МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахунково-графічна робота

з дисципліни «Дискретна математика»

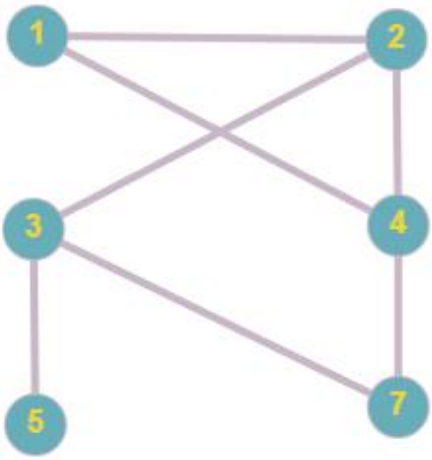
Виконав:
студент групи КН-115
Сирватка Максим
Викладач:
Мельникова Н.І.

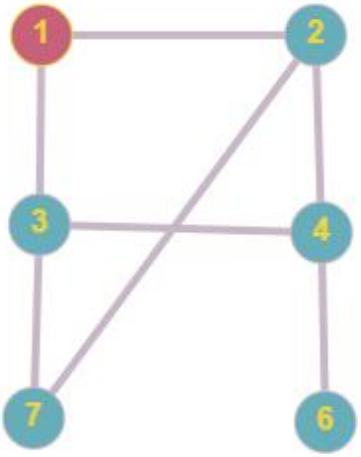
Львів – 2019 р.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

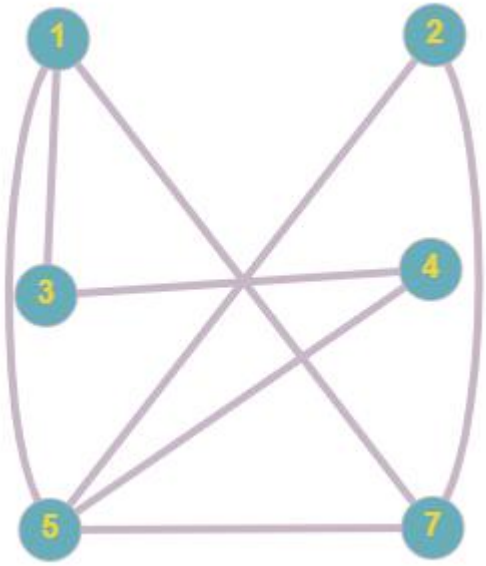
Варіант 14

1. Виконати наступні операції над графами:
1) знайти доповнення до першого графу;
2) об'єднання графів;
3) кільцеву суму G1 та G2 (G1+G2);
4) розмножити вершину у другому графі;
5) виділити підграф А - що складається з 3-х вершин в G1;
6) добуток графів.

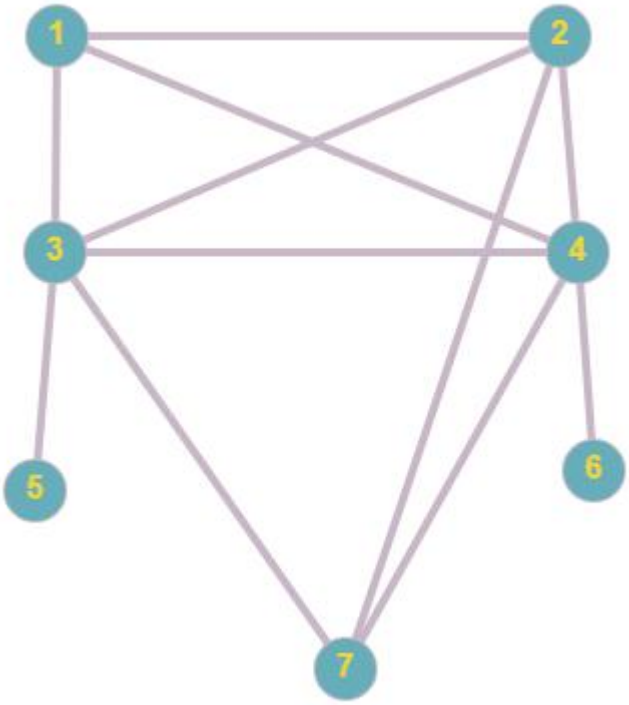




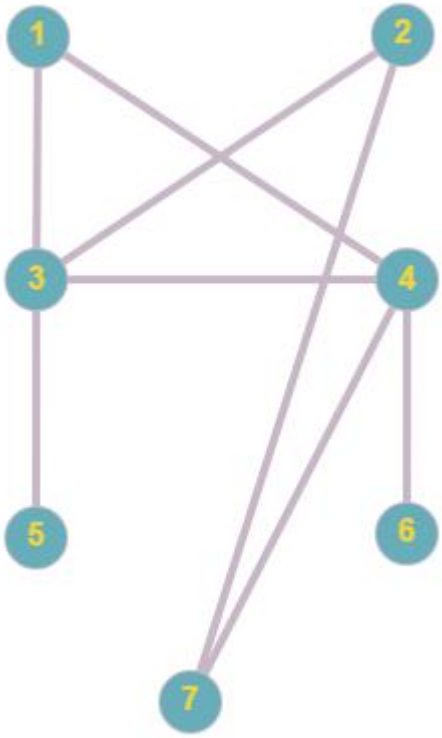
1) Доповнення до першого графу



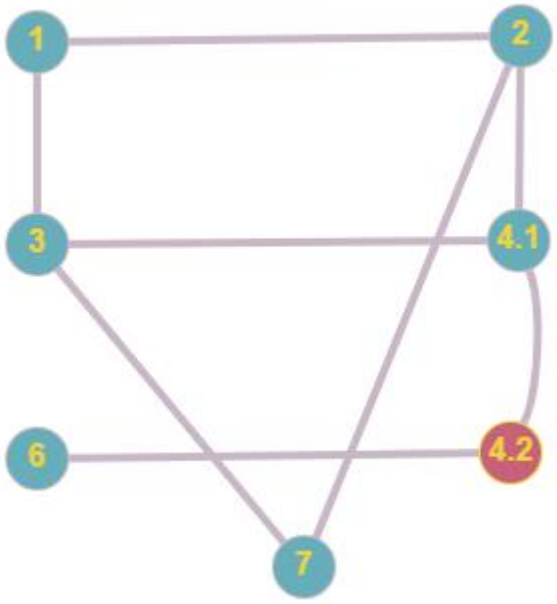
2) Об’єднання графів



3) Кільцева сума

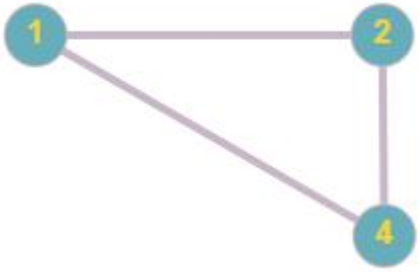


4) **Розмноження вершини в другому графі**



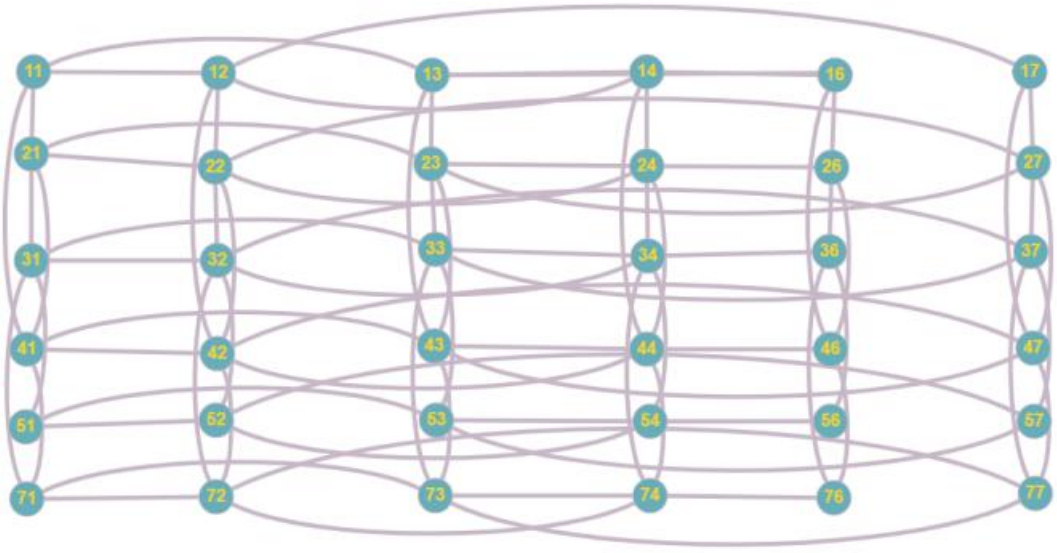
5) **Виділення підграфа А з трьох вершин в першому графі та стягнення А в G1**

Підграф А:

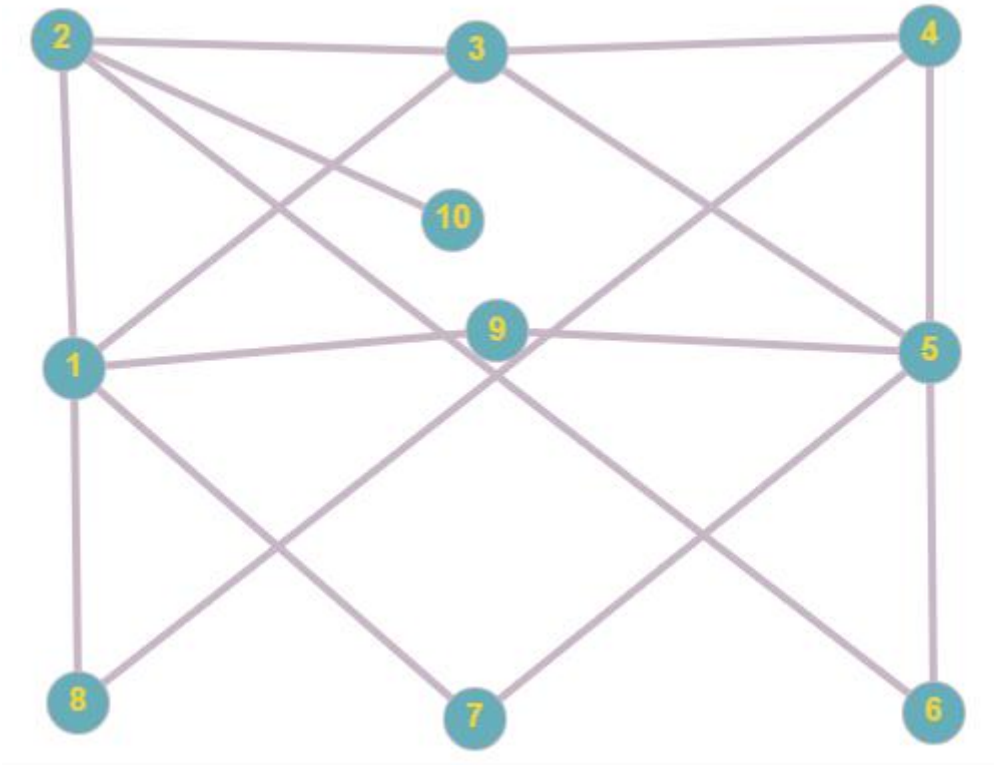


Стягнення А в G1:

6) Добуток графів



2. Скласти таблицю суміжності для орграфа.



Таблиця суміжності:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	1	1	0	0	0	1	1	1	0
2	1	0	1	0	0	1	0	0	0	1
3	1	1	0	1	1	0	0	0	0	0
4	0	0	1	0	1	0	0	1	0	0
5	0	0	1	1	0	1	1	0	1	0
6	0	1	0	0	1	0	0	0	0	0
7	1	0	0	0	1	0	0	0	0	0
8	1	0	0	1	0	0	0	0	0	0
9	1	0	0	0	1	0	0	0	0	0
10	0	1	0	0	0	0	0	0	0	0

3. Для графа з другого завдання знайти діаметр.

Діаметр: 3 (4->3->2->10)

4. Для графа з другого завдання виконати обхід дерева вшир.

V	**№**	**черга**
1	1	1
2	2	12
3	3	123
7	4	1237
8	5	12378
9	6	123789
-	-	23789
6	7	236789
10	8	23678910
-	-	3678910
4	9	36789104
5	10	367891045
-	-	67891045
-	-	7891045
-	-	891045
-	-	91045
-	-	1045
-	-	45
-	-	4
-	-	ø

Програмна реалізація

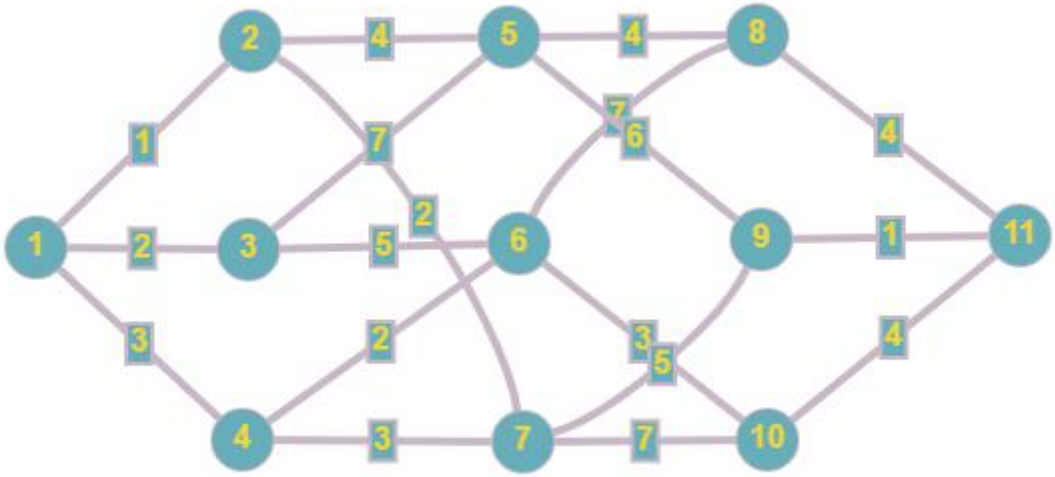
```
#include <iostream>

using namespace std;

const int n = 10;

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

int main()
{
    int start;
    cout << "Enter the start vertex: ";
    cin >> start;
    bool visited[n];
    cout << "Adjacency matrix: " << endl;
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
        for (int j = 0; j < n; j++)
            cout << " " << graph[i][j];
```

```
}

int vertex = start - 1;
int count = 0, head = 0;
int queue[n];
for (int i = 0; i < n; i++)
{
    queue[i] = 0;
}
queue[count++] = vertex;
visited[vertex] = true;
cout << "Bypass of the graph: " << endl;
while (head < count)
{
    vertex = queue[head++];
    cout << vertex + 1 << " ";
    for (int i = 0; i < n; i++)
        if (graph[vertex][i] && !visited[i])
        {
            queue[count++] = i;
            visited[i] = true;
        }
}
cout << endl;
system("pause");
return 0;
```

Результат виконання програми

```
Enter the start vertex: 1
Adjacency matrix:
 0 1 1 0 0 0 1 1 1 0
 1 0 1 0 0 1 0 0 0 1
 1 1 0 1 1 0 0 0 0 0
 0 0 1 0 1 0 0 1 0 0
 0 0 1 1 0 1 1 0 1 0
 0 1 0 0 1 0 0 0 0 0
 1 0 0 0 1 0 0 0 0 0
 1 0 0 1 0 0 0 0 0 0
 1 0 0 0 1 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0
Bypass of the graph:
1 2 3 7 8 9 6 10 4 5
```

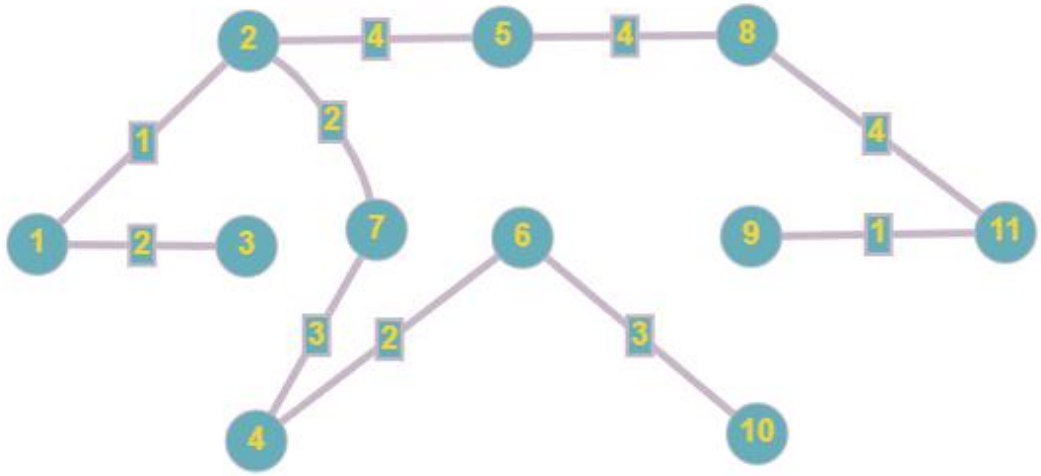
5. *Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.*



Алгоритм Краскала:

$$V = \{1; 2; 9; 11; 3; 4; 6; 7; 10; 5; 8\}$$

$$E = \{(1; 2); (9; 11); (1; 3); (4; 6); (2; 7); (4; 7); (6; 10); (2; 5); (5; 8); (8; 11)\}$$



Програмна реалізація

```
#include <iostream>

const int n = 12;

using namespace std;

int min, path[12];

int main()
{
    int i, j;
    int a, b, u, v;
    int l = 1;
    cout << "The cost adjacency matrix: " << endl;
    int graph[12][12] = {
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0},
{0, 1, 0, 0, 0, 4, 0, 2, 0, 0, 0, 0},
{0, 2, 0, 0, 0, 7, 5, 0, 0, 0, 0, 0},
{0, 3, 0, 0, 0, 0, 2, 3, 0, 0, 0, 0},
{0, 0, 4, 7, 0, 0, 0, 0, 4, 6, 0, 0},
{0, 0, 0, 5, 2, 0, 0, 0, 7, 0, 3, 0},
{0, 0, 2, 0, 3, 0, 0, 0, 0, 5, 7, 0},
{0, 0, 0, 0, 0, 4, 7, 0, 0, 0, 0, 4},
{0, 0, 0, 0, 0, 6, 0, 5, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 3, 7, 0, 0, 0, 4},
{0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 4, 0},
    };
    for (i = 1; i < n; i++)
    {
        for (j = 1; j < n; j++)
        {
            cout << graph[i][j] << " ";
            if (graph[i][j] == 0)
                graph[i][j] = 999;
        }
        cout << endl;
    }
    cout << "The verteces of the minimal tree are: " << endl;
    while (1)
    {
        for (i = 1, min = 999;i < n;i++)
        {
            for (j = 1;j < n;j++)
            {
                if (graph[i][j] < min)
                {
                    min = graph[i][j];
                    a = u = i;
                    b = v = j;
                }
            }
        }
        while (path[u])
            u = path[u];
        while (path[v])
            v = path[v];
        bool uni = false;
        if (u != v)
        {
            path[v] = u;
            uni = true;
        }
        if (uni)
        {
            cout << l++ <<  " edge (" << a << ", " << b << ") = " << min << endl;
        }
        graph[a][b] = graph[b][a] = 999;
    }
    return 0;
```

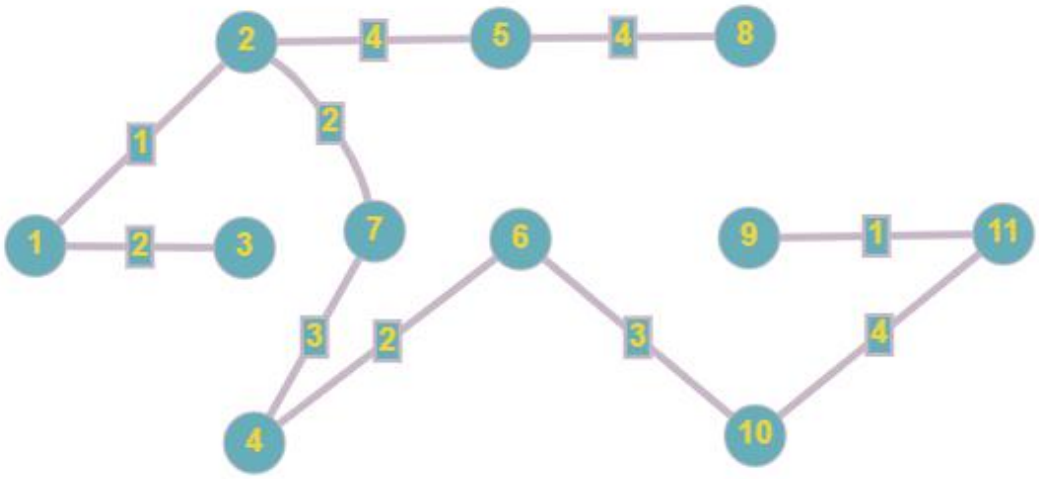
Результат виконання програми

```
The cost adjacency matrix:
0 1 2 3 0 0 0 0 0 0 0
1 0 0 0 4 0 2 0 0 0 0
2 0 0 0 7 5 0 0 0 0 0
3 0 0 0 0 2 3 0 0 0 0
0 4 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
The verteces of the minimal tree are:
1 edge (1, 2) = 1
2 edge (9, 11) = 1
3 edge (1, 3) = 2
4 edge (2, 7) = 2
5 edge (4, 6) = 2
6 edge (1, 4) = 3
7 edge (6, 10) = 3
8 edge (2, 5) = 4
9 edge (5, 8) = 4
10 edge (8, 11) = 4
```

Алгоритм Прима:

$$V = \{1; 2; 3; 7; 4; 6; 10; 11; 9; 5; 8\}$$

$$E = \{(1; 2); (1; 3); (2; 7); (4; 7); (4; 6); (6; 10); (10; 11); (9; 11); (2; 5); (5; 8)\}$$



Програмна реалізація

```
#include <iostream>
#include <windows.h>
#include <conio.h>

using namespace std;

const int MAX = 2000;

int main()
{
    char s;
    do
    {
        system("cls");
        setlocale(LC_ALL, "Ukr");
        const int n = 11;


        int graph[11][11] =
        {
            {0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0},
            {1, 0, 0, 0, 4, 0, 2, 0, 0, 0, 0},
            {2, 0, 0, 0, 7, 5, 0, 0, 0, 0, 0},
            {3, 0, 0, 0, 0, 2, 3, 0, 0, 0, 0},
            {0, 4, 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},
        };
```

```
cout << "Adjacency matrix : " << endl;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        cout << graph[i][j] << " ";
    }
    cout << endl;
}
cout << endl;
int start;
do {
    cout << "Enter the start vertex: \n";
    cin >> start;
} while (start < 1 || start > 11);
cout << "\nMinimum spanning tree: " << endl;
bool *visited = new bool[n];

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (graph[i][j] == 0)
            graph[i][j] = MAX;
    }
    visited[i] = false;
}
int v1, v2;
int min, weight = 0;
visited[start - 1] = true;
cout << "E = { ";
for (int tek = 1; tek < n; tek++)
{
    min = MAX;
    for (int i = 0; i < n; i++)
```

```
        {
            if (visited[i] == true)
            {
                for (int d = 0; d < n; d++)
                {
                    if (min > graph[i][d] && !visited[d])
                    {
                        v1 = i;
                        min = graph[i][d];
                        v2 = d;
                    }
                }
            }
        }
        weight += min;
        visited[v2] = true;
        if (tek != n - 1)
        {
            cout << "(" << v1 + 1 << ',' << v2 + 1 << "),";
        }
        else if (tek == n - 1)
        {
            cout << "(" << v1 + 1 << ',' << v2 + 1 << ") }";
        }
    }
    cout << "\n\nMinimal weight of tree: " << weight << endl;
    cout << "One more? (y/n)" << endl;
    s = _getch();
} while (s != 'n');

return 0;
```

Результат виконання програми

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

Enter the start vertex:
1

Minimum spanning tree:
E = { (1,2),(1,3),(2,7),(1,4),(4,6),(6,10),(2,5),(5,8),(8,11),(11,9) }

Minimal weight of tree: 26
One more? (y/n)
```

Отже, вага мінімального остового дерева в обох випадках рівний та **рівний 26.**

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

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

Розв'язання:

Проаналізувавши всі можливі шляхи для розв'язання задачі комівояжера, був знайдений найоптимальніший та найраціональніший шлях.

Розпочнемо з вершини 4:

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

	41	2	3	5	6	7	8
41	∞	*1*	*1*	*1*	*1*	*3*	*1*
2	*1*	∞	*5*	*2*	*1*	*3*	*3*
3	*1*	*5*	∞	*5*	*4*	*1*	*5*
5	*1*	*2*	*5*	∞	*1*	*5*	*1*
6	*1*	*1*	*4*	*1*	∞	*5*	*6*
7	*3*	*3*	*1*	*5*	*5*	∞	*1*
8	*1*	*3*	*5*	*1*	*6*	*1*	∞

	412	3	5	6	7	8
412	∞	*5*	*2*	*1*	*3*	*3*
3	*5*	∞	*5*	*4*	*1*	*5*
5	*2*	*5*	∞	*1*	*5*	*1*
6	*1*	*4*	*1*	∞	*5*	*6*
7	*3*	*1*	*5*	*5*	∞	*1*
8	*3*	*5*	*1*	*6*	*1*	∞

	3	5	4126	7	8
3	∞	*5*	*4*	*1*	*5*
5	*5*	∞	*1*	*5*	*1*
4126	*4*	*1*	∞	*5*	*6*
7	*1*	*5*	*5*	∞	*1*
8	*5*	*1*	*6*	*1*	∞

	3	41265	7	8
3	∞	*5*	*1*	*5*
41265	*5*	∞	*5*	*1*
7	*1*	*5*	∞	*1*
8	*5*	*1*	*1*	∞

	3	7	412658
3	∞	*1*	*5*
7	*1*	∞	*1*
~~**412658**~~	~~*5*~~	~~*1*~~	~~∞~~

	3	4126587
3	∞	*1*
7	*1*	∞

Останнім буде ребро (3;4), вага якого рівна 2(ребро потрібне для того, щоб повернутися у початкову вершину та створити цикл). Відповідно, вага даного мінімального шляху рівна 1+1+1+1+1+1+1+2 = **9.**

Програмна реалізація

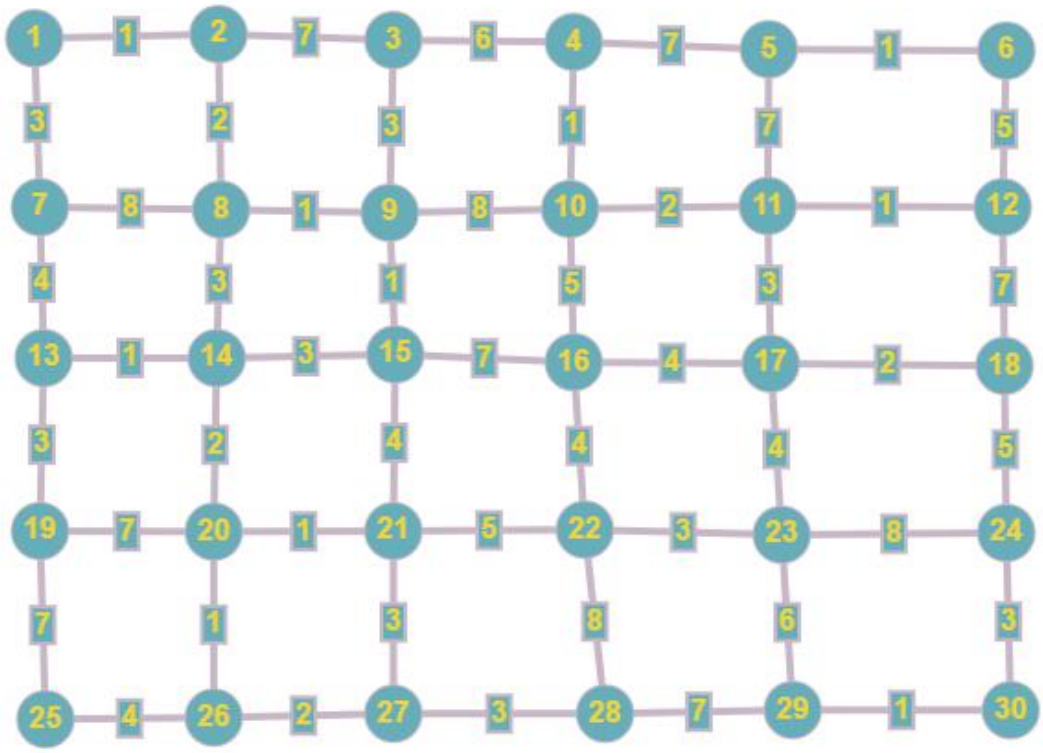
```cpp
#include <iostream>

using namespace std;

const int n = 9;
const int inf = 20000;
bool check(int key, int* mas, int kol);

int main()
{
    int arr[n][n] = {
        {0,  0,  0, 0,  0,  0,  0,  0,  0},
        {0, inf, 1, 1,  1,  1,  1,  3,  1},
        {0, 1,  inf, 5, 1,  2,  1,  3,  3},
        {0, 1,  5, inf, 2,  5,  4,  1,  5},
        {0, 1,  1,  2, inf, 5,  5,  6,  1},
        {0, 1,  2,  5,  5, inf, 1,  5,  1},
        {0, 1,  1,  4,  5,  1, inf, 5,  6},
        {0, 3,  3,  1,  6,  5,  5, inf, 1},
        {0, 1,  3,  5,  1,  1,  6,  1, inf},
    };

    int vertex[n - 1];
    int start;
    char c;
    do {
        system("cls");
        for (int i = 1; i < n; i++)
            vertex[i] = inf;
        do
        {
            cout << "Enter the start vertex: ";
            cin >> start;
        } while (start < 1 || start > n - 1);

        vertex[0] = start;
        int current = start;
        int path = 0;
        cout << "Path:" << endl;
        for (int i = 2; i < n; i++)
        {
            int min = inf;
            int min_t;
            for (int j = 1; j < n; j++)
            {
                if (check(j, vertex, n) && arr[current][j] < min && arr[current][j] > 0)
                {
                    min = arr[current][j];
                    min_t = j;
                }
            }

            path += min;
            vertex[i] = min_t;
            cout << current << " -> " << vertex[i] << " = " << min  << endl;
            current = vertex[i];
        }
        path += arr[start][current];
        cout << current << " -> " << start << " = " << arr[start][current] << endl;
        cout << "Total path: " << path << endl;

        cout << endl << "Press 'n' to stop: ";
        cin >> c;

    } while (c != 'n');

    system("pause");
    return 0;
}

bool check(int l, int* arr, int num) {
    for (int j = 0; j < num; j++)
    {
        if (arr[j] == l)
        {
            return false;
        }
    }
    return true;
}
```

Результат виконання програми

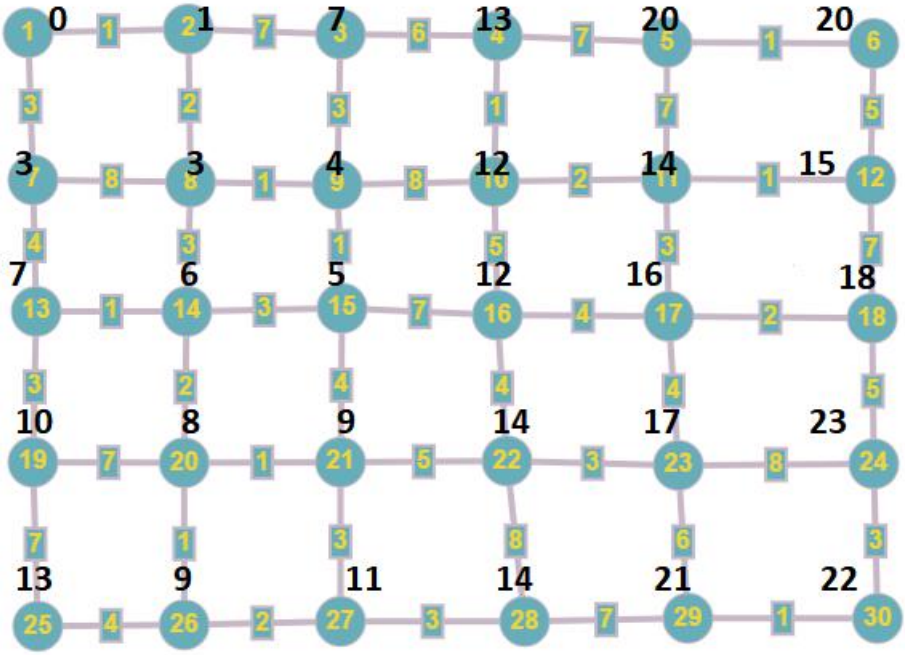
```
Enter the start vertex: 1
Path:
1 -> 2 = 1
2 -> 4 = 1
4 -> 8 = 1
8 -> 5 = 1
5 -> 6 = 1
6 -> 3 = 4
3 -> 7 = 1
7 -> 1 = 3
Total path: 13

Press 'n' to stop: н
Enter the start vertex: 2
Path:
2 -> 1 = 1
1 -> 3 = 1
3 -> 7 = 1
7 -> 8 = 1
8 -> 4 = 1
4 -> 5 = 5
5 -> 6 = 1
6 -> 2 = 1
Total path: 12

Press 'n' to stop: y
Enter the start vertex: 3
Path:
3 -> 1 = 1
1 -> 2 = 1
2 -> 4 = 1
4 -> 8 = 1
8 -> 5 = 1
5 -> 6 = 1
6 -> 7 = 5
7 -> 3 = 1
Total path: 12

Press 'n' to stop: y
Enter the start vertex: 4
Path:
4 -> 1 = 1
1 -> 2 = 1
2 -> 6 = 1
6 -> 5 = 1
5 -> 8 = 1
8 -> 7 = 1
7 -> 3 = 1
3 -> 4 = 2
Total path: 9
```

```
Enter the start vertex: 5
Path:
5 -> 1 = 1
1 -> 2 = 1
2 -> 4 = 1
4 -> 8 = 1
8 -> 7 = 1
7 -> 3 = 1
3 -> 6 = 4
6 -> 5 = 1
Total path: 11

Press 'n' to stop: y
Enter the start vertex: 6
Path:
6 -> 1 = 1
1 -> 2 = 1
2 -> 4 = 1
4 -> 8 = 1
8 -> 5 = 1
5 -> 3 = 5
3 -> 7 = 1
7 -> 6 = 5
Total path: 16

Press 'n' to stop: y
Enter the start vertex: 7
Path:
7 -> 3 = 1
3 -> 1 = 1
1 -> 2 = 1
2 -> 4 = 1
4 -> 8 = 1
8 -> 5 = 1
5 -> 6 = 1
6 -> 7 = 5
Total path: 12

Press 'n' to stop: y
Enter the start vertex: 8
Path:
8 -> 1 = 1
1 -> 2 = 1
2 -> 4 = 1
4 -> 3 = 2
3 -> 7 = 1
7 -> 5 = 5
5 -> 6 = 1
6 -> 8 = 6
Total path: 18
```

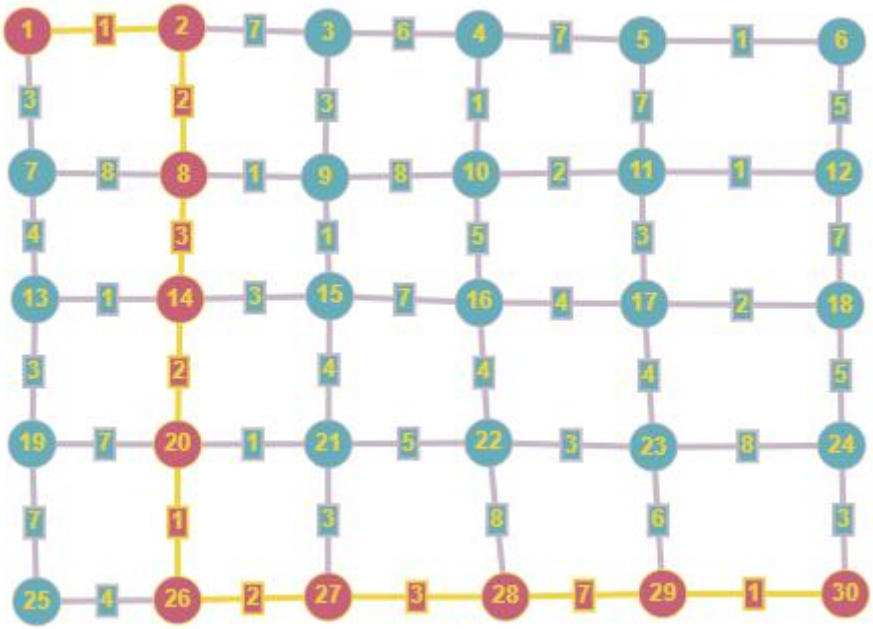
7. За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V_0 (вершина 1) та V(вершина 30) .*



Розв’язання: знайдемо шлях від початкової точки V_0 до кожної з вершин графа:



Тепер покажемо відстань від вершини 1 до 30:



Отже, відстань від V_0 до V* рівна **22**(1+2+3+2+1+2+3+7+1).

Програмна реалізація

```
#include <iostream>
#include <conio.h>
#include <windows.h>
#include <iomanip>

using namespace std;

const int INF = 1000;
const int VERTEX = 30;

int main()
{
    setlocale(LC_ALL, "rus");
    bool x[30];
    int path[30];
    int previous[30];
    int p, v;
    p = VERTEX;
    int graph[30][30] = {
    {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, 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, 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, 1, 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, 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, 4, 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, 1, 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, 3, 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, 4, 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, 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, 1, 0, 0, 0, 0, 3, 0, 7, 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, 7, 0, 4, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 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},
```

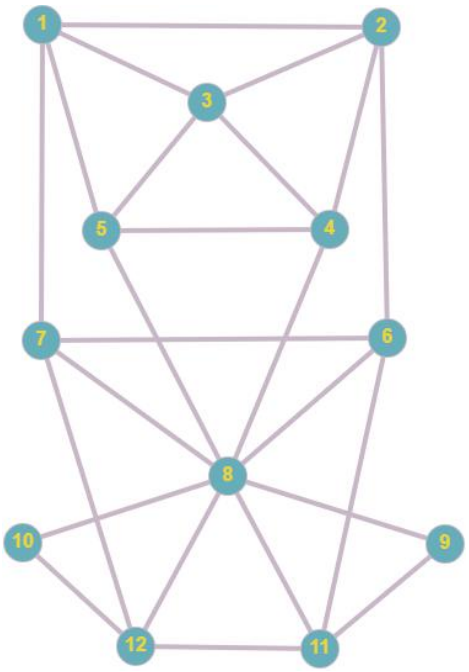
```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 7, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 1, 0, 5, 0, 0, 0, 0, 3, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 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, 6, 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, 3},
{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},
{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, 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, 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, 6, 0, 0, 0, 0, 7, 0, 1},
{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, 1, 0},
};

int start;
int end;
cout << "Enter the start vertex: ";
cin >> start;
if (start > (p - 1) && start < 0)
    cout << "Please, enter the correct vertex!" << endl;
start = start - 1;

for (int i = 0; i < VERTEX; i++)
{
    end = i;
    if (end == start)
        continue;
    else
    {
        int u;
        for (u = 0; u < p; u++)
        {
            path[u] = INF;
            x[u] = 0;
        }
        previous[start] = 0;
        path[start] = 0;
```

```
    x[start] = 1;
    v = start;
    while (1)
    {
        for (u = 0; u < p; u++)
        {
            if (graph[v][u] == 0)
                continue;
            if (x[u] == 0 && path[u] > path[v] + graph[v][u])
            {
                path[u] = path[v] + graph[v][u];

                previous[u] = v;

            }
        }
        int w = INF;
        v = -1;
        for (u = 0; u < p; u++)
        {
            if (x[u] == 0 && path[u] < w)
            {
                v = u;
                w = path[u];
            }
        }
        if (v == end)
        {
            cout << start + 1 << " --> " << end + 1 << "| Path :";
            u = end;
            while (u != start)
            {
                cout << " " << u + 1;
                u = previous[u];
            }
            cout << " " << start + 1 << "| Length = " << path[end] << endl;
            break;
        }
        x[v] = 1;
    }
}
    cout << endl << "V0" << "-->" << "V* = " << path[end];
    return 0;
}
```

Результат виконання програми

```
Enter the start vertex: 1
1 --> 2| Path : 2 1| Length = 1
1 --> 3| Path : 3 9 8 2 1| Length = 7
1 --> 4| Path : 4 3 9 8 2 1| Length = 13
1 --> 5| Path : 5 4 3 9 8 2 1| Length = 20
1 --> 6| Path : 6 12 11 10 9 8 2 1| Length = 20
1 --> 7| Path : 7 1| Length = 3
1 --> 8| Path : 8 2 1| Length = 3
1 --> 9| Path : 9 8 2 1| Length = 4
1 --> 10| Path : 10 9 8 2 1| Length = 12
1 --> 11| Path : 11 10 9 8 2 1| Length = 14
1 --> 12| Path : 12 11 10 9 8 2 1| Length = 15
1 --> 13| Path : 13 7 1| Length = 7
1 --> 14| Path : 14 8 2 1| Length = 6
1 --> 15| Path : 15 9 8 2 1| Length = 5
1 --> 16| Path : 16 15 9 8 2 1| Length = 12
1 --> 17| Path : 17 16 15 9 8 2 1| Length = 16
1 --> 18| Path : 18 17 16 15 9 8 2 1| Length = 18
1 --> 19| Path : 19 13 7 1| Length = 10
1 --> 20| Path : 20 14 8 2 1| Length = 8
1 --> 21| Path : 21 15 9 8 2 1| Length = 9
1 --> 22| Path : 22 21 15 9 8 2 1| Length = 14
1 --> 23| Path : 23 22 21 15 9 8 2 1| Length = 17
1 --> 24| Path : 24 18 17 16 15 9 8 2 1| Length = 23
1 --> 25| Path : 25 26 20 14 8 2 1| Length = 13
1 --> 26| Path : 26 20 14 8 2 1| Length = 9
1 --> 27| Path : 27 26 20 14 8 2 1| Length = 11
1 --> 28| Path : 28 27 26 20 14 8 2 1| Length = 14
1 --> 29| Path : 29 28 27 26 20 14 8 2 1| Length = 21
1 --> 30| Path : 30 29 28 27 26 20 14 8 2 1| Length = 22

V0-->V* = 22
```

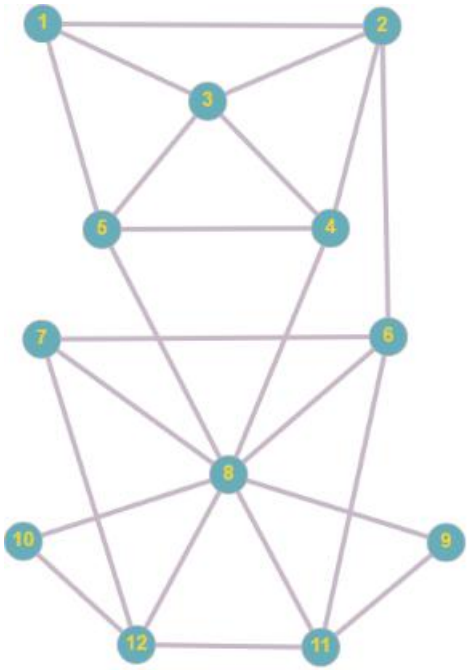
8. Знайти ейлеровий цикл в ейлеровому графі двома методами:
а) Флері;
б) елементарних циклів.



Розв’язання:

а) Алгоритм Флері полягає в тому, що ми повинні пройти та видалити кожне ребро графа лише один раз, і при цьому перевіряти, чи є ребро графа мостом (чи не спричине видалення графа розбиття на дві зв’язні компоненти).

Почнемо з вершини V1 та ребра V1 → V7:



Після чого послідовно видаляємо ребра та перевіряємо, чи є вони мостами:
V7 → V12 → V10 → V8 → V12 → V11→ V9 → V8 → V11 → V6 → V8 → V5 → V4 → V8 → V7 → V6 → V2 → V4 → V3 → V5 → V1 → V3 → V2 → V1.
Оскільки ми проходимо ребра лише один раз та повертаємося в початкову вершину, то знайдений цикл є ейлоровим.

б) Скористаємося тим, що ейлеровий цикл - це об’єднання всіх простих циклів. Для цього виділимо прості цикли у даному графі:

1) V1→ V3 → V5 →V1(виберемо як початковий);
2) V2 → V3 → V4 → V2;
3) V1→ V2 → V6 → V7 → V1;
4) V4 → V5 → V8 → V4;
5) V7 → V8 → V12 → V7;
6) V6 → V8 → V11 → V6;
7) V8 → V10 → V12 → V11 → V9 → V8.

Спочатку об’єднаємо початковий цикл з циклом №3:
V1→ V3 → V5 → V1→ V2 → V6 → V7 → V1.
Після цього додамо цикл №2:

V1→ V3 → V5 → V1→ V2 → V3 → V4 → V2 → V6 → V7 → V1.
Послідовно додаємо решту циклів:
V1→ V3 → V5 → V1→ V2 → V3 → V4 → V5 → V8 → V4 → V2 → V6 → V7 → V1.
V1→ V3 → V5 → V1→ V2 → V3 → V4 → V5 → V8 → V4 → V2 → V6 → V7 → V8 → V12 → V7 → V1.

V1→ V3 → V5 → V1→ V2 → V3 → V4 → V5 → V8 → V4 → V2 → V6 → V8 → V11 → V6 → V7 → V8 → V12 → V7 → V1.

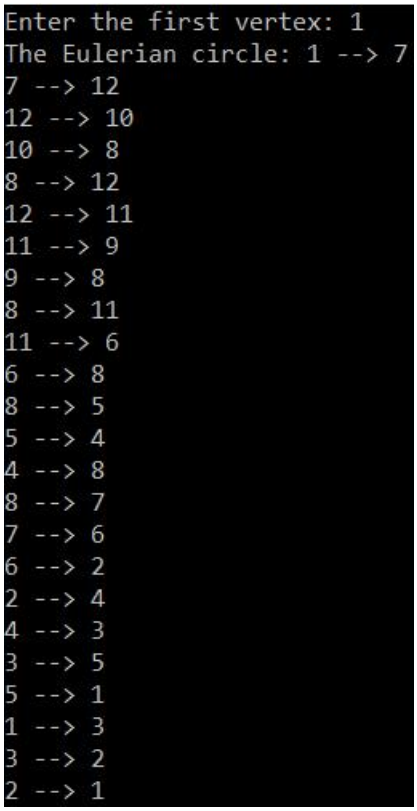
V1→ V3 → V5 → V1→ V2 → V3 → V4 → V5 → V8 → V10 → V12 → V11 → V9 → V8 → V4 → V2 → V6 → V8 → V11 → V6 → V7 → V8 → V12 → V7 → V1.

Отже, ми пройшли усі ребра графа по одному разу та повернулися у початкову вершину V1, тому цикл є правильним.

Програмна реалізація

```
#include <iostream>

using namespace std;

const int n = 13;
const int s = 100;

int k;
int path[s];

void Fleury(int);

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

int main()
{
    int s = 0;
    int vertex;
    bool T = true;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            s += graph[i][j];
        }
        if (s % 2)
            T = false;
    }
    k = -1;

    cout << "Enter the first vertex: ";
    cin >> vertex;
    cout << "The Eulerian circle: ";
    if (T)
    {
        Fleury(vertex);
        for (int j = 0; j < k; j++)
        {
            cout << path[j] << " --> " << path[j+1] << endl;
        }
    }
    else
        cout << "There is no Eulerian circle!!!" << endl;
    return 0;
}

void Fleury(int v)
{
    int i;
    for (i = 0; i < n; i++)
        if (graph[v][i])
        {
            graph[v][i] = graph[i][v] = 0;
            Fleury(i);
        }
    path[++k] = v;
}
```

Результат виконання програми



9. Спростити формулу (привести їх до скороченої ДНФ)

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

Розв'язання:

Використаємо метод Блейка-Порецького:

$$\bar{x}yz \vee xy = \bar{x}yz \vee xy \vee xz$$

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

$$\bar{x}\bar{z} \vee xy = \bar{x}\bar{z} \vee xy \vee y\bar{z}$$

$$\bar{x}\bar{z} \vee xy \vee y\bar{z} \vee \bar{x}yz \vee x\bar{z} \vee \bar{x}yz \vee xy \vee xz = y\bar{z} \vee x\bar{z} \vee \bar{x}yz \vee xy \vee xz$$

Відповідь: $y\bar{z} \vee x\bar{z} \vee \bar{x}yz \vee xy \vee xz$.