МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

“ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахунково-графічна робота

з дисципліни

«Дискретна математика»

Виконав:

студент групи КН-112

Калітовський Роман

Викладач:

Мельникова Н.І.

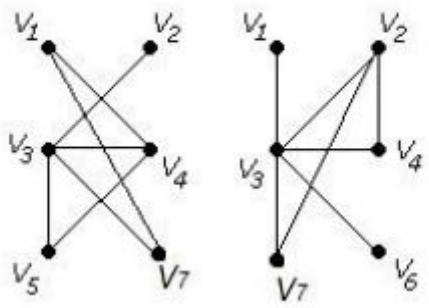
Львів – 2019 р.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

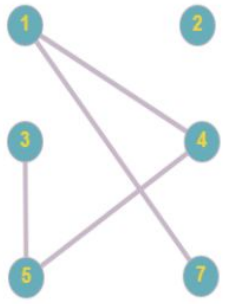
Завдання № 1

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.

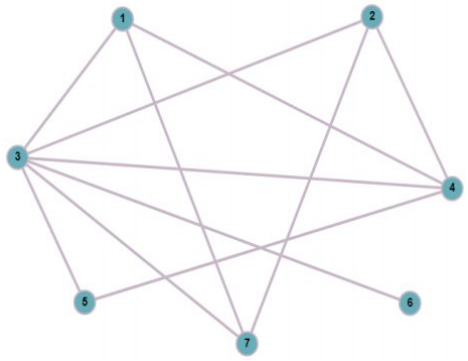
12)



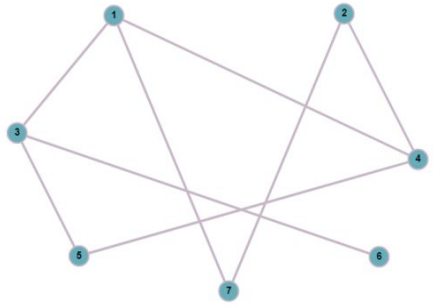
1)Доповнення
G1\G2:



2)об’єднання графів:

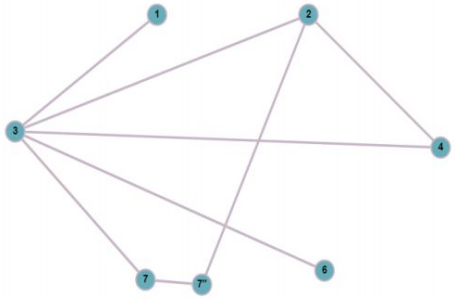


3)Кільцева сума:



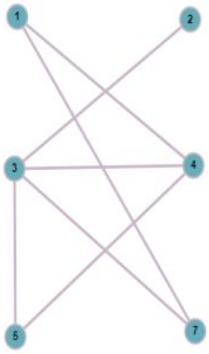
4)Розщепити вершину у другому графі:

Розщепимо вершину 7

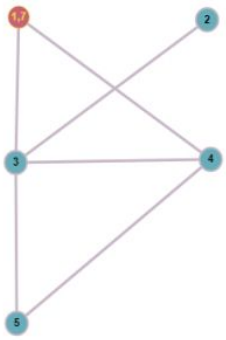


5)Виділити підграф A,що складається з V={1,3,7} в G1 і знайти стягнення A в G1

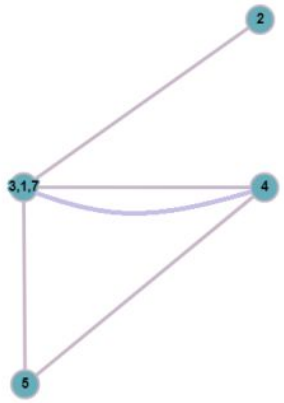
G1



Стягуємо 7 в 1

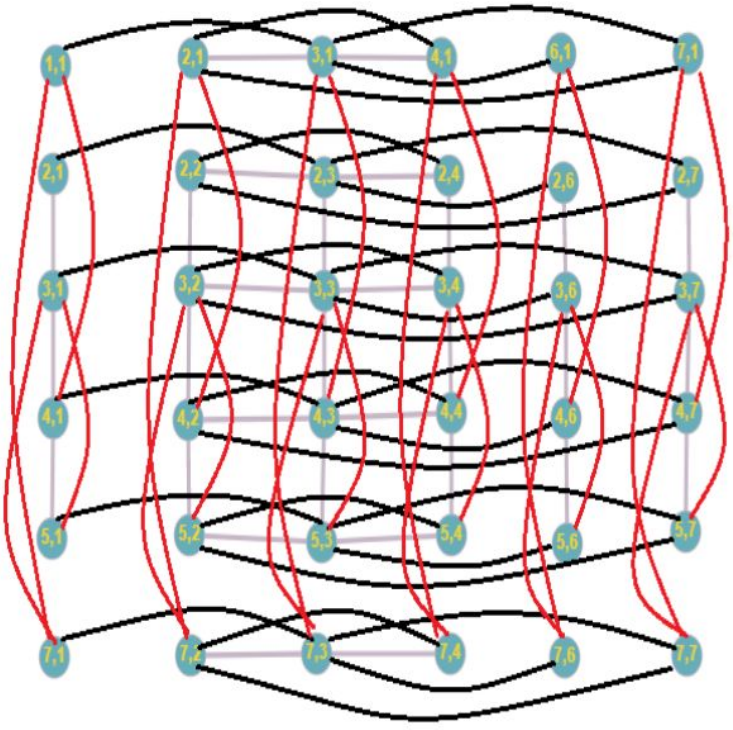


Стягуємо 1,7 в 3



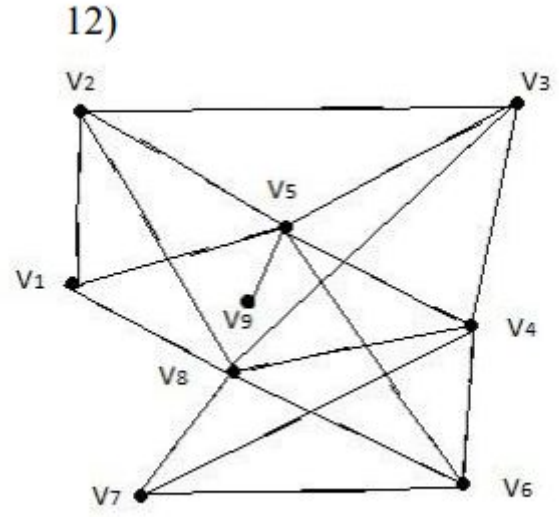
Стягуємо 3,7,1 в 4

6)Добуток графів



Завдання № 2

Скласти таблицю суміжності для графа.



Таблиця суміжності:

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

Завдання № 3

Для графа з другого завдання знайти діаметр.

Найдовший шлях від V7 до V9 і V8 до V9

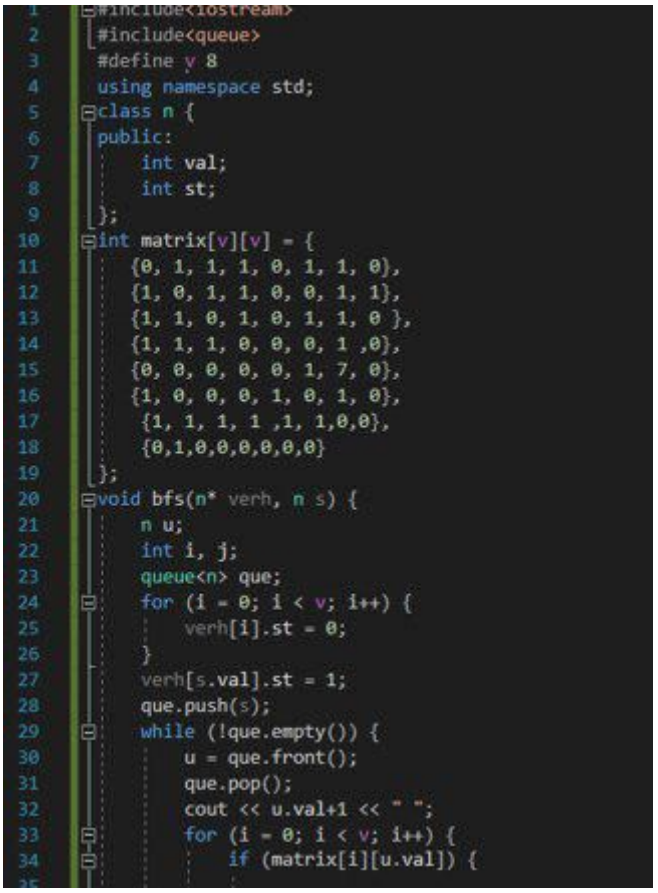
Діаметр = 3

Завдання № 4

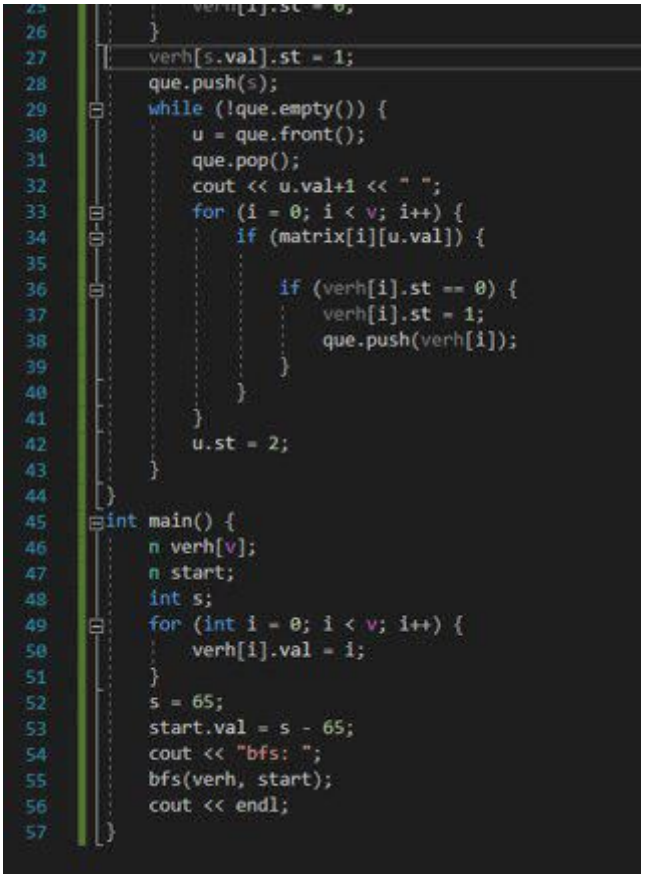
Для графа з другого завдання виконати обхід дерева вглиб (варіант закінчується на непарне число) або вшир (закінчується на парне число).

Обхід вшир:

Номер	Номер вершини	Черга
0	-	-
1	1	1
2	2	12
3	5	125
4	8	1258
5	-	258
6	3	2583
7	-	583
8	4	5834
9	6	58346
10	9	583469
11	-	83469
12	7	834697
13	-	34697
14	-	4697
15	-	697
16	-	97
17	-	7
18	-	-



```cpp
#include<iostream>
#include<queue>
#define v 8
using namespace std;
class n {
public:
    int val;
    int st;
};
int matrix[v][v] = {
   {0, 1, 1, 1, 0, 1, 1, 0},
   {1, 0, 1, 1, 0, 0, 1, 1},
   {1, 1, 0, 1, 0, 1, 1, 0 },
   {1, 1, 1, 0, 0, 0, 1 ,0},
   {0, 0, 0, 0, 0, 1, 7, 0},
   {1, 0, 0, 0, 1, 0, 1, 0},
    {1, 1, 1, 1 ,1, 1,0,0},
    {0,1,0,0,0,0,0,0}
};
void bfs(n* verh, n s) {
    n u;
    int i, j;
    queue<n> que;
    for (i = 0; i < v; i++) {
        verh[i].st = 0;
    }
    verh[s.val].st = 1;
    que.push(s);
    while (!que.empty()) {
        u = que.front();
        que.pop();
        cout << u.val+1 << " ";
        for (i = 0; i < v; i++) {
            if (matrix[i][u.val]) {
```



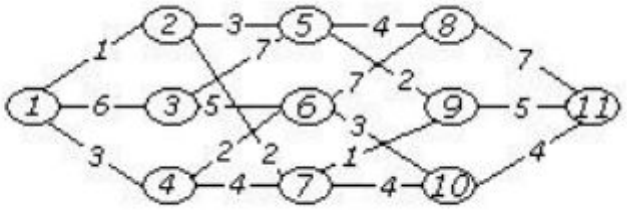
```cpp
    }
    verh[s.val].st = 1;
    que.push(s);
    while (!que.empty()) {
        u = que.front();
        que.pop();
        cout << u.val+1 << " ";
        for (i = 0; i < v; i++) {
            if (matrix[i][u.val]) {

                if (verh[i].st == 0) {
                    verh[i].st = 1;
                    que.push(verh[i]);
                }
            }
        }
        u.st = 2;
    }
}
int main() {
    n verh[v];
    n start;
    int s;
    for (int i = 0; i < v; i++) {
        verh[i].val = i;
    }
    s = 65;
    start.val = s - 65;
    cout << "bfs: ";
    bfs(verh, start);
    cout << endl;
}
```

```
btf
0       -       -
1       1       1
2       2       12
3       5       125
4       8       1258
5       -       258
6       3       2583
7       -       583
8       4       5834
9       6       58346
10      9       583469
11      -       83469
12      7       834697
13      -       34697
14      -       4697
15      -       697
16      -       97
17      -       7
18      -       -
```

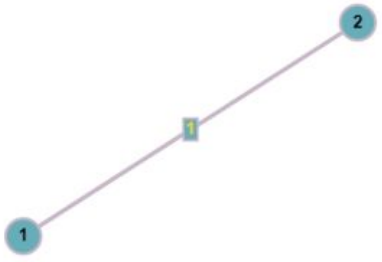
Завдання № 5

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

12)

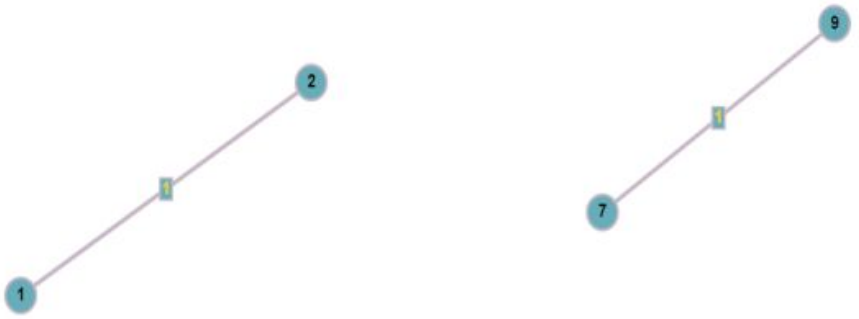


Метод Краскала V={1,2} E={(1,2)}

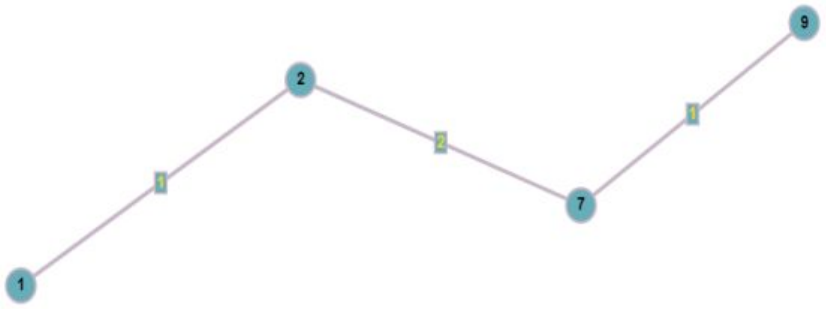


V={1,2,7,9}

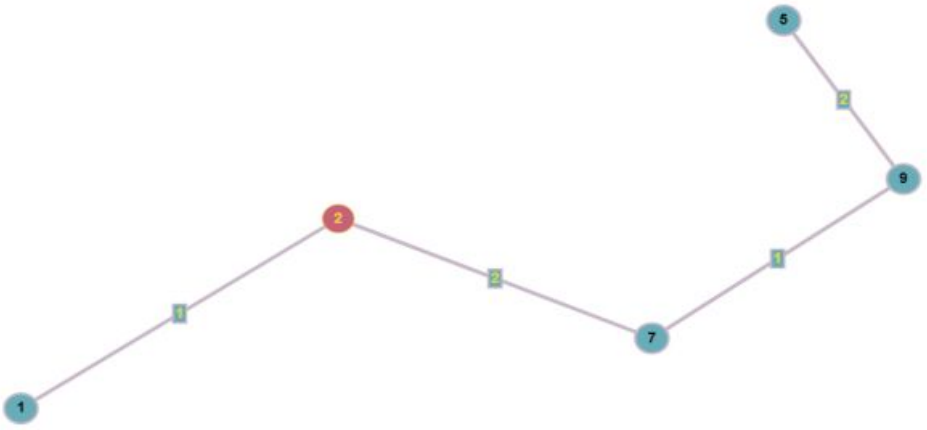
E={(1,2),(7,9)}



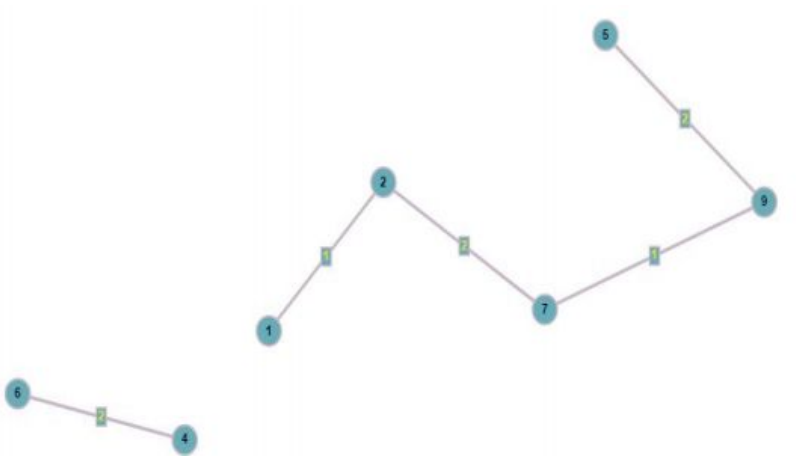
V={1,2,7,9}
E={(1,2),(7,9),(2,7)}



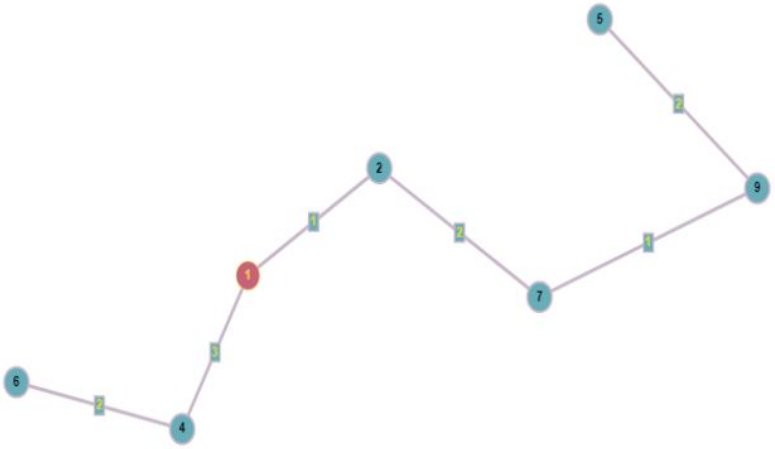
V={1,2,7,9,5} E={(1,2),(7,9),(2,7),(5,9)}



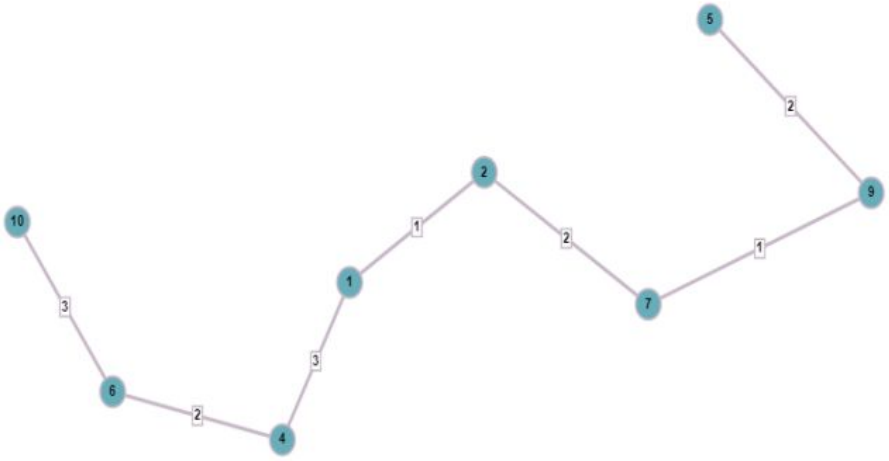
V={1,2,7,9,5,4,6} E={(1,2),(7,9),(2,7),(5,9),(4,6)}



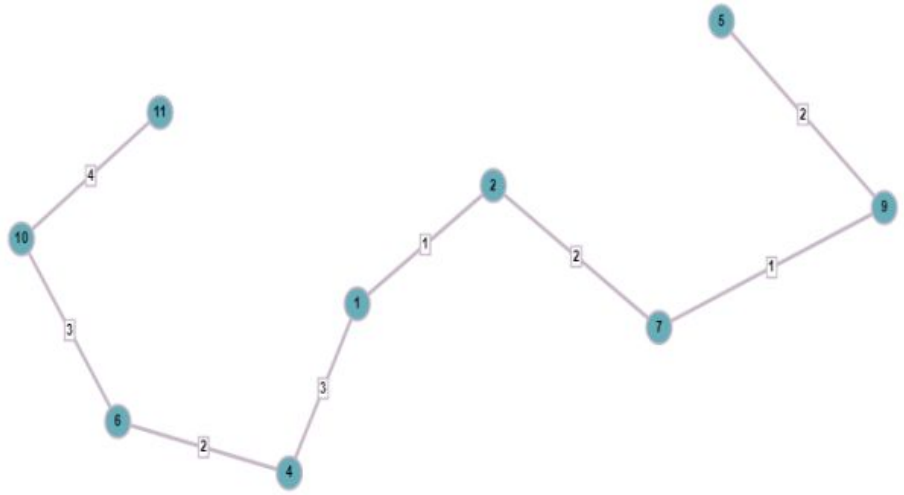
V={1,2,7,9,5,4,6} E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4)}



V={1,2,7,9,5,4,6,10} E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4)(6,10)}



V={1,2,7,9,5,4,6,10,11} E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4),(6,10),(10,11)}

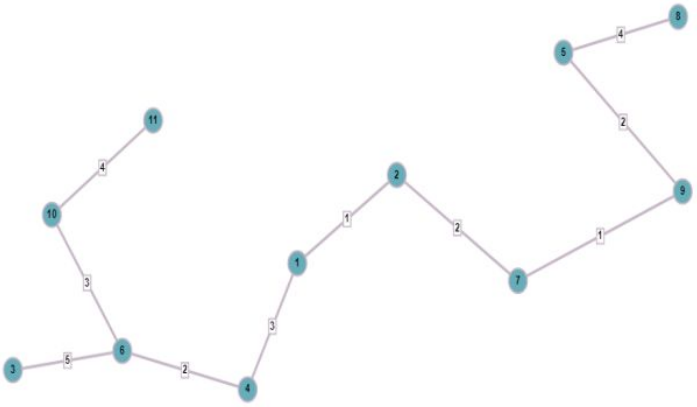


V={1,2,7,9,5,4,6,10,11,8}

E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4),(6,10),(10,11),(5,8)}

V={1,2,7,9,5,4,6,10,11,8,3}

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



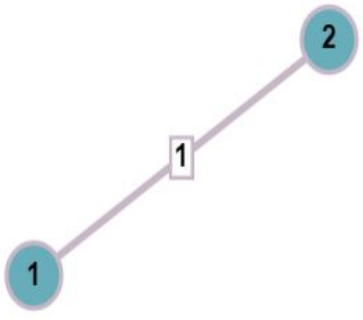
Weight=27

```cpp
#include <iostream>
#define INT_MAX 2147483647

using namespace std;

#define V 11
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 Kruskal(int cost[][V]) {
    cout << "The Kruskal Method" << endl;
    int min_cost = 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;
        if (min == INT_MAX) {
            cout << "There is no minimum spanning tree." << endl;
            exit(0);
        }

        union1(a, b);
        printf("Edge %d:(%d~%d) \t cost:%d \n",
               edge_count++ + 1, a + 1, b + 1, min);
        min_cost += min;
    }
    printf("\n Minimum cost= %d \n", min_cost);
}

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

    Kruskal(cost);
    return 0;
```

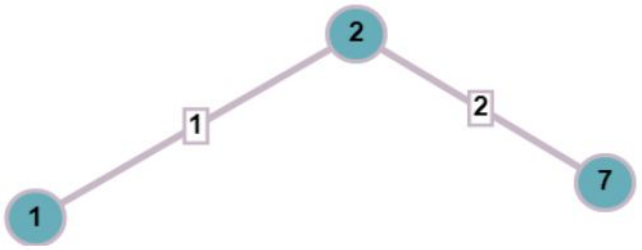
```
V={1,2,7,9,5,4,6,10,11,8,3}
E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4),(6,10),(10,11),(5,8),(6,3)}
```

Метод Прима:

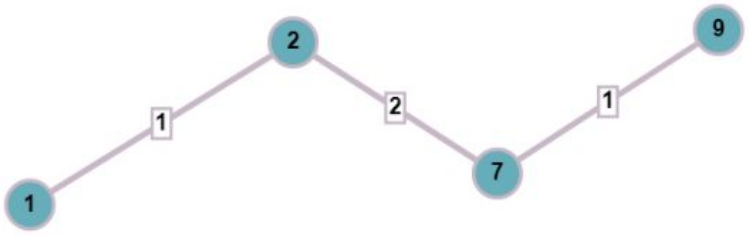
V={1,2} E={(1,2)}



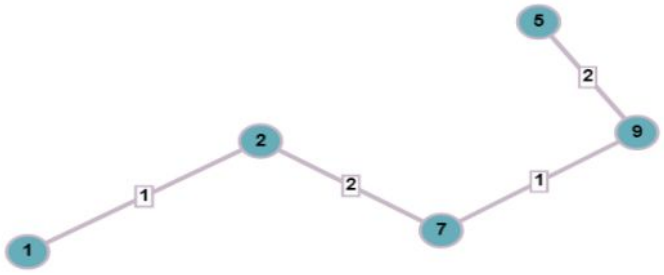
V={1,2,7} E={(1,2),(2,7)}



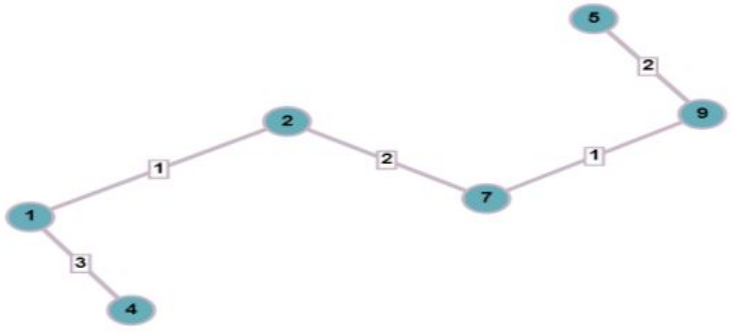
V={1,2,7,9} E={(1,2),(2,7),(7,9)}



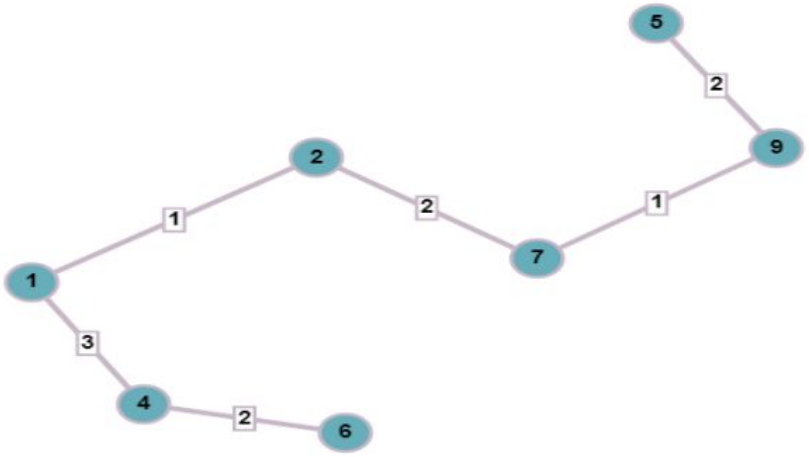
V={1,2,7,9,5} E={(1,2),(2,7),(7,9),(9,5)}



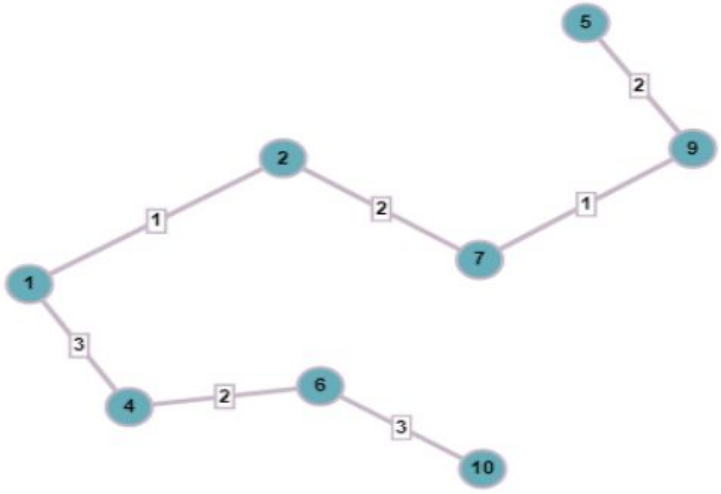
V={1,2,7,9,5,4} E={(1,2),(2,7),(7,9),(9,5),(1,4)}



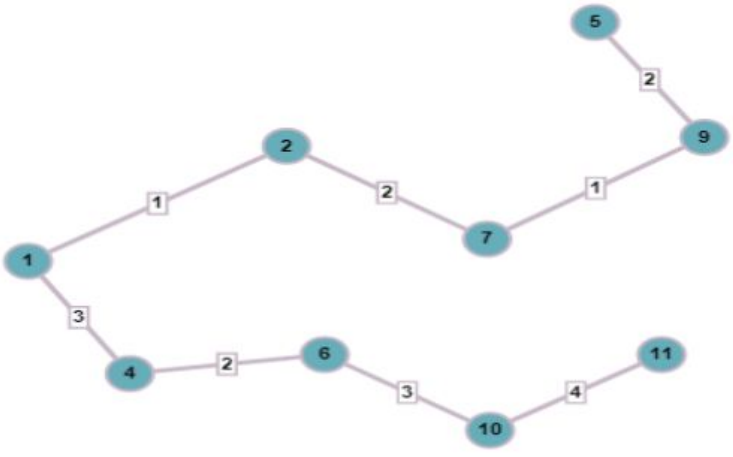
V={1,2,7,9,5,4,6} E={(1,2),(2,7),(7,9),(9,5),(1,4),(4,6)}



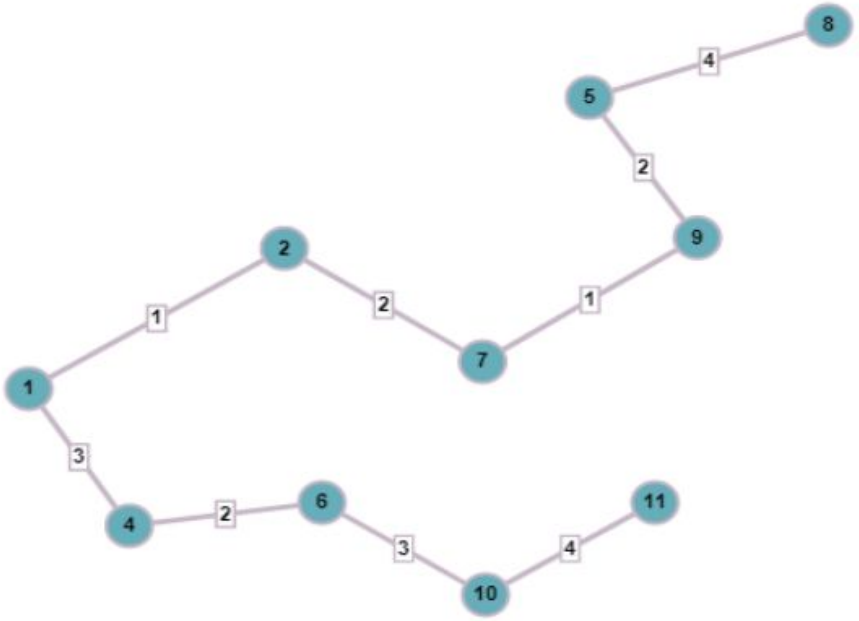
V={1,2,7,9,5,4,6,10} E={(1,2),(2,7),(7,9),(9,5),(1,4),(4,6),(6,10)}



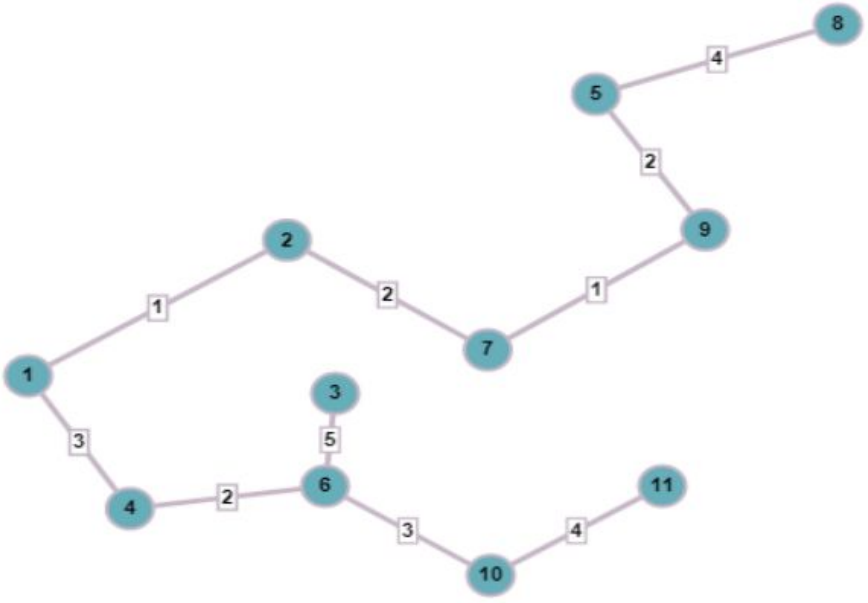
V={1,2,7,9,5,4,6,10,11} E={(1,2),(2,7),(7,9),(9,5),(1,4),(4,6),(6,10),(10,11)}



V={1,2,7,9,5,4,6,10,11,8}
E={(1,2),(2,7),(7,9),(9,5),(1,4),(4,6),(6,10),(10,11),(5,8)}



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



Weight=27.

```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "Source.h"

using namespace std;

int main(void)
{
    int versh, cnt = 0, min_ = 0, n, m;
    bool c = false;

    cout << "Vershini : "; cin >> versh; cout << "\n";

    int** graph = new int* [versh];
    for (int i = 0; i < versh; ++i)
        graph[i] = new int[versh];

    int** rebr = new int* [versh - 1];
    for (int i = 0; i < versh - 1; ++i)
        rebr[i] = new int[2];

    for (int i = 0; i < versh; ++i)
        for (int j = 0; j < versh; ++j)
            cin >> graph[i][j];

    int* tops = new int[versh];
    tops[cnt] = 1;
    ++cnt;

    for (int i = 0; cnt < versh; ++i) {
        for (int j = 0; j < cnt; ++j) {
            for (int x = 0; x < versh; ++x) {
                for (int y = 0; y < cnt; ++y)
                    if (tops[y] == x + 1)
                        c = true;
                if (c == true)
                    if (tops[y] == x + 1)
                        c = true;
                if (c == true)
                {
                    c = false;
                    continue;
                }

                if (min_ == 0 && graph[tops[j] - 1][x] > 0)
                {
                    min_ = graph[tops[j] - 1][x];
                    n = rebr[cnt - 1][0] = tops[j];
                    m = rebr[cnt - 1][1] = x + 1;
                    continue;
                }

                if (graph[tops[j] - 1][x] > 0 && graph[tops[j] - 1][x] < min_)
                {
                    min_ = graph[tops[j] - 1][x];
                    n = rebr[cnt - 1][0] = tops[j];
                    m = rebr[cnt - 1][1] = x + 1;
                }
            }
        }

        graph[n - 1][m - 1] = 0;
        graph[m - 1][n - 1] = 0;
        tops[cnt] = m;
        ++cnt;
        min_ = 0;
    }

    cout << endl << "Rebra: ";
    for (int i = 0; i < versh - 1; ++i)
        cout << "(" << rebr[i][0] << ", " << rebr[i][1] << ") ";
}
```

```
V={1,2,7,9,5,4,6,10,11,8,3}
E={(1,2),(7,9),(2,7),(5,9),(4,6),(1,4),(6,10),(10,11),(5,8),(6,3)}
```

Завдання № 6

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

12)

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

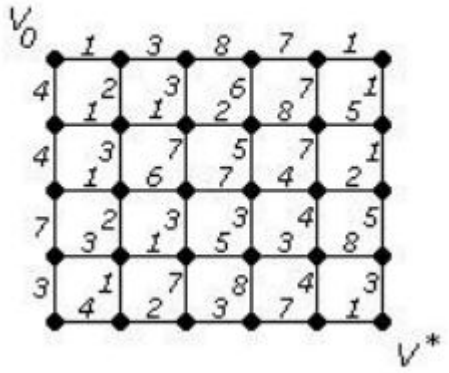
Почнемо з 1-ої вершини:
1
Найближча до 1-ої вершина - 5
1->5
Довжина шляху: 4
найближча до 5-ої вершини - 2
1->5->2
Довжина шляху: 4+1
найближча до 2-ої вершини - 3
1->5->2->3
Довжина шляху: 4+1+1
найближча до 3-ої вершини - 4
1->5->2->3->4
Довжина шляху: 4+1+1+1
найближча до 4-ої вершини - 6
1->5->2->3->4->6
Довжина шляху: 4+1+1+1+5
найближча до 6-ої вершини - 7
1->5->2->3->4->6->7
Довжина шляху: 4+1+1+1+5+5
найближча до 7-ої вершини - 8
1->5->2->3->4->6->7->8
Довжина шляху: 4+1+1+1+5+5+1
Всі вершини пройдені, повертаємось у початкову
1->5->2->3->4->6->7->8->1

Довжина шляху: 4+1+1+1+5+5+1+5=23.

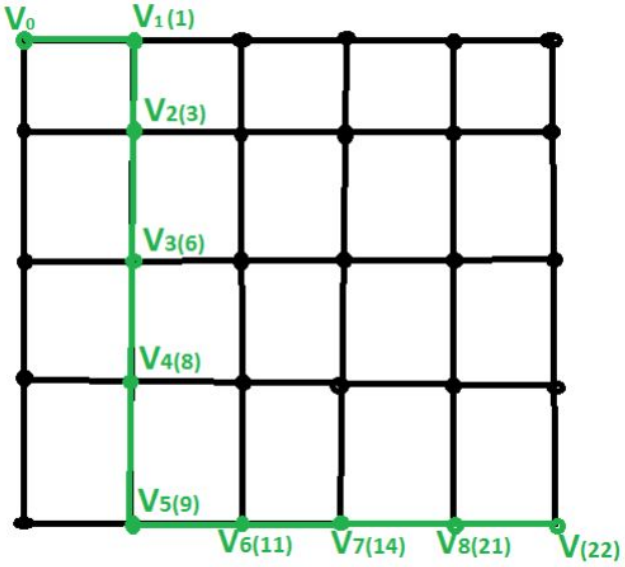
Завдання № 7

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

12)



Найкоротший шлях:



```
#include <iostream>
#define SIZE 30
using namespace std;
int main()
{
  int a[SIZE][SIZE] =
  //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
  {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, 0, 0, 0, 0, 0,
   8, 0, 5, 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, 5, 0, 3, 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, 3, 0, 8, 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, 8, 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, 0,
   0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 6, 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, 7, 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, 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, 1, 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, 1, 0, 2, 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, 2, 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, 6, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 4, 0, 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, 5, 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, 5, 0, 2, 0, 0, 0, 0, 3, 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, 7, 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, 7, 0, 7, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 7, 0, 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, 3, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 8, 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, 3, 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, 3, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 5, 0, 0, 0, 0, 5, 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, 0, 0, 0, 2, 0, 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, 2, 0, 0, 0, 0, 1, 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, 1, 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, 3, 0, 0, 0, 6, 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, 8, 0, 0, 0, 3, 0,
```

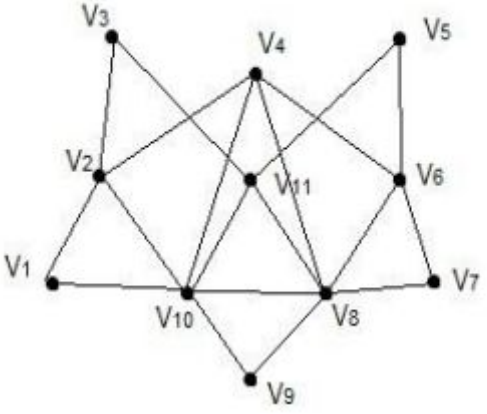
```
  };

  int d[SIZE];
  int v[SIZE];
  int temp, minindex, min;
  int begin_index = 0;

  for (int i = 0; i<SIZE; i++)
  {
    d[i] = 10000;
    v[i] = 1;
  }
  d[begin_index] = 0;

  do {
    minindex = 10000;
    min = 10000;
    for (int i = 0; i<SIZE; i++)
    {
      if ((v[i] == 1) && (d[i]<min))
      {
        min = d[i];
        minindex = i;
      }
    }

    if (minindex != 10000)
    {
      for (int i = 0; i<SIZE; i++)
      {
        if (a[minindex][i] > 0)
        {
          temp = min + a[minindex][i];
          if (temp < d[i])
          {
            d[i] = temp;
          }
        }
      }
      v[minindex] = 0;
    }
  } while (minindex < 10000);


  cout<<"Вивід найкоротшого шляху: "<<endl;
  for (int i = k - 1; i >= 0; i--)
    cout<<ver[i];
  getchar(); getchar();
  return 0;
}
```

```
1->2->8->14->20->26->27->28->29->30
```

Завдання № 8

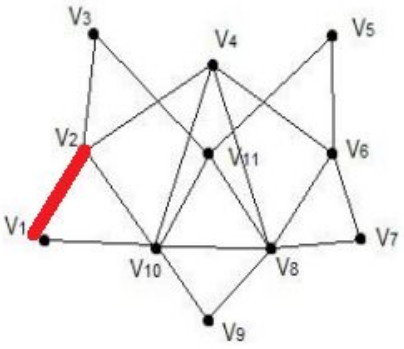
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.

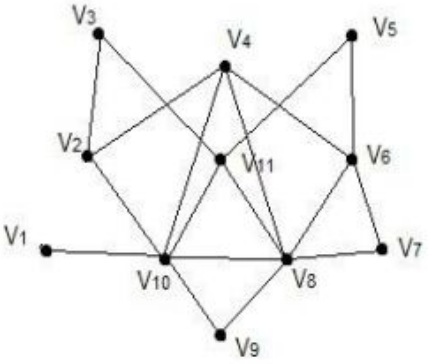
12)



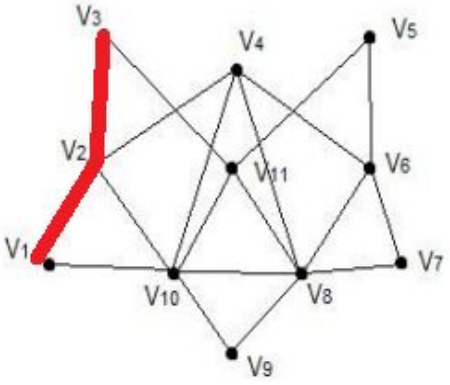
Метод флері:

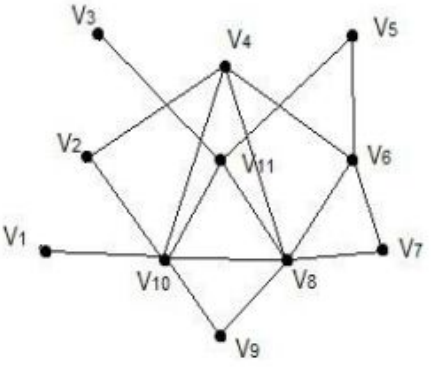
1



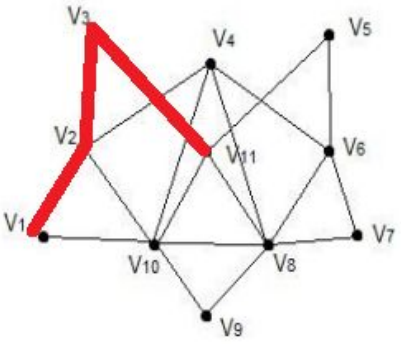


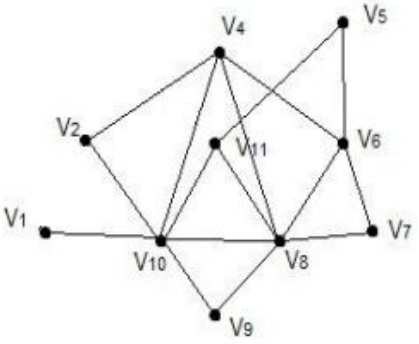
2



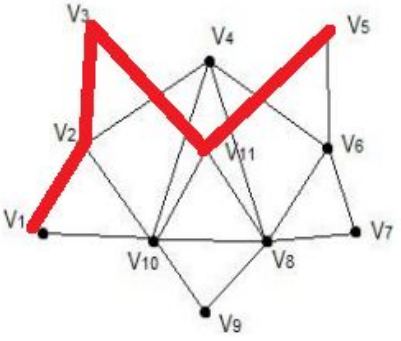


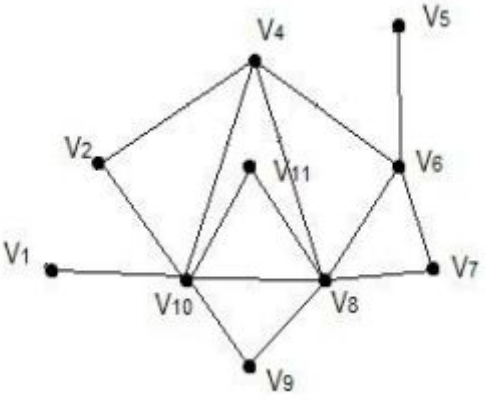
3


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

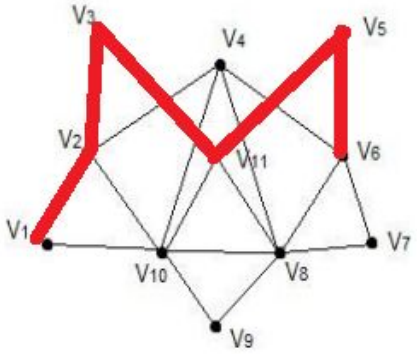

V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

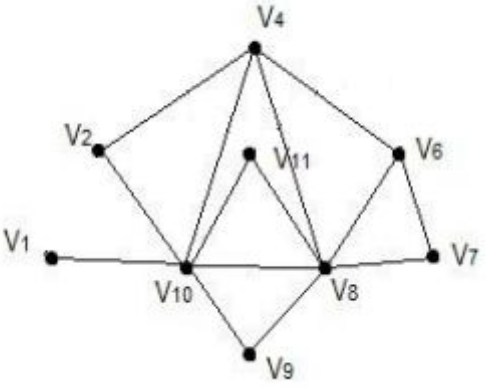
4


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

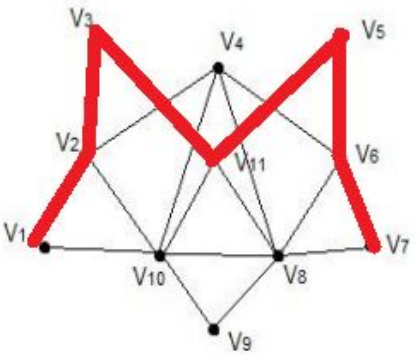

V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

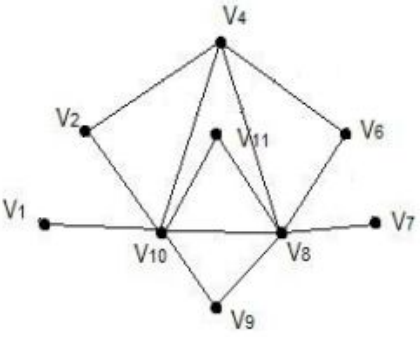
5


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

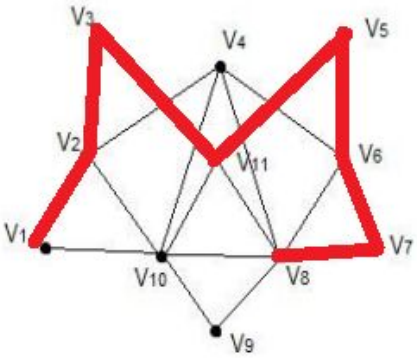

V4
V2
V11
V6
V1
V7
V10
V8
V9

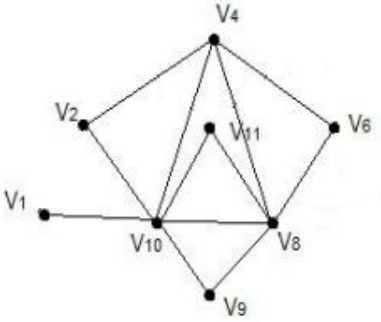
6


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

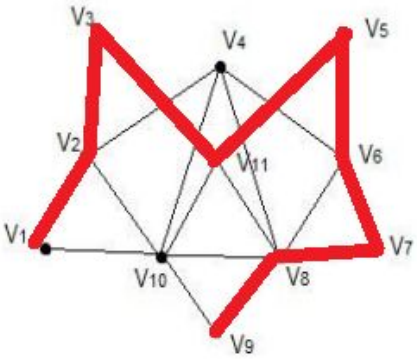

V4
V2
V11
V6
V1
V7
V10
V8
V9

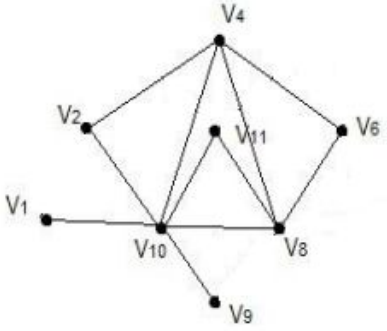
7


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

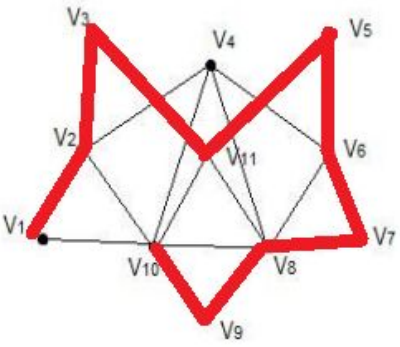

V4
V2
V11
V6
V1
V10
V8
V9

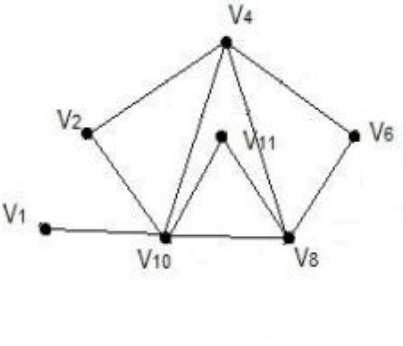
8


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

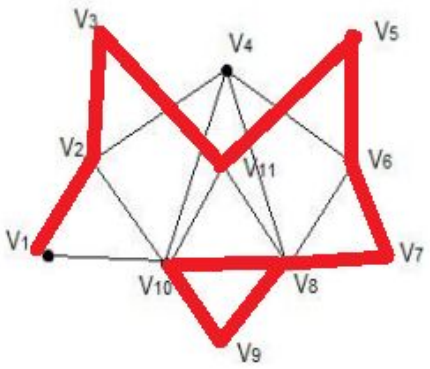

V4
V2
V11
V6
V1
V10
V8
V9

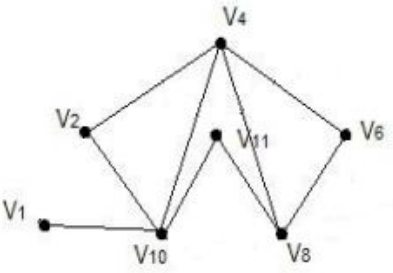
9


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

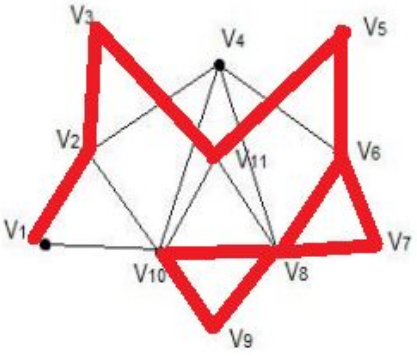

V4
V2
V11
V6
V1
V10
V8

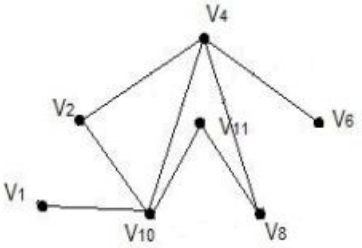
10


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

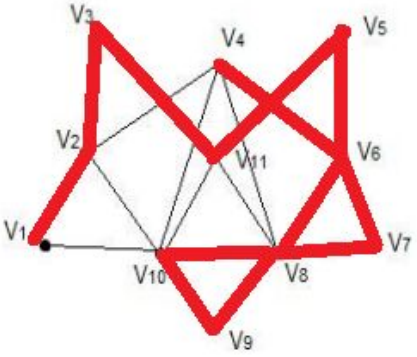

V4
V2
V11
V6
V1
V10
V8

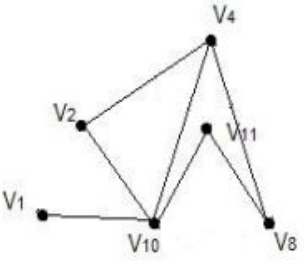
11


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

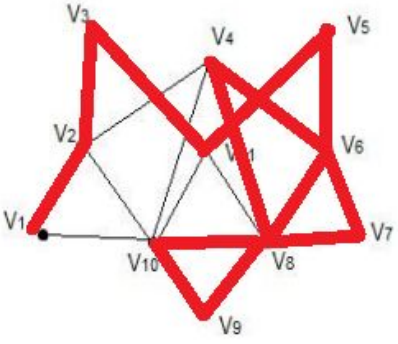

V4
V2
V11
V6
V1
V10
V8

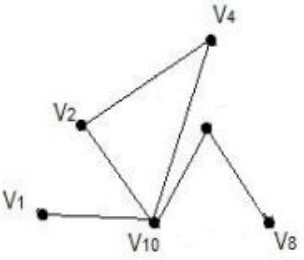
12


V3
V4
V5
V2
V11
V6
V1
V7
V10
V8
V9

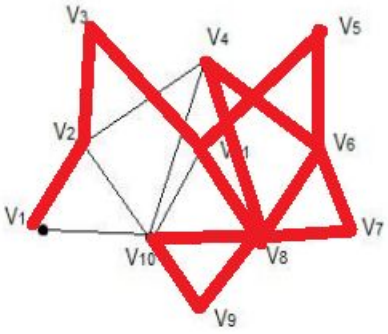

V4
V2
V11
V1
V10
V8

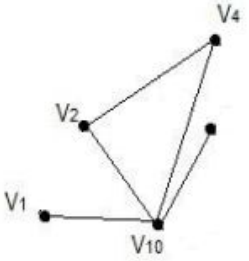
13


V3
V4
V5
V2
V6
V1
V7
V10
V8
V9

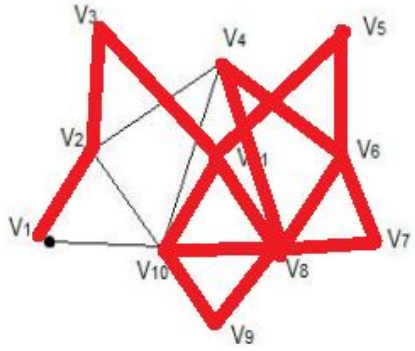

V4
V2
V1
V10
V8

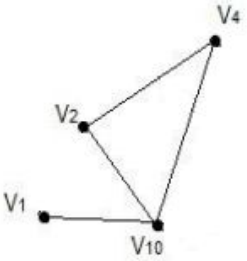
14


V3
V4
V5
V2
V6
V1
V7
V10
V8
V9

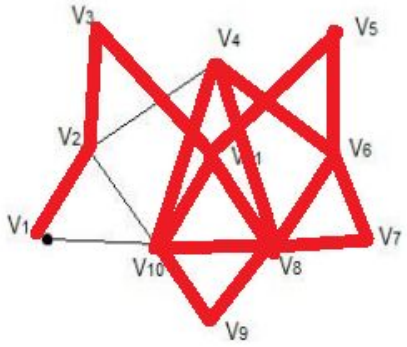

V4
V2
V1
V10

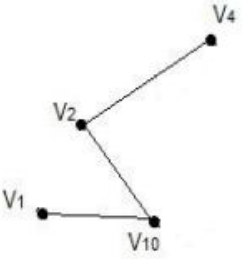
15


V3
V4
V5
V2
V6
V1
V7
V10
V8
V9

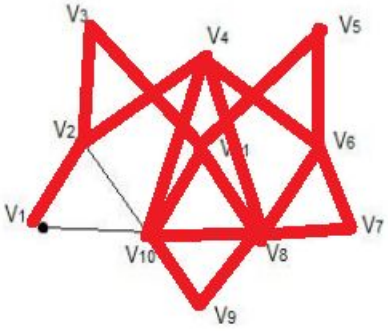

V4
V2
V1
V10

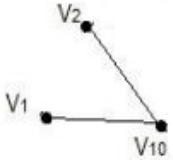
16


V3
V4
V5
V2
V6
V1
V7
V10
V8
V9

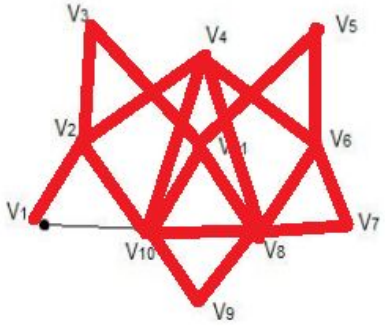

V4
V2
V1
V10

17

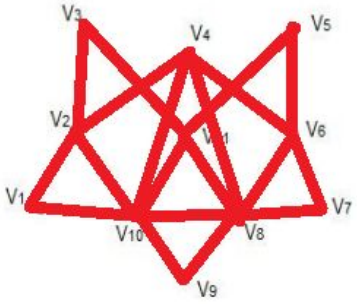

V3
V4
V5
V2
V6
V1
V7
V10
V8
V9


V2
V1
V10

18



19

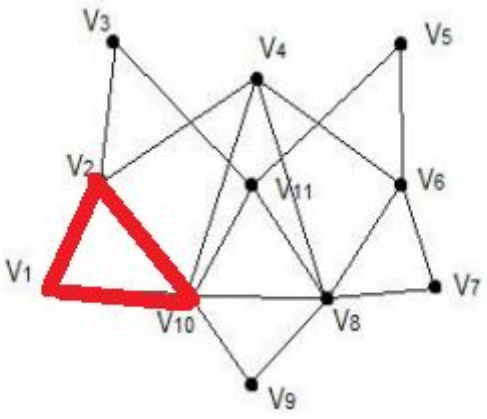


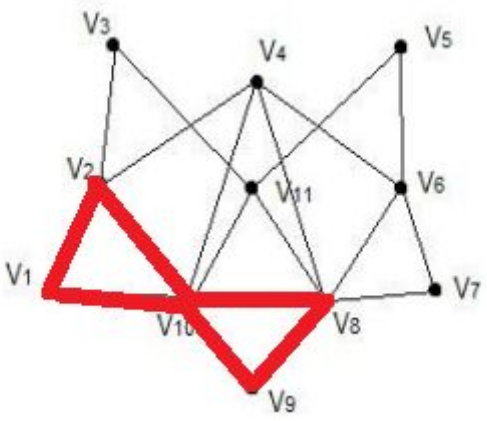

```
#include<iostream>
#include<vector>
#define v 12
using namespace std;
int matrix[v][v] = {
    {0,0,0,0,0,0,1,0,1,0,0},
    {0,0,1,0,0,0,1,0,1,0,1,0},
    {0,1,0,1,0,1,0,1,0,1,0,1},
    {0,0,1,0,0,0,1,0,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0,0,1},
    {0,0,1,0,1,0,0,0,0,0,0,0},
    {0,1,0,1,0,0,0,1,0,0,0,1},
    {1,0,1,0,0,0,1,0,0,0,1,0},
    {0,1,0,0,0,0,0,0,0,1,0,0},
    {1,0,1,0,0,0,0,0,1,0,1,0},
    {0,1,0,0,0,0,0,1,0,1,0,1},
    {0,0,1,0,1,0,1,0,0,0,1,0}
};
int temp[v][v];
int findstartvr() {
    for (int i = 0; i < v; i++) {
        int stp = 0;
        for (int j = 0; j < v; j++) {
            if (temp[i][j])
                stp++;
        }
        if (stp % 2 != 0)
            return i;
    }
    return 0;
}
bool most(int u, int vr) {
    int stp = 0;
    for (int i = 0; i < v; i++)
        if (temp[vr][i])
```

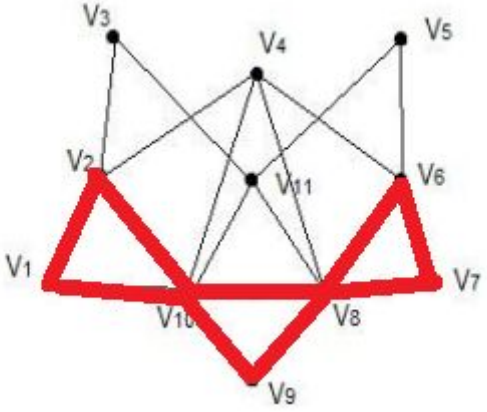
```
    for (int i = 0; i < v; i++)
        if (temp[vr][i])
            stp++;
    if (stp > 1) {
        return false;
    }
    return true;
}
int edgecount() {
    int counter = 0;
    for (int i = 0; i < v; i++)
        for (int j = i; j < v; j++)
            if (temp[i][j])
                counter++;
    return counter;
}
void fleri(int start) {
    static int edge = edgecount();
    for (int i = 0; i < v; i++) {
        if (temp[start][i]) {
            if (edge <= 1 || !most(start, i)) {
                cout << start+1 << "-" << i+1 << " ";
                temp[start][i] = temp[i][start] = 0;
                edge--;
                fleri(i);
            }
        }
    }
}
int main() {
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            temp[i][j] = matrix[i][j];
    cout << "Euler Path Or Circuit: ";
    fleri(findstartvr());
```

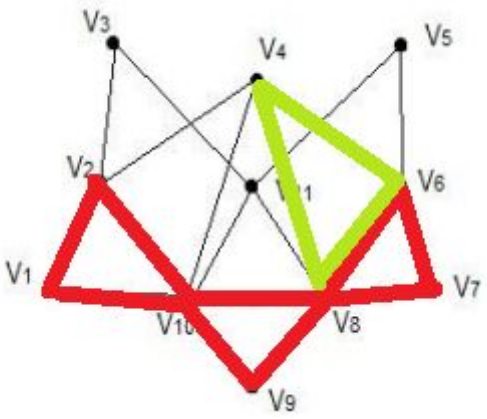
```
1->2->3->11->5->6->7->8->9->10->8->6->4->8->11->10->4->2->10->1
```

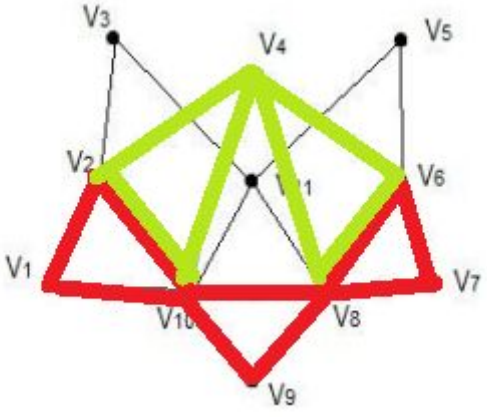
метод елементарних циклів:

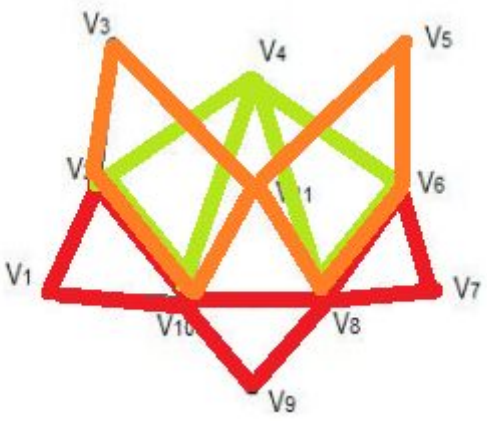



V3
V4
V5
V2
V6
V11
V1
V7
V10
V8
V9

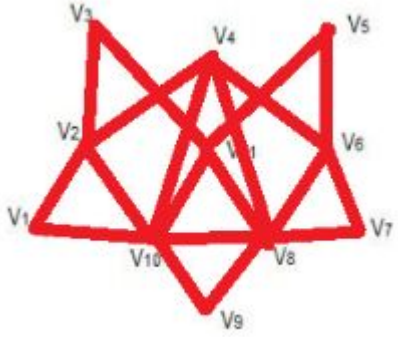

V3
V4
V5
V2
V6
V11
V1
V7
V10
V8
V9


V3
V4
V5
V2
V6
V1
V7
V8
V9


V3
V4
V5
V2
V6
V1
V7
V8
V9



Знайдено всі елементарні цикли, ось результат їх поєднання:




```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
#include <list>
using namespace std;
stack<int> head ;
stack<int>vidpovid;
vector <int> stp;
vector < list<int> > graf;

int main()
{
    int n, a, x, y;
    cin >> n >> a;
    graf.resize(n + 1);
    stp.resize(n + 1);
    for (; a--;)
    {
        cin >> x >> y;
        graf[x].push_back(y);
        graf[y].push_back(x);
        ++stp[x];
        ++stp[y];
    }
    if (any_of(stp.begin() + 1, stp.end(), [](int i) {return i & 1; }))
```

```
        graf[y].push_back(x);
        ++stp[x];
        ++stp[y];
    }
    if (any_of(stp.begin() + 1, stp.end(), [](int i) {return i & 1; }))
        cout << "Vidsutnii eilerovi zikl";
    else
    {
        head.push(1);
        while (!head.empty())
        {
            while (stp[head.top()])
            {
                int v = graf[head.top()].back();
                graf[head.top()].pop_back();
                graf[v].remove(head.top());
                --stp[head.top()];
                head.push(v);
                --stp[v];
            }
            while (!head.empty() && !stp[head.top()])
            {
                vidpovid.push(head.top());
                head.pop();
            }
        }
        while (!vidpovid.empty())
        {
            cout << vidpovid.top() << ' ';
            vidpovid.pop();
        }
    }
}
```

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

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

12. $\bar{x}y \vee x\bar{y}\bar{z}$

Скорочена ДНФ (англ. Reduced disjunctive normal form) - форма запису функції, що володіє наступними властивостями:

1) будь-які два доданки відрізняються як мінімум в двох позиціях,
2) жоден з Кон'юнктів не міститься в іншому.

Наша функція відповідає цим вимогам, тому можемо стверджувати що ця функція вже є скороченою ДНФ, і спростити її не можна.