Міністерство освіти і науки України
Нацональний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Лабораторна робота № 5
з дисципліни
«Дискретна математика»

Виконав:
Студент групи КН-114
Кратко Денис

Викладач:
Мельникова Н.І.

Львів – 2019

Тема: Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи
Мета роботи: набуття практичних вмінь та навичок з використання алгоритму Дейкстри.

Теоретичні відомості

Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G. Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри.

«Жадібними» називаються алгоритми, які на кожному кроці вибирають оптимальний із можливих варіантів.

Задача про найкоротший ланцюг. Алгоритм Дейкстри.

Дано n-вершинний граф $G=(V, E)$, у якому виділено пару вершин $v_0, v^* \in V$, і кожне ребро зважене числом $w(e) \geq 0$. Нехай $X=\{x\}$ – множина усіх простих ланцюгів, що з'єднують v_0 з v^*, $x=(V_x, E_x)$. Цільова функція $F(x)=\sum_{e \in E_x} w(e) \to \min$. Потрібно знайти найкоротший ланцюг, тобто $x_0 \in X: F(x_0)=\min_{x \in X} F(x)$

Перед описом алгоритму Дейкстри подамо визначення термінів "k-а найближча вершина і "дерево найближчих вершин". Перше з цих понять визначається індуктивно так.

1-й крок індукції. Нехай зафіксовано вершину x_0, E_1 – множина усіх ребер $e \in E$, інцидентних v_0. Серед ребер $e \in E_1$ вибираємо ребро $e(1)=(v_0, v_1)$, що має мінімальну вагу, тобто $w(e(1))=\min_{e \in E_1} w(e)$. Тоді v_1 називаємо першою найближчою вершиною (НВ), число $w(e(1))$ позначаємо $l(1) = l(v_1)$ і називаємо відстанню до цієї НВ. Позначимо $V_1=\{v_0, v_1\}$ – множину найближчих вершин.

2-й крок індукції. Позначимо E_2 – множину усіх ребер $e=(v', v'')$, $e \in E$, таких що $v' \in V_1$, $v'' \in (V \setminus V_1)$. Найближчим вершинам $v \in V_1$ приписано відстані $l(v)$ до кореня v_0, причому $l(v_0)=0$. Введемо позначення: $\overline{V_1}$ – множина таких вершин $v'' \in (V \setminus V_1)$, що $\exists$ ребра виду $e=(v, v'')$, де $v \in V_1$. Для всіх ребер $e \in E_2$ знаходимо таке ребро $e_2=(v', v_2)$, що величина $l(v')+w(e_2)$ найменша. Тоді v_2 називається другою найближчою вершиною, а ребра e_1, e_2 утворюють зростаюче дерево для виділених найближчих вершин $D_2=\{e_1, e_2\}$.

(s+1)-й крок індукції. Нехай у результаті s кроків виділено множину найближчих вершин $Vs=\{v_0, v_1, \ldots, v_s\}$ і відповідне їй зростаюче дерево $D_s=\{e_1, e_2, \ldots, e_s\}$... Для кожної вершини $v \in V_s$ обчислена відстань $l(v)$ від кореня v_0 до v; $\overline{V_s}$ – множина вершин $v \in (V \setminus V_s)$, для яких існують ребра вигляду $e=(v_t, v)$, де $v_t \in V_s$, $v \in (V \setminus V_s)$. На кроці s+1 для кожної вершини $v_t \in V_s$ обчислюємо відстань до вершини v_t : $L(s+1)(v_r) = l(v_r) + \min_{v^* \in \overline{V_s}} w(v_r, v^*)$, де min береться по всіх ребрах $e=(v_t, v*)$, $v^* \in \overline{V_s}$, після чого знаходимо min серед величин $L(s+1)(v_t)$. Нехай цей min досягнуто для вершин v_{r0} і відповідної їй $v^* \in \overline{V_s}$, що назвемо v_{s+1}. Тоді вершину v_{s+1} називаємо (s+1)-ю НВ, одержуємо множину $V_{s+1}=V_s \cup v_{s+1}$ і зростаюче дерево $D_{s+1}=D_s \cup (v_{r0}, v_{s+1})$. (s+1)-й крок завершується перевіркою: чи є чергова НВ v_{s+1} відзначеною вершиною, що повинна бути за умовою задачі зв'язано найкоротшим ланцюгом з вершиною v_0. Якщо так, то довжина шуканого ланцюга дорівнює $l(v_{s+1})=l(v_{r0})+w(v_{r0}, v_{s+1})$; при цьому шуканий ланцюг однозначно відновлюється з ребер зростаючого дерева D_{s+1}. У противному випадку випливає перехід до кроку s+2.

Плоскі і планарні графи

Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається *планарним*, якщо він є ізоморфним плоскому графу.

Гранню плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані.

Алгоритм γ укладання графа G являє собою процес послідовного приєднання до деякого укладеного підграфа $\widetilde{G}$ графа G нового ланцюга, обидва кінці якого належать $\widetilde{G}$. При цьому в якості початкового плоского графа $\widetilde{G}$ вибирається будь-який простий цикл графа G. Процес продовжується доти, поки не буде побудовано плоский граф, ізоморфний графові G, або приєднання деякого ланцюга виявиться неможливим. В останньому випадку граф G не є планарним.

Нехай побудоване деяке укладання підграфа $\widetilde{G}$ графа G.

Сегментом S відносно $\widetilde{G}$ будемо називати підграф графа G одного з наступних виглядів:

- ребро $e \in E$, $e = (u, v)$, таке, що $e \notin \widetilde{E}$, $u, v \in \widetilde{V}$, $\widetilde{G} = (\widetilde{V}, \widetilde{E})$;
- зв'язний компонент графа $G - \widetilde{G}$, доповнений всіма ребрами графа G, інцидентними вершинам узятого компонента, і кінцями цих ребер.

Вершину v сегмента S відносно $\widetilde{G}$ будемо називати *контактною*, якщо $v \in \widetilde{V}$.

Припустимою гранню для сегмента S відносно $\widetilde{G}$ називається грань Г графа $\widetilde{G}$, що містить усі контактні вершини сегмента S. Через Г(S) будемо позначати множину припустимих граней для S.

Назвемо α-*ланцюгом* простий ланцюг L сегмента S, що містить дві різні контактні вершини і не містить інших контактних вершин.

Тепер формально опишемо алгоритм γ.

0. Виберемо деякий простий цикл C графа G і укладемо його на площині; покладемо $\widetilde{G} = G$.

1. Знайдемо грані графа $\widetilde{G}$ і сегменти відносно $\widetilde{G}$. Якщо множина сегментів порожня, то перейдемо до пункту 7.
2. Для кожного сегмента S визначимо множину Г(S).
3. Якщо існує сегмент S, для якого Г(S)=∅, то граф G не планарний. Кінець. Інакше перейдемо до п. 4.
4. Якщо існує сегмент S, для якого міститься єдина припустима грань Г, то перейдемо до п. 6. Інакше до п. 5.
5. Для деякого сегмента S Г(S)>1. У цьому випадку вибираємо довільну припустиму грань Г.
6. Розмістимо довільний α- ланцюг L∈S у грань Г; замінимо $\widetilde{G}$ на $\widetilde{G} \cup L$ і перейдемо до п. 1.
7. Побудовано укладання $\widetilde{G}$ графа G на площині. Кінець.

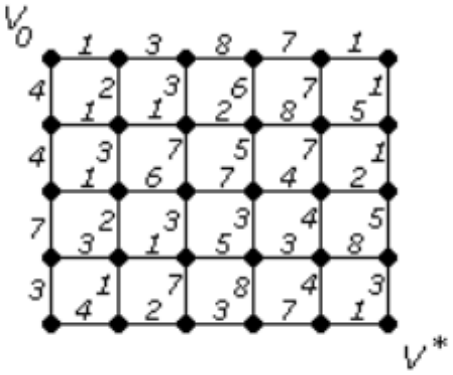
Кроком алгоритму γ будемо вважати приєднання до $\widetilde{G}$ α- ланцюга L.

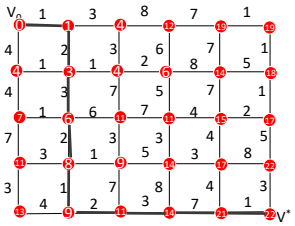
Завдання (12 варіант)

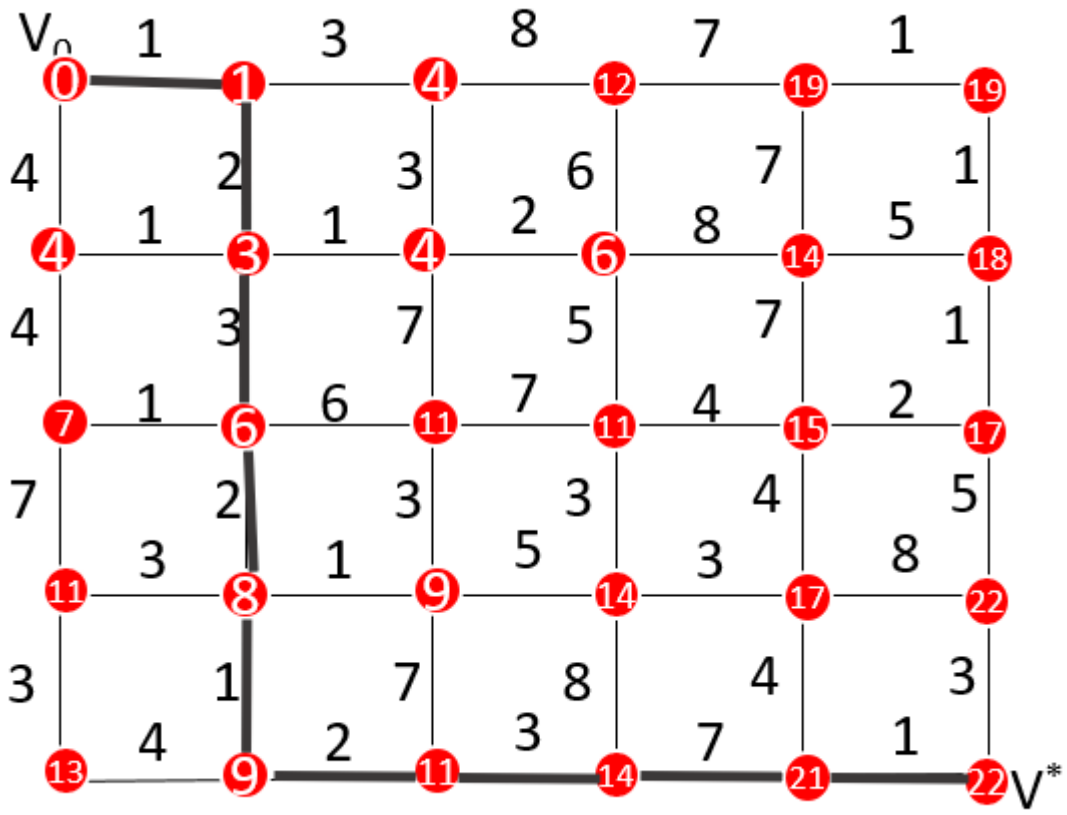
Додаток 1

1. За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі поміж парою вершин V_0 і V*.

12



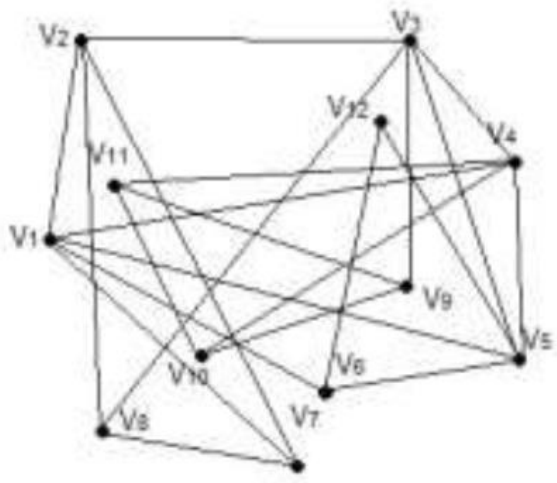


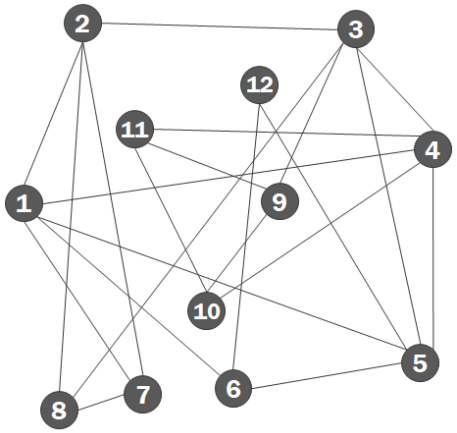


Довжина мінімального шляху – 22.

2. За допомогою γ-алгоритма зробити укладку графа у площині або довести, що вона неможлива.

12

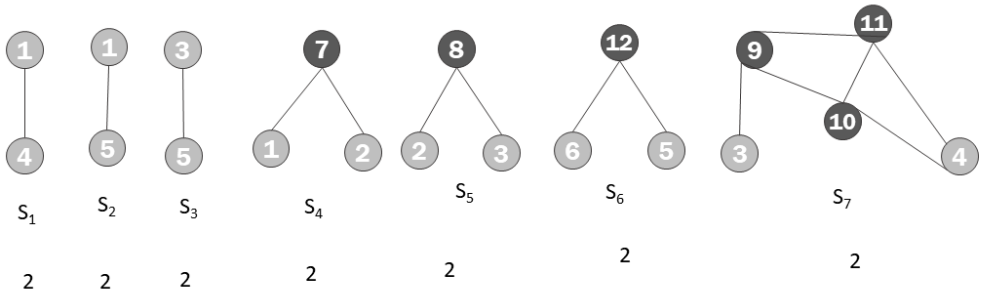




Граф зв’язний
Граф містить хоча б один цикл
Граф не містить мостів

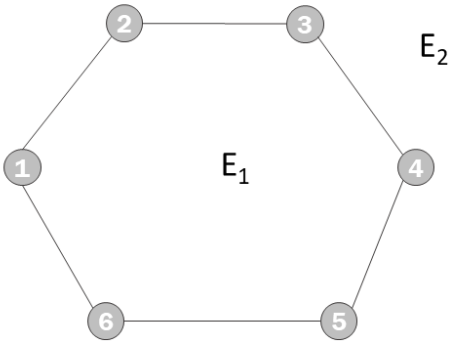
Отже, можна скористатися гамма-алгоритмом укладання графа.

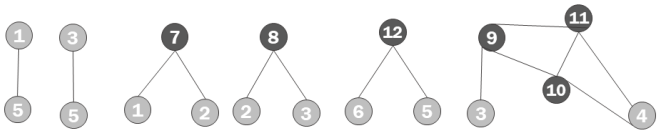
Обираємо будь-який простий цикл.



S_1 S_2 S_3 S_4 S_5 S_6 S_7

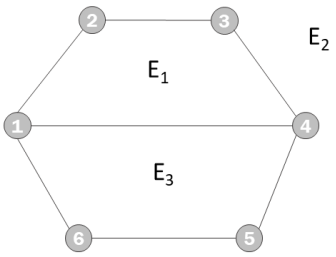
2 2 2 2 2 2 2

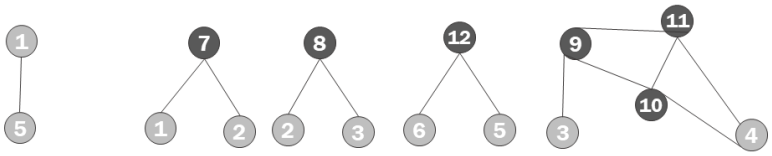




S_2 S_3 S_4 S_5 S_6 S_7

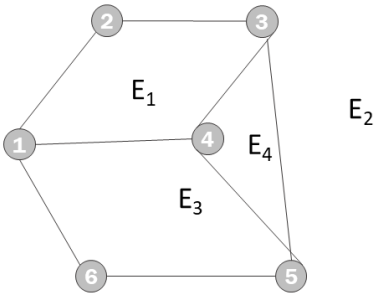
2 1 2 2 2 2

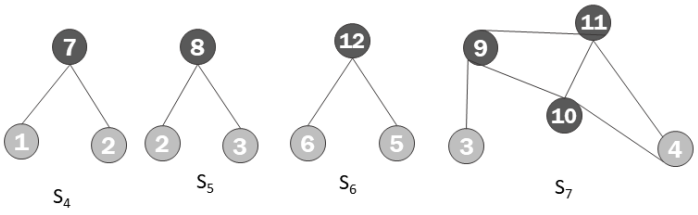




S_2 S_4 S_5 S_6 S_7

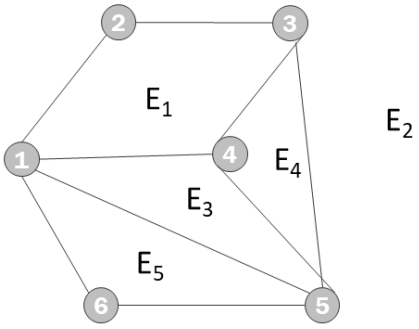
2 2 2 2 2

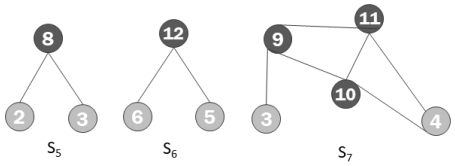




S_4 S_5 S_6 S_7

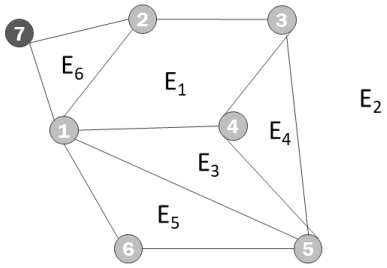
2 2 2 2

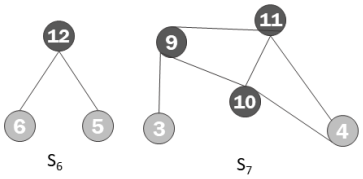




S_5 S_6 S_7

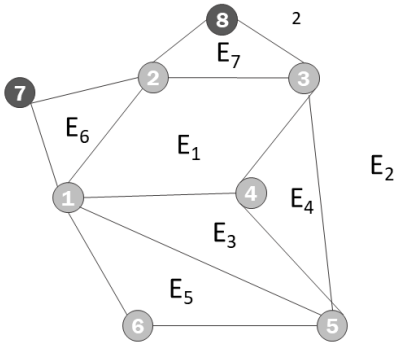
2 2 2

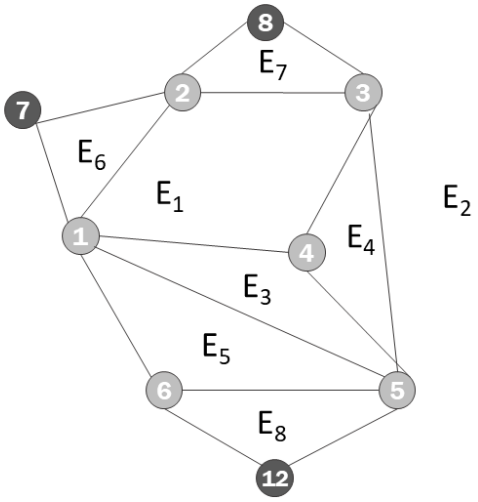


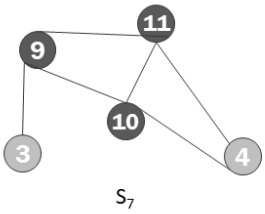


S_6 S_7

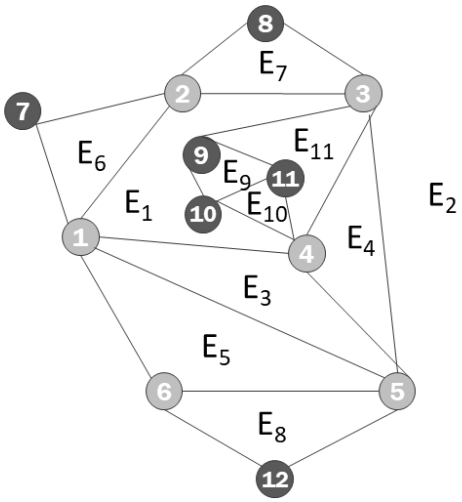
2 2







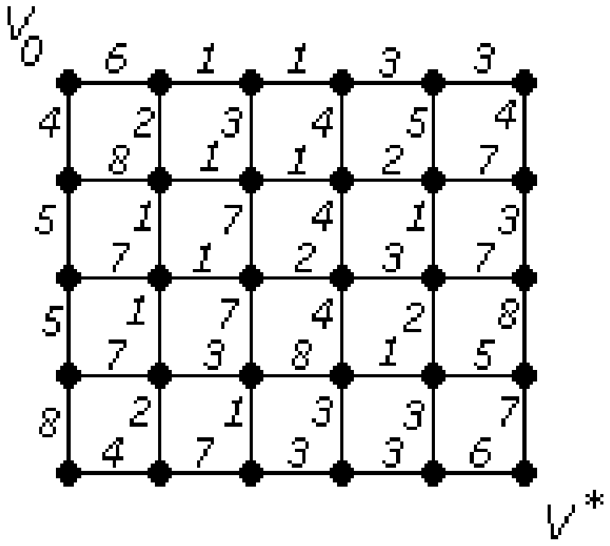
2



Додаток 2

Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.

12



Розв’язок

```
#include <bits/stdc++.h>

using namespace std;

const short int EXCEPTION_GRAPH_VERTEXOUTOFRANGE = 1;
const short int PARSE_ERROR = 2;

typedef struct {
    unsigned long int a;
    unsigned long int b;
    long int w;
} neograph_edge;
typedef neograph_edge* neograph_matrix_elem;
typedef neograph_matrix_elem* neograph_matrix_row;
typedef neograph_matrix_row* neograph_matrix;
typedef struct {
    neograph_matrix matrix;
    unsigned long int vertex_count;
    neograph_edge** edges;
    unsigned long int edge_count;
} neograph;
typedef struct {
    unsigned long int a;
    unsigned long int b;
    long int w;
} parsedinp;

neograph_edge* NeographEdge(unsigned long int a, unsigned long int b, long int w) {
    neograph_edge* new_edge = (neograph_edge*)malloc(sizeof(neograph_edge));
    new_edge->a = a;
    new_edge->b = b;
    new_edge->w = w;
    return new_edge;
};

neograph* Neograph(unsigned long int n_vertices) {
    neograph_matrix new_matrix = (neograph_matrix)calloc(n_vertices,
sizeof(neograph_matrix_row));
```

```
    unsigned long j;
    for (unsigned long int i = 0; i < n_vertices; i++) {
        new_matrix[i] = (neograph_matrix_row)calloc(n_vertices,
sizeof(neograph_matrix_elem));
        for (j = 0; j < n_vertices; j++) {
            new_matrix[i][j] = NULL;
        }
    };
    neograph* new_neograph = (neograph*)malloc(sizeof(neograph));
    new_neograph->matrix = new_matrix;
    new_neograph->vertex_count = n_vertices;
    new_neograph->edges = (neograph_edge**)calloc(255, sizeof(neograph_edge*));
    new_neograph->edge_count = 0;
    return new_neograph;
};

void remove_neograph(neograph* ng) {
    free(ng);
};

void print_edge(neograph_edge* e) {
    cout << "(a=" << e->a + 1 << ", b=" << e->b + 1 << ", weight=" << e->w << ")";
};

void print_edge_array(neograph* ng) {
    printf("Edge array: [");
    for (unsigned long i = 0; i < ng->edge_count; i++) {
        print_edge(ng->edges[i]);
        printf(", ");
    };
    printf("]\n");
};

bool add_edge(neograph* p_gr, neograph_edge* e) {
    if ((e->a < p_gr->vertex_count) && (e->b < p_gr->vertex_count)) {
        if (p_gr->matrix[e->a][e->b] == NULL) {
            p_gr->edge_count++;
            p_gr->edges[p_gr->edge_count - 1] = e;
            p_gr->matrix[e->a][e->b] = e;
            p_gr->matrix[e->b][e->a] = e;
            return true;
        } else {
            return false;
        };
    }
    else {
        throw EXCEPTION_GRAPH_VERTEXOUTOFRANGE;
    };
};

parsedinp parse_input(char* s) {
    parsedinp result;
    if (sscanf(s, "%u %u %li", &result.a, &result.b, &result.w) == 3) {
        return result;
    }
    else {
        throw PARSE_ERROR;
    };
};

neograph* read_neograph(unsigned long int n_vertices, FILE* f) {
    neograph* new_neograph = Neograph(n_vertices);
```

```
    char* temp = (char*)calloc(255, sizeof(char));
    parsedinp parsed;
    neograph_edge* debug_t;
    while (true) {
        try {
            parsed = parse_input(fgets(temp, 255, f));
            debug_t = NeographEdge(parsed.a - 1, parsed.b - 1, parsed.w);
            add_edge(new_neograph, debug_t);
        } catch (const short int e) {
            if (e == PARSE_ERROR) {
                break;
            }
            else {
                printf("[Invalid vertex number]\n");
            };
        };
    };
    return new_neograph;
};

neograph* fileinput_neograph(const char* filename) {
    FILE* f = fopen(filename, "r");
    unsigned long int n_vertices;
    fscanf(f, "%u\n", &n_vertices);
    neograph* new_neograph = read_neograph(n_vertices, f);
    fclose(f);
    return new_neograph;
};

void print_neograph_matrix(neograph* ng) {
    printf("Neograph:\n");
    unsigned long int i, j;
    printf("    | ");
    for (i = 0; i < ng->vertex_count; i++) {
        printf("%3u  ", i + 1);
    };
    printf("\n----+-");
    for (i = 0; i < ng->vertex_count; i++) {
        printf("-----");
    };
    printf("\n");
    for (i = 0; i < ng->vertex_count; i++) {
        printf("%3u |", i + 1);
        for (j = 0; j < ng->vertex_count; j++) {
            printf("  ");
            if (ng->matrix[i][j] != NULL) {
                printf("1");
            }
            else {
                printf("0");
            };
            printf("  ");
        };
        printf("\n");
    };
};

void swap(neograph_edge** xp, neograph_edge** yp) {
    neograph_edge* temp = *xp;
    *xp = *yp;
    *yp = temp;
};
```

```
int main() {
    printf("[ Algorytm Deikstry ]\n");
    // import neograph
    char* filename = new char[256];
    printf("Enter filename (don't use spaces):\n");
    cin >> filename;
    neograph* a = fileinput_neograph(filename);

    print_neograph_matrix(a);

    print_edge_array(a);

    bool* closed = (bool*)calloc(a->vertex_count, sizeof(bool));
    unsigned long int inf = 10000;
    unsigned long int* distances = (unsigned long int*)calloc(a->vertex_count,
sizeof(unsigned long int));
    for (unsigned long int i=0; i < a->vertex_count; i++) {
        closed[i] = false;
        distances[i] = inf;
    };
    distances[0] = 0;
    unsigned long int min_vertex_i;
    unsigned long int vertices_left = a->vertex_count;
    unsigned long int j=0;
    while (vertices_left) {
        min_vertex_i = a->vertex_count - 1;
        for (j=0; j<a->vertex_count; j++) { // шукаємо чергову вершину
            if (!closed[j] && (distances[j] < distances[min_vertex_i])) {
                min_vertex_i = j;
            };
        };

        for (j=0; j<a->vertex_count; j++) {
            if (!closed[j] && (a->matrix[min_vertex_i][j] != NULL)) { // перевіряємо,
чи сусід і чи переглянута
                distances[j] = min(distances[min_vertex_i] + a-
>matrix[min_vertex_i][j]->w, distances[j]);
            };
        };

        closed[min_vertex_i] = true;
        vertices_left--;
    };

    printf("Distances:\n");
    for (unsigned long int i=0; i < a->vertex_count; i++) {
        printf("v%u = %u, ", i + 1, distances[i]);
    };
    printf("\n");

    printf("Edges of path (reversed):\n");
    unsigned long int vertex_i = a->vertex_count - 1;
    unsigned long int min_distance_vertex_i;
    while (vertex_i) {
        min_distance_vertex_i = vertex_i;
        for (j = 0; j < a->vertex_count; j++) {
            if ((a->matrix[j][vertex_i] != NULL) && (distances[j] <
distances[min_distance_vertex_i])) {
                min_distance_vertex_i = j;
            };
        };
```

```
        print_edge(a->matrix[min_distance_vertex_i][vertex_i]);
        printf(", ");
        vertex_i = min_distance_vertex_i;
    };
    printf("\n");

    remove_neograph(a);
    free(closed);
    free(distances);

    return 0;
};
```

Результат виконання програми

1 випадок

Зчитуємо граф з файлу file.txt (граф з умови)

```
30
1 2 6
2 3 1
3 4 1
4 5 3
5 6 3
7 8 8
8 9 1
9 10 1
10 11 2
11 12 7
13 14 7
14 15 1
15 16 2
16 17 3
17 18 7
19 20 7
20 21 3
21 22 8
22 23 1
23 24 5
25 26 4
26 27 7
27 28 3
28 29 3
29 30 6
1 7 4
7 13 5
13 19 5
19 25 8
2 8 2
8 14 1
14 20 1
20 26 2
3 9 3
9 15 7
15 21 7
21 27 1
4 10 4
10 16 4
16 22 4
```

```
22 28 3
5 11 5
11 17 1
17 23 2
23 29 3
6 12 4
12 18 3
18 24 8
24 30 7
```

```
"D:\Dyskretna laboratorni\5\cmake-build-debug\5.exe"
[ Algorytm Deikstry ]
Enter filename (don't use spaces):
file.txt
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
17   18   19   20   21   22   23
  24   25   26   27   28   29   30
----+-------------------------------------------------------------------------------
-----------------------------------
-----------------------------------
  1 |  0    1    0    0    0    0    1    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  2 |  1    0    1    0    0    0    0    1    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  3 |  0    1    0    1    0    0    0    0    1    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  4 |  0    0    1    0    1    0    0    0    0    1    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  5 |  0    0    0    1    0    1    0    0    0    0    1    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  6 |  0    0    0    0    1    0    0    0    0    0    0    1    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  7 |  1    0    0    0    0    0    0    1    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  8 |  0    1    0    0    0    0    1    0    1    0    0    0    0    1    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  9 |  0    0    1    0    0    0    0    1    0    1    0    0    0    0    1    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
 10 |  0    0    0    1    0    0    0    0    1    0    1    0    0    0    0    1
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
 11 |  0    0    0    0    1    0    0    0    0    1    0    1    0    0    0    0
1    0    0    0    0    0    0
  0    0    0    0    0    0    0
 12 |  0    0    0    0    0    1    0    0    0    0    1    0    0    0    0    0
0    1    0    0    0    0    0
  0    0    0    0    0    0    0
 13 |  0    0    0    0    0    0    1    0    0    0    0    0    0    1    0    0
0    0    1    0    0    0    0
  0    0    0    0    0    0    0
```

```
 14 |  0    0    0    0    0    0    0    1    0    0    0    0    1    0    1    0
0    0    0    1    0    0    0
  0    0    0    0    0    0    0
 15 |  0    0    0    0    0    0    0    0    1    0    0    0    0    1    0    1
0    0    0    0    1    0    0
  0    0    0    0    0    0    0
 16 |  0    0    0    0    0    0    0    0    0    1    0    0    0    0    1    0
1    0    0    0    0    1    0
  0    0    0    0    0    0    0
 17 |  0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    1
0    1    0    0    0    0    1
  0    0    0    0    0    0    0
 18 |  0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
1    0    0    0    0    0    0
  1    0    0    0    0    0    0
 19 |  0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
0    0    0    1    0    0    0
  0    1    0    0    0    0    0
 20 |  0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
0    0    1    0    1    0    0
  0    0    1    0    0    0    0
 21 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
0    0    0    1    0    1    0
  0    0    0    1    0    0    0
 22 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
0    0    0    0    1    0    1
  0    0    0    0    1    0    0
 23 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1    0    0    0    0    1    0
  1    0    0    0    0    1    0
 24 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    1    0    0    0    0    1
  0    0    0    0    0    0    1
 25 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    1    0    0    0    0
  0    0    1    0    0    0    0
 26 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    1    0    0    0
  0    1    0    1    0    0    0
 27 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    1    0    0
  0    0    1    0    1    0    0
 28 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    1    0
  0    0    0    1    0    1    0
 29 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    1
  0    0    0    0    1    0    1
 30 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  1    0    0    0    0    1    0
Edge array: [(a=1, b=2, weight=6), (a=2, b=3, weight=1), (a=3, b=4, weight=1), (a=4,
b=5, weight=3), (a=5, b=6, weight=3
), (a=7, b=8, weight=8), (a=8, b=9, weight=1), (a=9, b=10, weight=1), (a=10, b=11,
weight=2), (a=11, b=12, weight=7), (a
=13, b=14, weight=7), (a=14, b=15, weight=1), (a=15, b=16, weight=2), (a=16, b=17,
weight=3), (a=17, b=18, weight=7), (a
=19, b=20, weight=7), (a=20, b=21, weight=3), (a=21, b=22, weight=8), (a=22, b=23,
weight=1), (a=23, b=24, weight=5), (a
=25, b=26, weight=4), (a=26, b=27, weight=7), (a=27, b=28, weight=3), (a=28, b=29,
weight=3), (a=29, b=30, weight=6), (a
=1, b=7, weight=4), (a=7, b=13, weight=5), (a=13, b=19, weight=5), (a=19, b=25,
```

```
weight=8), (a=2, b=8, weight=2), (a=8, b
=14, weight=1), (a=14, b=20, weight=1), (a=20, b=26, weight=2), (a=3, b=9, weight=3),
(a=9, b=15, weight=7), (a=15, b=21
, weight=7), (a=21, b=27, weight=1), (a=4, b=10, weight=4), (a=10, b=16, weight=4),
(a=16, b=22, weight=4), (a=22, b=28,
 weight=3), (a=5, b=11, weight=5), (a=11, b=17, weight=1), (a=17, b=23, weight=2),
(a=23, b=29, weight=3), (a=6, b=12, w
eight=4), (a=12, b=18, weight=3), (a=18, b=24, weight=8), (a=24, b=30, weight=7), ]
Distances:
v1 = 0, v2 = 6, v3 = 7, v4 = 8, v5 = 11, v6 = 14, v7 = 4, v8 = 8, v9 = 9, v10 = 10,
v11 = 12, v12 = 18, v13 = 9, v14 = 9
, v15 = 10, v16 = 12, v17 = 13, v18 = 20, v19 = 14, v20 = 10, v21 = 13, v22 = 16, v23
= 15, v24 = 20, v25 = 16, v26 = 12
, v27 = 14, v28 = 17, v29 = 18, v30 = 24,
Edges of path (reversed):
(a=29, b=30, weight=6), (a=23, b=29, weight=3), (a=17, b=23, weight=2), (a=11, b=17,
weight=1), (a=10, b=11, weight=2),
(a=4, b=10, weight=4), (a=3, b=4, weight=1), (a=2, b=3, weight=1), (a=1, b=2,
weight=6),

Process finished with exit code 0
```

2 випадок

Зчитуємо граф з файлу file1.txt (граф з задачі 1 додатку 1)

```
30
1 2 1
2 3 3
3 4 8
4 5 7
5 6 1
7 8 1
8 9 1
9 10 2
10 11 8
11 12 5
13 14 1
14 15 6
15 16 7
16 17 4
17 18 2
19 20 3
20 21 1
21 22 5
22 23 3
23 24 8
25 26 4
26 27 2
27 28 3
28 29 7
29 30 1
1 7 4
7 13 4
13 19 7
19 25 3
2 8 2
8 14 3
14 20 2
20 26 1
```

```
3 9 3
9 15 7
15 21 3
21 27 7
4 10 6
10 16 5
16 22 3
22 28 8
5 11 7
11 17 7
17 23 4
23 29 4
6 12 1
12 18 1
18 24 5
24 30 3
```

```
"D:\Dyskretna laboratorni\5\cmake-build-debug\5.exe"
[ Algorytm Deikstry ]
Enter filename (don't use spaces):
file1.txt
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
17   18   19   20   21   22   23
  24   25   26   27   28   29   30
----+--------------------------------------------------------------------------------
-----------------------------------
-----------------------------------
  1 |  0    1    0    0    0    0    1    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  2 |  1    0    1    0    0    0    0    1    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  3 |  0    1    0    1    0    0    0    0    1    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  4 |  0    0    1    0    1    0    0    0    0    1    0    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  5 |  0    0    0    1    0    1    0    0    0    0    1    0    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  6 |  0    0    0    0    1    0    0    0    0    0    0    1    0    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  7 |  1    0    0    0    0    0    0    1    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  8 |  0    1    0    0    0    0    1    0    1    0    0    0    0    1    0    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  9 |  0    0    1    0    0    0    0    1    0    1    0    0    0    0    1    0
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
 10 |  0    0    0    1    0    0    0    0    1    0    1    0    0    0    0    1
0    0    0    0    0    0    0
  0    0    0    0    0    0    0
 11 |  0    0    0    0    1    0    0    0    0    1    0    1    0    0    0    0
1    0    0    0    0    0    0
```

```
  0    0    0    0    0    0    0
 12 |  0    0    0    0    0    1    0    0    0    0    1    0    0    0    0    0
0    1    0    0    0    0    0
  0    0    0    0    0    0    0
 13 |  0    0    0    0    0    0    1    0    0    0    0    0    0    1    0    0
0    0    1    0    0    0    0
  0    0    0    0    0    0    0
 14 |  0    0    0    0    0    0    0    1    0    0    0    0    1    0    1    0
0    0    0    1    0    0    0
  0    0    0    0    0    0    0
 15 |  0    0    0    0    0    0    0    0    1    0    0    0    0    1    0    1
0    0    0    0    1    0    0
  0    0    0    0    0    0    0
 16 |  0    0    0    0    0    0    0    0    0    1    0    0    0    0    1    0
1    0    0    0    0    1    0
  0    0    0    0    0    0    0
 17 |  0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    1
0    1    0    0    0    0    1
  0    0    0    0    0    0    0
 18 |  0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
1    0    0    0    0    0    0
  1    0    0    0    0    0    0
 19 |  0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
0    0    0    1    0    0    0
  0    1    0    0    0    0    0
 20 |  0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
0    0    1    0    1    0    0
  0    0    1    0    0    0    0
 21 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
0    0    0    1    0    1    0
  0    0    0    1    0    0    0
 22 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
0    0    0    0    1    0    1
  0    0    0    0    1    0    0
 23 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1    0    0    0    0    1    0
  1    0    0    0    0    1    0
 24 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    1    0    0    0    0    1
  0    0    0    0    0    0    1
 25 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    1    0    0    0    0
  0    0    1    0    0    0    0
 26 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    1    0    0    0
  0    1    0    1    0    0    0
 27 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    1    0    0
  0    0    1    0    1    0    0
 28 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    1    0
  0    0    0    1    0    1    0
 29 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    1
  0    0    0    0    1    0    1
 30 |  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0
  1    0    0    0    0    1    0
Edge array: [(a=1, b=2, weight=1), (a=2, b=3, weight=3), (a=3, b=4, weight=8), (a=4,
b=5, weight=7), (a=5, b=6, weight=1
), (a=7, b=8, weight=1), (a=8, b=9, weight=1), (a=9, b=10, weight=2), (a=10, b=11,
weight=8), (a=11, b=12, weight=5), (a
```

```
=13, b=14, weight=1), (a=14, b=15, weight=6), (a=15, b=16, weight=7), (a=16, b=17,
weight=4), (a=17, b=18, weight=2), (a
=19, b=20, weight=3), (a=20, b=21, weight=1), (a=21, b=22, weight=5), (a=22, b=23,
weight=3), (a=23, b=24, weight=8), (a
=25, b=26, weight=4), (a=26, b=27, weight=2), (a=27, b=28, weight=3), (a=28, b=29,
weight=7), (a=29, b=30, weight=1), (a
=1, b=7, weight=4), (a=7, b=13, weight=4), (a=13, b=19, weight=7), (a=19, b=25,
weight=3), (a=2, b=8, weight=2), (a=8, b
=14, weight=3), (a=14, b=20, weight=2), (a=20, b=26, weight=1), (a=3, b=9, weight=3),
(a=9, b=15, weight=7), (a=15, b=21
, weight=3), (a=21, b=27, weight=7), (a=4, b=10, weight=6), (a=10, b=16, weight=5),
(a=16, b=22, weight=3), (a=22, b=28,
 weight=8), (a=5, b=11, weight=7), (a=11, b=17, weight=7), (a=17, b=23, weight=4),
(a=23, b=29, weight=4), (a=6, b=12, w
eight=1), (a=12, b=18, weight=1), (a=18, b=24, weight=5), (a=24, b=30, weight=3), ]
Distances:
v1 = 0, v2 = 1, v3 = 4, v4 = 12, v5 = 19, v6 = 19, v7 = 4, v8 = 3, v9 = 4, v10 = 6,
v11 = 14, v12 = 18, v13 = 7, v14 = 6
, v15 = 11, v16 = 11, v17 = 15, v18 = 17, v19 = 11, v20 = 8, v21 = 9, v22 = 14, v23 =
17, v24 = 22, v25 = 13, v26 = 9, v
27 = 11, v28 = 14, v29 = 21, v30 = 22,
Edges of path (reversed):
(a=29, b=30, weight=1), (a=28, b=29, weight=7), (a=27, b=28, weight=3), (a=21, b=27,
weight=7), (a=20, b=21, weight=1),
(a=14, b=20, weight=2), (a=8, b=14, weight=3), (a=2, b=8, weight=2), (a=1, b=2,
weight=1),

Process finished with exit code 0
```