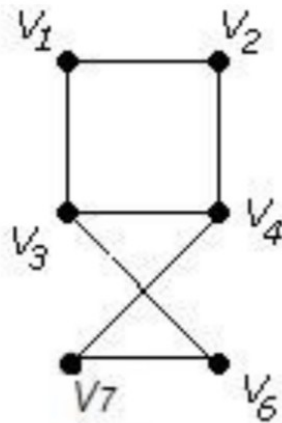
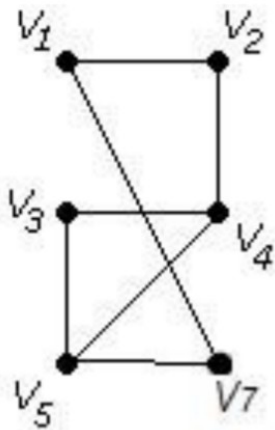


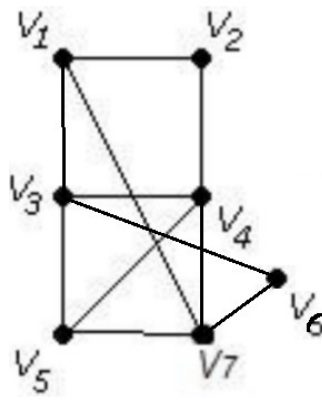
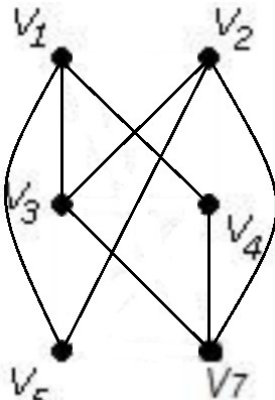
Завдання №1

Виконати наступні операції над графами: 1) знайти доповнення до першого графу, 2) об'єднання графів, 3) кільцеву сумму G_1 та G_2 (G_1+G_2), 4) розмножити вершину у другому графі, 5) виділити підграф A - що складається з 3-х вершин в G_1 , 6) добуток графів.



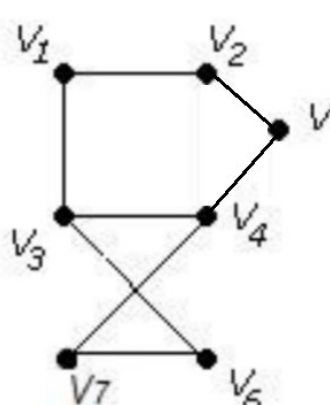
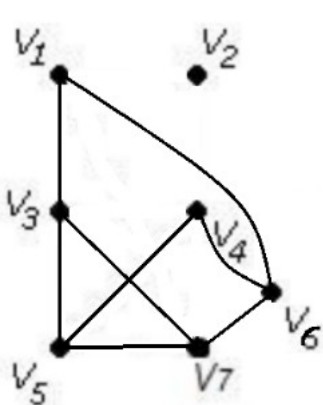
1)

2)

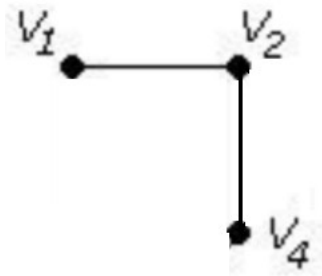


3)

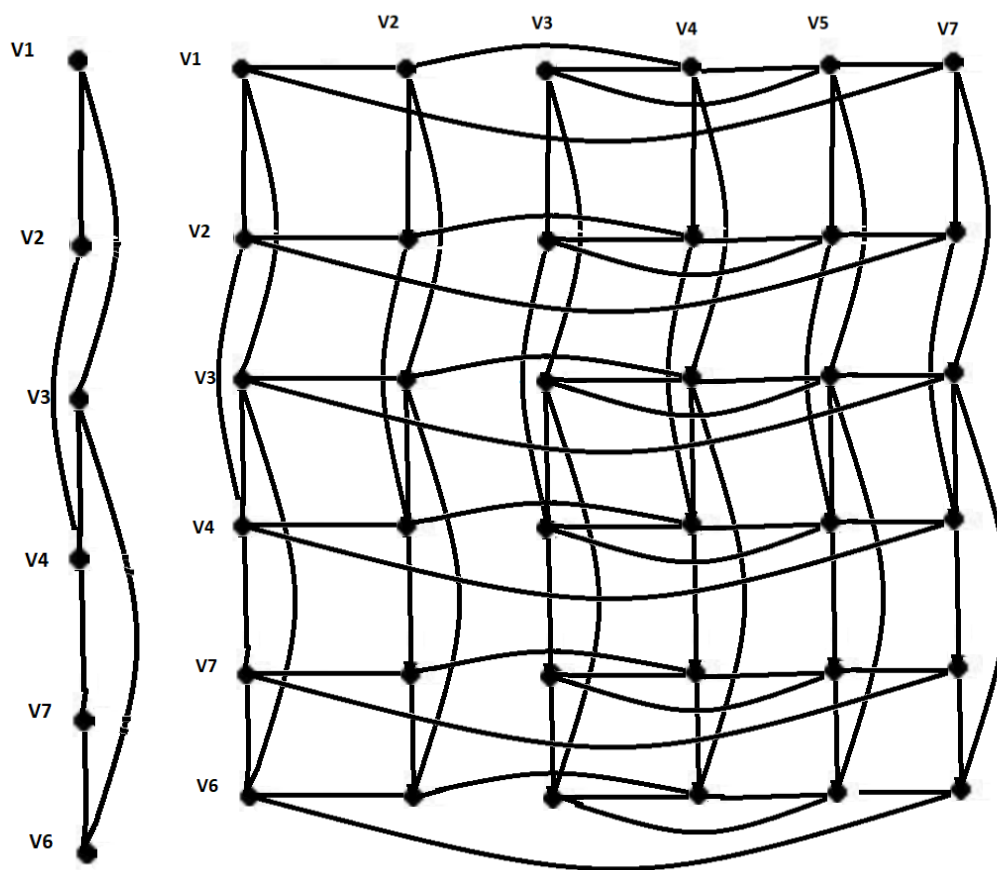
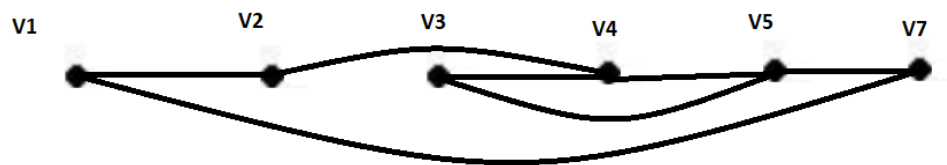
4)



5)



6)



Завдання №2

Скласти таблицю суміжності для орграфа.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
V1	0	1	0	0	0	0	1	1	0	0
V2	1	0	1	0	0	0	1	0	0	0

V3	0	1	0	1	0	0	0	0	1	0
V4	0	0	1	0	1	0	1	0	1	0
V5	0	0	0	1	0	1	1	0	0	0
V6	0	0	0	0	1	0	1	1	0	0
V7	1	1	0	1	1	1	0	1	0	0
V8	1	0	0	0	0	1	1	0	1	0
V9	0	0	1	1	0	0	1	1	0	1
V10	0	0	0	0	0	0	0	0	1	0

Завдання №3

Для графа з другого завдання знайти діаметр.

Діаметр: 3.

Завдання №4

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

Код:

Вершини	№	Стек
1	1	1
2	2	12
3	3	123
9	4	1239
10	5	123910
-	-	1239
8	6	12398
7	7	123987
6	8	1239876
5	9	12398765
4	10	123987654
-	-	12398765
-	-	1239876
-	-	123987
-	-	12398
-	-	1239
-	-	1239
-	-	123
-	-	12
-	-	1
-	-	Ø

```

#include <iostream>

using namespace std;

const int n = 10;
int i, j;
bool* visited = new bool[n];
int graph[n][n] =
{
    {0,1,0,0,0,0,1,1,0,0},
    {1,0,1,0,0,0,1,0,0,0},
    {0,1,0,1,0,0,0,0,1,0},
    {0,0,1,0,1,0,1,0,1,0},
    {0,0,0,1,0,1,1,0,0,0},
    {0,0,0,0,1,0,1,1,0,0},
    {1,1,0,1,1,1,0,1,0,0},
    {1,0,0,0,0,1,1,0,1,0},
    {0,0,1,1,0,0,1,1,0,1},
    {0,0,0,0,0,0,0,0,1,0}
};

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

int main()
{
    int start, x;
    cout << "Matrix: " << endl;
    for (i = 0; i < n; i++)
    {
        visited[i] = false;
        for (j = 0; j < n; j++)
            cout << " " << graph[i][j];
        cout << endl;
    }
    start = 1;
    bool* vis = new bool[n];
    cout << "Path: "; cin >> x;
    DFS(start - 1);
    delete[] visited;
    system("pause>>void");
}

```

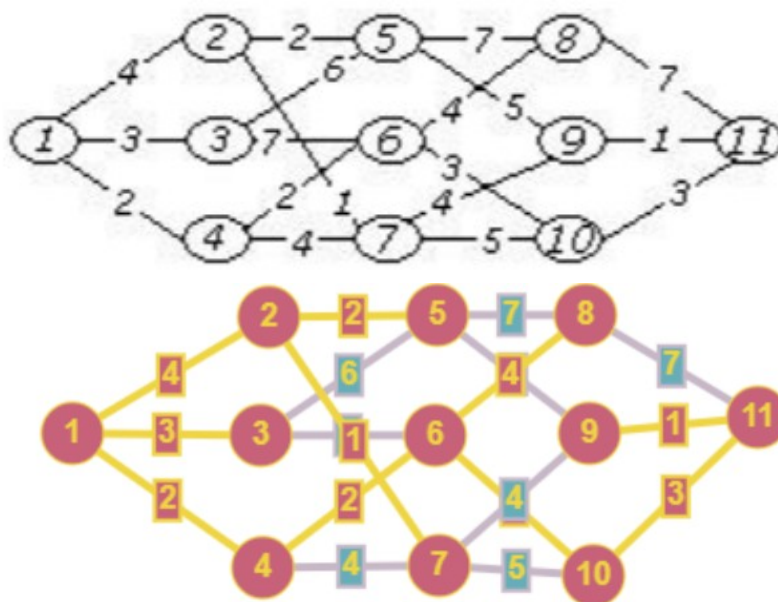
Вивід:

```
"D:\лэ|тхЁ\—шёьЁхСэр ьрСхьрСшър.ў 1\чочЁрїєэьютр Ё... — □ ×
Matrix:
0 1 0 0 0 0 1 1 0 0
1 0 1 0 0 0 1 0 0 0
0 1 0 1 0 0 0 0 1 0
0 0 1 0 1 0 1 0 1 0
0 0 0 1 0 1 1 0 0 0
0 0 0 0 1 0 1 1 0 0
1 1 0 1 1 1 0 1 0 0
1 0 0 0 0 1 1 0 1 0
0 0 1 1 0 0 1 1 0 1
0 0 0 0 0 0 0 0 1 0
Path: 1 2 3 9 10 8 7 6 5 4

Process returned 0 (0x0)   execution time : 42.742 s
Press any key to continue.
■
```

Завдання №5

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



Краскала:

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

Прима:

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

Код(Прима):

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

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

    cout << "The quantity of tops: "; cin >> versh; cout <<
    "\nMatrix: \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)
        {
            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;
        }
    }

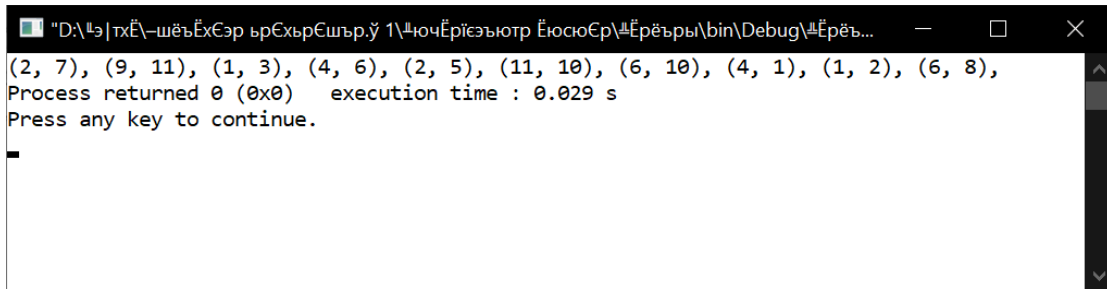
    graph[n - 1][m - 1] = 0;
    graph[m - 1][n - 1] = 0;
    tops[cnt] = m;
    ++cnt;
    min_ = 0;
}

for (int i = 0; i < versh - 1; ++i)
    cout << "(" << rebr[i][0] << ", " << rebr[i][1] << ") ";

}

```

Вивід:



```
"D:\...-ш...\... 1\... \bin\Debug\...  
(2, 7), (9, 11), (1, 3), (4, 6), (2, 5), (11, 10), (6, 10), (4, 1), (1, 2), (6, 8),  
Process returned 0 (0x0)   execution time : 0.029 s  
Press any key to continue.  
_
```

Код(Краскала):

```
#include <iostream>  
#include <vector>  
  
struct edge{  
    int u, v, weight;  
    edge(){};  
    edge(int u_, int v_, int weight_){  
        u = u_;  
        v = v_;  
        weight = weight_;  
    };  
};  
void print_edge(edge e){  
    std::cout << "(" << e.u << ", " << e.v  
    << ")", ";  
}  
int id[11];  
  
int find(int p){  
    while(p!=id[p]){  
        p = id[p];  
    }  
    return p;  
}  
  
bool connected(int a, int b){  
    return(find(a) == find(b));  
}  
  
void unio(int p, int q){  
    int root1 = find(p);  
    int root2 = find(q);  
    if(root1 == root2){  
        return;  
    }  
}
```



```

    id[root1] = root2;
}
int main() {
    for (int i = 0; i < 12; i++) {
        id[i] = i;
    }

```

```

const int num_of_edges = 18;
edge edges[num_of_edges] =
    {edge(1, 2, 4),
     edge(2, 5, 2),
     edge(5, 8, 7),
     edge(8, 11, 7),
     edge(11, 10, 2),
     edge(10, 7, 4),
     edge(7, 4, 4),
     edge(4, 1, 3),
     edge(1, 3, 2),
     edge(3, 5, 6),
     edge(3, 6, 7),
     edge(4, 6, 2),
     edge(2, 7, 1),
     edge(6, 8, 4),
     edge(6, 10, 3),
     edge(5, 9, 5),
     edge(7, 9, 5),
     edge(9, 11, 1)
    };

```

```

bool sorted = false;
while (!sorted) {
    sorted = true;
    for (int i = 0; i < 17; i++) {
        for (int j = i + 1; j < 18; j++) {
            if (edges[j].weight < edges[i].weight) {
                sorted = false;
                edge tmp = edges[j];
                edges[j] = edges[i];
                edges[i] = tmp;
            }
        }
    }
}

```

```

edge mst[num_of_edges];
int mstsize = 0;

```

```

for (edge e:edges) {
    // print_edge(e);
}

for (edge e: edges) {

    if(!connected(e.v, e.u)){
        unio(e.v, e.u);
        mst[mstsize] = e;
        mstsize++;
    }
}
for (int i = 0; i<mstsize; i++) {
    print_edge(mst[i]);
}
return 0;
}

```

Вивід:

```

"D:\...-ш...\... \... \... \bin\Debug\...
(2, 7), (9, 11), (1, 3), (4, 6), (2, 5), (11, 10), (6, 10), (4, 1), (1, 2), (6, 8),
Process returned 0 (0x0) execution time : 0.029 s
Press any key to continue.

```

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

1 -> 8 -> 6 -> 3 -> 7 -> 5 -> 2 -> 4 -> 1;

$2 + 1 + 5 + 1 + 1 + 2 + 1 + 2 = 15;$

Код:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

class Komivoiser {
public:
    string name;
    int number;
    Komivoiser() {}
};

int main()
{
    setlocale(LC_ALL, "Ukrainian");
    int v = 0;
    cout << "Кількість вершин : ";
    cin >> v;
    int** graph = new int* [v];

    for (int j = 0; j < v; j++) {
        graph[j] = new int[v];
    }

    cout << "Бага ребер : " << endl;

    for (int a = 0; a < v; a++) {
        for (int j = 0; j < v; j++) {
            cin >> graph[a][j];
        }
    }

    ///////////
    int* a = new int[v];
```

```

for (int i = 0; i < v; i++)
    a[i] = i + 1;
int n = sizeof(a) / sizeof(a[0]);

vector<Komivoiser> Path;
int min_path=0;
sort(a, a + v);
for (int i = 1; i < v; i++) {

    min_path += graph[a[i] - 1][a[i] - 1];

}
min_path += graph[a[v - 1] - 1][a[0] - 1];
do {
    Komivoiser t;
    t.name = to_string(a[0]); t.number = 0;
    for (int i = 1; i < v; i++) {
        t.name += "->" + to_string(a[i]);
        t.number += graph[a[i-1]-1][a[i]-1];

    }
    t.name += "->" + to_string(a[0]);
    t.number += graph[a[v-1] - 1][a[0] - 1];
    Path.push_back(t);
    if (min_path > t.number) min_path = t.number;
} while (next_permutation(a, a + v));

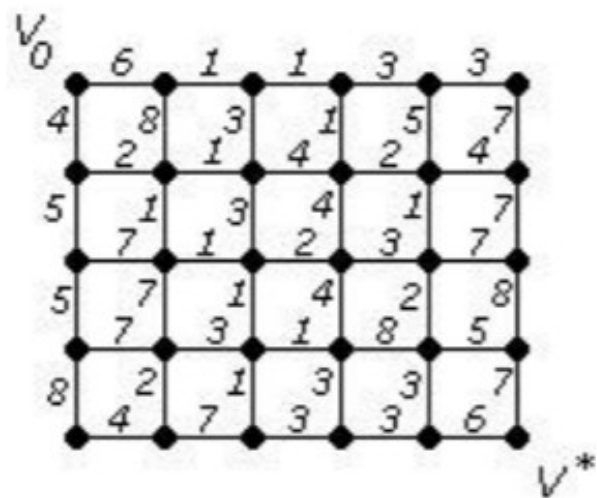
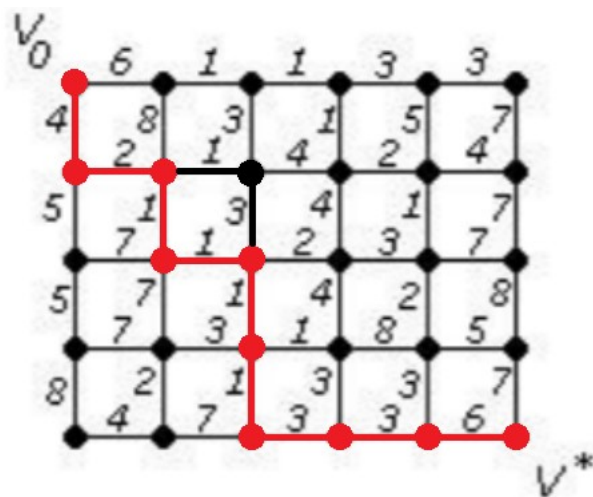
cout << "Оптимальні шляхи:" << endl;
for (int i = 0; i < Path.size(); i++) {
    if(Path[i].number == min_path){
        cout << "Path: " << Path[i].name << " " << "weight: "
<< Path[i].number << endl;
    }
}

return 0;
}

```

Завдання №7

За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V_0 і V^* .



Код:

```
#include <iostream>

using namespace std;

int choose_min(int** arr,int n)
{
    int x;

    for (int i = 0; i < n; i++)
        if (arr[i][1])
        {
            x = i;
            break;
        }
}
```

```

        for (int i = 1; i < n; i++)
            if (arr[x][0] >= arr[i][0] && arr[i][1] == 1)
                x = i;

        return x;
    }

int main()
{
    int inf = 1000000;
    int a, b, c;
    int v;
    cout << "Number of tops: "; cin >> v;
    cout << "Weight of ribs : " << "\n";

    int** gr = new int* [v];

    for (int j = 0; j < v; j++)
        gr[j] = new int[v];

    for (int a = 0; a < v; a++)
        for (int j = 0; j < v; j++)
            gr[a][j] = 0;

    while(1)
    {
        cin >> a; if (a == 0) break;
        cin >> b;
        cin >> c;
        gr[a-1][b-1] = gr[b-1][a-1] = c;
    }

    int f;
    int** tops = new int*[v];
    for (int i = 0; i < v; i++)
        tops[i] = new int[2];

    int* path = new int[v];

    cout << "From: "; cin >> f;

    for (int i = 0; i < v; i++)
    {
        if (i == f-1)
        {
            tops[i][0] = 0;
            tops[i][1] = 1;
        }
        else

```

```

        {
            tops[i][0] = inf;
            tops[i][1] = 1;
        }
    }
    path[f - 1] = 0;

    int ch;

    for (int i = 0; i < v; i++)
    {
        ch = choose_min(tops, v);
        for (int j = 0; j < v; j++)
            if (gr[ch][j])
                if (tops[j][0] > tops[ch][0] + gr[ch][j])
                {
                    tops[j][0] = tops[ch][0] + gr[ch][j];
                    path[j] = ch;
                }
        tops[ch][1] = 0;
    }

    cout << "To: ";
    int k; cin >> k;
    cout << "Min path: ";
    cout << tops[k - 1][0];
    cout << endl << k << " - ";
    k--;
    for (int i = 0; path[k] != f - 1; i++)
    {
        cout << path[k] + 1 << " - ";
        k = path[k];
    }
    cout << f << "\n";
    return 0;
}

```

Вивід:

```

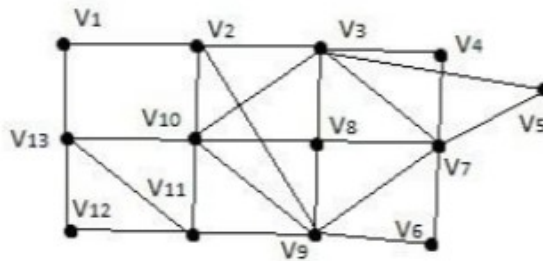
24 30 7
25 26 4
26 27 7
27 28 3
28 29 3
29 30 6
0
From: 1
To: 30
Min path: 22
30 - 29 - 28 - 27 - 21 - 15 - 14 - 8 - 7 - 1

Process returned 0 (0x0)   execution time : 118.558 s
Press any key to continue.

```

Завдання №8

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



1 -> 2 -> 3 -> 4 -> 7 -> 5 -> 3 -> 10 -> 9 -> 8 -> 3 -> 7 -> 6 -> 9 -> 11 -> 12 -> 13 -> 11 -> 10 -> 8 -> 7 -> 9 -> 2 -> 10 -> 13 -> 1.

Код(Флері):

```
#include <iostream>
#include <string.h>
#include <algorithm>
#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 << 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 << "-" << v << " ";
            rmvEdge(u, v);
            printEulerUtil(v);
        }
    }
}

bool Graph::isValidNextEdge(int u, int v)
{
    int count = 0;
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
        if (*i != -1)
            count++;
    if (count == 1)
        return true;
    bool visited[V];
    memset(visited, false, V);
    int count1 = DFSCount(u, visited);

    rmvEdge(u, v);
    memset(visited, false, V);
    int count2 = DFSCount(u, visited);
    addEdge(u, v);
    return (count1 > count2)? false: true;
}

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

```

```
}
```

```
int Graph::DFSCount(int v, bool visited[])
{
    visited[v] = true;
    int count = 1;

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

    return count;
}
```

```
int main()
{
    Graph g1(13);
    g1.addEdge(1-1, 2-1);
    g1.addEdge(13-1, 1-1);
    g1.addEdge(2-1, 3-1);
    g1.addEdge(2-1, 9-1);
    g1.addEdge(2, 10-1);
    g1.addEdge(3-1, 4-1);
    g1.addEdge(3-1, 7-1);
    g1.addEdge(3-1, 5-1);
    g1.addEdge(3-1, 8-1);
    g1.addEdge(10-1, 3-1);
    g1.addEdge(4-1, 7-1);
    g1.addEdge(7-1, 5-1);
    g1.addEdge(7-1, 6-1);
    g1.addEdge(6-1, 9-1);
    g1.addEdge(9-1, 7-1);
    g1.addEdge(8-1, 7-1);
    g1.addEdge(8-1, 9-1);
    g1.addEdge(10-1, 8-1);
    g1.addEdge(10-1, 9-1);
    g1.addEdge(9-1, 11-1);
    g1.addEdge(10-1, 11-1);
    g1.addEdge(13-1, 10-1);

    g1.addEdge(11-1, 12-1);
    g1.addEdge(13-1, 11-1);
    g1.addEdge(12-1, 13-1);
    return 0;
}
```

Вивід(Від 0 до 12):

Код(элементарні цикли):

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
#include <list>
using namespace std;

vector < list<int> > graph;
vector <int> deg;
stack<int> head,tail ;

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

        cout << "-1";
    else
    {
        head.push(1);
        while(!head.empty())
        {
            while(deg[head.top()])
            {

                int v = graph[head.top()].back();
```

```

graph[head.top()].pop_back();
graph[v].remove(head.top());
--deg[head.top()] ;
head.push(v);
--deg[v];

}
while(!head.empty()&&!deg[head.top()])
{
    tail.push( head.top());
    head.pop();
}
}
while(!tail.empty())
{
    cout << tail.top() <<' ';
    tail.pop();
}

}
}

```

Вивід:

```

10 3
10 9
10 8
3 8
8 9
3 7
9 7
8 7
7 5
2 9
3 5
1 13 11 10 8 7 5 3 7 9 2 10 9 8 3 10 13 12 11 9 6 7 4 3 2 1
Process returned 0 (0x0)   execution time : 90.012 s
Press any key to continue.

```
