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:

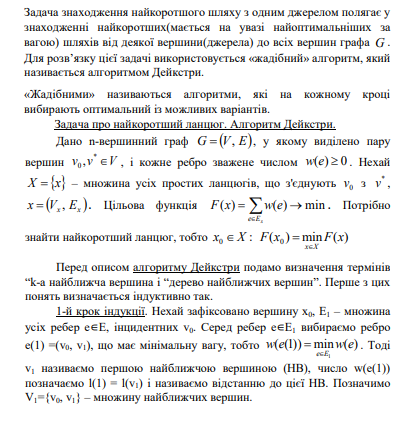
**Мельникова Н.І.**

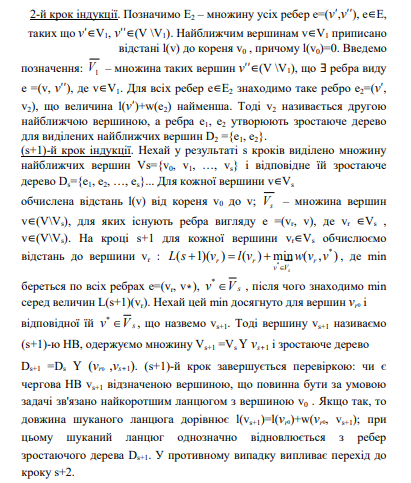
Львів – 2019

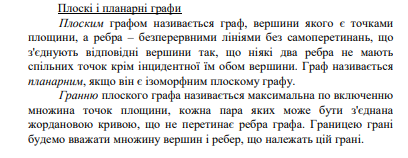
**Тема:** Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи

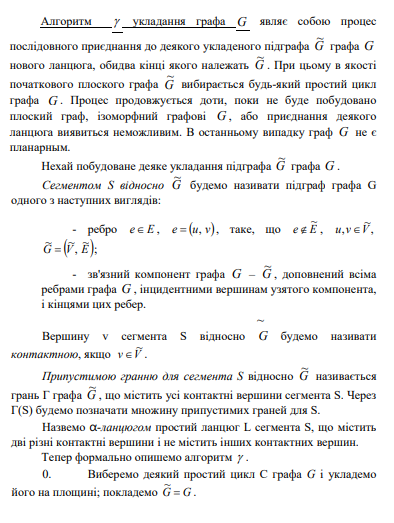
**Мета роботи:** набуття практичних вмінь та навичок з використання алгоритму Дейкстри.

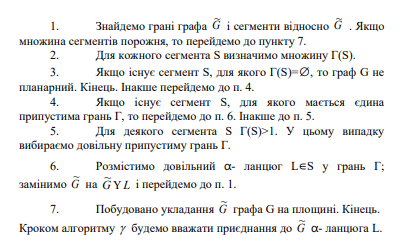
**Теоретичні відомості**

****

****

****

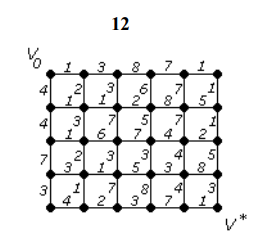
****

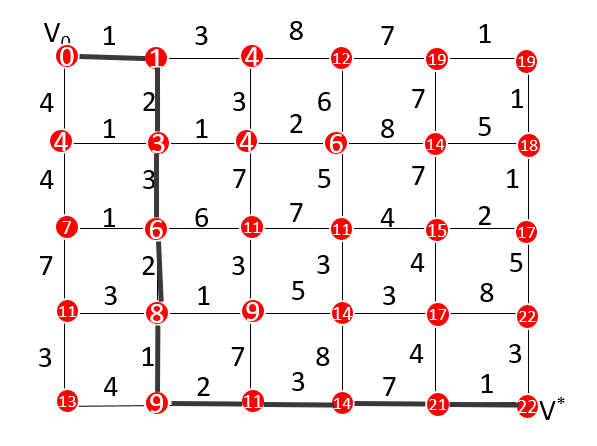
****

**Завдання (12 варіант)**

Додаток 1

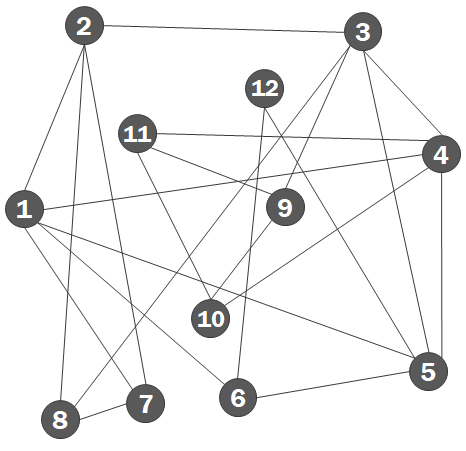
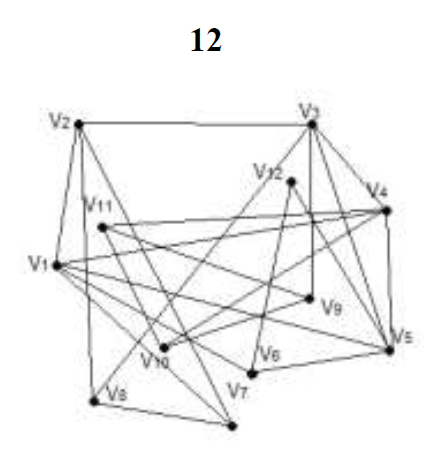
1. За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі поміж парою вершин V0 і V\*.





Довжина мінімального шляху – 22.

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



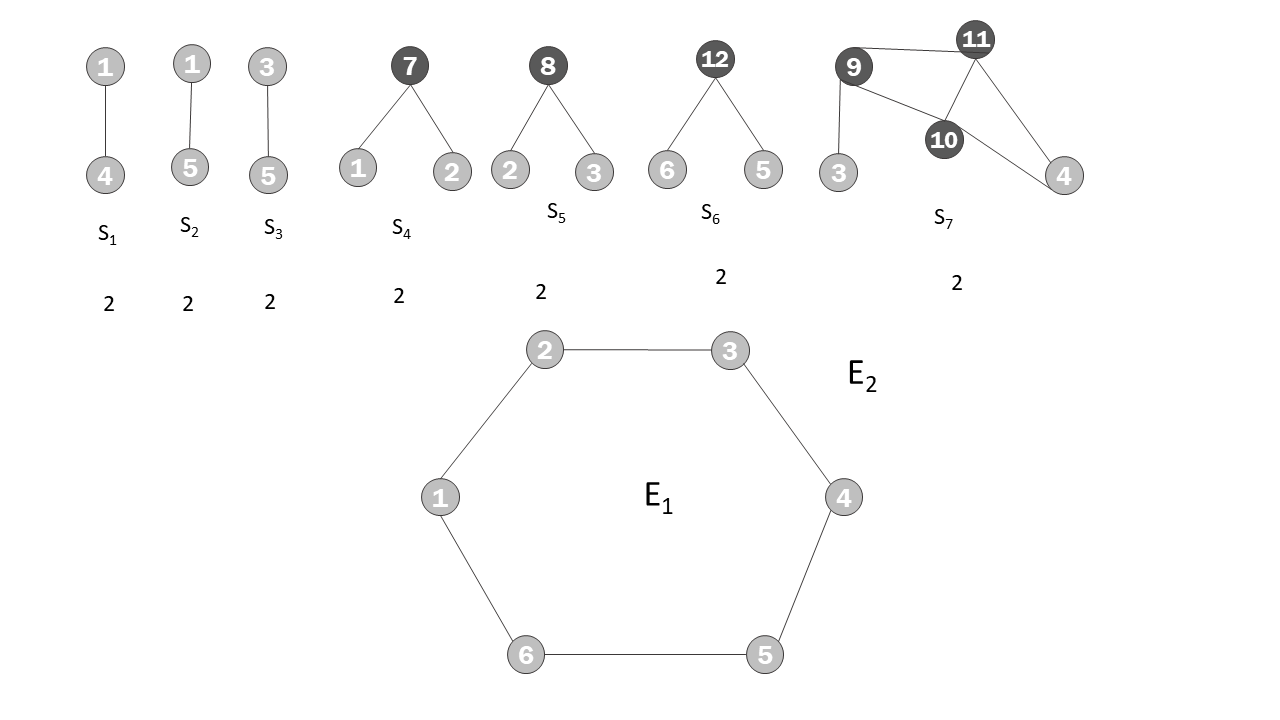
Граф зв’язний

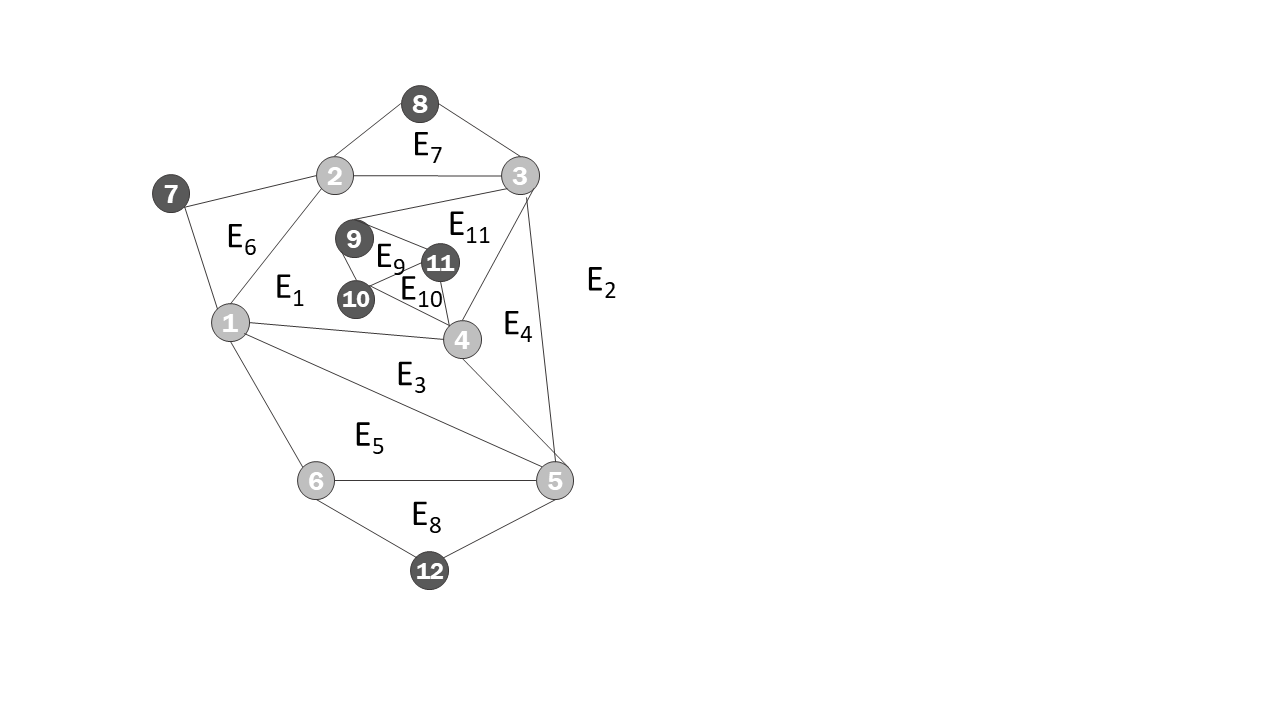
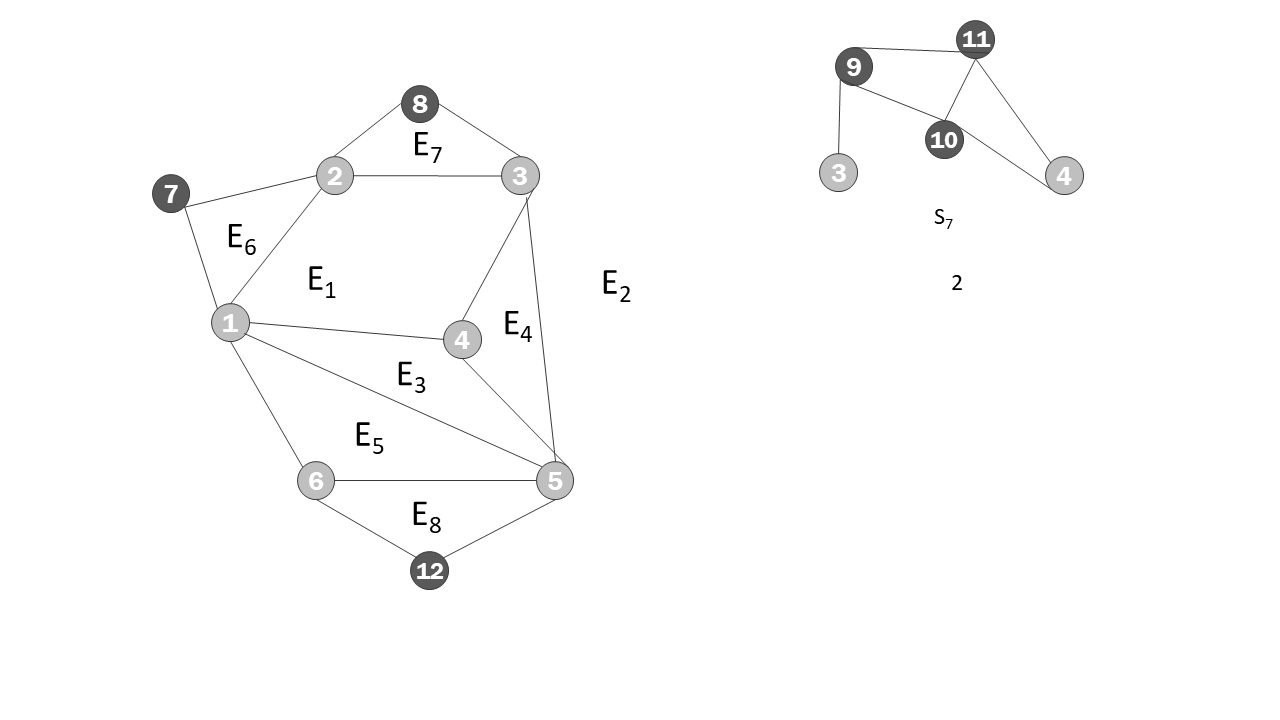
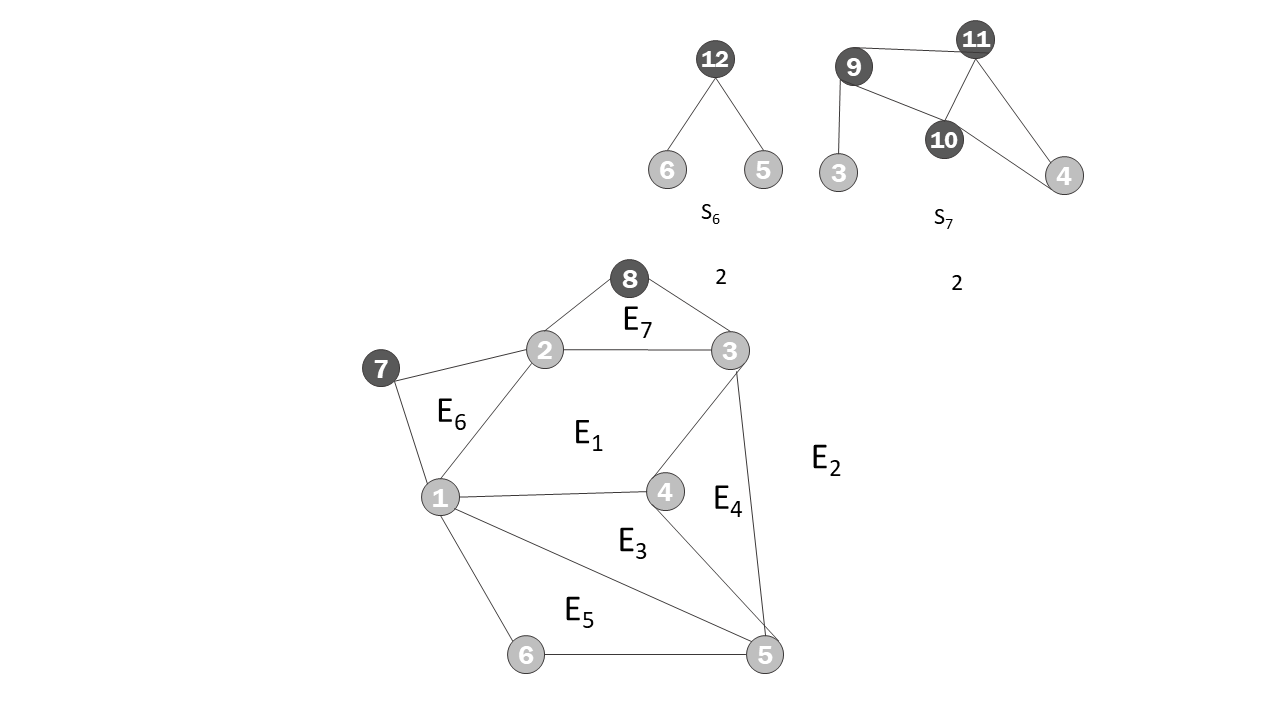
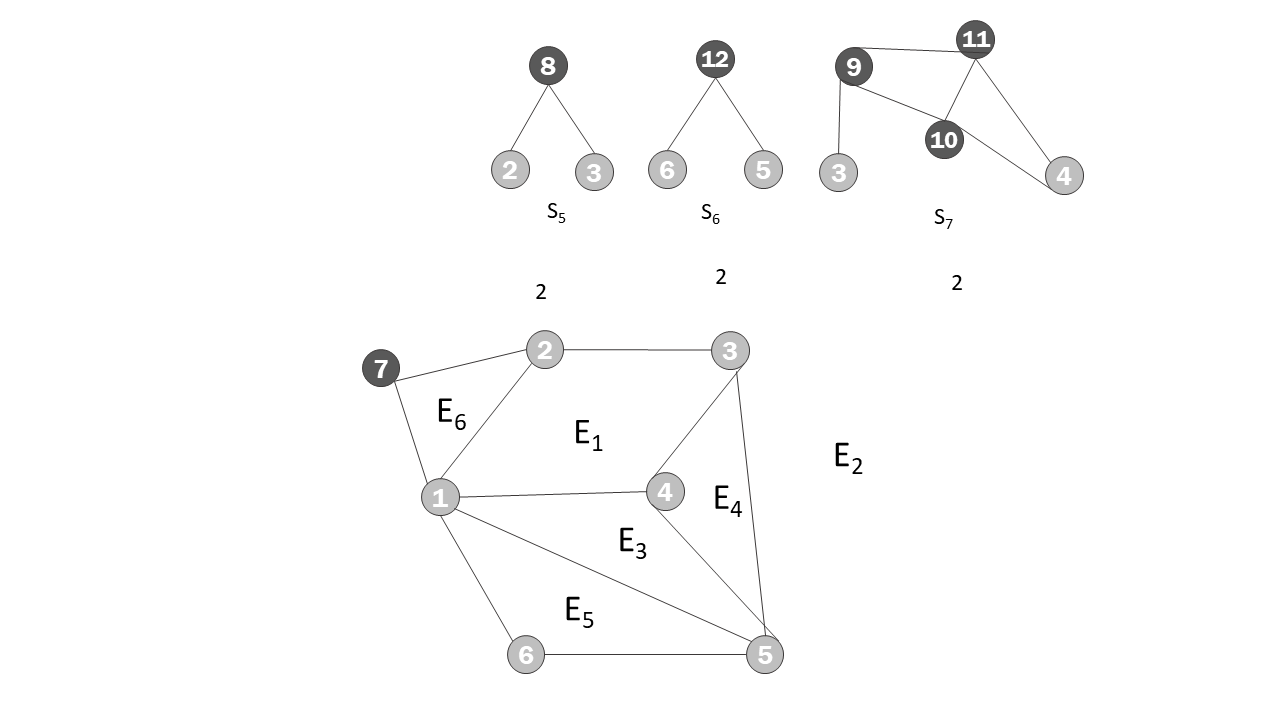
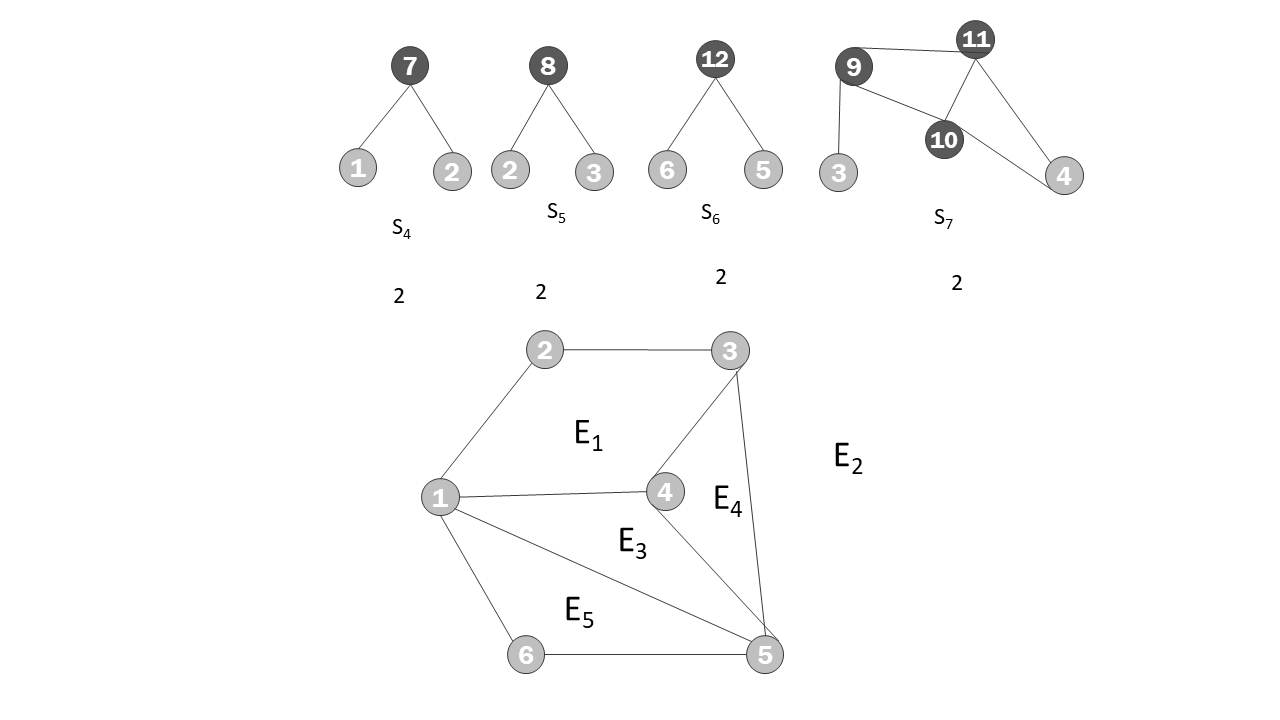
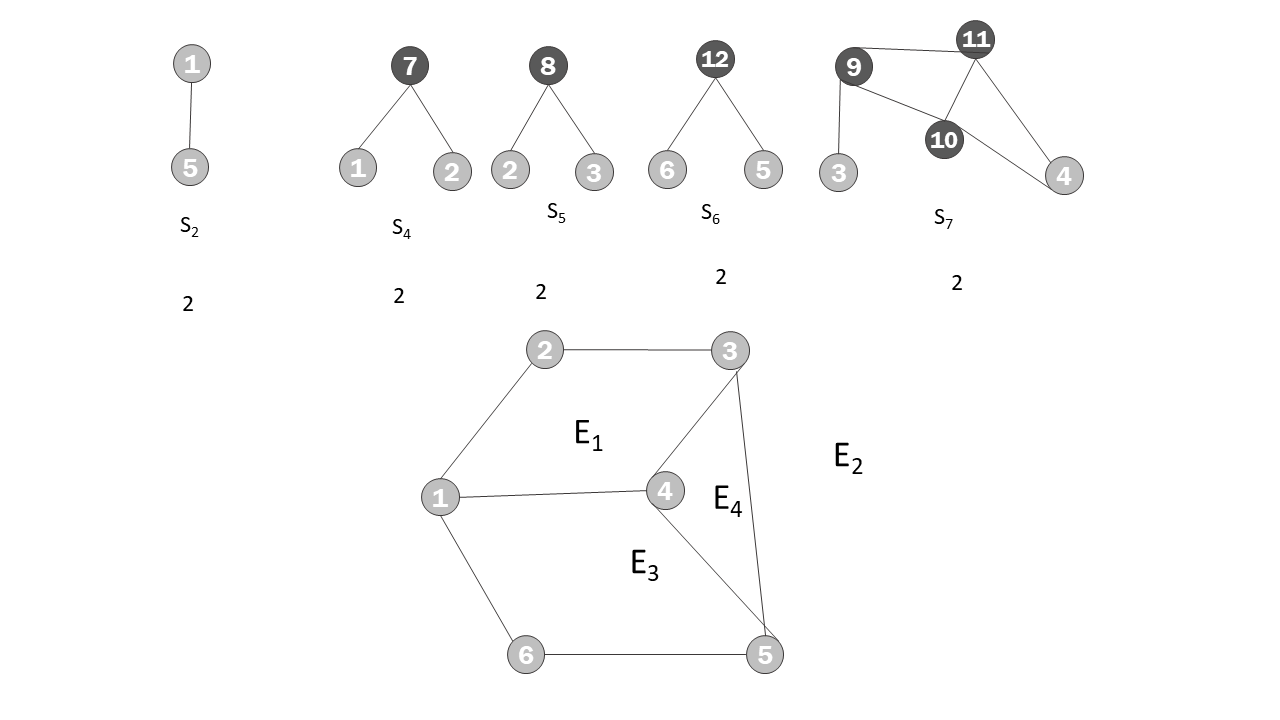
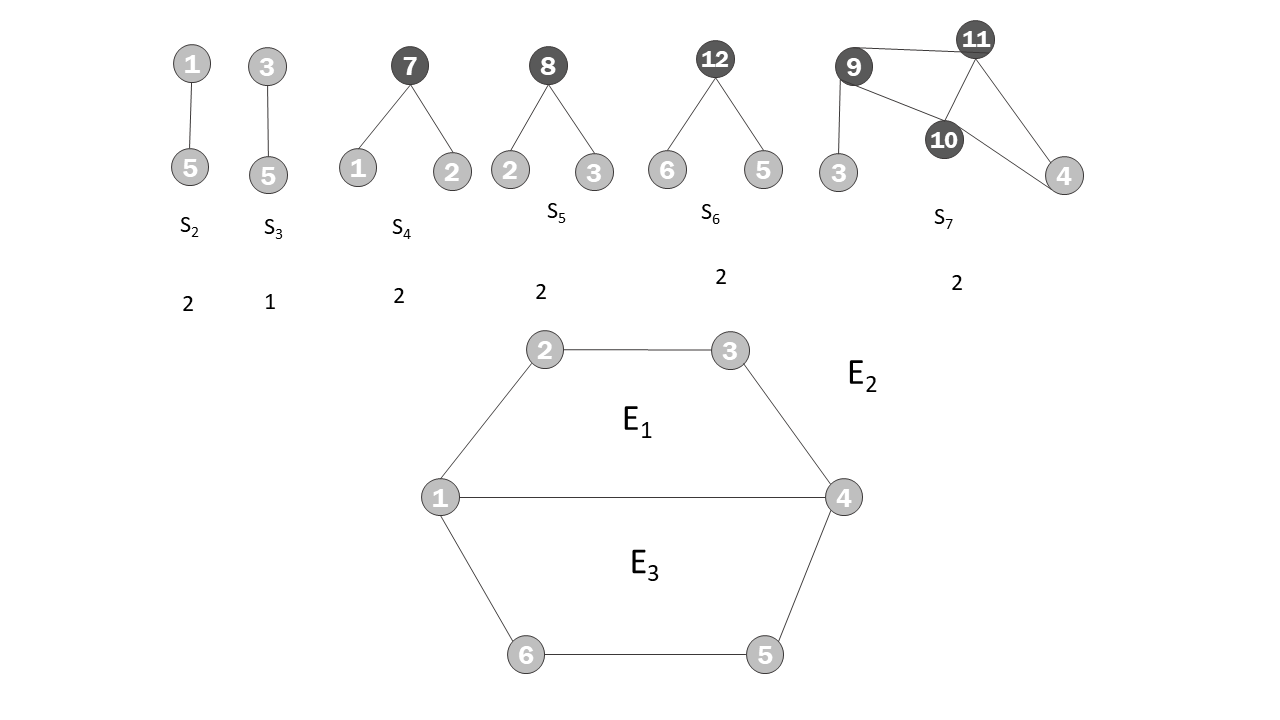
Граф містить хоча б один цикл

Граф не містить мостів

Отже, можна скористатися гамма-алгоритмом укладання графа.

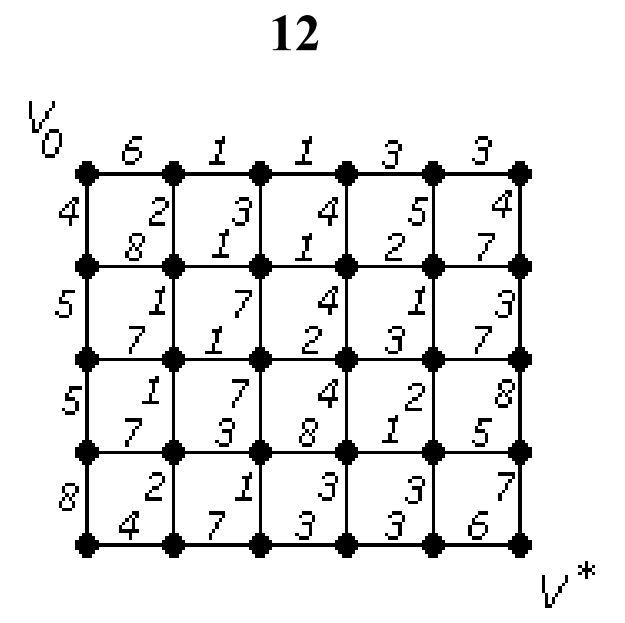
Обираємо будь-який простий цикл.





Додаток 2

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



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