МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ В НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахунково-графічні завдання

з дисципліни
«Дискретна математика»

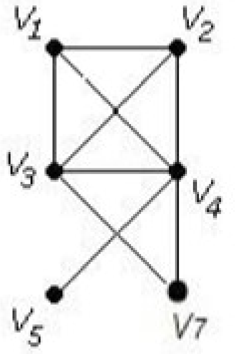
Виконала:
студентка групи КН-115
Дзямба Аліна
Викладач:
Мельникова Н. І.

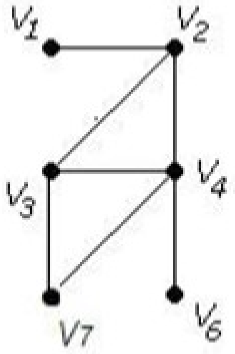
Львів – 2019 р.

Завдання № 1

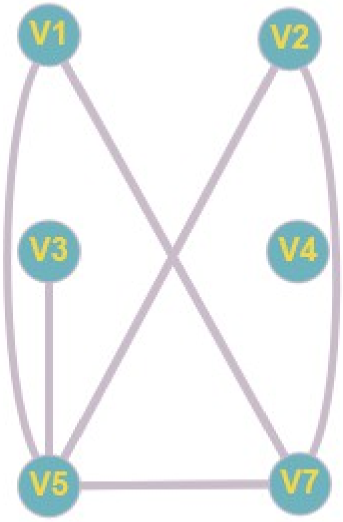
Виконати наступні операції над графами:

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

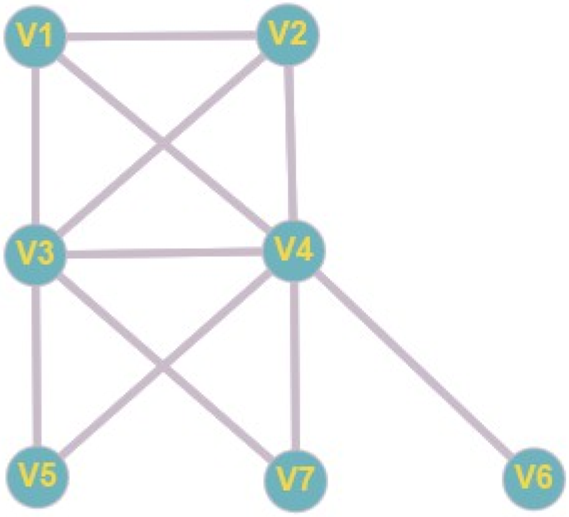




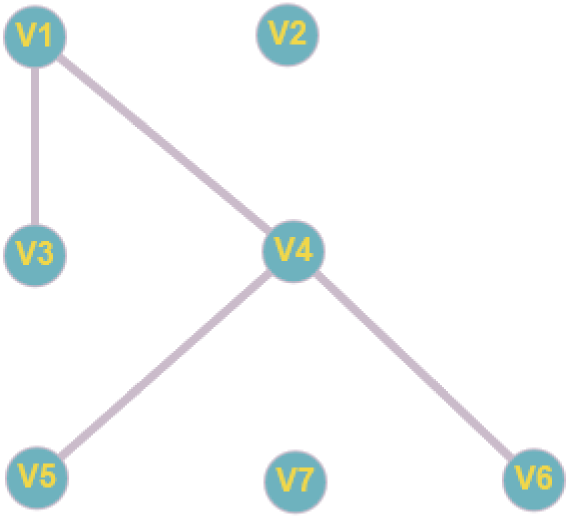
1)



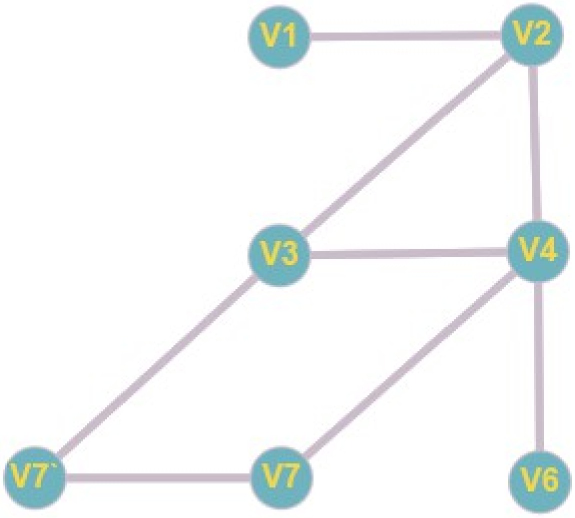
2)



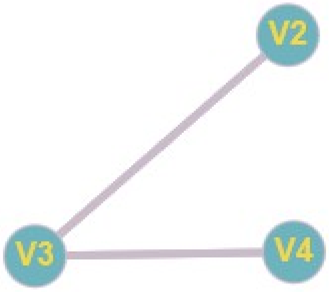
3)



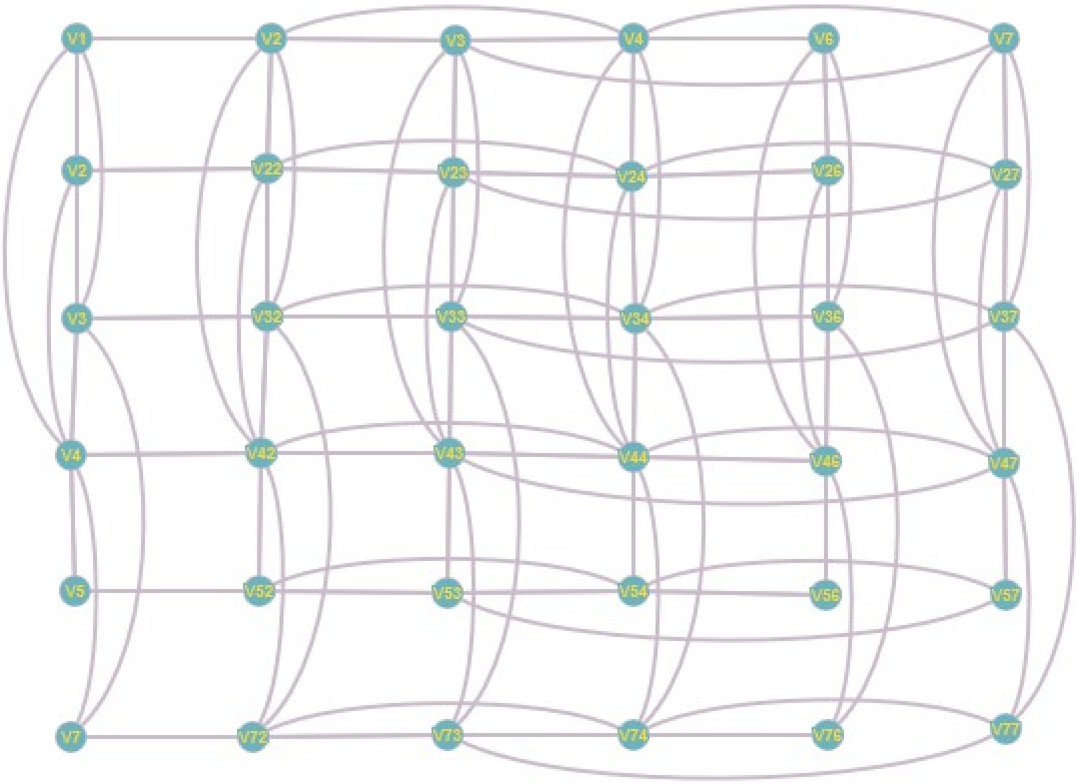
4)



5)

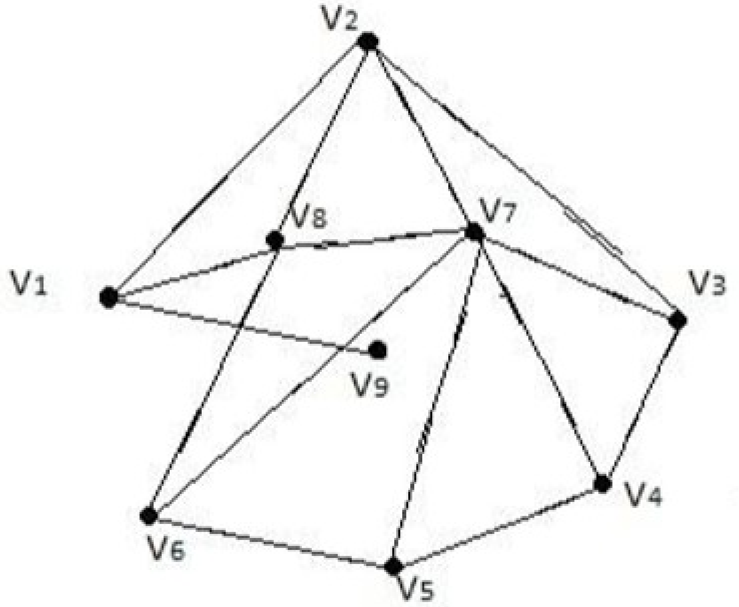


6)



Завдання № 2

Скласти таблицю суміжності для орграфа.



	V1	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**
V1	0	1	0	0	0	0	0	1	1
V2	1	0	1	0	0	0	1	1	0
V3	0	1	0	1	0	0	1	0	0
V4	0	0	1	0	1	0	1	0	0
V5	0	0	0	1	0	1	1	0	0
V6	0	0	0	0	1	0	1	1	0
V7	0	1	1	1	1	1	0	1	0
V8	1	1	0	0	0	1	1	0	0
V9	1	0	0	0	0	0	0	0	0

Завдання № 3

Для графа з другого завдання знайти діаметр.

d(V9, V4) = 4

Завдання № 4

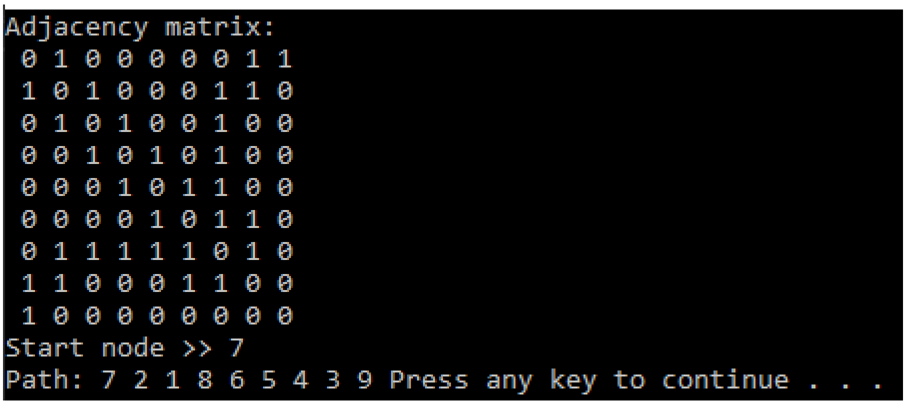
Для графа з другого завдання виконати обхід дерева вглиб.

Вершина	DFS	Стек
V7	1	V7
V8	2	V7 V8
V1	3	V7 V8 V1
V9	4	V7 V8 V1 V9
--	--	V7 V8 V1
V2	5	V7 V8 V1 V2
V3	6	V7 V8 V1 V2 V3
V4	7	V7 V8 V1 V2 V3 V4
V5	8	V7 V8 V1 V2 V3 V4 V5
V6	9	V7 V8 V1 V2 V3 V4 V5 V6
--	--	V7 V8 V1 V2 V3 V4 V5
--	--	V7 V8 V1 V2 V3 V4
--	--	V7 V8 V1 V2 V3
--	--	V7 V8 V1 V2
--	--	V7 V8 V1
--	--	V7 V8
--	--	V7
--	--	--

Програмна реалізація:

```
#include <iostream>
using namespace std;
const int n = 9;
int i, j;
bool* visited = new bool[n];

int graph[n][n] =
{
{0, 1, 0, 0, 0, 0, 0, 1, 1},
{1, 0, 1, 0, 0, 0, 1, 1, 0},
{0, 1, 0, 1, 0, 0, 1, 0, 0},
{0, 0, 1, 0, 1, 0, 1, 0, 0},
{0, 0, 0, 1, 0, 1, 1, 0, 0},
{0, 0, 0, 0, 1, 0, 1, 1, 0},
{0, 1, 1, 1, 1, 1, 0, 1, 0},
{1, 1, 0, 0, 0, 1, 1, 0, 0},
{1, 0, 0, 0, 0, 0, 0, 0, 0}
};

void DFS(int st)
{
        int r;
        cout << st + 1 << " ";
        visited[st] = true;
        for (r = 0; r <= n; r++)
                if ((graph[st][r] != 0) && (!visited[r]))
                        DFS(r);
}

int main()
{
        int start;
        cout << "Adjacency matrix: " << endl;

        for (i = 0; i < n; i++)
        {
                visited[i] = false;
                for (j = 0; j < n; j++)
                        cout << " " << graph[i][j];
                cout << endl;
        }
        cout << "Start node >> "; cin >> start;

        bool* vis = new bool[n];
        cout << "Path: ";
        DFS(start - 1);
        delete[]visited;
        system("pause");
}
```

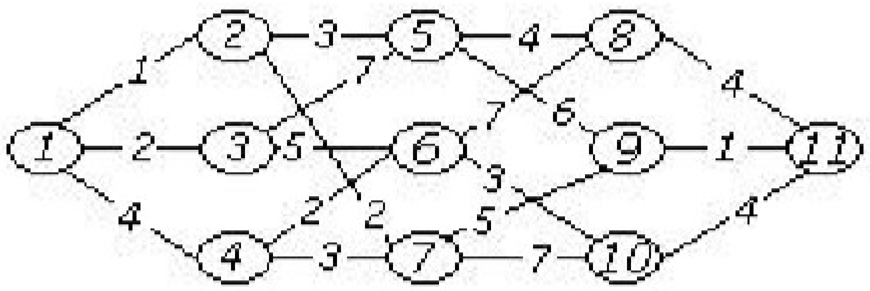
Результат:



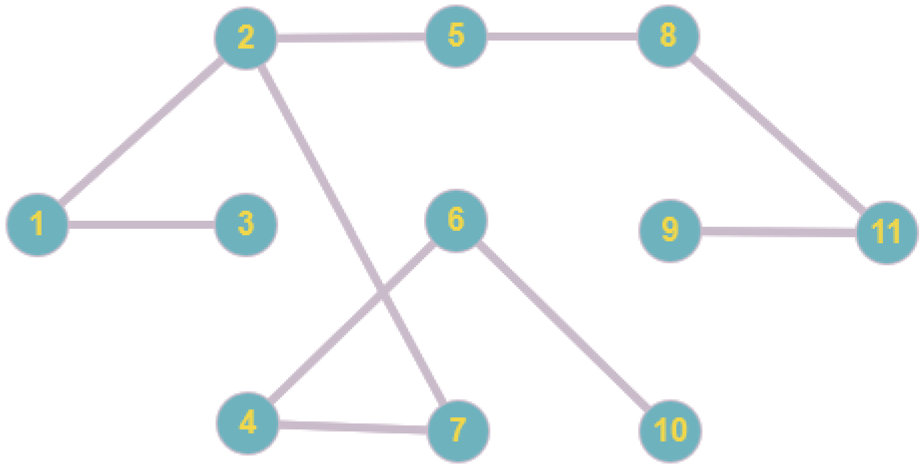
```
Adjacency matrix:
 0 1 0 0 0 0 0 1 1
 1 0 1 0 0 0 1 1 0
 0 1 0 1 0 0 1 0 0
 0 0 1 0 1 0 1 0 0
 0 0 0 1 0 1 1 0 0
 0 0 0 0 1 0 1 1 0
 0 1 1 1 1 1 0 1 0
 1 1 0 0 0 1 1 0 0
 1 0 0 0 0 0 0 0 0
Start node >> 7
Path: 7 2 1 8 6 5 4 3 9 Press any key to continue . . .
```

Завдання № 5

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



Метод Краскала:
V = {1, 2, 9, 11, 3, 4, 6, 7, 10, 5, 8}
E = {(1; 2), (9; 11), (1; 3), (4; 6), (2; 7), (6; 10), (4; 7), (2; 5), (5; 8), (8; 11)}
Вага дерева = 25



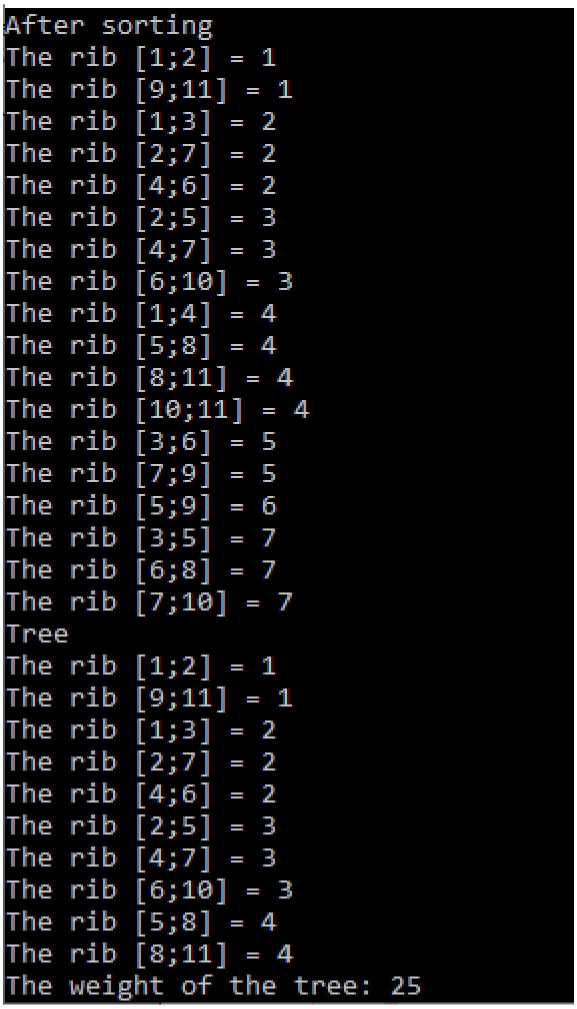
Програмна реалізація:

```
#include <iostream>
using namespace std;

struct Rib
{
	int v1;
	int v2;
	int weight;
}Graph[100];

struct sort_rib
{
	int v1;
	int v2;
	int weight;
}sort;

void Fill_Struct(int number_of_ribs)
{
	for (int i = 0; i < number_of_ribs; i++)
	{
		cout << "Firts point: ";
		cin >> Graph[i].v1;
		cout << "Second point: ";
		cin >> Graph[i].v2;
```

```
        int sort;

        if (Graph[i].v1 > Graph[i].v2)
        {
            sort = Graph[i].v1;
            Graph[i].v1 = Graph[i].v2;
            Graph[i].v2 = sort;
        }
        cout << "The rib [" << Graph[i].v1 << ";" << Graph[i].v2 << "] = ";
        cin >> Graph[i].weight;
        cout << endl;
    }
}

void Sort_Sructure(int number_of_ribs)
{
    for (int s = 1; s < number_of_ribs; s++)
    {
        for (int i = 0; i < number_of_ribs - s; i++)
        {
            if (Graph[i].weight > Graph[i + 1].weight)
            {
                sort.v1 = Graph[i].v1;
                sort.v2 = Graph[i].v2;
                sort.weight = Graph[i].weight;
                Graph[i].v1 = Graph[i + 1].v1;
                Graph[i].v2 = Graph[i + 1].v2;
                Graph[i].weight = Graph[i + 1].weight;
                Graph[i + 1].v1 = sort.v1;
                Graph[i + 1].v2 = sort.v2;
                Graph[i + 1].weight = sort.weight;
            }
        }
    }
}

void Show_Struct(int number_of_ribs)
{
    for (int i = 0; i < number_of_ribs; i++)
    {
        cout << "The rib [" << Graph[i].v1 << ";" << Graph[i].v2 << "] = " <<
Graph[i].weight << endl;
    }
}

void Algo_Kraskala(int number_of_ribs, int amount_of_points)
{
    int weighttree = 0;
    int* parent = new int[amount_of_points];
    int v1, v2, weight;
    int to_change, changed;
    for (int i = 0; i < amount_of_points; i++)
    {
        parent[i] = i;
    }
    for (int i = 0; i < number_of_ribs; i++)
    {
        v1 = Graph[i].v1;
        v2 = Graph[i].v2;
        weight = Graph[i].weight;
        if (parent[v2] != parent[v1])
        {
            cout << "The rib [" << Graph[i].v1 << ";" << Graph[i].v2 << "] = " <<
Graph[i].weight << endl;
            weighttree += weight;
```

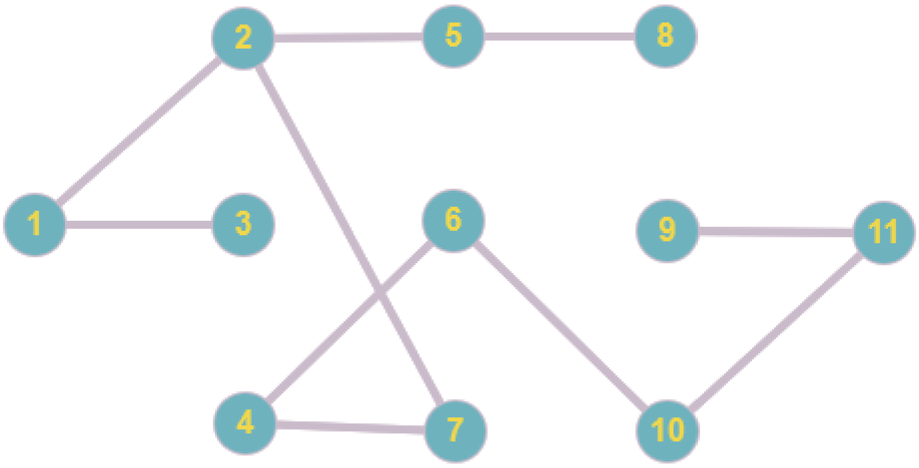
```
                to_change = parent[v1];
                changed = parent[v2];
                for (int j = 0; j < amount_of_points; j++)
                {
                    if (parent[j] == changed) { parent[j] = to_change;
                    }
                }
            }
        }
        delete[] parent;
        cout << "The weight of the tree: " << weighttree;
}

int main()
{
        cout << "Enter an amount of points" << endl;
        int q;
        cin >> q;
        int amount_of_points = q + 1;
        cout << "Enter a number of ribs" << endl;
        int number_of_ribs;
        cin >> number_of_ribs;
        Fill_Struct(number_of_ribs);
        Sort_Sructure(number_of_ribs);
        cout << "After sorting" << endl;
        Show_Struct(number_of_ribs);
        cout << "Tree" << endl;
        Algo_Kraskala(number_of_ribs, amount_of_points);
}
```

Результат:



```
After sorting
The rib [1;2] = 1
The rib [9;11] = 1
The rib [1;3] = 2
The rib [2;7] = 2
The rib [4;6] = 2
The rib [2;5] = 3
The rib [4;7] = 3
The rib [6;10] = 3
The rib [1;4] = 4
The rib [5;8] = 4
The rib [8;11] = 4
The rib [10;11] = 4
The rib [3;6] = 5
The rib [7;9] = 5
The rib [5;9] = 6
The rib [3;5] = 7
The rib [6;8] = 7
The rib [7;10] = 7
Tree
The rib [1;2] = 1
The rib [9;11] = 1
The rib [1;3] = 2
The rib [2;7] = 2
The rib [4;6] = 2
The rib [2;5] = 3
The rib [4;7] = 3
The rib [6;10] = 3
The rib [5;8] = 4
The rib [8;11] = 4
The weight of the tree: 25
```

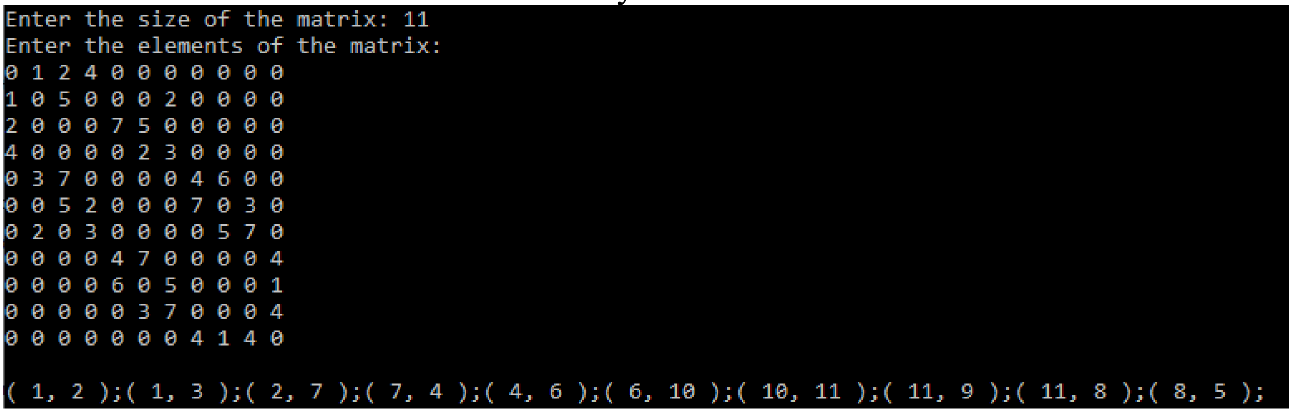
Метод Прима:
V = {1, 2, 3, 7, 5, 4, 6, 10, 8, 11, 9}
E = {(1; 2), (1; 3), (2; 7), (2; 5), (7; 4), (4; 6), (6; 10), (5; 8), (10; 11), (11; 9)}



Програмна реалізація:

```
#include <iostream>
using namespace std;
int main()
{
        int n, i, j, k;
        cout << "Enter the size of the matrix: ";
        cin >> n;
        int a[100][100];
        cout << "Enter the elements of the matrix: \n";

        for (i = 0; i < n; i++)
        {
                for (j = 0; j < n; j++)
                {
                        cin >> a[i][j];
                }
        }
        cout << endl;

        int numb[100]{ 0 };
        int size = 1;
        int min_n, min_i, min_j;

        while (size < n)
        {
                min_n = 1000;

                for (k = 0; k < size; k++)
                {
                        for (i = 0; i < n; i++)
                        {
                                if (a[numb[k]][i] < min_n && a[numb[k]][i] != 0)
                                {
                                        min_n = a[numb[k]][i];
                                        min_i = i;
                                        min_j = numb[k];
                                }
                        }
        }
        a[min_j][min_i] = 0;
        a[min_i][min_j] = 0;

        bool true_i = 0;

        for (i = 0; i < size; i++)
            if (min_i == numb[i])
                true_i = 1;

        if (true_i == 0)
        {
            size++;
            numb[size - 1] = min_i;
            cout << "( " << min_j + 1 << ", " << min_i + 1 << " )" << ';';
        }
    }
}
```

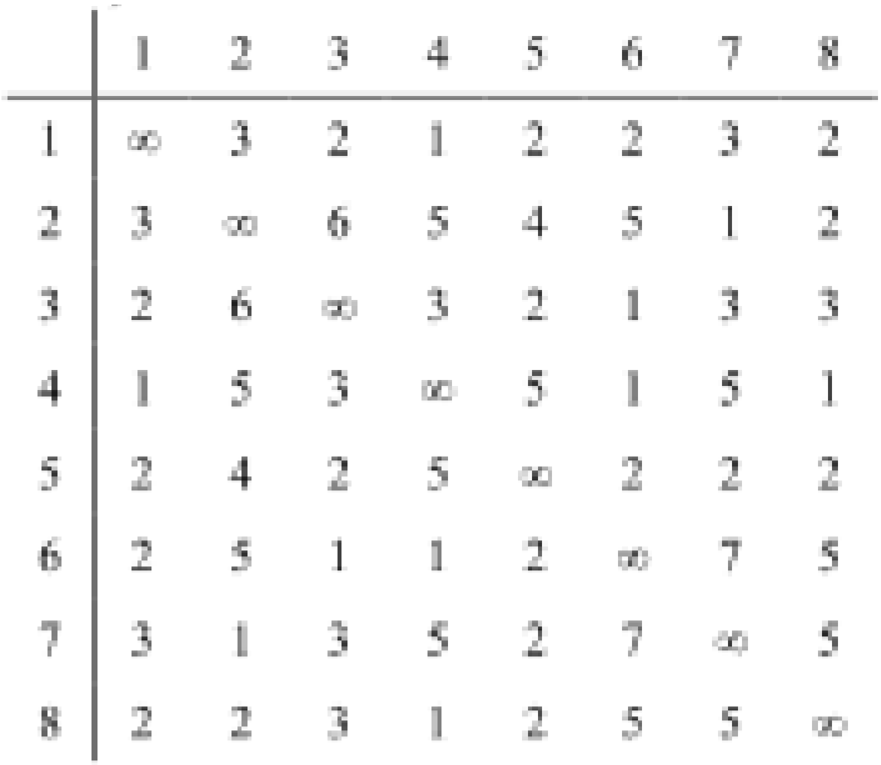
Результат:



```
Enter the size of the matrix: 11
Enter the elements of the matrix:
0 1 2 4 0 0 0 0 0 0 0
1 0 5 0 0 0 2 0 0 0 0
2 0 0 0 7 5 0 0 0 0 0
4 0 0 0 0 2 3 0 0 0 0
0 3 7 0 0 0 0 4 6 0 0
0 0 5 2 0 0 0 7 0 3 0
0 2 0 3 0 0 0 0 5 7 0
0 0 0 0 4 7 0 0 0 0 4
0 0 0 0 6 0 5 0 0 0 1
0 0 0 0 0 3 7 0 0 0 4
0 0 0 0 0 0 0 4 1 4 0

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

Завдання № 6

Розв’язати задачу комівояжера для повного 8-ми вершинного графа методом «іди у найближчий», матриця вагів якого має вигляд:



	1	2	3	4	5	6	7	8
1	∞	3	2	1	2	2	3	2
2	3	∞	6	5	4	5	1	2
3	2	6	∞	3	2	1	3	3
4	1	5	3	∞	5	1	5	1
5	2	4	2	5	∞	2	2	2
6	2	5	1	1	2	∞	7	5
7	3	1	3	5	2	7	∞	5
8	2	2	3	1	2	5	5	∞

	2	**3**	**14**	**5**	**6**	**7**	**8**
2	∞	6	5	4	5	1	2
3	6	∞	3	2	1	3	3
14	5	3	∞	5	1	5	1
5	4	2	5	∞	2	2	2
6	5	1	1	2	∞	7	5
7	1	3	5	2	7	∞	5
8	2	3	1	2	5	5	∞

	2	**3**	**5**	**146**	**7**	**8**
2	∞	6	4	5	1	2
3	6	∞	2	1	3	3
5	4	2	∞	2	2	2
146	5	1	2	∞	7	5
7	1	3	2	7	∞	5
8	2	3	2	5	5	∞

	2	**1463**	**5**	**7**	**8**
2	∞	6	4	1	2
1463	6	∞	2	3	3
5	4	2	∞	2	2
7	1	3	2	∞	5
8	2	3	2	5	∞

	2	**14635**	**7**	**8**
2	∞	4	1	2
14635	4	∞	2	2
7	1	2	∞	5
8	2	2	5	∞

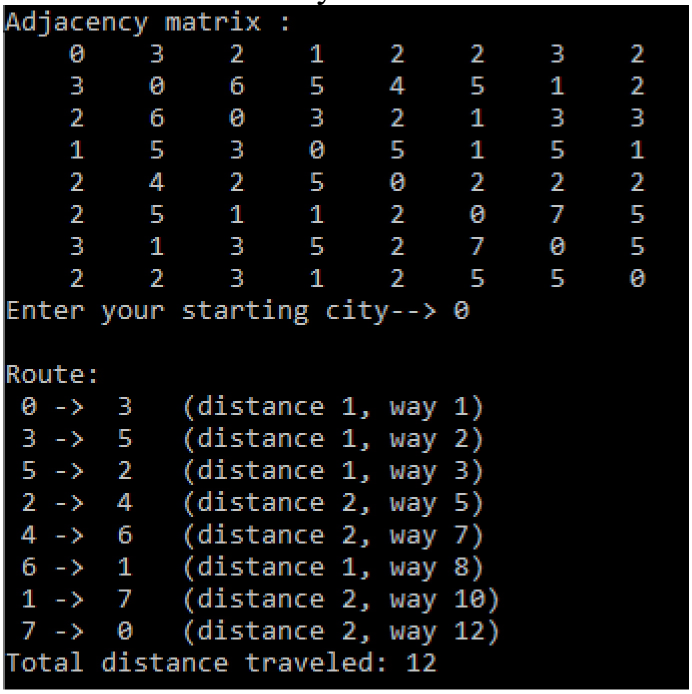
	2	**146357**	**8**
2	∞	1	2
146357	1	∞	5
8	2	5	∞

	1463572	**8**
1463572	∞	2
8	2	∞

Приклад реалізації:

```
#include <iostream>

using namespace std;


bool check(int key, int* mas, int kol) {
    for (int j = 0; j < kol; j++)
        if (mas[j] == key)
            return false;
    return true;
}

int main() {
    int kol;
    do
    {
        cout << "Enter the number of cities(2-10) --> ";
        cin >> kol;
    } while (kol < 2 || kol > 10);
    int** arr = new int* [kol];
    for (int i = 0; i < kol; i++)
        arr[i] = new int[kol];

    int rasst;
    for (int i = 0; i < kol; i++) {
        for (int j = i; j < kol; j++) {
            if (i == j)
            {
                arr[i][j] = 0;
                continue;
            }
            do
            {
                cout << "Enter the distance from the city " << i << " to the city
" << j << " --> ";
                cin >> rasst;
            } while (rasst < 1);
            arr[i][j] = arr[j][i] = rasst;
        }
    }
    system("cls");
    cout << endl << "Adjacency matrix : ";
    for (int i = 0; i < kol; i++) {
        cout << endl;
        for (int j = 0; j < kol; j++)
            cout << setw(5) << arr[i][j];
    }

    int* route = new int[kol];

    cout << endl;
    char ans;
    int start;
    do {
        for (int i = 0; i < kol; i++)
            route[i] = -1;
        do
        {
            cout << "Enter your starting city--> ";
            cin >> start;
        } while (start < 0 || start > kol - 1);

        route[0] = start;
        int now = start;
        int path = 0;
        cout << "\nRoute:" << endl;
        for (int i = 1; i < kol; i++) {
            int min = INT_MAX, min_town;
            for (int j = 0; j < kol; j++) {
                if (check(j, route, kol) && arr[now][j] < min && arr[now][j] > 0)
{
                    min = arr[now][j];
                    min_town = j;
                }
            }

            path += min;
            route[i] = min_town;
            cout << setw(2) << now << " -> " << setw(2) << route[i] << "
(distance " << min << ", way " << path << ")" << endl;
            now = route[i];
        }
        path += arr[start][now];
        cout << setw(2) << now << " -> " << setw(2) << start << "   (distance " <<
arr[start][now] << ", way " << path << ")" << endl;
        cout << "Total distance traveled: " << path << endl;

        cout << endl << "Would you like to continue your search for paths? (+, If yes)
--> ";
        cin >> ans;

    } while (ans == '+');

    delete[] route;
    for (int i = 0; i < kol; i++)
        delete[] arr[i];
    delete[] arr;

    system("pause");
    return 0;
}
```

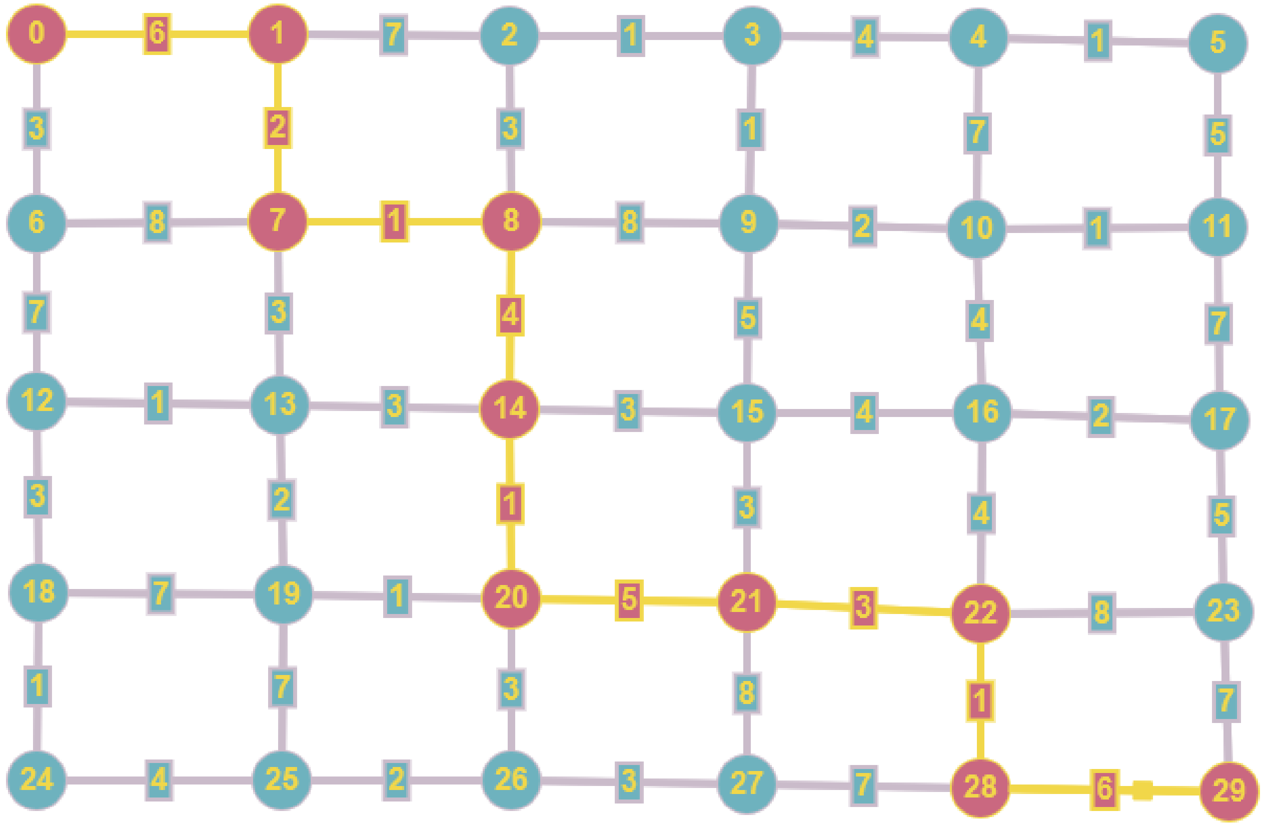
Результат:



```
Adjacency matrix :
    0    3    2    1    2    2    3    2
    3    0    6    5    4    5    1    2
    2    6    0    3    2    1    3    3
    1    5    3    0    5    1    5    1
    2    4    2    5    0    2    2    2
    2    5    1    1    2    0    7    5
    3    1    3    5    2    7    0    5
    2    2    3    1    2    5    5    0
Enter your starting city--> 0

Route:
 0 ->  3   (distance 1, way 1)
 3 ->  5   (distance 1, way 2)
 5 ->  2   (distance 1, way 3)
 2 ->  4   (distance 2, way 5)
 4 ->  6   (distance 2, way 7)
 6 ->  1   (distance 1, way 8)
 1 ->  7   (distance 2, way 10)
 7 ->  0   (distance 2, way 12)
Total distance traveled: 12
```

Завдання № 7

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



V0 → 1 → 7 → 8 → 14 → 20 → 21 → 22 → 28 → 29 = 29

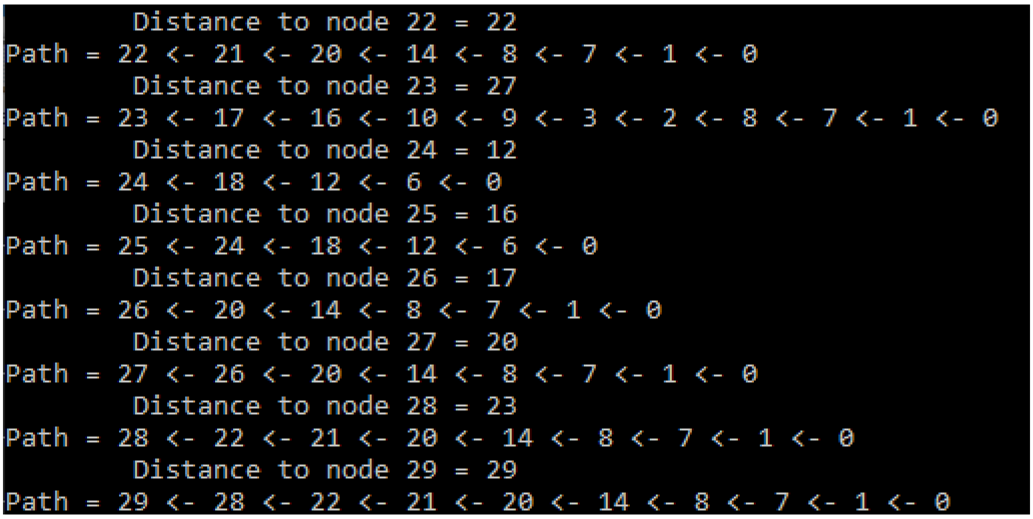
Програмна реалізація:

```
#include <iostream>
```

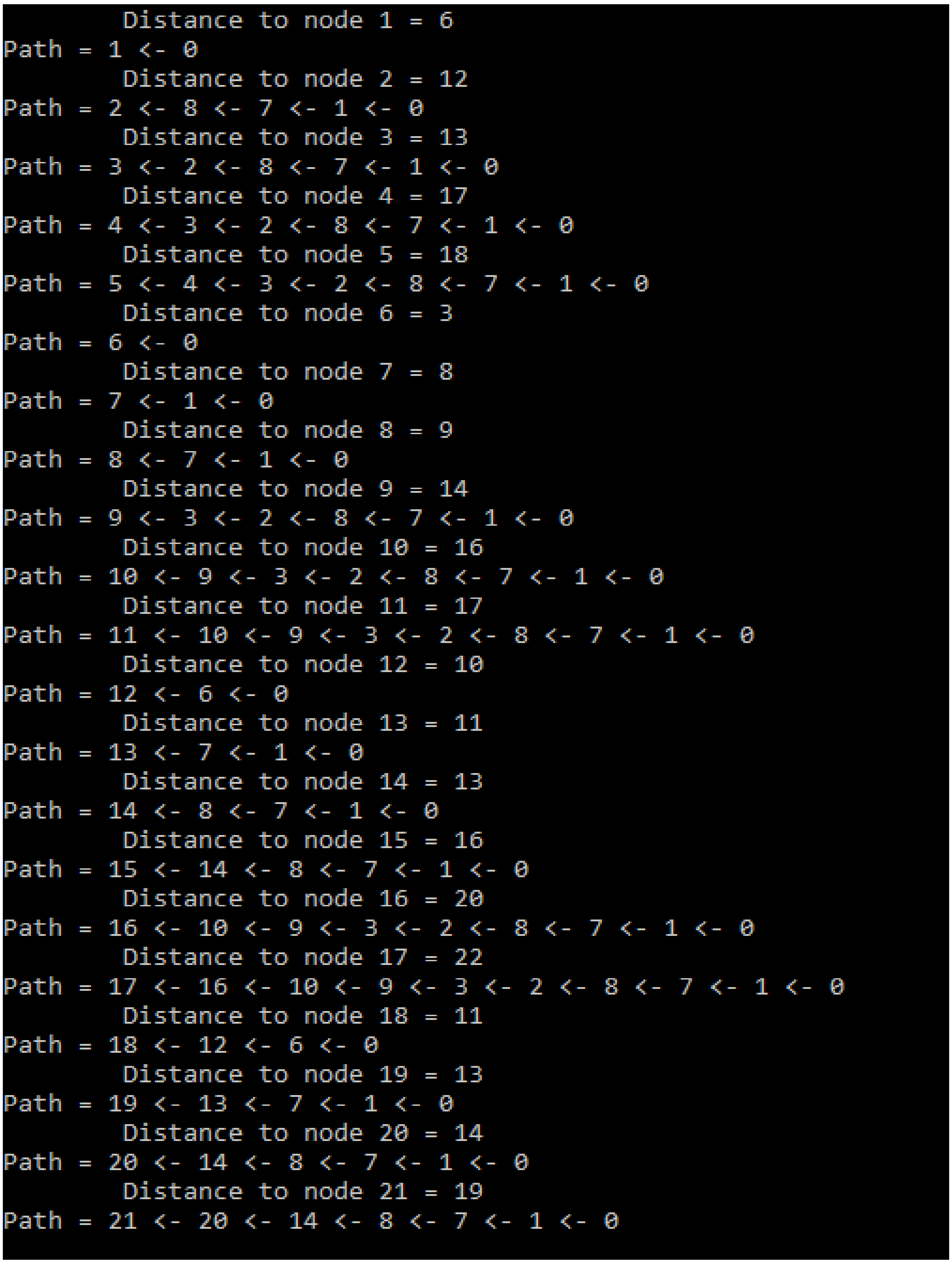
```
#include <stdio.h>
using namespace std;
#define INFINITY 9999
#define max 30

void algorithm(int G[max][max], int n, int start);

int main()
{
	int G[max][max] = {
{0, 6, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{6, 0, 7, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 7, 0, 1, 0, 0, 0, 0, 3, 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, 4, 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, 4, 0, 1, 0, 0, 0, 0, 7, 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, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{3, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 2, 0, 0, 0, 0, 8, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 3, 0, 0, 0, 0, 1, 0, 8, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0, 0, 8, 0, 2, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 3, 0, 4, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 4, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 7, 0, 1, 0, 0, 0, 0, 7, 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, 5, 0, 0, 0, 0, 3, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 5, 0, 3, 0, 0, 0, 0, 8, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 3, 0, 8, 0, 0, 0, 0, 1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 7},
{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, 4, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 4, 0, 2, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 3, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 3, 0, 7, 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, 7, 0, 6},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 6, 0}
	};
	int n = 30;
	int u = 0;
	algorithm(G, n, u);
	return 0;
}

void algorithm(int G[max][max], int n, int start)
{
	int cost[max][max];
	int distance[max];
	int pred[max];
	int visited[max];
	int count, min_d, next;

	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			if (G[i][j] == 0)
				cost[i][j] = INFINITY;
			else
				cost[i][j] = G[i][j];

	for (int i = 0; i < n; i++)
	{
		distance[i] = cost[start][i];
```

```
            pred[i] = start;
            visited[i] = 0;
        }
        distance[start] = 0;
        visited[start] = 1;
        count = 1;

        while (count < n - 1)
        {
            min_d = INFINITY;

            for (int i = 0; i < n; i++)
                if (distance[i] < min_d && !visited[i])
                {
                    min_d = distance[i];
                    next = i;
                }
            visited[next] = 1;

            for (int i = 0; i < n; i++)
                if (!visited[i])
                    if (min_d + cost[next][i] < distance[i])
                    {
                        distance[i] = min_d + cost[next][i];
                        pred[i] = next;
                    }
            count++;
        }

        for (int i = 0; i < n; i++)
            if (i != start)
            {
                cout << "\n\tDistance to node " << i << " = " << distance[i];
                cout << "\nPath = " << i;
                int j = i;
                do
                {
                    j = pred[j];
                    cout << " <- " << j;
                } while (j != start);
            }
}
```

Результат:



```
        Distance to node 22 = 22
Path = 22 <- 21 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 23 = 27
Path = 23 <- 17 <- 16 <- 10 <- 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 24 = 12
Path = 24 <- 18 <- 12 <- 6 <- 0
        Distance to node 25 = 16
Path = 25 <- 24 <- 18 <- 12 <- 6 <- 0
        Distance to node 26 = 17
Path = 26 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 27 = 20
Path = 27 <- 26 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 28 = 23
Path = 28 <- 22 <- 21 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 29 = 29
Path = 29 <- 28 <- 22 <- 21 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
```



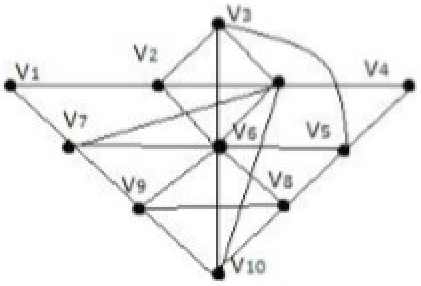
```
        Distance to node 1 = 6
Path = 1 <- 0
        Distance to node 2 = 12
Path = 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 3 = 13
Path = 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 4 = 17
Path = 4 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 5 = 18
Path = 5 <- 4 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 6 = 3
Path = 6 <- 0
        Distance to node 7 = 8
Path = 7 <- 1 <- 0
        Distance to node 8 = 9
Path = 8 <- 7 <- 1 <- 0
        Distance to node 9 = 14
Path = 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 10 = 16
Path = 10 <- 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 11 = 17
Path = 11 <- 10 <- 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 12 = 10
Path = 12 <- 6 <- 0
        Distance to node 13 = 11
Path = 13 <- 7 <- 1 <- 0
        Distance to node 14 = 13
Path = 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 15 = 16
Path = 15 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 16 = 20
Path = 16 <- 10 <- 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 17 = 22
Path = 17 <- 16 <- 10 <- 9 <- 3 <- 2 <- 8 <- 7 <- 1 <- 0
        Distance to node 18 = 11
Path = 18 <- 12 <- 6 <- 0
        Distance to node 19 = 13
Path = 19 <- 13 <- 7 <- 1 <- 0
        Distance to node 20 = 14
Path = 20 <- 14 <- 8 <- 7 <- 1 <- 0
        Distance to node 21 = 19
Path = 21 <- 20 <- 14 <- 8 <- 7 <- 1 <- 0
```

Завдання № 8

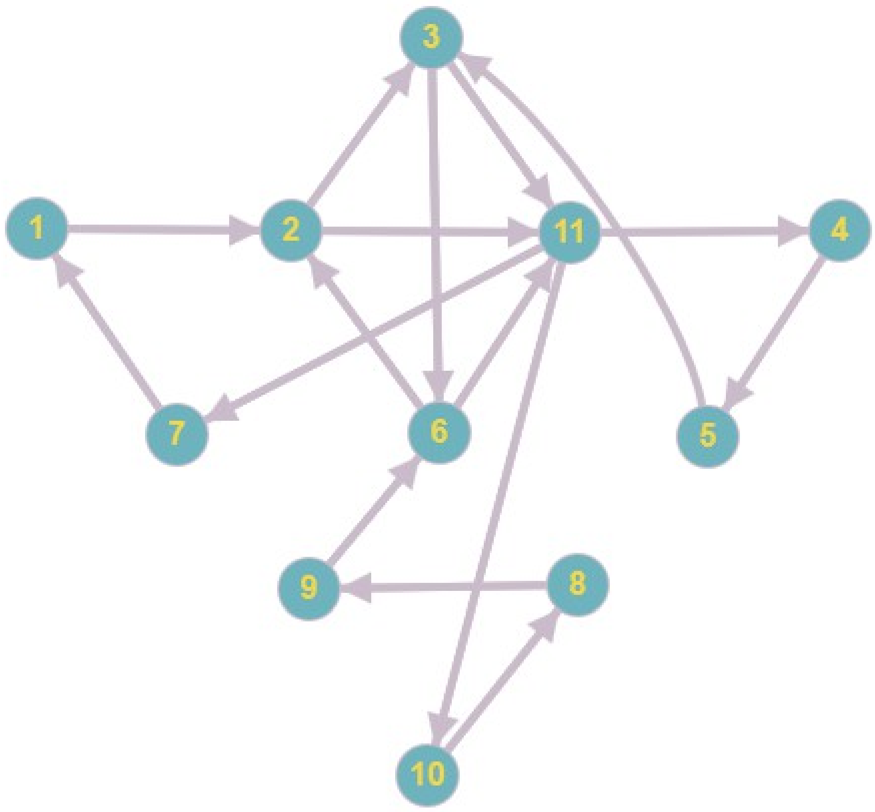
Знайти ейлеровий цикл в ейлеровому графі двома методами:

а) Флері;

б) елементарних циклів.



Розв'язання:



а)

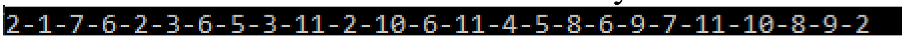
V1⇒V2⇒V11⇒V10⇒V8⇒V9⇒V6⇒V10⇒V9⇒V7⇒V6⇒V5⇒V8⇒V6⇒V2⇒V3⇒V11⇒V4⇒V5⇒
V3⇒V6⇒V11⇒V7⇒V1

Програмна реалізація:

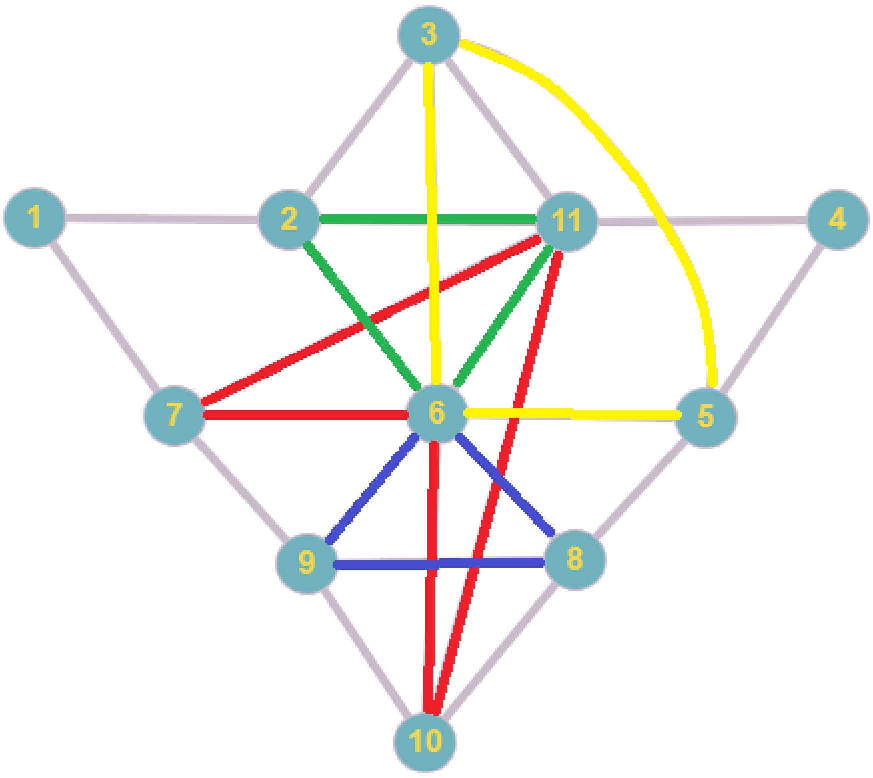
```
#include <iostream>
#include <string.h>
#include <list>
using namespace std;

class Graph
{
        int V;
        list<int>* adj;
public:
        Graph(int V)
        {
                this->V = V;
                adj = new list<int>[V];
        }

        ~Graph() { delete[] adj; }

        void addEdge(int u, int v)
        {
                adj[u].push_back(v);
                adj[v].push_back(u);
        }

        void rmvEdge(int u, int v);
        void printEulerTour();
        void printEulerUtil(int s);
        int DFSCount(int v, bool visited[]);
        bool isValidNextEdge(int u, int v);
```

```
};

void Graph::printEulerTour()
{
	int u = 0;
	for (int i = 0; i < V; i++)
		if (adj[i].size() & 1)
		{
			u = i;
			break;
		}

	printEulerUtil(u);
	cout << u + 1 << endl;
}

void Graph::printEulerUtil(int u)
{
	list<int>::iterator i;
	for (i = adj[u].begin(); i != adj[u].end(); ++i)
	{
		int v = *i;
		if (v != -1 && isValidNextEdge(u, v))
		{
			cout << u + 1 << "-";
			rmvEdge(u, v);
			printEulerUtil(v);
		}
	}
}

bool Graph::isValidNextEdge(int u, int v)
{
	int count = 0;
	list<int>::iterator i;
	for (i = adj[u].begin(); i != adj[u].end(); ++i)
		if (*i != -1)
			count++;

	if (count == 1)
		return true;

	bool* visited = new bool[V];
	memset(visited, false, V);

	int count1 = DFSCount(u, visited);
	rmvEdge(u, v);

	memset(visited, false, V);
	int count2 = DFSCount(u, visited);
	addEdge(u, v);

	return (count1 > count2) ? false : true;
}

void Graph::rmvEdge(int u, int v)
{
	list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);
	*iv = -1;
	list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);
	*iu = -1;
}

int Graph::DFSCount(int v, bool visited[])
{
        visited[v] = true;
        int count = 1;
        list<int>::iterator i;

        for (i = adj[v].begin(); i != adj[v].end(); ++i)
                if (*i != -1 && !visited[*i])
                        count += DFSCount(*i, visited);

        return count;
}

int main()
{
        Graph g1(17);
        g1.addEdge(0, 1);
        g1.addEdge(0, 6);
        g1.addEdge(1, 2);
        g1.addEdge(1, 5);
        g1.addEdge(1, 10);
        g1.addEdge(1, 9);
        g1.addEdge(2, 5);
        g1.addEdge(2, 10);
        g1.addEdge(2, 4);
        g1.addEdge(3, 10);
        g1.addEdge(3, 4);
        g1.addEdge(4, 5);
        g1.addEdge(4, 7);
        g1.addEdge(5, 10);
        g1.addEdge(5, 9);
        g1.addEdge(5, 7);
        g1.addEdge(5, 8);
        g1.addEdge(5, 6);
        g1.addEdge(6, 10);
        g1.addEdge(6, 8);
        g1.addEdge(7, 8);
        g1.addEdge(7, 9);
        g1.addEdge(8, 9);
        g1.addEdge(9, 10);
        g1.printEulerTour();
}
```

Результат:



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



б) 7 → 11 → 10 → 6 → 7
2 → 11 → 6 → 2
6 → 8 → 9 → 2
3 → 5 → 6 → 3

Завдання №9

Спростити формулу (привести їх до скороченої ДНФ).

$$x\bar{z} \vee xy \vee yz$$

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$(\bar{x}yz)$ v $(x\overline{yz})$ v $(xy\bar{z})$ v (xyz)