МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота

З дисципліни

“Дискретна математика”

Виконав:

Студент групи КН-115

Конопльов Павло

Викладач:

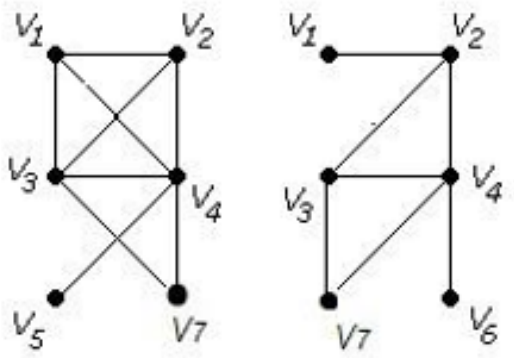
Мельникова Н.І.

Львів-2019р,

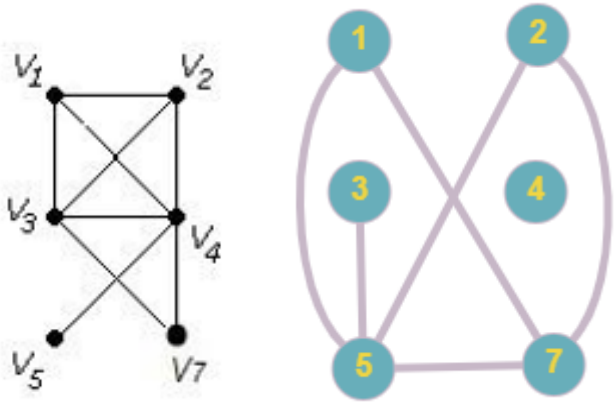
Варіант 15

Завдання № 1

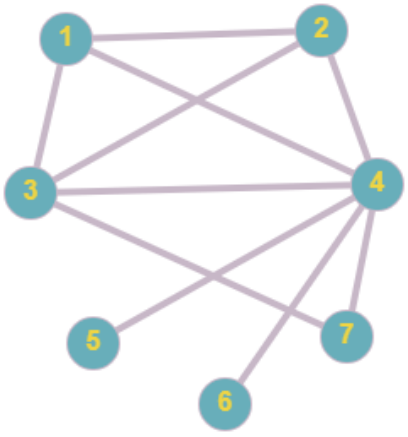
Виконати наступні операції над графами:



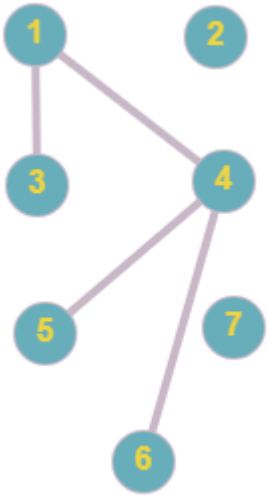
1) знайти доповнення до першого графу



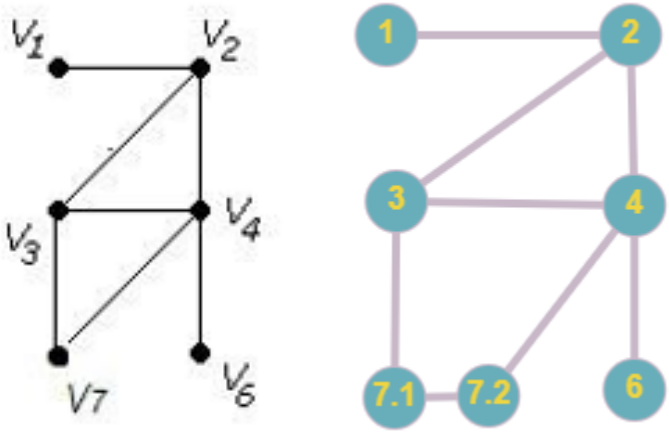
2) об’єднання графів



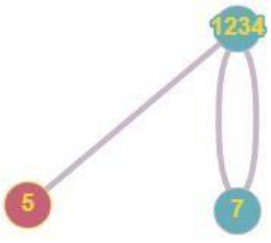
3) кільцеву сумму G1 та G2 (G1+G2)



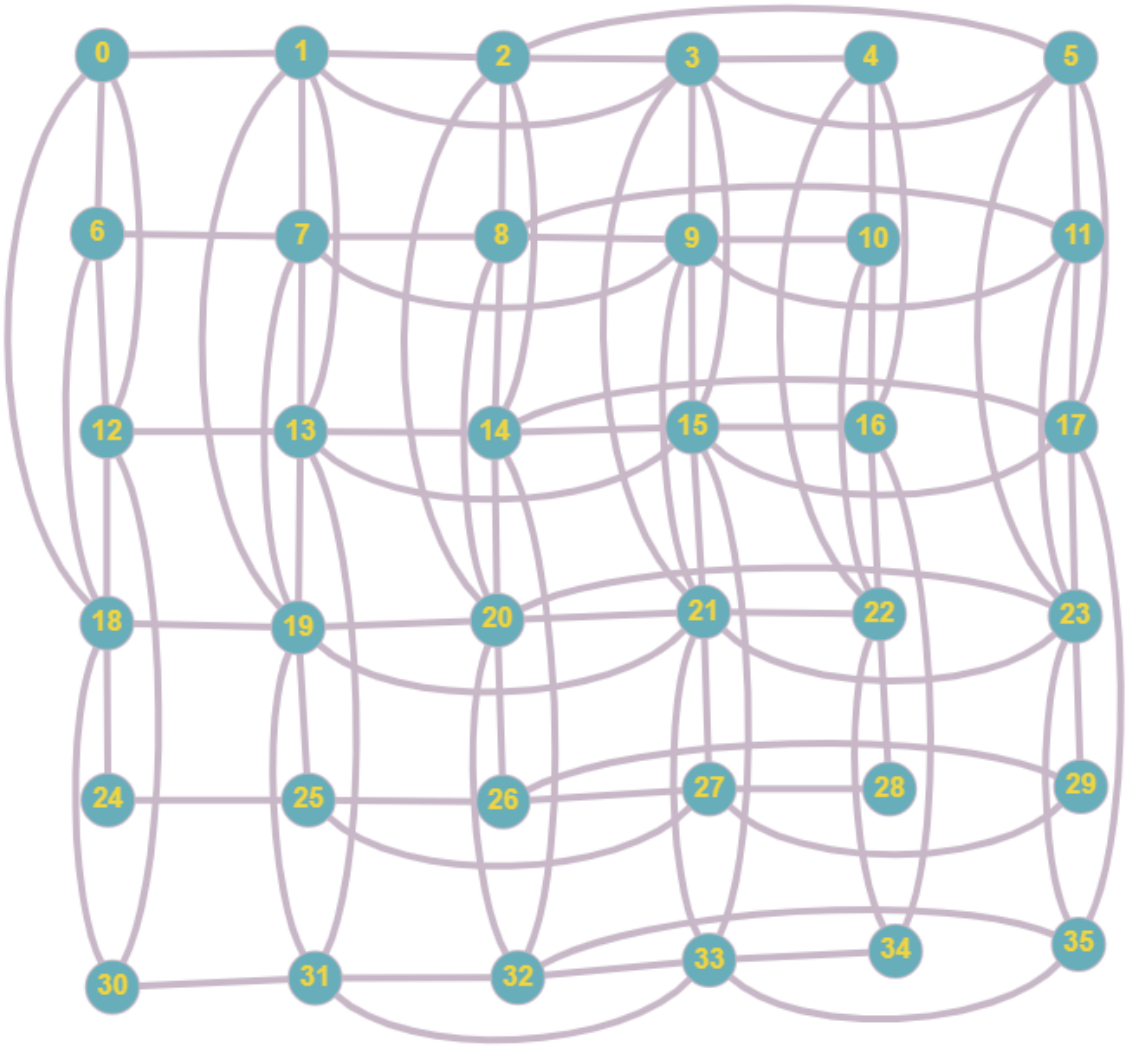
4) розмножити вершину у другому графі



5) виділити підграф А - що скадається з 3-х вершин в G1

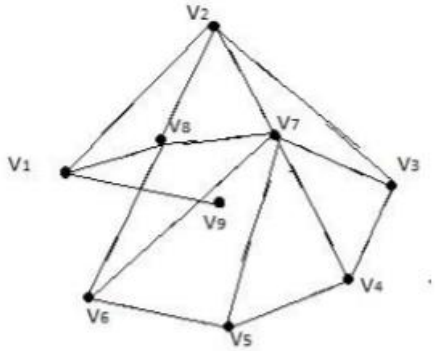


6) добуток графів



Завдання № 2

Скласти таблицю суміжності для орграфа:



	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_1	-	1	2	3	3	2	2	1	1
V_2	1	-	1	2	2	2	1	1	2
V_3	2	1	-	1	2	2	1	2	3
V_4	3	2	1	-	1	2	1	2	4
V_5	3	2	2	1	-	1	1	2	4
V_6	2	2	2	2	1	-	1	1	3
V_7	2	1	1	1	1	1	-	1	3
V_8	1	1	2	2	2	1	1	-	2
V_9	1	2	3	4	4	3	3	2	-

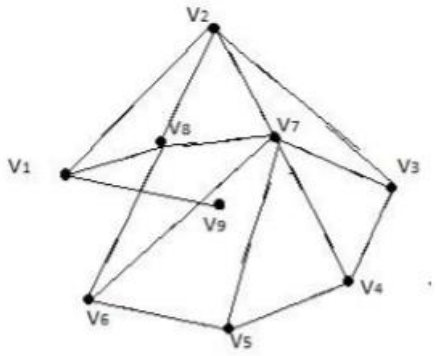
Завдання № 3

Для графа з другого завдання знайти діаметр.

Діаметр графа: 4

Завдання № 4

Для графа з другого завдання виконати обхід дерева вглиб (варіант закінчується на непарне число) або вшир (закінчується на парне число).



Вершина	№	Стек
V_1	0	V_1
V_2	1	V_1V_2
V_3	2	$V_1V_2V_3$
V_4	3	$V_1V_2V_3V_4$
V_5	4	$V_1V_2V_3V_4V_5$
V_6	5	$V_1V_2V_3V_4V_5V_6$
V_7	6	$V_1V_2V_3V_4V_5V_6V_7$
V_8	7	$V_1V_2V_3V_4V_5V_6V_7V_8$
-	-	$V_1V_2V_3V_4V_5V_6V_7$
-	-	$V_1V_2V_3V_4V_5V_6$
-	-	$V_1V_2V_3V_4V_5$
-	-	$V_1V_2V_3V_4$
-	-	$V_1V_2V_3$
-	-	V_1V_2
-	-	V_1
V_9	8	V_1V_9

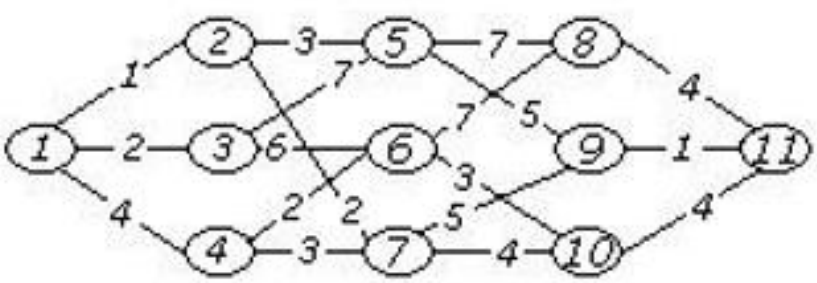
```cpp
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    stack<int> Stack;
    int vert[9];
    int Mtx[9][9] = { { 0, 1, 0, 0, 0, 0, 1, 1, 1 },
                      { 1, 0, 1, 0, 0, 0, 1, 1, 0 },
                      { 0, 1, 0, 1, 0, 0, 1, 0, 0 },
                      { 0, 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 } };

    for (int i = 0; i < 9; i++)
    {
        vert[i] = 0;
    }
    Stack.push(0);
    cout << "DFS: " << endl;
    while (!Stack.empty())
    {
        int node = Stack.top();
        Stack.pop();
        if (vert[node] == 2) continue;
        vert[node] = 2;
        for (int j = 8; j >= 0; j--)
        {
            if (Mtx[node][j] == 1 && vert[j] != 2)
            {
                Stack.push(j);
                vert[j] = 1;
            }
        }
        cout << node + 1 << endl;
    }
}
```

Результат виконання програми

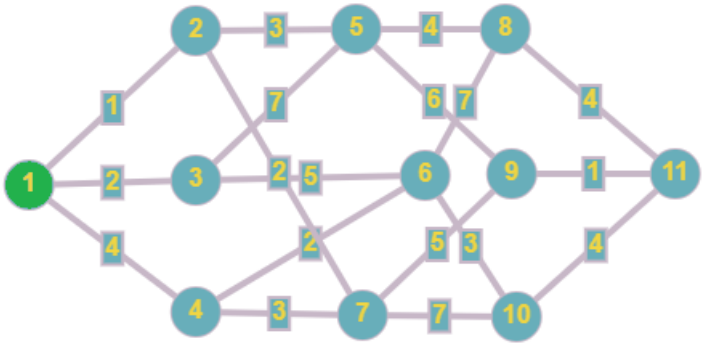
```
DFS:
1
2
3
4
5
6
7
8
9
```

Завдання № 5

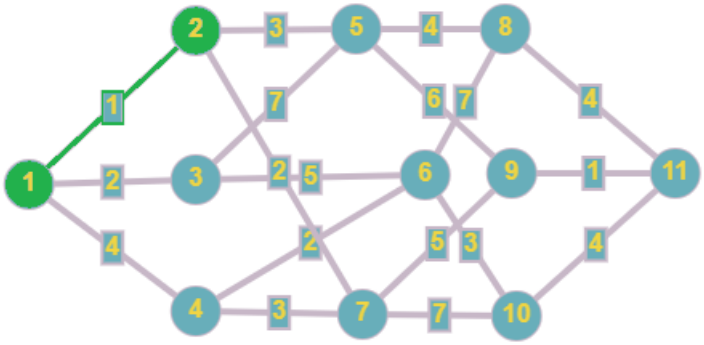
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.



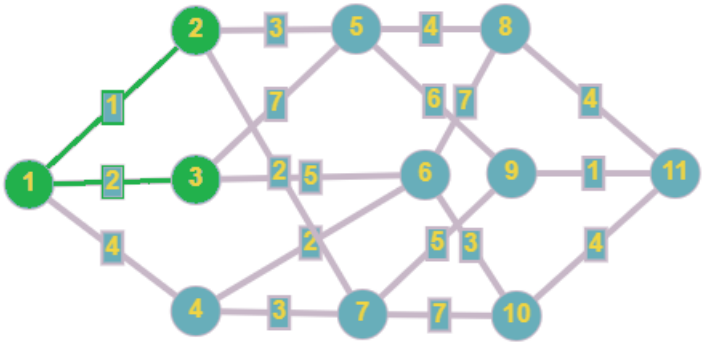
Алгоритм Прима



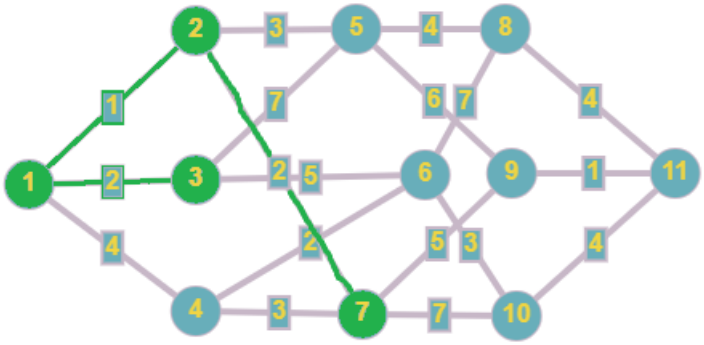
1



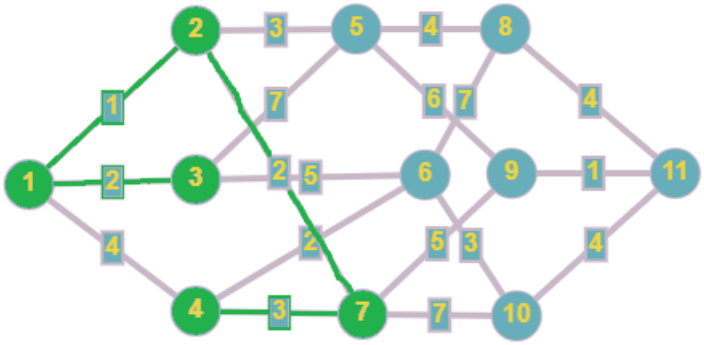
2


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

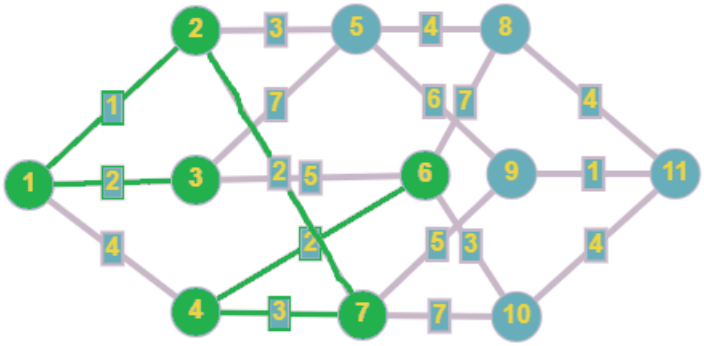
3


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

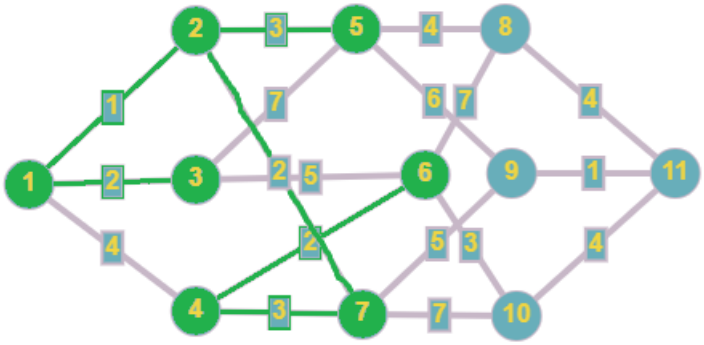
4


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

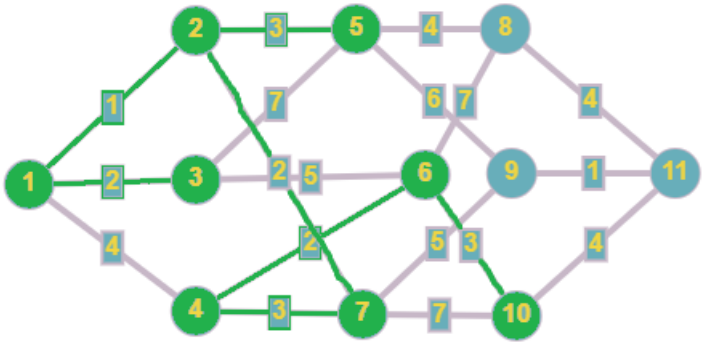
5


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

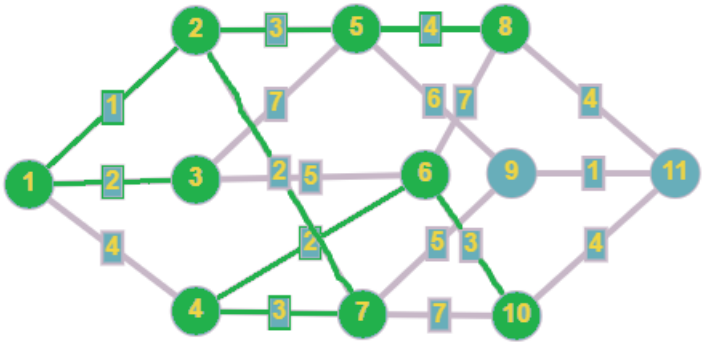
6


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

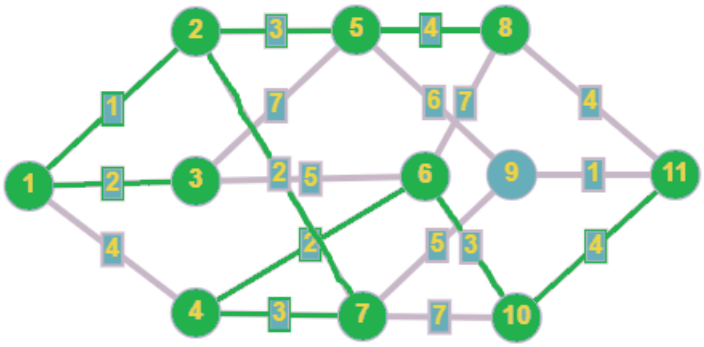
7


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

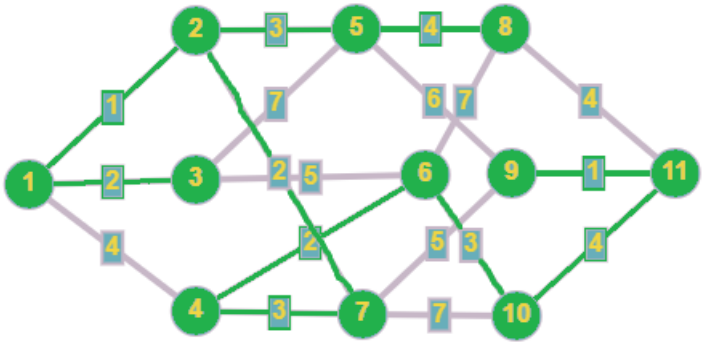
8


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

9


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

10



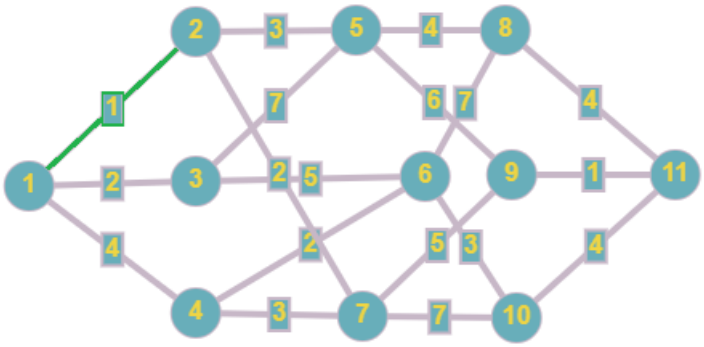
11

Алгоритм Прима

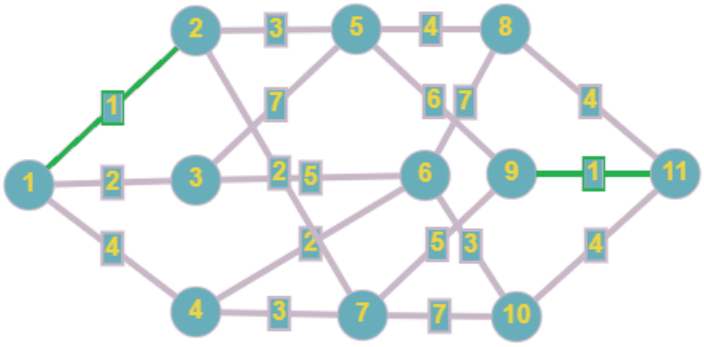
```cpp
#include<iostream>
using namespace std;

void main()
{
    int n, i, j, a, b, x, y, counter = 1;
    int visited[15] = { 0 }, min, mas[15][15];
    int path[100] = {0};

    cout << "Enter size:  ";
    cin >> n;
    cout << "Enter matrix: " << endl;

    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            cin >> mas[i][j];
            if (mas[i][j] == 0)
            {
                mas[i][j] = 999;
            }
        }
    }
    visited[1] = 1;

    while (counter < n)
    {
        for (i = 1, min = 999; i <= n; i++)
            for (j = 1; j <= n; j++)
                if (mas[i][j] < min)
                    if (visited[i] != 0)
                    {
                        min = mas[i][j];
                        a = x = i;
                        b = y = j;
                    }
        if (visited[x] == 0 || visited[y] == 0)
        {
            path[counter-1] = b;
            counter++;
            visited[b] = 1;
        }
        mas[a][b] = mas[b][a] = 999;
    }
    cout << " " << endl;
    cout << 1 << " --> ";
    for (int i = 0; i < n - 1; i++)
    {
        cout << path[i];
        if (i < n - 2)
        {
            cout << " --> ";
        }
```

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


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

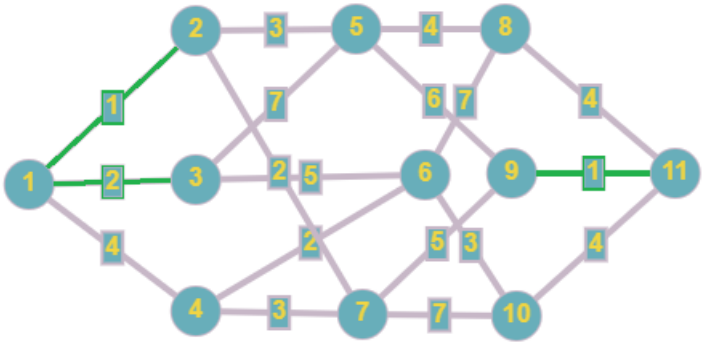
Алгоритм Краскала



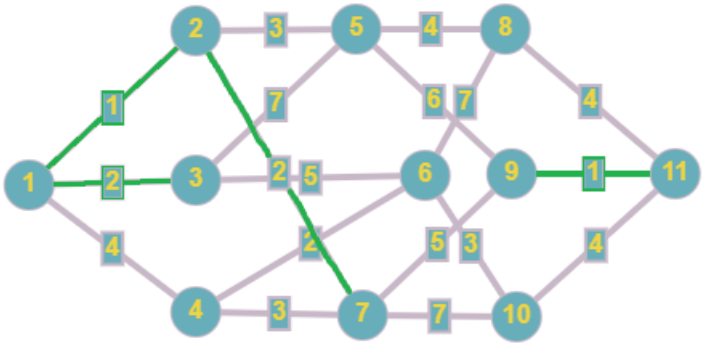
1



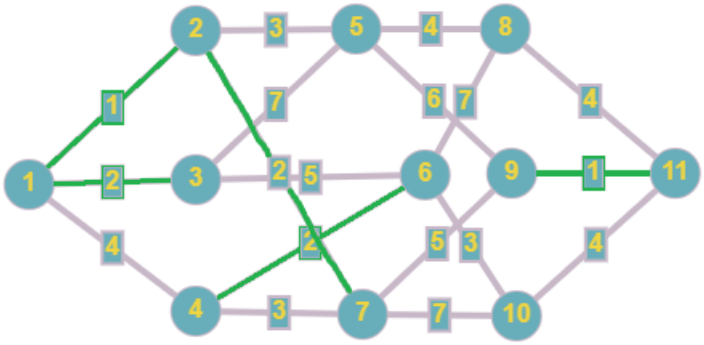
2



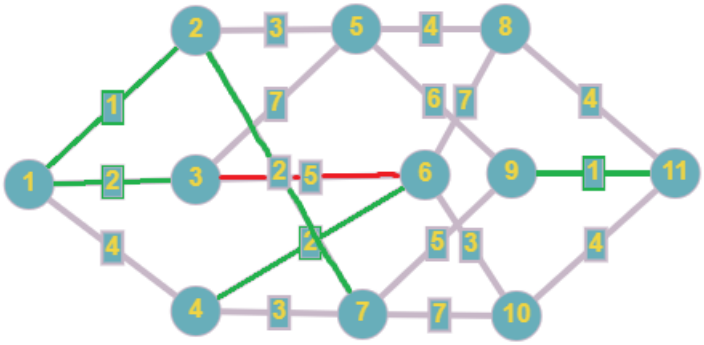
3



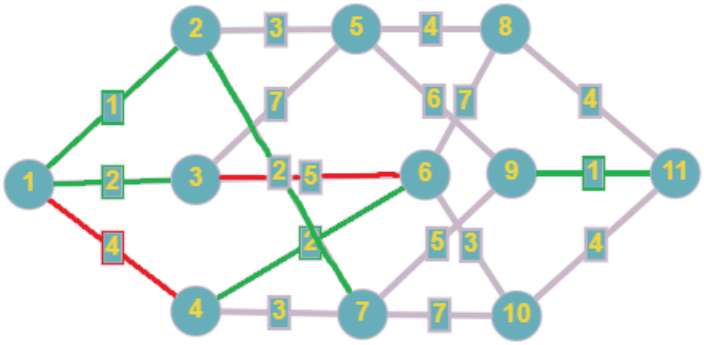
4


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

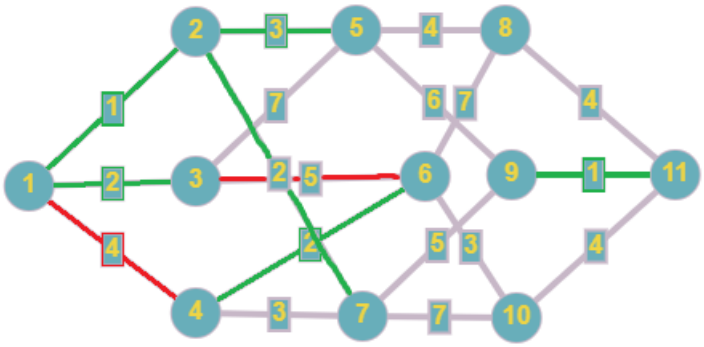
5


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

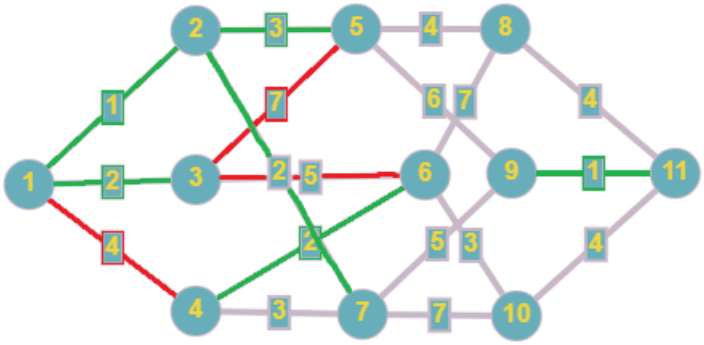
6


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

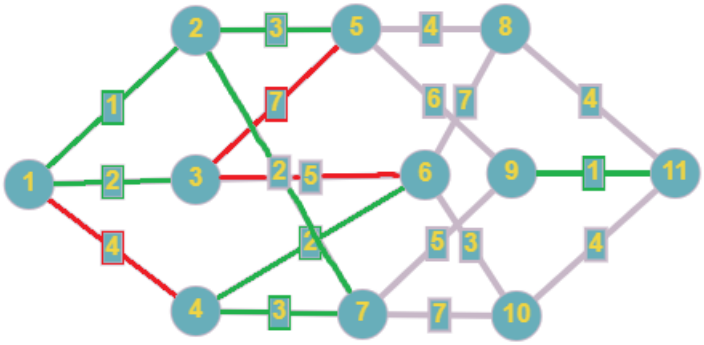
7


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

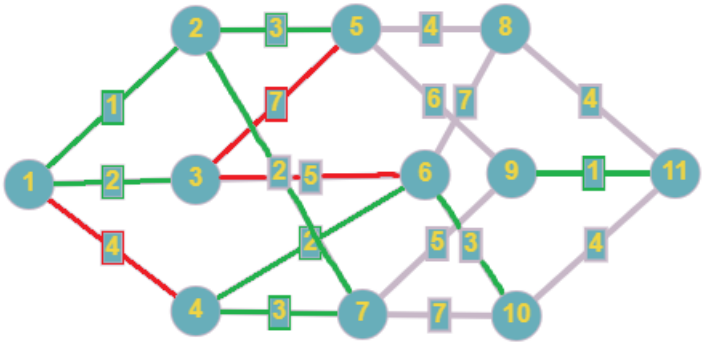
8


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

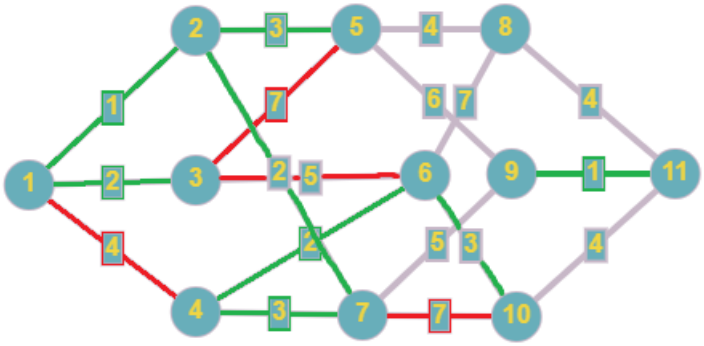
9


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

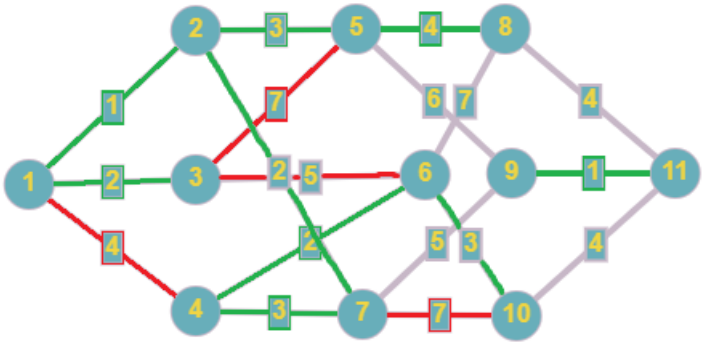
10


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

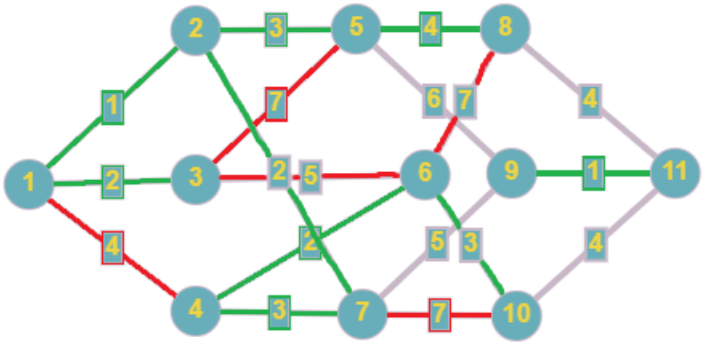
11


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

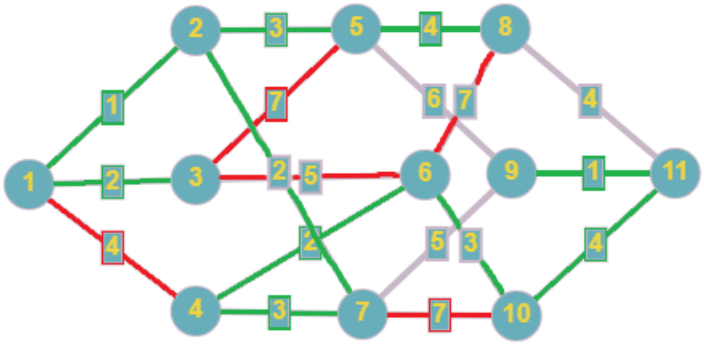
12


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

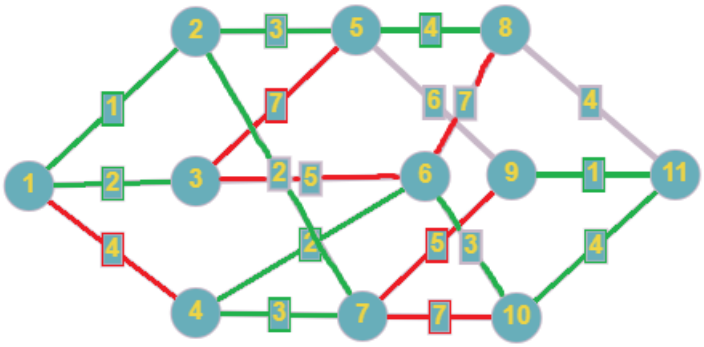
13


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

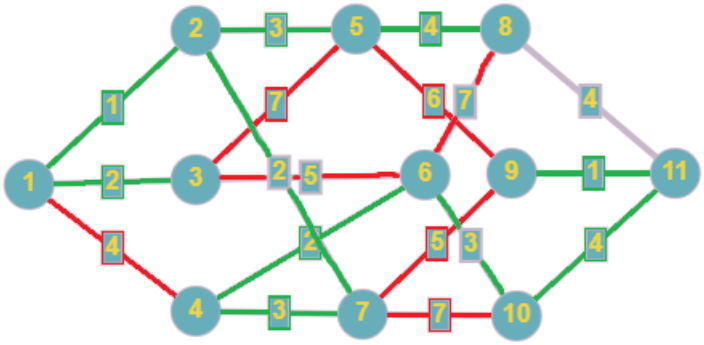
14


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

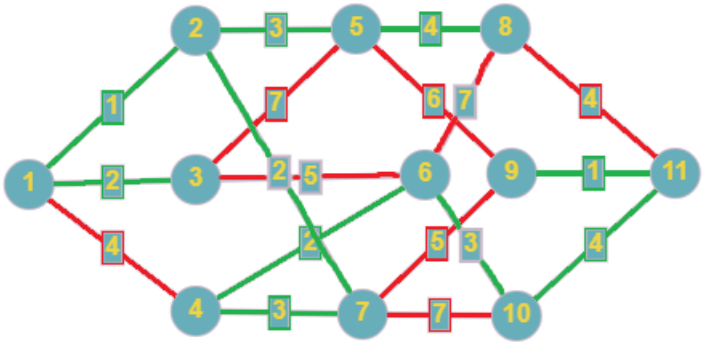
15


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

16


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

17


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

18

```
#define V 11
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
int parent[V];

int find(int i)
{
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void union1(int i, int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

void kruskalMST(int cost[][V])
{
    int mincost = 0;
    for (int i = 0; i < V; i++)
        parent[i] = i;

    int edge_count = 0;
    while (edge_count < V - 1) {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (find(i) != find(j) && cost[i][j] < min) {
                    min = cost[i][j];
                    a = i;
                    b = j;
                }
            }
        }
        union1(a, b);
        cout << "Edge " << edge_count++ << ':' << '(' << a << ", " << b << ')' << " cost: " << min << endl;
        mincost += min;
    }
    cout << "Min cost : " << mincost;
}
```

```
int main()
{
    int cost[][V] = {
        { INT_MAX, 1, 2, 4, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX },
        { 1, INT_MAX, INT_MAX, INT_MAX, 3, INT_MAX, 2, INT_MAX, INT_MAX, INT_MAX, INT_MAX },
        { 2, INT_MAX, INT_MAX, INT_MAX, 7, 6, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX },
        { 4, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 2, 3, INT_MAX, INT_MAX, INT_MAX, INT_MAX },
        { INT_MAX, 3, 7, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 7, 5, INT_MAX, INT_MAX },
        { INT_MAX, INT_MAX, 6, 2, INT_MAX, INT_MAX, INT_MAX, 7, INT_MAX, 3, INT_MAX },
        { INT_MAX, 2, INT_MAX, 3, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 5, 4, INT_MAX },
        { INT_MAX, INT_MAX, INT_MAX, INT_MAX, 7, 7, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 4 },
        { INT_MAX, INT_MAX, INT_MAX, INT_MAX, 5, INT_MAX, 5, INT_MAX, INT_MAX, INT_MAX, 1 },
        { INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 3, 4, INT_MAX, INT_MAX, INT_MAX, 4 },
        { INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 4, 1, 4, INT_MAX },
    };
    kruskalMST(cost);
}
```

Результат виконання програми

```
Edge 0:(0, 1) cost: 1
Edge 1:(8, 10) cost: 1
Edge 2:(0, 2) cost: 2
Edge 3:(1, 6) cost: 2
Edge 4:(3, 5) cost: 2
Edge 5:(1, 4) cost: 3
Edge 6:(3, 6) cost: 3
Edge 7:(5, 9) cost: 3
Edge 8:(7, 10) cost: 4
Edge 9:(9, 10) cost: 4
Min cost : 25
```

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	∞

Матриця:

	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	∞

Переходимо в 4 вершину. Довжина шляху 1

	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	∞

Переходимо в 6 вершину. Довжина шляху 2

0	2	3	5	146	7	8
2	∞	3	2	1	3	3
3	3	∞	5	1	5	1
5	2	5	∞	2	2	2
146	1	1	2	∞	7	5
7	3	5	2	7	∞	5
8	3	1	2	5	5	∞

Переходимо в 2 вершину. Довжина шляху 3

∞	1462	3	5	7	8
1462	∞	5	1	5	1
3	5	∞	2	2	2
5	1	2	∞	7	5
7	5	2	7	∞	5
8	1	2	5	5	∞

Переходимо в 5 вершину. Довжина шляху 4

∞	3	14625	7	8
3	∞	2	2	2
14625	2	∞	7	5
7	2	7	∞	5
8	2	5	5	∞

Переходимо в 3 вершину. Довжина шляху 6

∞	146253	7	8
146253	∞	7	5
7	7	∞	5
8	5	5	∞

Переходимо в 8 вершину. Довжина шляху 11

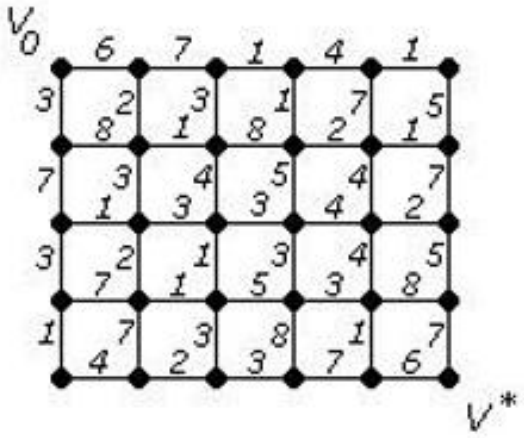
∞	7	1462538
7	∞	5
1462538	5	∞

Переходимо в 7 вершину. Довжина шляху 16. Найкоротший шлях 14625387

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define V 8

int travllingSalesmanProblem(int graph[][V], int s)
{
    vector<int> vertex; for (int i = 0; i < V; i++)
        if (i != s) vertex.push_back(i);
    int min_path = INT_MAX; do {
        int current_pathweight = 0;
        int k = s;

        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]]; k = vertex[i];
        }

        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);

    } while (next_permutation(vertex.begin(), vertex.end()));

    return min_path;
}

int main()
{
    int graph[][V] = { { 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 } };

    int s = 0;
    cout << travllingSalesmanProblem(graph, s) << endl;
}
```

Завдання № 7

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



Довжина найкоротшого шляху: 29

Алгоритм Дейкстри

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

using namespace std;

int main()
{
    int a = 1000;
    int Mtx[100][100];
    int x[100];
    int LengthWay[100];
    int PreVertex[100];
    int size;

    cout << "Amount of verticles: ";
    cin >> size;
    cout << "Enter weight matrix: " << endl;

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            cin >> Mtx[i][j];
        }
    }

    int start;
    int end;

    do
    {
        cout << "Enter start vertex: ";
        cin >> start;
    } while (start > (size - 1) || start < 0);

    int v;

    start = start - 1;

    for (int i = 0, j = 0; i < size; i++, j++)
    {
        end = i;
        if (start == end) continue;
        else
        {
            int u;
            for (u = 0; u < size; u++)
```

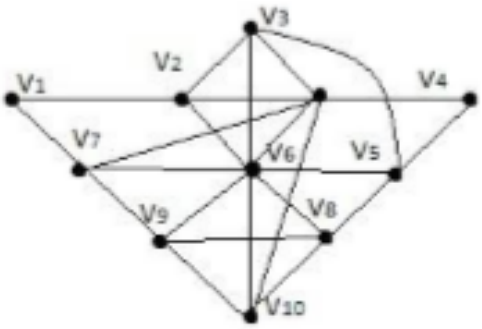
```
            int u;
            for (u = 0; u < size; u++)
            {
                LengthWay[u] = a;
                x[u] = 0;
            }

            PreVertex[start] = 0;
            LengthWay[start] = 0;
            x[start] = 1;
            v = start;

            while (1)
            {
                for (u = 0; u < size; u++)
                {
                    if (Mtx[v][u] == 0)continue;

                    if (x[u] == 0 && LengthWay[u] > LengthWay[v] + Mtx[v][u])
                    {
                        LengthWay[u] = LengthWay[v] + Mtx[v][u];
                        PreVertex[u] = v;
                    }
                }

                int w = a;


                for (u = 0; u < size; u++)
                {
                    if (x[u] == 0 && LengthWay[u] < w)
                    {
                        v = u;
                        w = LengthWay[u];
                    }
                }

                if (v == end) break;

                x[v] = 1;
            }
        }
    }
    int fin;
```

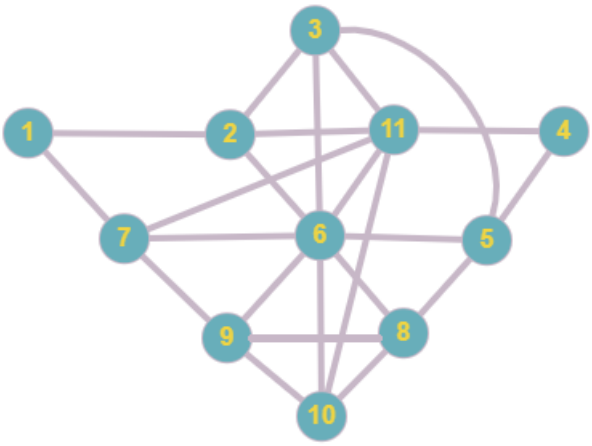
```
    int fin;

    do
    {
        cout << "Enter finish vertex: ";
        cin >> fin;
    } while (fin > (size+1) || fin < 0);

    end = fin-1;
    int u = fin-1;

    if (fin == end + 1)
    {
        cout << "Smallest way from " << start + 1 << " in vertex " << end + 1 << " :";
        int u = end;

        while (u != start)
        {
            cout << " " << u + 1;
            u = PreVertex[u];
        }
        cout << " " << start + 1 << " And cost --> " << LengthWay[end];
    }
}
```

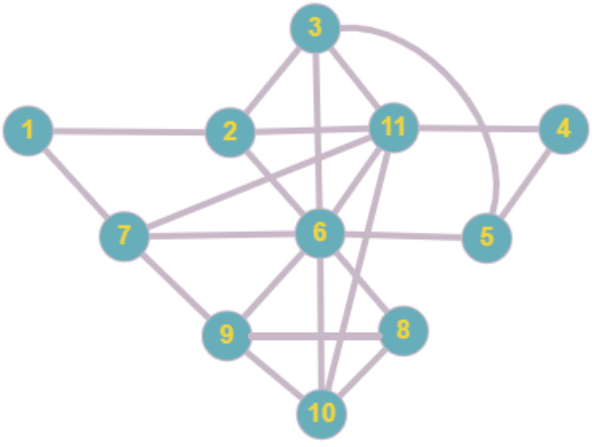
```
Amount of verticles: 30
Enter weight matrix:
 0 6  0  0 0 0  4 0   0  0  0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 6 0 1  0 0 0 0 8 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0
 0 1 0 1 0 0 0 0 3 0 0 0 0 0 0  0 0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 3 0 0 0 0 1 0 0 0 0  0 0 0  0 0 0 0 0 0 0  0  0 0 0 0 0
 0 0 0  3 0 3  0 0 0 0 5 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 0 0 0 7 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
 4 0 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 8  0 0 0 0 2 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 3 0 0 0 0 1 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 1 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 5 0 0 0 0 2 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 7 0 0 0 0 4 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 5 0 0 0 0 0  0 7 0 0  0 0 5 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 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 3  0 0 0 0 1 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0  0 0 0 0 0 0 0 0 4 0 0 0 0 2 0 3  0  0 0 0 4 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0  1 0 0 0 0  3 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 7 0 0 0  0 7 0  0 0  0 0 0 8 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 7 0 0  0 0 8 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 7 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 0  1 0 0 0 0 3 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 4 0 0 0 0 1 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 2 0 0 0 0 8 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 8 0 0 0 0 5 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 8 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 2 0 0 0 0 4  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 3 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 3 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 3 0 0 0 0 3 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 7 0 0 0 0 6 0
Enter start vertex: 1
Enter finish vertex: 30
Smallest way from 1 in vertex 30 : 30 29 28 22 21 15 14 8 7 1
And cost --> 22
```

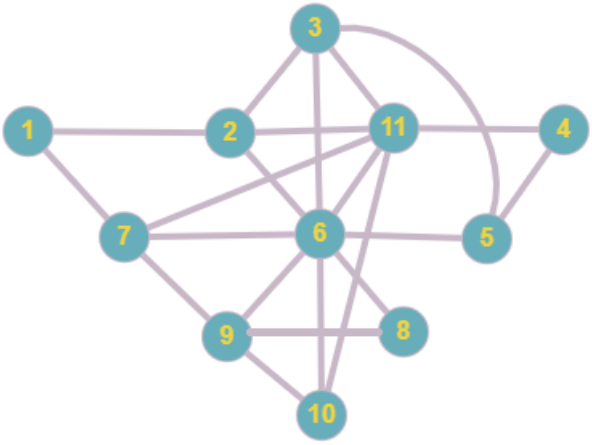
Завдання № 8

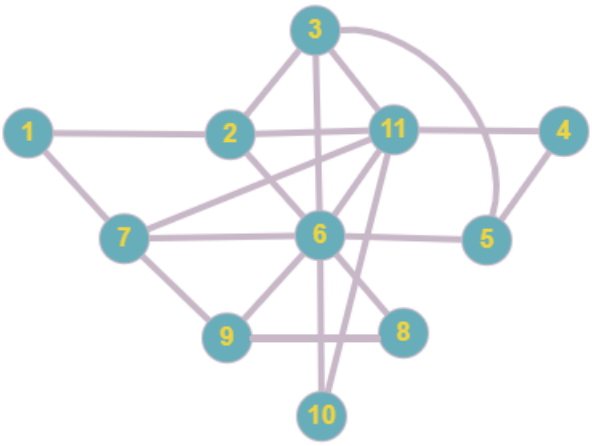
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері;

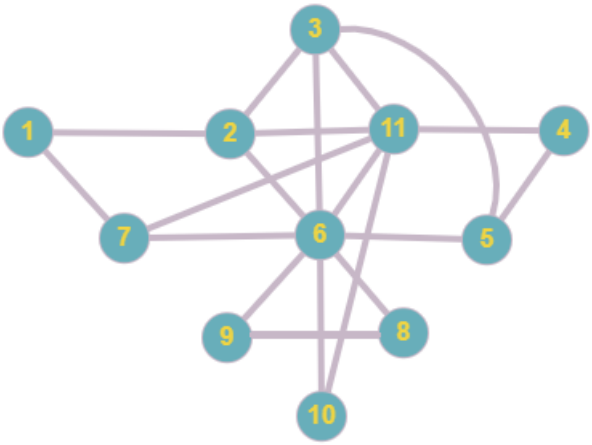


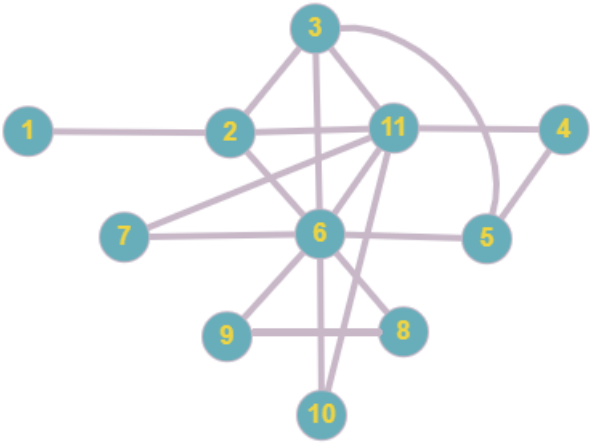


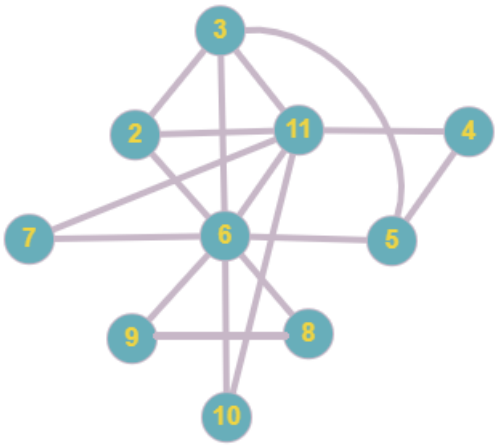


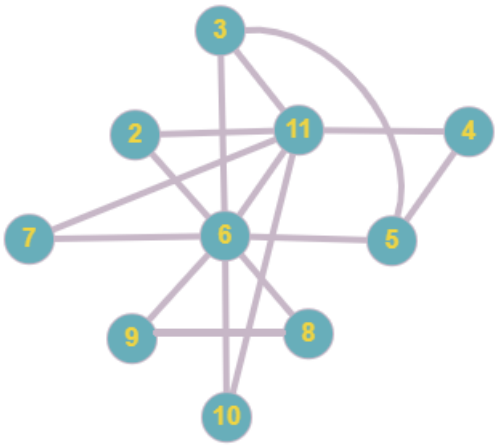

3
1
2
11
4
7
6
5
9
8
10

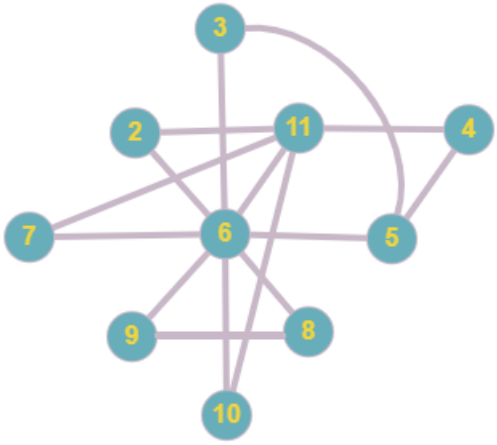

3
1
2
11
4
7
6
5
9
8
10

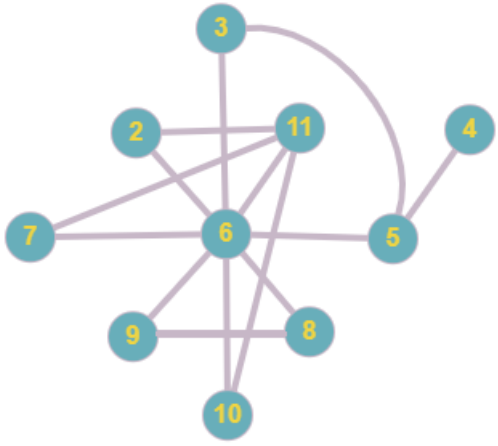

3
1
2
11
4
7
6
5
9
8
10

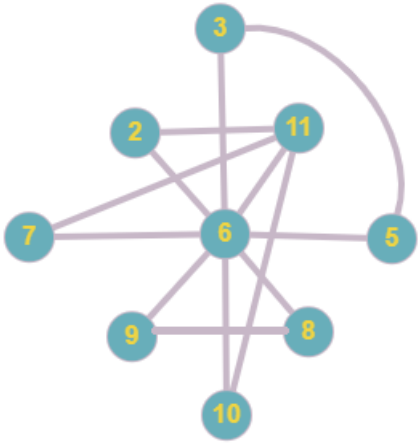

3
1
2
11
4
7
6
5
9
8
10

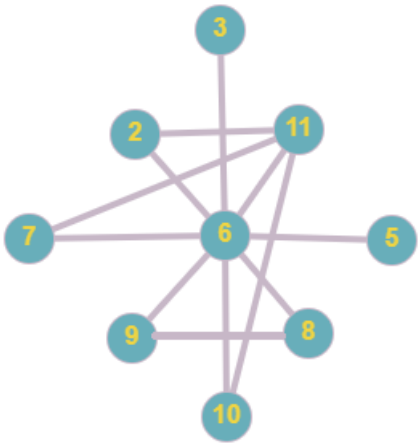

3
2
11
4
7
6
5
9
8
10

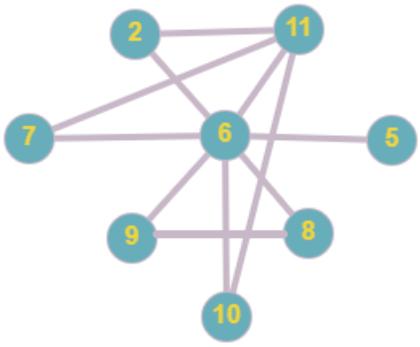

3
2
11
4
7
6
5
9
8
10

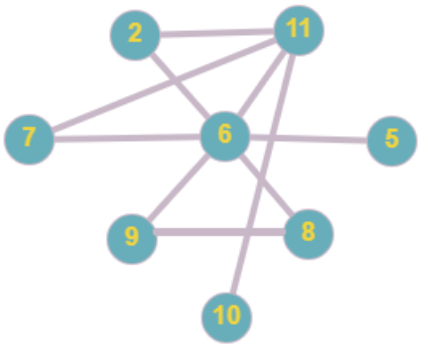

3
2
11
4
7
6
5
9
8
10

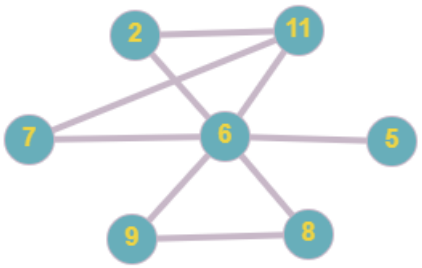

3
2
11
4
7
6
5
9
8
10

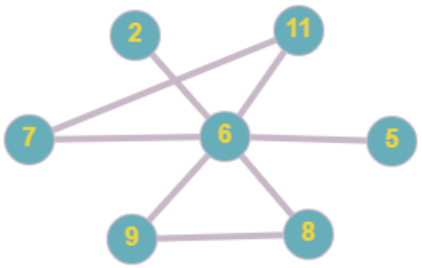

3
2
11
7
6
5
9
8
10

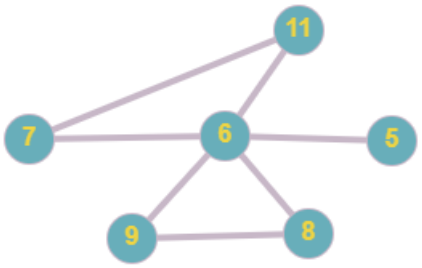

3
2
11
7
6
5
9
8
10

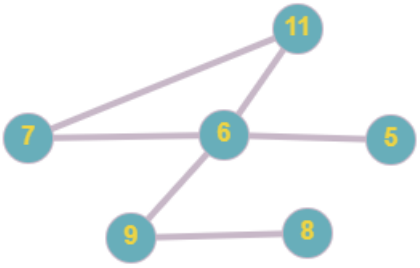

2
11
7
6
5
9
8
10

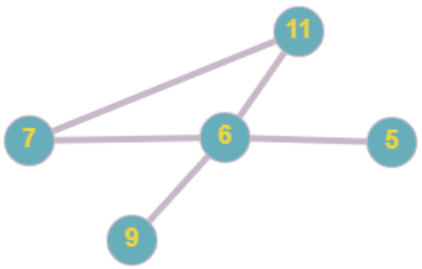

2
11
7
6
5
9
8
10

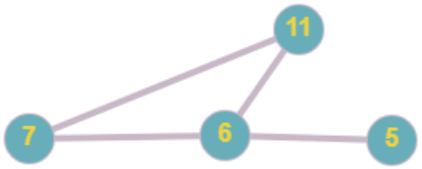

2
11
7
6
5
9
8


2
11
7
6
5
9
8


11
7
6
5
9
8

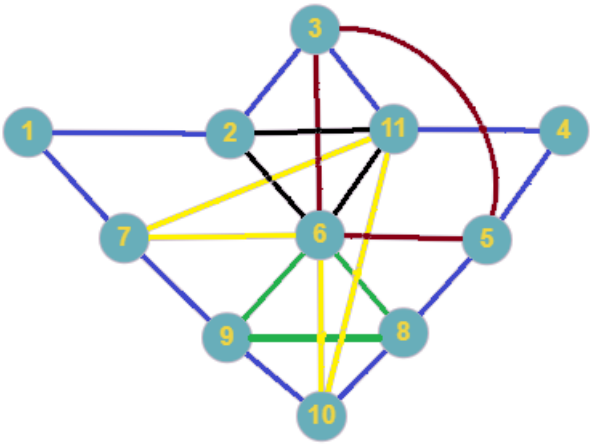

11
7
6
5
9
8


11
7
6
5
9


11
7
6
5


11
7
6
5

б) елементарних циклів.



```cpp
#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);
```

```cpp
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;
}
```

```cpp
int main()
{
    Graph g1(11);
    g1.addEdge(0, 1);
    g1.addEdge(0, 6);
    g1.addEdge(1, 2);
    g1.addEdge(1, 10);
    g1.addEdge(1, 5);
    g1.addEdge(2, 11);
    g1.addEdge(2, 5);
    g1.addEdge(2, 4);
    g1.addEdge(3, 4);
    g1.addEdge(4, 5);
    g1.addEdge(4, 7);
    g1.addEdge(5, 6);
    g1.addEdge(5, 7);
    g1.addEdge(5, 8);
    g1.addEdge(5, 9);
    g1.addEdge(6, 8);
    g1.addEdge(7, 8);
    g1.addEdge(7, 9);
    g1.addEdge(8, 9);
    g1.addEdge(9, 10);
    g1.addEdge(10, 3);
    g1.addEdge(10, 6);
    g1.addEdge(10, 5);

    g1.printEulerTour();
}
```

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

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

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

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