Міністерство освіти і науки України
Нацональний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Лабораторна робота № 4
з дисципліни
«Дискретна математика»

Виконав:
Студент групи КН-114
Кратко Денис

Викладач:
Мельникова Н.І.

Львів – 2019

Тема: Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Пріма-Краскала

Мета роботи: набуття практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.

Теоретичні відомості

Теорія графів дає простий, доступний і потужний інструмент побудови моделей прикладних задач, є ефективним засобом формалізації сучасних інженерних і наукових задач у різних областях знань.

Графом G називається пара множин (V, E), де V – множина вершин, перенумерованих числами 1, 2, ..., n = v, V = {v}, E − множина упорядкованих або неупорядкованих пар e = (v′,v″), v′∈V, v″∈V, називаних дугами або ребрами, E ={e}. При цьому немає примусового значення, як вершини розташовані в просторі або площині і які конфігурації мають ребра.

Неорієнтованим графом G називається граф, у якого ребра не мають напрямку. Такі ребра описуються неупорядкованою парою (v′,v″).

Орієнтований граф (орграф) – це граф ребра якого мають напрямок та можуть бути описані упорядкованою парою (v′,v″).

Упорядковане ребро називають дугою.

Граф є змішаним, якщо наряду з орієнтованими ребрами (дугами) є також і неорієнтовані. При

розв’язку задач змішаний граф зводиться до орграфа.

Кратними (паралельними) називаються ребра, які зв’язують одні і ті ж вершини. Якщо ребро виходить та й входить у дну і ту саму вершину, то таке ребро називається петлею.

Мультиграф – граф, який має кратні ребра.

Псевдограф – граф, який має петлі.

Простий граф – граф, який не має кратних ребер та петель.

Будь-яке ребро інцидентне двом вершинам (v′,v″), які воно з’єднує. У свою чергу вершини (v′,v″) інцендентні до ребра e. Дві вершини (v′,v″) називають суміжними, якщо вони належать до одного й того самого ребра e, і несуміжні у протилежному випадку.

Два ребра називають суміжними, якщо вони мають спільну вершину.

Відношення суміжності як для вершин, так і для ребер є симетричним відношенням.

Степенем вершини графа G називається число інцидентних їй ребер.

Граф, який не має ребер, називається пустим графом, нульграфом.

Вершина графа, яка не інцедентна до жодного ребра, називається ізольованою.

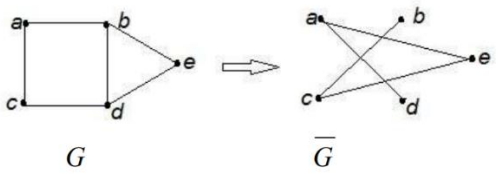
Вершина графа, яка інцедентна тільки до одного ребра, називається звисаючою.

Частина G’=(V’, E’) графа G={V, E} називається підграфом графа G , якщо V′ ⊆V і E′ складається з тих і тільки тих ребер e = (v′,v″), у яких обидві кінцеві вершини v′,v″∈V′. Частина G’={V’, E’}називається суграфом або остовим підграфом графа G, якщо виконано умови: V′ =V , E′ ⊆ E .

Операції над графами

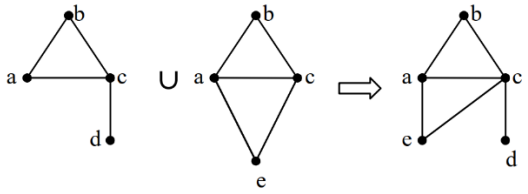
1. Вилученням ребра e (e∈ E) з графа G=(V, E) – є така операція, внаслідок якої отримаємо новий граф G1, для якого G1 =(V,E \{e}).

2. Доповненням графа G=(V, E) називається граф G=(V, E'), якщо він має одну і ту саму кількість вершин та дві його вершини суміжні тоді і тільки тоді, коли вони не суміжні в G (тобто ребро (v_i ,v_j)∈E′ тоді, коли (v_i,v_j)∉E). Наприклад:



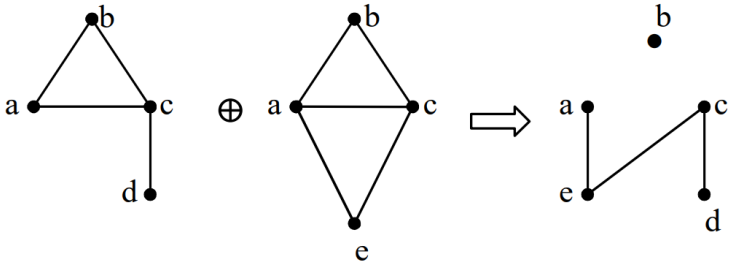
G $\overline{G}$

3. Об'єднанням графів $G_1=(V_1, E_1)$ та $G_2=(V_2, E_2)$ називається граф $G=(V, E)=G_1 \cup G_2$, у якому V =V1 ∪V2 та E = E1∪ E2 . Наприклад:



G_1 G_2 $G_1 \cup G_2$

4. Кільцевою сумою графів $G_1=(V_1, E_1)$ та $G_2=(V_2, E_2)$ називається граф G = (V, E) = G1 ⊕G2 у якому V=V1 ∪V2 та E = E1ΔE2 = (E1 ∪ E2) \ (E1 ∩ E2). Наприклад:

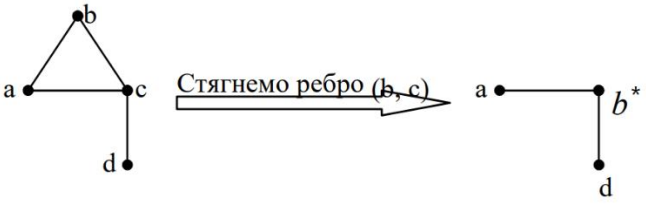


G_1 G_2 $G_1 \oplus G_2$

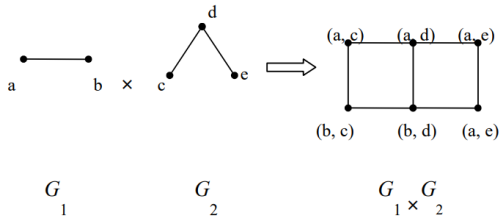
5. Розщеплення (роздвоєння) вершини графа. Нехай v - вершина графа G = (V, E). Множину усіх суміжних з нею вершин довільним чином розділимо на дві множини $N_1(v)$ та $N_2(v)$, таких що $N_1(\upsilon) \bigcup N_2(\upsilon) = V$. Видаливши вершину v разом з інцидентними їй ребрами, додамо дві нові вершини v_1 та v_2, які з'єднані ребром (v_1, v_2). Вершину v_1 з'єднаємо ребром з кожною вершиною множини $N_1(v)$, а вершину v_2– з кожною вершиною множини $N_2(v)$. Таким чином з графа G отримаємо новий граф G_v^*. Виконана операція називається розщепленням вершини v. Наприклад:

6. Стягування ребра (дуги). Ця операція означає видалення ребра та ототожнення його суміжних вершин. Граф G_1 стягується до графа G_2 , якщо граф G_2 може бути отриманим з G_1 в

результаті деякої послідовності стягування ребер (дуг). Наприклад:



7. Добутком графів $G_1=(V_1, E_1)$ та $G_2=(V_2, E_2)$ називається граф $G = G_1 \times G_2$ у якого $V = V_1 \times V_2$ а множина ребер визначається наступним чином: вершини (u_1, v_1), (u_2, v_2), суміжні у G тоді і тільки тоді коли $u_1 = u_2$ і v_1 та v_2 суміжні у G_2 , або $v_1 = v_2$ і u_1, u_2 суміжні у G_1. Наприклад:



Таблицею (матрицею) суміжності $R = [r_{ij}]$ графа G=(V, E) називається квадратна матриця порядку n (n – число вершин графа), елементи якої $r_{i,j}$ (i=1, 2, …n; j=1,2, …n) визначаються наступним чином:

$$r_{ij} = \begin{cases} 1, \text{ якщо існує дуга з } \upsilon_i \text{ в } \upsilon_j; \\ 0, \text{ в іншому випадку.} \end{cases}$$

Матриця суміжності повністю визначає структуру графа.

Ексцентриситет вершини графа – відстань до максимально віддаленої від неї вершини. Для графа, для якого не визначена вага його ребер, відстань визначається у вигляді числа ребер.

Радіус графа – мінімальний ексцентриситет вершин.

Діаметр графа – максимальний ексцентриситет вершин.

Діаметром зв’язного графа називається максимально можлива довжина між двома його вершинами.

Нехай дано неорієнтований граф G=(V, E). Маршрутом довжини j –1 з вершини

v_1 у v_j називається послідовність $M = \{(\upsilon_1,\upsilon_2), (\upsilon_2,\upsilon_3),...,(\upsilon_i,\upsilon_{i+1}), ..., (\upsilon_{j-1},\upsilon_j)\}$, яка складається з ребер $j = (v_s,v_{s+1}) \in E$, при цьому кожні два сусідніх ребра мають спільну кінцеву вершину.

Маршрут називається ланцюгом, якщо всі його ребра різні.

Відкритий ланцюг називається шляхом, якщо всі його вершини різні.

Замкнений ланцюг називається циклом, якщо різні всі його вершини, за винятком кінцевих.

Шлях і цикл називаються гамільтоновими, якщо вони проходять через усі вершини графа.

Алгоритми знаходження найкоротшого кістякового дерева

Алгоритм Прима для даного n-вершинного графа G=(V, E) будує по кроках s=1, 2, …, l≤n–1 зростаюче дерево Ds=(Vs, Es), Vs⊆V, Es⊆E. S=1. Фіксуємо довільну вершину v_0, серед усіх ребер,

інцидентних вершині v_0 знаходимо найкоротше ребро $e_1=(v_0, v_1)$.

Покладемо, що $D_1=(V_1, E_1)$, $V_1=\{v_0, v_1\}$, $E_1=\{e_1\}$ і переходимо до кроку s =2.

Нехай здійснено $s<n-1$ кроків, у результаті чого в графі G виділено зростаюче дерево $Ds=(Vs, Es)$. Тоді на кроці (s+1) серед усіх ребер $e=(v',v'')$, таких що $v'\in Vs$, $v''\in(V \setminus Vs)$, знаходимо найкоротше ребро $es+1=(vr, vs+1)$ і приєднуємо його до дерева Ds, у результаті чого одержуємо дерево $D_{s+1}=(V_{s+1}, E_{s+1})$, $V_{s+1}= V_s \cup \{v_{s+1}\}$, $E_{s+1}= E_s \cup \{e_{s+1}\}$.

Алгоритм закінчує свою роботу в двох випадках: 1) результативно на кроці $s=n-1$ у випадку, якщо граф G зв'язний; 2) безрезультатно, якщо G – незв'язний граф.

Алгоритм Краскала. Перший етап – підготовчий, для даного графа G упорядковуються ребра $e \in E$ у послідовність $e_1, e_2, ..., e_m$, $m = |E|$, у порядку неспадання ваг цих ребер: $w(e_1) \le w(e_2) \le \ldots \le w(e_s) \le \ldots \le w(e_m)$.

Другий етап виконується по кроках $s=1, 2, \ldots, m0 \le m$ у такий спосіб. На кроках s=1, 2 ребра e1, e2 з послідовності офарблюються. На кожному наступному кроці s розглядається ребро e_s з послідовності, і воно офарблюється тоді і тільки тоді, коли не утворює циклу з ребрами, пофарбованими на попередніх кроках. У противному випадку ребро e_s умовно викреслюється з графа G = (V, E).

Алгоритм закінчує роботу на кроці s=m0, коли пофарбованим виявиться (n–1) по рахунку ребро e_s, $n = |V|$, тому що по необхідності n–1 пофарбованих ребер утворюють кістякове дерево n-вершинного графа.

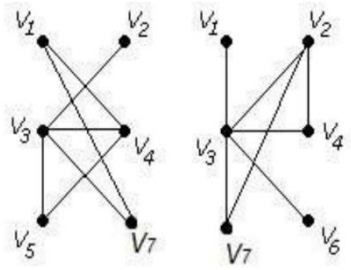
Завдання (12 варіант)

Додаток 1

1. **Виконати наступні операції над графами:**

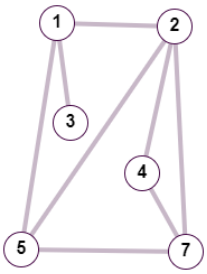
   1) знайти доповнення до першого графу,
   2) об'єднання графів,
   3) кільцеву суму G1 та G2 (G1+G2),
   4) розщепити вершину у другому графі,
   5) виділити підграф А, що складається з 3-х вершин в G1 і знайти стягнення А в G1 (G1\ А), 6) добуток графів.

12

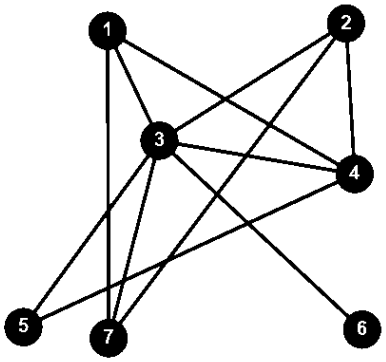


Розв'язування:

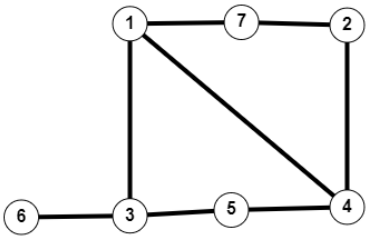
1. $\overline{G_1}$ – доповнення до G_1:



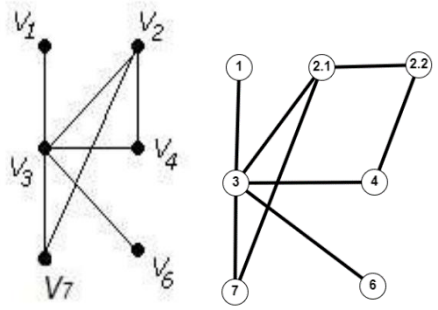
2. Об'єднання G_1 і G_2:



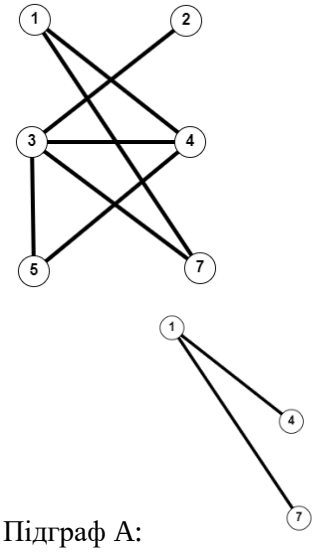
3. Кільцева сума G_1 і G_2:



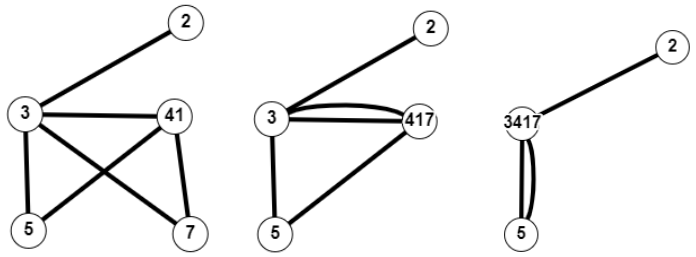
4. Розщепимо вершину V_2 у G_2 (з $V_{2.1}$ з'єднаємо V_3, V_7, з $V_{2.2}$ – V_4):



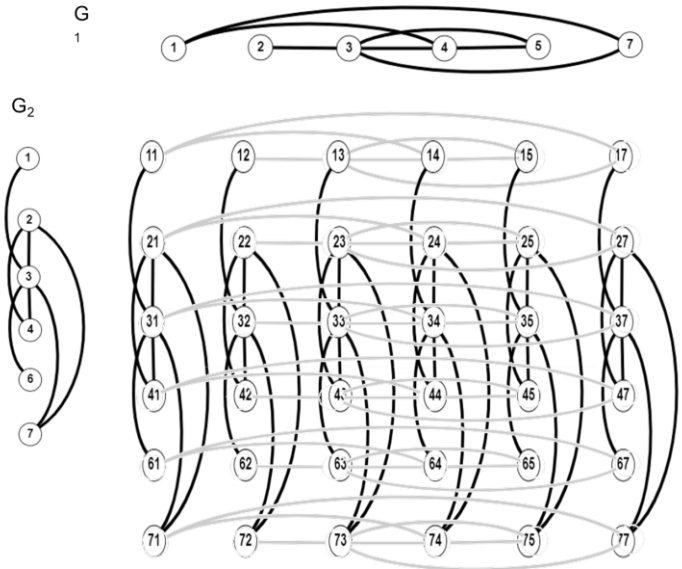
5. Виділити підграф A, що складається з трьох вершин, в G_1 і знайти стягнення A в G_1 (G_1 \ A).



Підграф A:

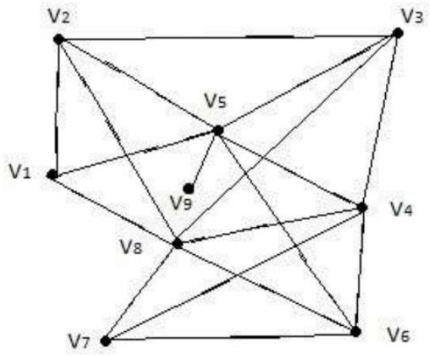


6. Добуток графів G_1 та G_2:



2. Знайти таблицю суміжності та діаметр графа

12



	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_1	0	1	0	0	1	0	0	1	0
V_2	1	0	1	0	1	0	0	1	0
V_3	0	1	0	1	1	0	0	1	0
V_4	0	0	1	0	1	1	1	1	0
V_5	1	1	1	1	0	1	0	0	1
V_6	0	0	0	1	1	0	1	1	0
V_7	0	0	0	1	0	1	0	1	0
V_8	1	1	1	1	0	1	1	0	0
V_9	0	0	0	0	1	0	0	0	0

Діаметр графа – максимальний ексцентриситет його вершин.

Побудуємо таблицю відстаней від вершини до вершини. За ваги ребер приймемо 1. В кінці рядка визначимо ексцентриситет вершини, що відповідає рядку.

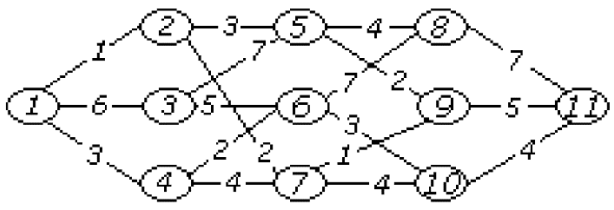
	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	e(v)
V_1	0	1	2	2	1	2	2	1	2	2
V_2	1	0	1	2	1	2	2	1	2	2
V_3	2	1	0	1	1	2	2	1	2	2
V_4	2	2	1	0	1	1	1	1	2	2
V_5	1	1	1	1	0	1	2	2	1	2
V_6	2	2	2	1	1	0	1	1	2	2
V_7	2	2	2	1	2	1	0	1	3	3
V_8	1	1	1	1	2	1	1	0	3	3
V_9	2	2	2	2	1	2	3	3	0	3

Отже, діаметр графа – 3.

3. Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

Граф:

12



Методом Краскала:

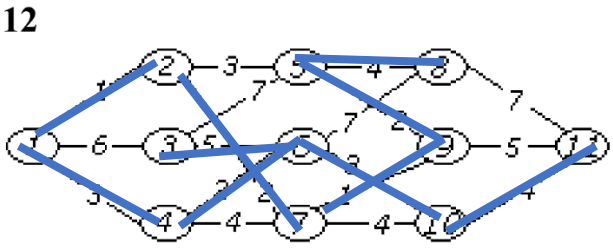
В порядку зростання ваг випишемо ребра графа:

(1, 2), (7, 9), (2, 7), (4, 6), (5, 9), (1, 4), (2, 5), (6, 10), (4, 7), (5, 8), (7, 10), (10, 11), (3, 6), (9, 11), (1, 3), (3, 5), (6, 8), (8, 11).

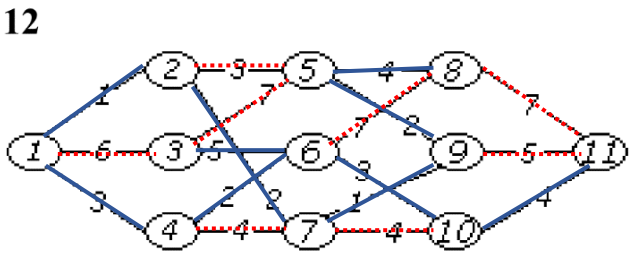
По черзі дивимося, чи не утвориться цикл, якщо додати ребро до графа B, якщо так, то додаємо. Виконуємо це, поки новий граф не міститиме n – 1 ребер, де n – кількість вершин у заданому графі.

n = 11

Ребра одержаного мінімального остового дерева: (1, 2), (7, 9), (2, 7), (4, 6), (5, 9), (1, 4), (6, 10), (5, 8), (10, 11), (3, 6).



Методом Прима:



Результат побудови наведений на рисунку.

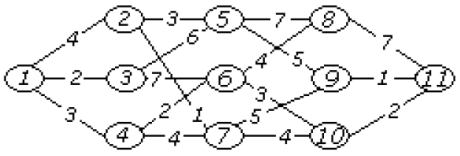
Додаток 2

Умова

Написати програму, яка реалізує алгоритм знаходження остового дерева мінімальної ваги згідно свого варіанту.

Варіант № 12

За алгоритмом Краскала знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:



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) {
    //printf("Neograph debug");
    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;
};

void sort_array_by_weight(neograph_edge** arr, unsigned long int n) {
    // bubblesort from https://www.geeksforgeeks.org/bubble-sort/
    unsigned long int i, j;
    for (i = 0; i < n-1; i++) {
        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++) {
            if (((arr[j])->w) > ((arr[j+1])->w)) {
                swap(arr + j, arr + (j+1));
            };
        };
    };
};

void print_stage_info(unsigned long int stage_number, neograph* ng) {
    printf("After stage %u: {\n", stage_number);
    print_neograph_matrix(ng);
    printf("}\n");
};

int main() {
    printf("[ Algorytm Kraskala ]\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);

    sort_array_by_weight(a->edges, a->edge_count);

    print_edge_array(a);

    neograph* b = Neograph(a->vertex_count);

    print_stage_info(0, b);

    unsigned long int j, old_id, new_id;
    unsigned long int* vertex_tree_ids = (unsigned long int*)calloc(a->vertex_count,
```

```
sizeof(unsigned long int));
    for (unsigned long int i = 0; i < a->vertex_count; i++) {
        vertex_tree_ids[i] = i;
    };
    for (unsigned long int i = 0; i < a->edge_count; i++) {
        printf("Edge ");
        print_edge(a->edges[i]);
        if (vertex_tree_ids[a->edges[i]->a] != vertex_tree_ids[a->edges[i]->b]) { //
add
            add_edge(b, a->edges[i]);
            printf(" added\n");
            old_id = vertex_tree_ids[a->edges[i]->b];
            new_id = vertex_tree_ids[a->edges[i]->a];
            for (j = 0; j < a->vertex_count; j++) {
                if (vertex_tree_ids[j] == old_id) {
                    vertex_tree_ids[j] = new_id;
                };
            };
        }
        else { // skip
            printf(" skipped\n");
        }
        print_stage_info(i + 1, b);
    };
    print_edge_array(b);

    remove_neograph(a);
    remove_neograph(b);
    free(vertex_tree_ids);

    return 0;
};
```

Результат виконання програми

Програма як вхідні дані приймає файл, де в першому рядку записано кількість вершин, у наступних – три числа (номер першої вершини, номер другої вершини, вага ребра) через пробіл. Коли програма зустрічає некоректний (у нашому випадку порожній) рядок, зчитування завершується.

1 випадок

file.txt, розташований в одній директорії з виконуваним файлом

```
11
1 2 4
1 3 2
1 4 3
2 5 3
2 7 1
3 5 6
3 6 7
4 6 2
4 7 4
5 8 7
5 9 5
6 8 4
6 10 3
7 9 5
7 10 4
8 11 7
```

```
9 11 1
10 11 2
```

```
"D:\Dyskretna laboratorni\4\cmake-build-debug\4.exe"
[ Algorytm Kraskala ]
Enter filename (don't use spaces):
file.txt
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    1    1    0    0    0    0    0
  4 |  1    0    0    0    0    1    1    0    0    0    0
  5 |  0    1    1    0    0    0    0    1    1    0    0
  6 |  0    0    1    1    0    0    0    1    0    1    0
  7 |  0    1    0    1    0    0    0    0    1    1    0
  8 |  0    0    0    0    1    1    0    0    0    0    1
  9 |  0    0    0    0    1    0    1    0    0    0    1
 10 |  0    0    0    0    0    1    1    0    0    0    1
 11 |  0    0    0    0    0    0    0    1    1    1    0
Edge array: [(a=1, b=2, weight=4), (a=1, b=3, weight=2), (a=1, b=4,
weight=3), (a=2, b=5, weight=3), (a=2, b=7, weight=1
), (a=3, b=5, weight=6), (a=3, b=6, weight=7), (a=4, b=6, weight=2),
(a=4, b=7, weight=4), (a=5, b=8, weight=7), (a=5, b
=9, weight=5), (a=6, b=8, weight=4), (a=6, b=10, weight=3), (a=7, b=9,
weight=5), (a=7, b=10, weight=4), (a=8, b=11, wei
ght=7), (a=9, b=11, weight=1), (a=10, b=11, weight=2), ]
Edge array: [(a=2, b=7, weight=1), (a=9, b=11, weight=1), (a=1, b=3,
weight=2), (a=4, b=6, weight=2), (a=10, b=11, weigh
t=2), (a=1, b=4, weight=3), (a=2, b=5, weight=3), (a=6, b=10,
weight=3), (a=1, b=2, weight=4), (a=4, b=7, weight=4), (a=
6, b=8, weight=4), (a=7, b=10, weight=4), (a=5, b=9, weight=5), (a=7,
b=9, weight=5), (a=3, b=5, weight=6), (a=3, b=6, w
eight=7), (a=5, b=8, weight=7), (a=8, b=11, weight=7), ]
After stage 0: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    0    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    0    0    0    0    0
  3 |  0    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    0    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    0    0    0    0    0    0    0    0
  7 |  0    0    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    0
 10 |  0    0    0    0    0    0    0    0    0    0    0
 11 |  0    0    0    0    0    0    0    0    0    0    0
}
Edge (a=2, b=7, weight=1) added
After stage 1: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    0    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  0    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    0    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    0    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
```

```
  9 |  0    0    0    0    0    0    0    0    0    0    0
 10 |  0    0    0    0    0    0    0    0    0    0    0
 11 |  0    0    0    0    0    0    0    0    0    0    0
}
Edge (a=9, b=11, weight=1) added
After stage 2: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    0    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  0    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    0    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    0    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    0
 11 |  0    0    0    0    0    0    0    0    1    0    0
}
Edge (a=1, b=3, weight=2) added
After stage 3: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    1    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    0    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    0    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    0
 11 |  0    0    0    0    0    0    0    0    1    0    0
}
Edge (a=4, b=6, weight=2) added
After stage 4: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    1    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    1    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    0
 11 |  0    0    0    0    0    0    0    0    1    0    0
}
Edge (a=10, b=11, weight=2) added
After stage 5: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+---------------------------------------------------------
  1 |  0    0    1    0    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  0    0    0    0    0    1    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    0    0
```

```
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=1, b=4, weight=3) added
After stage 6: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
----+-----------------------------------------------------
  1 |  0    0    1    1    0    0    0    0    0    0    0
  2 |  0    0    0    0    0    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    0    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=2, b=5, weight=3) added
After stage 7: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
----+-----------------------------------------------------
  1 |  0    0    1    1    0    0    0    0    0    0    0
  2 |  0    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    0    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    0    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=6, b=10, weight=3) added
After stage 8: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
----+-----------------------------------------------------
  1 |  0    0    1    1    0    0    0    0    0    0    0
  2 |  0    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=1, b=2, weight=4) added
After stage 9: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
----+-----------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
```

```
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=4, b=7, weight=4) skipped
After stage 10: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
-----+---------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    0    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    0    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=6, b=8, weight=4) added
After stage 11: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
-----+---------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=7, b=10, weight=4) skipped
After stage 12: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
-----+---------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=5, b=9, weight=5) skipped
After stage 13: {
Neograph:
     |   1    2    3    4    5    6    7    8    9   10   11
-----+---------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
```

```
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=7, b=9, weight=5) skipped
After stage 14: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+-------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=3, b=5, weight=6) skipped
After stage 15: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+-------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=3, b=6, weight=7) skipped
After stage 16: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+-------------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=5, b=8, weight=7) skipped
After stage 17: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+-------------------------------------------------------
```

```
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge (a=8, b=11, weight=7) skipped
After stage 18: {
Neograph:
    |   1    2    3    4    5    6    7    8    9   10   11
----+-----------------------------------------------------
  1 |  0    1    1    1    0    0    0    0    0    0    0
  2 |  1    0    0    0    1    0    1    0    0    0    0
  3 |  1    0    0    0    0    0    0    0    0    0    0
  4 |  1    0    0    0    0    1    0    0    0    0    0
  5 |  0    1    0    0    0    0    0    0    0    0    0
  6 |  0    0    0    1    0    0    0    1    0    1    0
  7 |  0    1    0    0    0    0    0    0    0    0    0
  8 |  0    0    0    0    0    1    0    0    0    0    0
  9 |  0    0    0    0    0    0    0    0    0    0    1
 10 |  0    0    0    0    0    1    0    0    0    0    1
 11 |  0    0    0    0    0    0    0    0    1    1    0
}
Edge array: [(a=2, b=7, weight=1), (a=9, b=11, weight=1), (a=1, b=3,
weight=2), (a=4, b=6, weight=2), (a=10, b=11, weigh
t=2), (a=1, b=4, weight=3), (a=2, b=5, weight=3), (a=6, b=10,
weight=3), (a=1, b=2, weight=4), (a=6, b=8, weight=4), ]

Process finished with exit code 0
```

2 випадок

file1.txt, розташований в одній директорії з виконуваним файлом

```
4
1 2 4
1 3 5
4 1 6
4 2 8
```

```
"D:\Dyskretna laboratorni\4\cmake-build-debug\4.exe"
[ Algorytm Kraskala ]
Enter filename (don't use spaces):
file1.txt
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    1    1    1
  2 |  1    0    0    1
  3 |  1    0    0    0
  4 |  1    1    0    0
Edge array: [(a=1, b=2, weight=4), (a=1, b=3, weight=5), (a=4, b=1, weight=6), (a=4,
b=2, weight=8), ]
Edge array: [(a=1, b=2, weight=4), (a=1, b=3, weight=5), (a=4, b=1, weight=6), (a=4,
b=2, weight=8), ]
After stage 0: {
```

```
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    0    0    0
  2 |  0    0    0    0
  3 |  0    0    0    0
  4 |  0    0    0    0
}
Edge (a=1, b=2, weight=4) added
After stage 1: {
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    1    0    0
  2 |  1    0    0    0
  3 |  0    0    0    0
  4 |  0    0    0    0
}
Edge (a=1, b=3, weight=5) added
After stage 2: {
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    1    1    0
  2 |  1    0    0    0
  3 |  1    0    0    0
  4 |  0    0    0    0
}
Edge (a=4, b=1, weight=6) added
After stage 3: {
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    1    1    1
  2 |  1    0    0    0
  3 |  1    0    0    0
  4 |  1    0    0    0
}
Edge (a=4, b=2, weight=8) skipped
After stage 4: {
Neograph:
    |   1    2    3    4
----+--------------------
  1 |  0    1    1    1
  2 |  1    0    0    0
  3 |  1    0    0    0
  4 |  1    0    0    0
}
Edge array: [(a=1, b=2, weight=4), (a=1, b=3, weight=5), (a=4, b=1, weight=6), ]

Process finished with exit code 0
```