МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота

З дисципліни

“Дискретна математика”

Виконала:

Студентка групи КН-115

Попів Христина

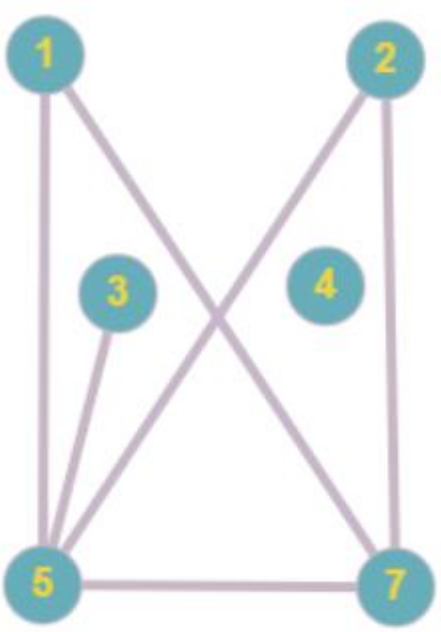
Викладач:

Мельникова Н.І.

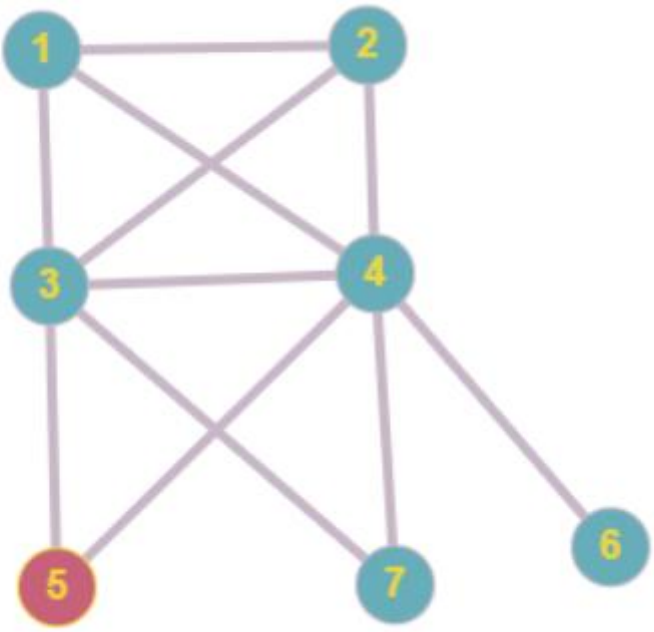
Варіант 15

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 А - що скадається з 3-х вершин в G1 6) добуток графів.

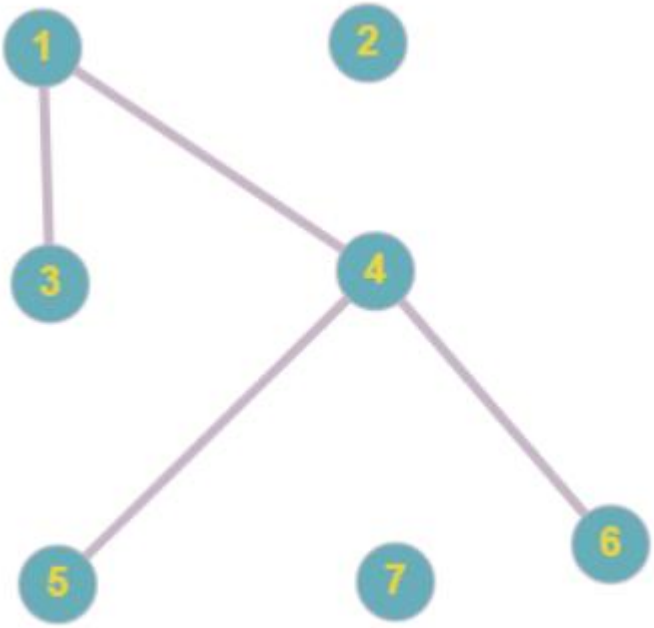
1)



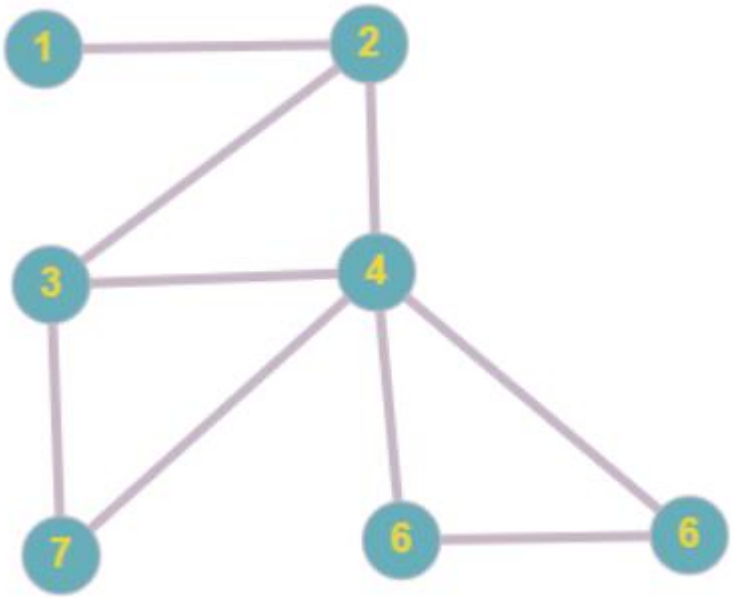
2)



3)



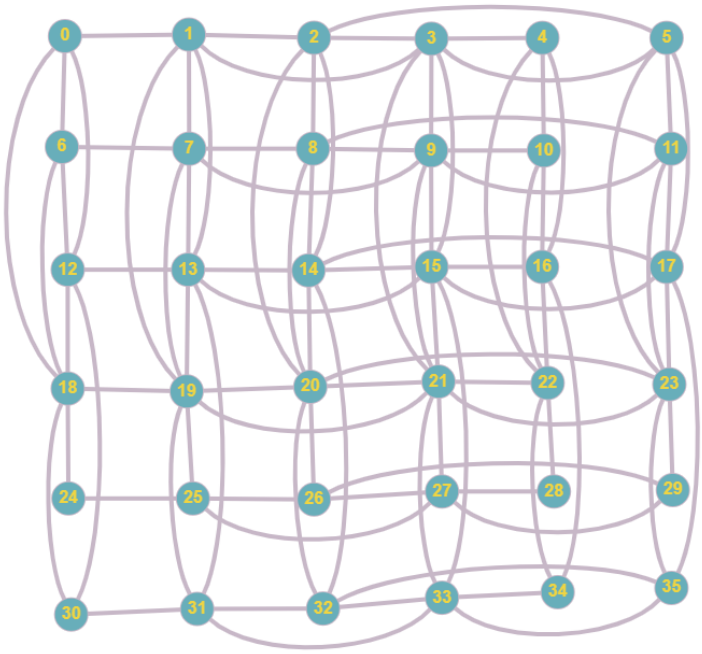
4)



5)

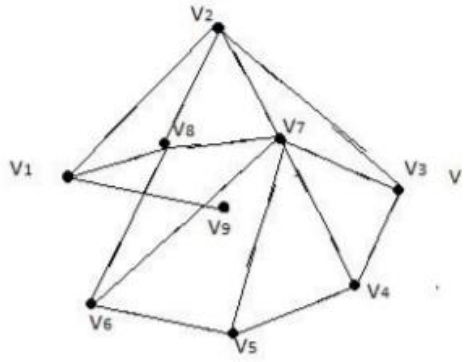

2
3
4

6)


0
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
31
32
33
34
35

Завдання № 2 Скласти таблицю суміжності для орграфа.

15)



	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	1	1
2	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	1	0	0
4	0	0	1	0	1	0	1	0	0
5	0	0	0	1	0	1	1	0	0
6	0	0	0	0	1	0	1	1	0
7	0	1	1	1	1	1	0	1	0
8	1	1	0	0	0	1	1	0	0
9	1	0	0	0	0	0	0	0	0

Завдання № 3 Для графа з другого завдання знайти діаметр.

Діаметр графа :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 V9**
V2	**5**	**V7 V8 V9 V2**
V3	**6**	**V7 V8 V9 V2 V3**
V4	**7**	**V7 V8 V9 V2 V3 V4**
V5	**8**	**V7 V8 V9 V2 V3 V4 V5**
V6	**9**	**V7 V8 V9 V2 V3 V4 V5 V6**
-	**-**	**V7 V8 V9 V2 V3 V4 V5**
-	**-**	**V7 V8 V9 V2 V3 V4**
-	**-**	**V7 V8 V9 V2 V3**
-	**-**	**V7 V8 V9 V2**
-	**-**	**V7 V8 V9**
-	**-**	**V7 V8**
-	**-**	**V7**
-	**-**	**-**

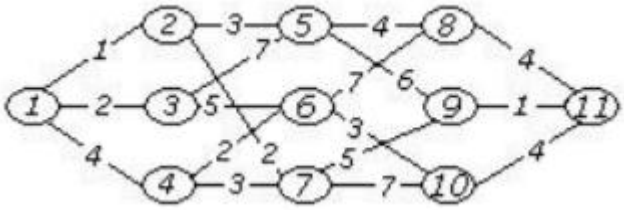
Програма:

```
#include <iostream>
#include <stack> // стек
using namespace std;
int main()
{
	stack<int> Stack;
	int mas[9][9] =
	{
	 {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},
	 };
	int nodes[9]; // вершини
	for (int i = 0; i < 9; i++) // всі вершини = 0
		nodes[i] = 0;
	Stack.push(0); // поміщаємо у стек пешу вершину
	while (!Stack.empty())
	{ // допоки стек не пустий
		int node = Stack.top(); // видаляємо вершину		Stack.pop();
		if (nodes[node] == 2) continue;
		nodes[node] = 2; // помічаємо її як пройдену
		for (int j = 8; j >= 0; j--)
		{ // перевіряємо для неї всі суміжні вершини
			if (mas[node][j] == 1 && nodes[j] != 2)
			{ // якщо вершина суміжна
				Stack.push(j); // добавляємо її у стек
				nodes[j] = 1; // помічаємо вершину як виявлену
			}
		}
		cout << node + 1 << endl; // вивід
	}

	return 0;
	}
```

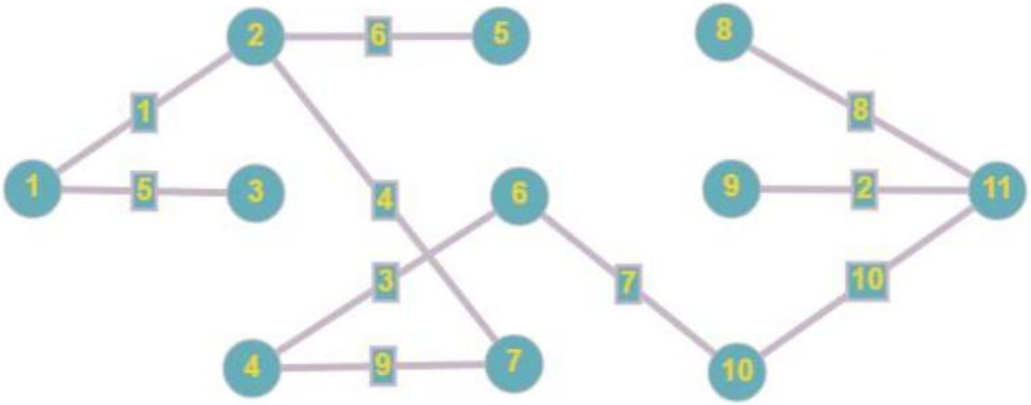
Результат:

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

Завдання № 5 Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.



Метод Краскала (номер на ребрах відповідає послідовності кроків):



Код програми:

```
#include <iostream>
using namespace std;
struct Rib
{
    int v1, v2, 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;
```

No issues found

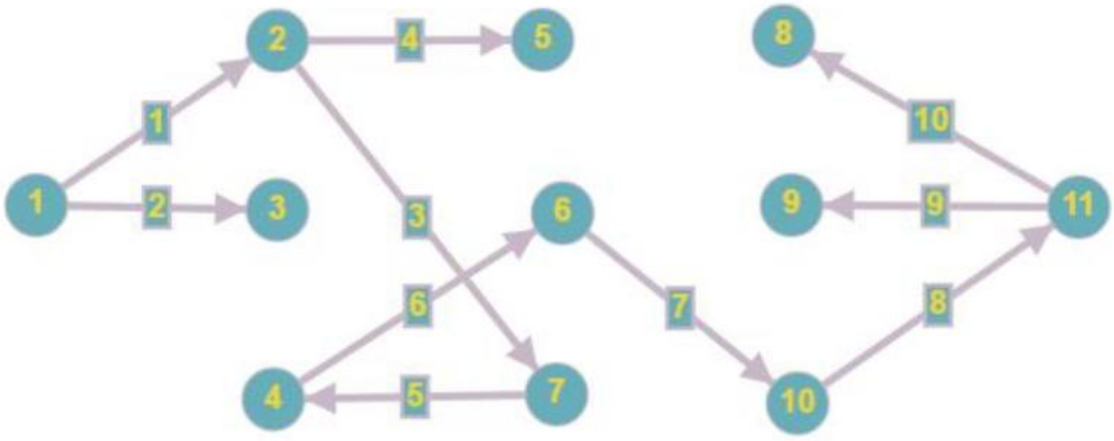
```
                sort.v2 = Graph[i].v2;
                sort.weight = Graph[i].weight;
                Graph[i].v1 = Graph[i + 1].v1;
                Graph[i].v2 = Graph[i + 1].v2;
                Graph[i].weight = Graph[i + 1].weight;
                Graph[i + 1].v1 = sort.v1;
                Graph[i + 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];
```

```
                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);
    //Show_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 :

```
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 [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 [1;4] = 4
The rib [5;8] = 4
The rib [3;6] = 5
The rib [5;9] = 6
The rib [3;5] = 7
The weight of the tree: 39
```

Метод Прима(номер на ребрах відповідає послідовності кроків):



Код програми :

```
#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:

15)

	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	**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**	∝

	1456372	**8**
1463572	∝	**5**
8	**5**	∝

Найкоротший шлях: 1456372

Код програми:

```
#include <iostream>
#include <iomanip>

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;
```

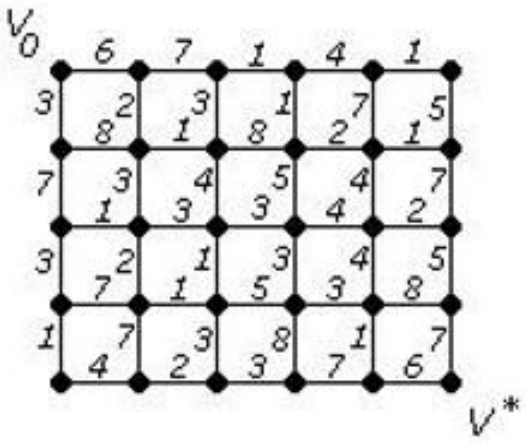
```
        }
        path += arr[start][now];
        cout << setw(2) << now << " -> " << setw(2) << start << "   (distance " << arr[start][now] << ", way " << path << ")" << endl;
        cout << "Total distance traveled: " << path << endl;

        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:

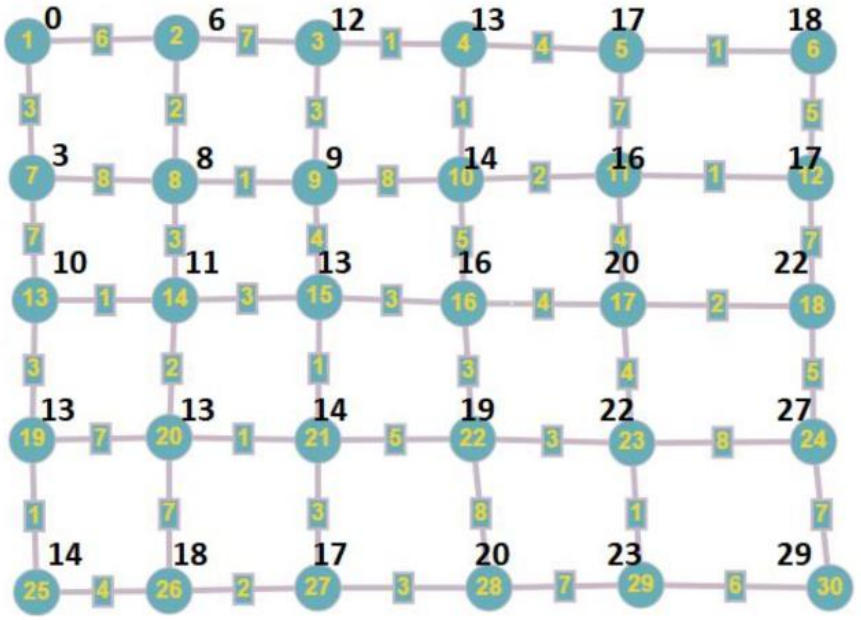
```
Adjacency matrix :
    0    3    2    1    2    2    3    2
    3    0    6    5    4    5    1    2
    2    6    0    3    2    1    3    3
    1    5    3    0    5    1    5    1
    2    4    2    5    0    2    2    2
    2    5    1    1    2    0    7    5
    3    1    3    5    2    7    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

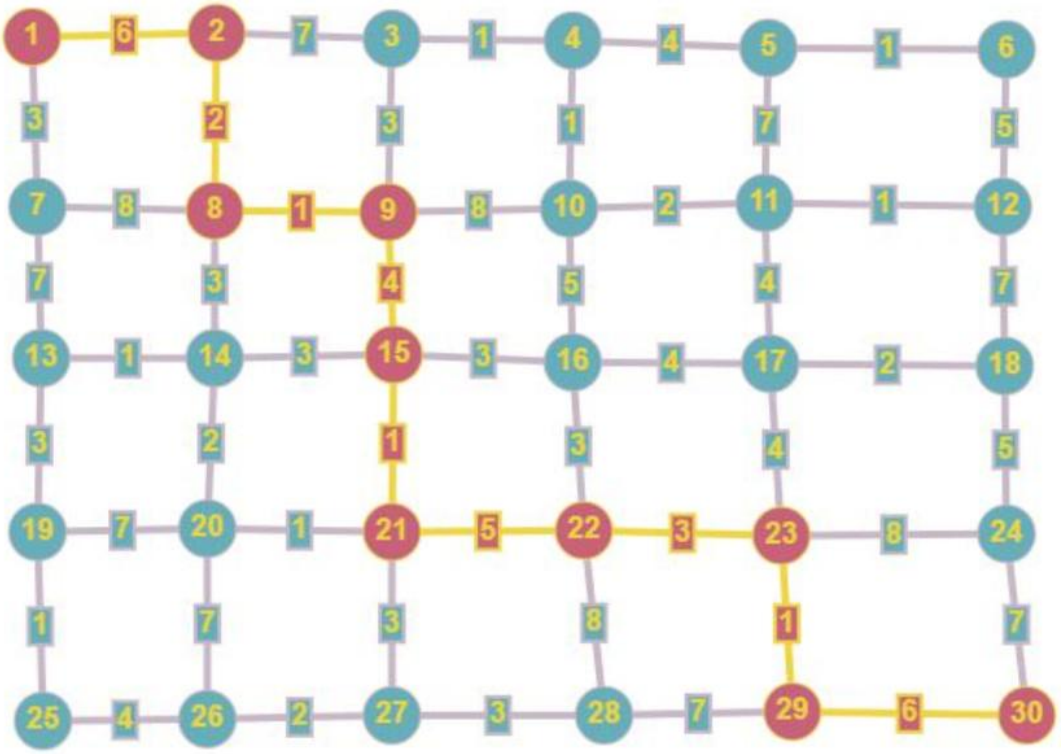
За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V0 і V* .



За допомогою алгоритму Дейкстри знайдемо найкортший шлях від V_0 до кожної вершини графа (мінімальний шлях відображений біля вершин):



Тепер покажемо найменшу відстань від вершини 1 до вершини 30:



Отже, найкоротша відстань від вершини 1 до вершини 30 рівна **29**.

Код програми:

```cpp
#include <iostream>

using namespace std;

const int N = 30;
const int INF = 20000;

void main()
{
    setlocale(LC_ALL, "ukr");
    int start;
    int graph[N][N] = {
        {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, 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, 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, 0, 2, 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, 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, 1, 0, 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, 3, 0, 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, 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, 1, 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, 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},
        {0, 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, 0, 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, 6},
        {0, 0, 0, 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, 0, 0, 0, 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, 6},
        {0, 0, 0, 0, 0, 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} };

    do
    {
        cout << "Початкова вершина: ";
        cin >> start;
    } while (start < 1 || start > N);
    int Dist[N];
    int count, index, i, u, m;

    m = start;
    bool visited[N];
    for (i = 0; i < N; i++)
    {
        Dist[i] = INF;
        visited[i] = false;
    }
    Dist[start - 1] = 0;

    for (count = 0; count < N; count++)
    {
        int min = INF;
        for (i = 0; i < N; i++)
        {
            if (!visited[i] && Dist[i] <= min)
            {
                min = Dist[i];
                index = i;
            }
        }
        u = index;
        visited[u] = true;
        for (i = 0; i < N; i++)
```

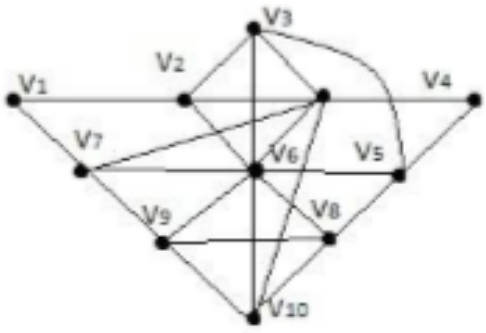
```
        visited[u] = true;
        for (i = 0; i < N; i++)
            if (!visited[i] && (graph[u][i] && Dist[u] != INF) && Dist[u] + graph[u][i] < Dist[i])
            {
                Dist[i] = Dist[u] + graph[u][i];
            }
    }

    cout << "Відстань від заданої вершини до всіх вершин графа: " << endl;
    for (i = 0; i < N; i++)
    {
        if (Dist[i] != INF)
        {
            cout << m << " -> " << i + 1 << " = " << Dist[i] << endl;
        }
    }
    system("pause");
}
```

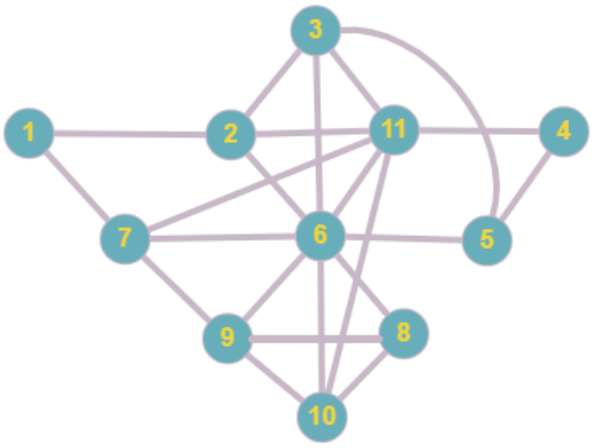
Результат

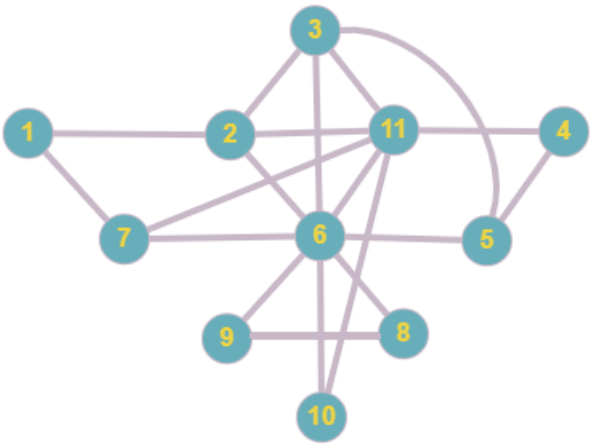
```
Початкова вершина: 4323
Початкова вершина: 33
Початкова вершина: -1
Початкова вершина: 1
Відстань від заданої вершини до всіх вершин графа:
1 -> 1 = 0
1 -> 2 = 6
1 -> 3 = 12
1 -> 4 = 13
1 -> 5 = 17
1 -> 6 = 18
1 -> 7 = 3
1 -> 8 = 8
1 -> 9 = 9
1 -> 10 = 14
1 -> 11 = 16
1 -> 12 = 17
1 -> 13 = 10
1 -> 14 = 11
1 -> 15 = 13
1 -> 16 = 16
1 -> 17 = 20
1 -> 18 = 22
1 -> 19 = 13
1 -> 20 = 13
1 -> 21 = 14
1 -> 22 = 19
1 -> 23 = 22
1 -> 24 = 27
1 -> 25 = 14
1 -> 26 = 18
1 -> 27 = 17
1 -> 28 = 20
1 -> 29 = 23
1 -> 30 = 29
Для продолжения нажмите любую клавишу . . .
```

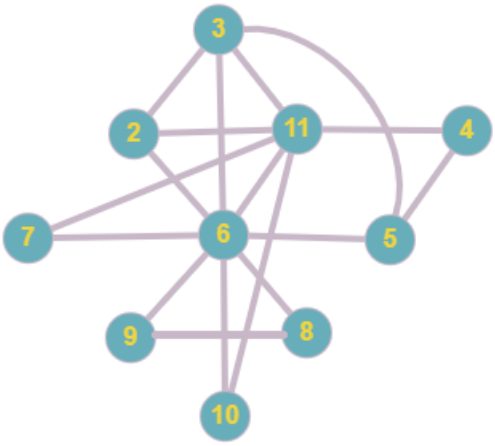
Завдання № 8

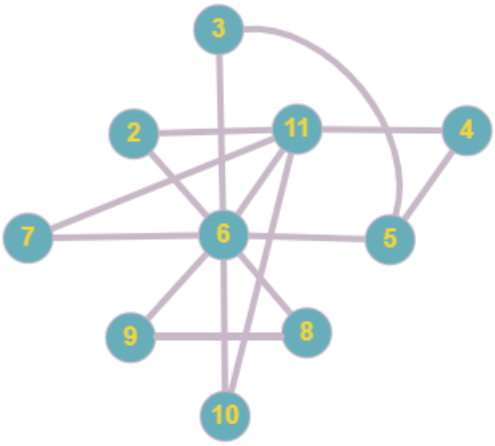
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері;

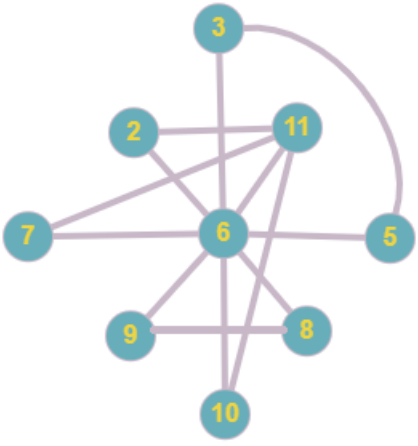


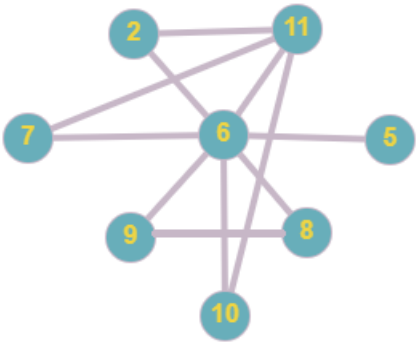

3
1
2
11
4
7
6
5
9
8
10

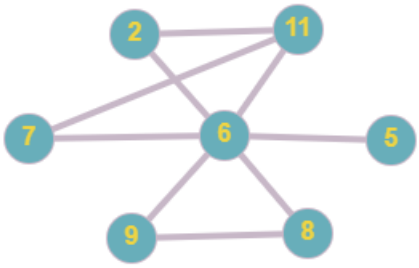

3
1
2
11
4
7
6
5
9
8
10

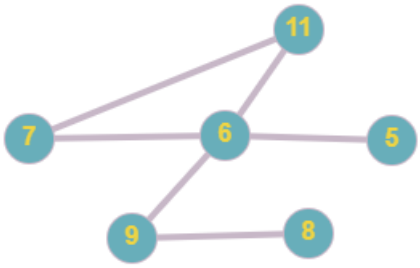

3
2
11
4
7
6
5
9
8
10

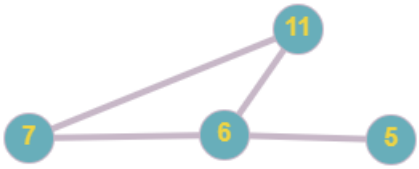

3
2
11
4
7
6
5
9
8
10

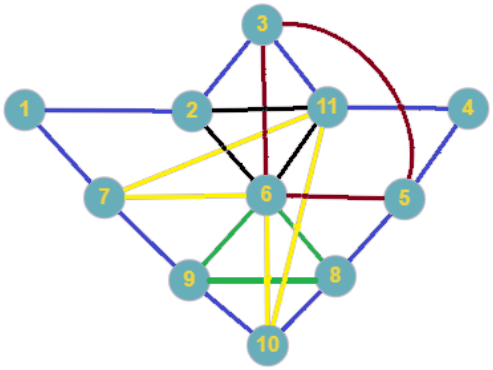

3
2
11
7
6
5
9
8
10


2
11
7
6
5
9
8
10









б)

Код програми:

```
#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)
```

```
{
	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
```

Завдання 9

Спростити формули (привести їх до скороченої ДНФ).

$$\mathbf{x}\,\overline{\mathbf{y}} \vee \mathbf{x}\,\mathbf{y} \vee \mathbf{y}\,\mathbf{z}$$

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

$$(\overline{x}\ y\ z) \vee (x\ \overline{y}\ \overline{z}) \vee (x\ y\ \overline{z}) \vee (x\ y\ z)$$