту(певна вершина графа). Відстань до неї задається 0, відстань до всіх інших точок стає рівна нескінченності(або максимальному числу, яке дозволяє вмістити в себе змінна у пам’яті). Таким чином відмічається, чи булла відвідана вершина, чи ні. Після цього на кожному кроці розглядається одна вершина, відстань до якої найменша від уже відвіданих вершин, тим самим зменшуючи відстань до початкової вершини і відмічаючи поточну вершину як уже відвідану. Так буде повторюватись, доки не залишиться невідвіданих вершин.

**3 ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА**

Для роботи програми користувач має мати встановлену 32-х бітну операційну систему сімейства Windows. Щоб почати працювати з програмою користувач має запустити файл KURSOVA.EXE, після цього буде запущено програму.

У меню користувач може обрати, вводити дані з клавіатури, чи з файлу рис 3.1.

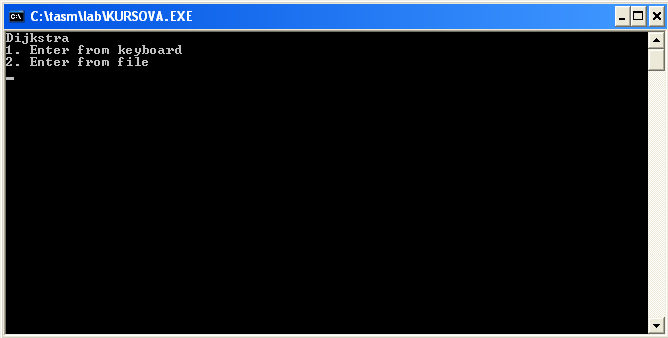


Рисунок 3.1 —Головне меню програми

**3.1 Введення даних з клавіатури**

Щоб розпочати введення даних(матриці) з клавіатури потрібно натиснути клавішу “1”, після цього користувачу буде запропоновано ввести кількість вершин графу(матриці суміжності) та елементи даної матриці(відстані між вершинами графу). Програма працює з неорієнтованими графами, тому потрібно вводити матрицю, дзеркально відображену відносно головної діагоналі (Рис 3.2). При введенні некоректних даних буде відображено відповідне повідомлення (Рис 3.3).

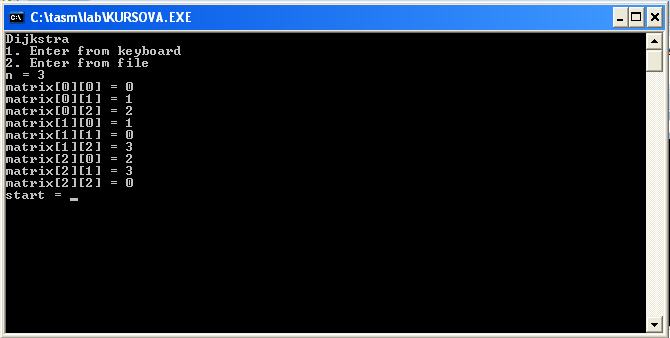


Рисунок 3.2 Введення даних з клавіатури

Після того, як всі елементи матриці будуть задані, користувачу буде запропоновано обрати початкову вершину, з якої буде здійснено обхід до всіх вершин графа.

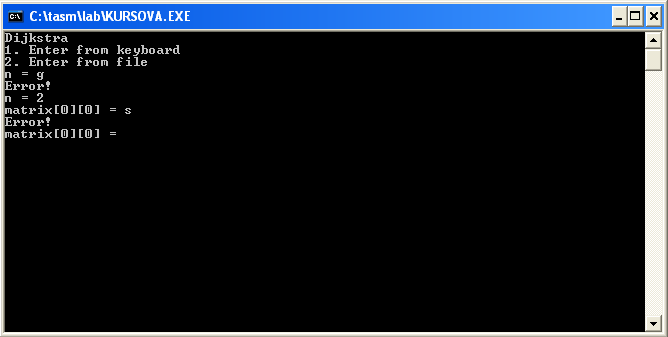


Рисунок 3.3 Реакція програми на введення некоректних даних

**3.2 Введення даних з файлу**

Щоб почати введення інформації з файлу у головному меню запущеної програми необхідно натиснути клавішу “2”. Після цього користувачу буде запропоновано ввести назву файлу, з якого буде зчитано інформацію (Рис. 3.4).

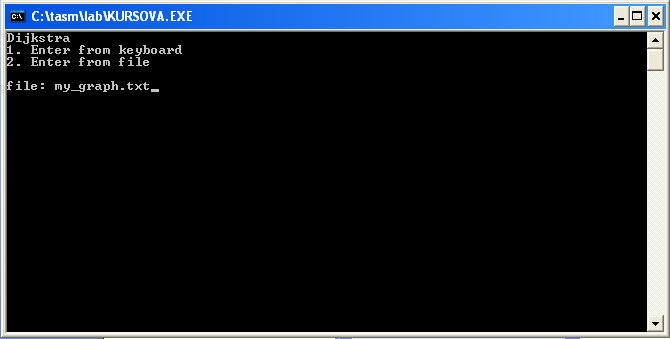


Рисунок 3.4 Введення даних з файлу

Файл, з якого вводиться інформація в програму має бути в одній папці з файлом KURSOVA.EXE. В першому рядку файлу має бути введена кількість вершин графа, в наступних n рядках(де n – кількість вершин графу) має бути введена матриця суміжності графу. Після останнього елемента має бути перенос строки (Рис.3.5).

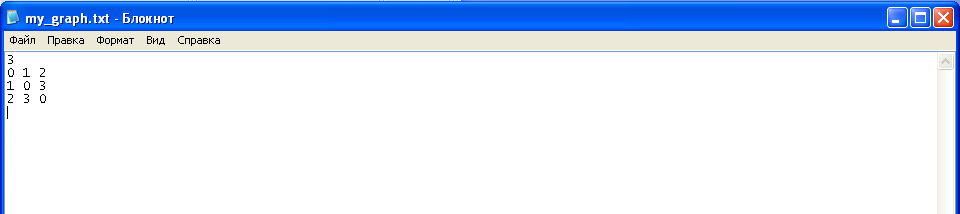


Рисунок 3.5 Файл зі вхідними даними

Після того, як буде введено назву файлу, буде виведено кількість вершин введеного графу і запропоновано ввести початкову вершину, з якої буде здійснено обхід по всім вершинам графу за алгоритмом Дейкстри (Рис.3.6).

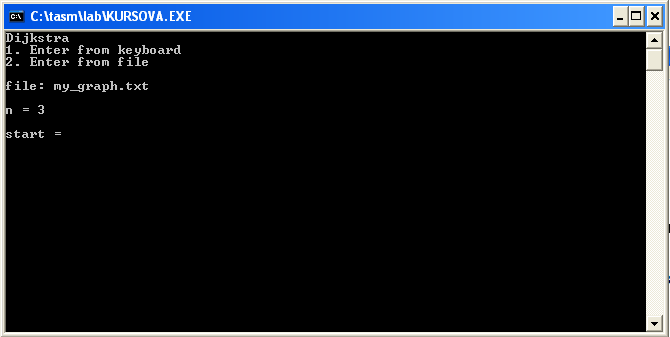


Рисунок 3.6. Введення даних з обраного файлу

**4** **ПРИКЛАДИ РОБОТИ ПРОГРАМИ**

Приклад 1:

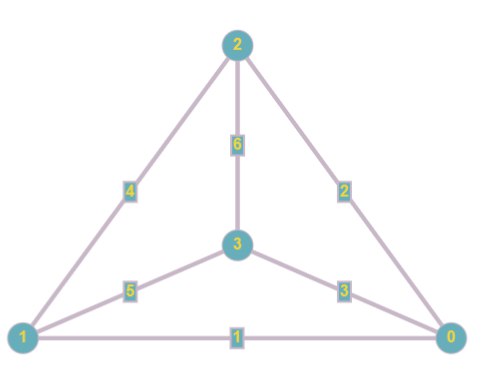


Рисунок 4.1 Тестовий граф №1

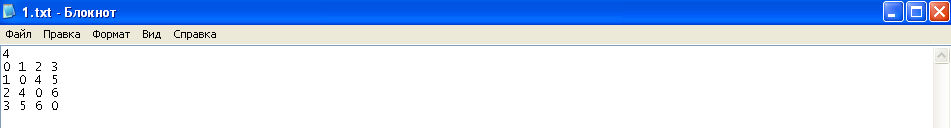


Рисунок 4.2 Файл зі вхідними даними тестового графа №1

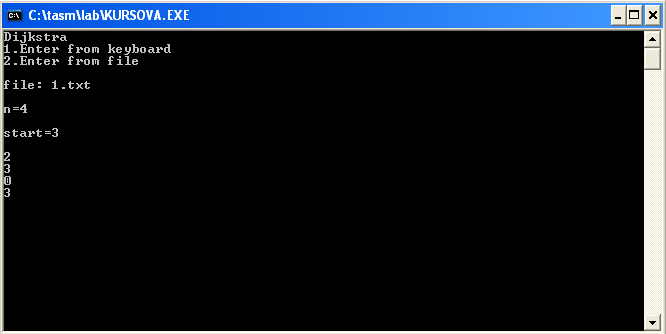


Рисунок 4.3 Результат виконання програми для тестового графа №1

Приклад 2:

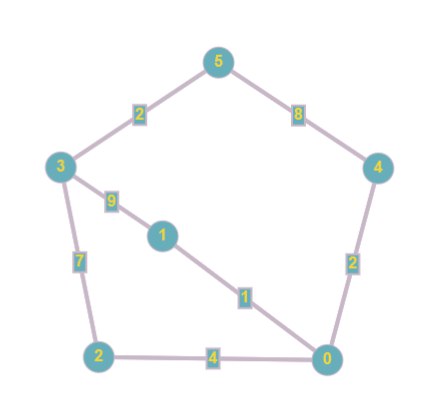


Рисунок 4.4 Тестовий граф №2

G= (3)

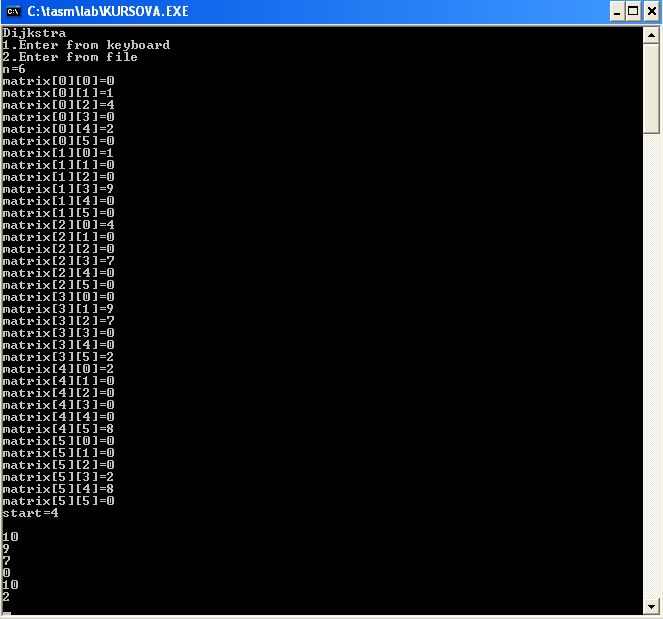


Рисунок 4.5 Результат виконання програми для тестового графа №2

**ВИСНОВКИ**

В даній курсовій роботі було розроблено програмну реалізацію алгоритму Дейкстри на мові ассемблеру. Програма була розроблена і протестована на операційній системі *Windows XP* з використанням пакету *Turbo Assembler* (*TASM*)*.*

Під час дослідження теоретичної частини алгоритмів на графах, розробки програми на мові Ассемблеру мною були покращені навички програмування на даній мові програмування, досліджені алгоритми на графах, написана программа для пошуку найкоротшого шляху між вершинами та набуті навички роботи у середовищі *Turbo Assembler.*. Було встановлено, що алгоритм Дейкстри є одним із найпотужніших алгоритмів для даної задачі, а Ассемблер і досі є актуальним через свою точність роботи та економію ресурсів та часу ЕОМ.

Програма успішно пройшла тестування та готова до роботи.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Голубь. Искусство программирования на Ассемблере. Лекции и упражнения. — ДиаСофт, 2002 —ISBN 5-93772-056-3
2. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы построение и анализ. — Москва, Санкт-Петербург, Киев, 2005 – 1292 с.
3. www.cyberforum.ru/asm-beginners

**ДОДАТКИ**

**Додаток А Лістинг програми**

.model small

.386

.stack 2000h

.data

dijk db "Dijkstra", endl, 0

keyboard\_in db '1. Enter from keyboard', endl, 0

file\_in db '2. Enter from file', endl, 0

str\_n db 'n = ', 0

str\_error db 'Error!', endl, 0

str\_end db 'Enter ESC for exit', endl, 0

str\_matr1 db 'matrix[', 0

str\_matr2 db '][', 0

str\_matr3 db '] = ', 0

str\_start db 'start = ', 0

str\_file db 'file: ', 0

n dw 0

beg dw 0

len dw ?

matrix dw ?

distance dw ?

visited dw ?

handle dw ?

file\_name db 80 dup(0)

m\_out db 80 dup(0)

tmp db 80 dup(0)

.code

LOCALS

include m\_io.inc

index proc c

arg i

uses ax, cx

mov ax, n

add ax, ax

mov cx, i

xor bx, bx

index\_cycle:

cmp cx, 0

je index\_end

add bx, ax

dec cx

jmp index\_cycle

index\_end:

add bx, matrix

ret

index endp

dijkstra proc c

uses ax, bx, cx, dx, si, di

local @count, @index, @i, @u, @min

mov cx, n

dec cx

dijkstra\_cycle1:

cmp cx, 0

je dijkstra\_end2

mov @min, -1

xor si, si

dijkstra\_cycle2:

cmp si, n

je dijkstra\_end

mov bx, visited

add bx, si

cmp byte ptr[bx], 0

jne dijkstra\_endif1

mov bx, distance

add bx, si

add bx, si

mov ax, @min

cmp word ptr[bx], ax

ja dijkstra\_endif1

mov ax, word ptr[bx]

mov @min, ax

mov @index, si

dijkstra\_endif1:

inc si

jmp dijkstra\_cycle2

dijkstra\_end:

mov ax, @index

mov @u, ax

mov bx, visited

add bx, ax

mov byte ptr[bx], 1

xor si, si

dijkstra\_cycle3:

cmp si, n

je equal

mov bx, visited

add bx, si

cmp byte ptr[bx], 0

jne dijkstra\_endif2

push @u

call index

add sp, 2

add bx, si

add bx, si

cmp word ptr[bx], 0

je dijkstra\_endif2

mov ax, word ptr[bx]

mov bx, distance

add bx, @u

add bx, @u

cmp word ptr[bx], -1

je dijkstra\_endif2

add ax, word ptr[bx]

mov bx, distance

add bx, si

add bx, si

cmp ax, word ptr[bx]

jnb dijkstra\_endif2

mov word ptr[bx], ax

dijkstra\_endif2:

inc si

jmp dijkstra\_cycle3

equal:

dec cx

jmp dijkstra\_cycle1

dijkstra\_end2:

ret

dijkstra endp

main proc

mov ax, @data

mov ds, ax

pochatok:

puts dijk

puts keyboard\_in

puts file\_in

check\_input:

call \_getch

cmp al, '1'

je keyboard\_input

cmp al, '2'

je file\_input

cmp ah, 1

jne check\_input

\_exit

file\_input:

putc endl

puts str\_file

push offset file\_name

call gets

putc endl

putc endl

push RO

push offset file\_name

call fopen

add sp, 4

test ax, ax

jne file\_er

puts str\_error

jmp pochatok

file\_er:

mov handle, ax

push offset tmp

push handle

call ifin

add sp, 4

cmp bl, 0

je file\_open

puts str\_error

jmp pochatok

file\_open:

mov n, ax

cwd

mul ax

add ax, ax

mov len, ax

sub sp, ax

mov matrix, sp

puts str\_n

puts tmp

push handle

push n

push matrix

call m\_ifin

add sp, 6

push ax

putc endl

pop ax

cmp al, 0

je prisv

puts str\_error

add sp, len

jmp pochatok

keyboard\_input:

push offset str\_n

call icin

add sp, 2

mov n, ax

cwd

mul ax

add ax, ax

mov len, ax

sub sp, ax

mov matrix, sp

push sp

push n

call matrix\_input

add sp, 4

prisv:

mov ax, n

sub sp, ax

mov visited, sp

add ax, ax

sub sp, ax

mov distance, sp

push offset str\_start

poch\_versh:

call icin

dec ax

cmp ax, n

jb norm\_versh

puts str\_error

jmp poch\_versh

norm\_versh:

add sp, 2

mov beg, ax

putc endl

mov si, visited

mov di, distance

mov cx, n

my\_cycle:

cmp cx, 0

je alg\_beg

mov byte ptr[si], 0

mov word ptr[di], -1

inc si

dec cx

jmp my\_cycle

alg\_beg:

mov si, distance

mov ax, beg

add si, ax

add si, ax

mov word ptr[si], 0

push beg

call dijkstra

add sp, 2

xor cx, cx

mov bx, distance

vivid:

cmp cx, n

jnb kin

push 10

push offset tmp

push word ptr[bx]

call itoa

add sp, 6

puts tmp

putc endl

inc cx

add bx, 2

jmp vivid

kin:

add sp, len

add sp, n

add sp, n

add sp, n

getc

\_exit

main endp

end main

include src\stdio.inc

include src\ctype.inc

ifin proc c

arg @handle, @str

uses cx

push @handle

@@cycle1:

call fgetc

push ax

call isspace

mov bx, ax

pop ax

cmp bx, 0

jne @@cycle1

mov bx, @str

mov [bx], al

inc bx

@@cycle2:

call fgetc

push ax

call isspace

mov cx, ax

pop ax

cmp cx, 0

jne @@end\_fin

mov [bx], al

inc bx

jmp @@cycle2

@@end\_fin:

mov byte ptr[bx], 0

push @str

call atoi

add sp, 4

ret

ifin endp

icin proc c

arg @s

uses bx, dx

sub sp, 80

mov dx, sp

mov dx, sp

push dx

@@cycle:

puts @s

call gets

putc endl

call strlen

cmp ax, 0

je @@err

call atoi

cmp bl, 0

je @@end

@@err:

puts str\_error

jmp @@cycle

@@end:

add sp, 82

ret

icin endp

include io.inc

matrix\_input proc c

arg @n, @matrix

uses bx, si, di

mov bx, @matrix

xor si, si

@@cycle1:

cmp si, @n

je @@end

xor di, di

@@cycle2:

cmp di, @n

je @@ns

push offset str\_matr1

push offset m\_out

call strcpy

add sp, 4

push 10

push offset tmp

push si

call itoa

add sp, 6

push offset tmp

push offset m\_out

call strcat

add sp, 4

push offset str\_matr2

push offset m\_out

call strcat

add sp, 4

push 10

push offset tmp

push di

call itoa

add sp, 6

push offset tmp

push offset m\_out

call strcat

add sp, 4

push offset str\_matr3

push offset m\_out

call strcat

add sp, 4

push offset m\_out

call icin

add sp, 2

mov word ptr[bx], ax

add bx, 2

inc di

jmp @@cycle2

@@ns:

inc si

jmp @@cycle1

@@end:

ret

matrix\_input endp

m\_ifin proc c

arg @m, @n, @handle

uses bx, cx, dx, si, di

mov ax, n

cwd

mul ax

mov di, ax

sub sp, 80

mov dx, sp

mov si, @m

xor cx, cx

push dx

push @handle

@@cycle:

cmp cx, di

je @@end

call ifin

cmp bl, 0

je @@ns

add sp, 84

mov ax, 1

ret

@@ns:

mov [si], ax

add si, 2

inc cx

jmp @@cycle

@@end:

add sp, 84

putc endl

xor ax, ax

ret

m\_ifin endp

**Додаток Б Блок схема алгоритму**