МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

РОЗРАХУНКОВО-ГРАФІЧНІ ЗАВДАННЯ

з дисципліни

“ДИСКРЕТНА МАТЕМАТИКА”

Виконав:

студент групи КН-113

Сеньків Максим

Викладач:

Мельникова Наталя Іванівна

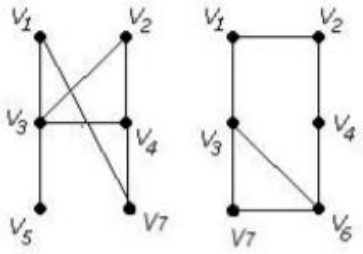
Львів – 2019 р.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

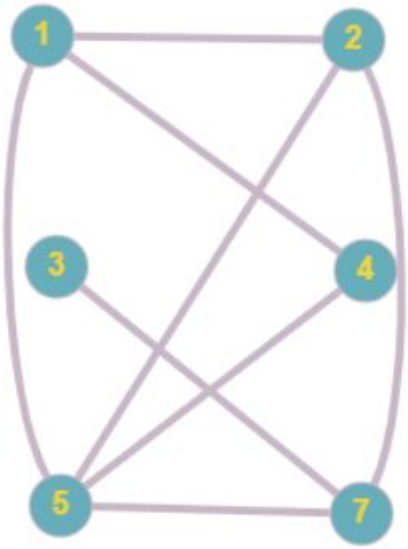
Завдання № 1

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.

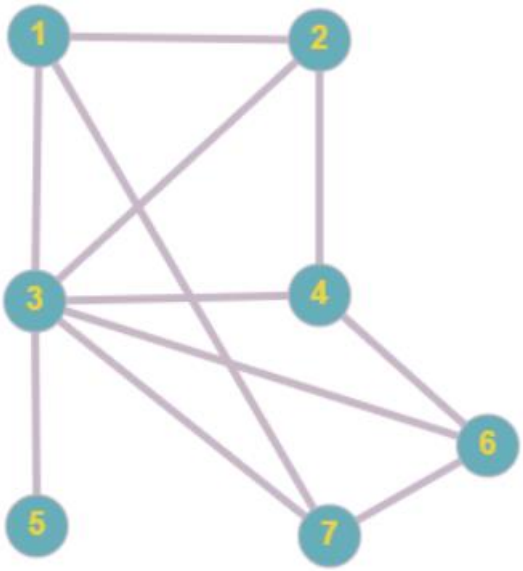
23)



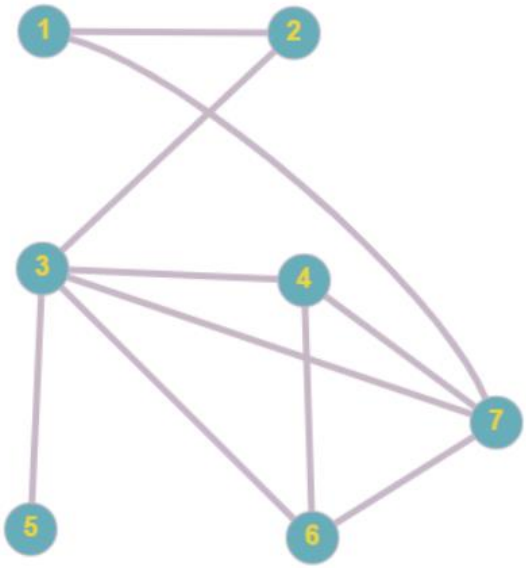
1)



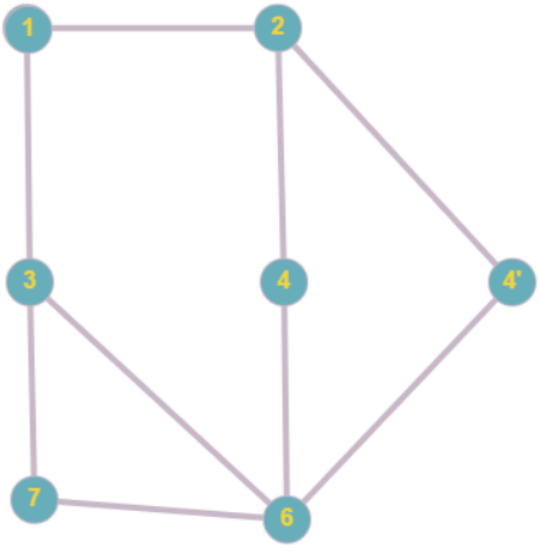
2)



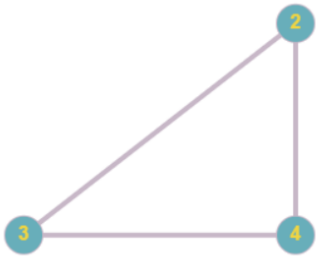
3)



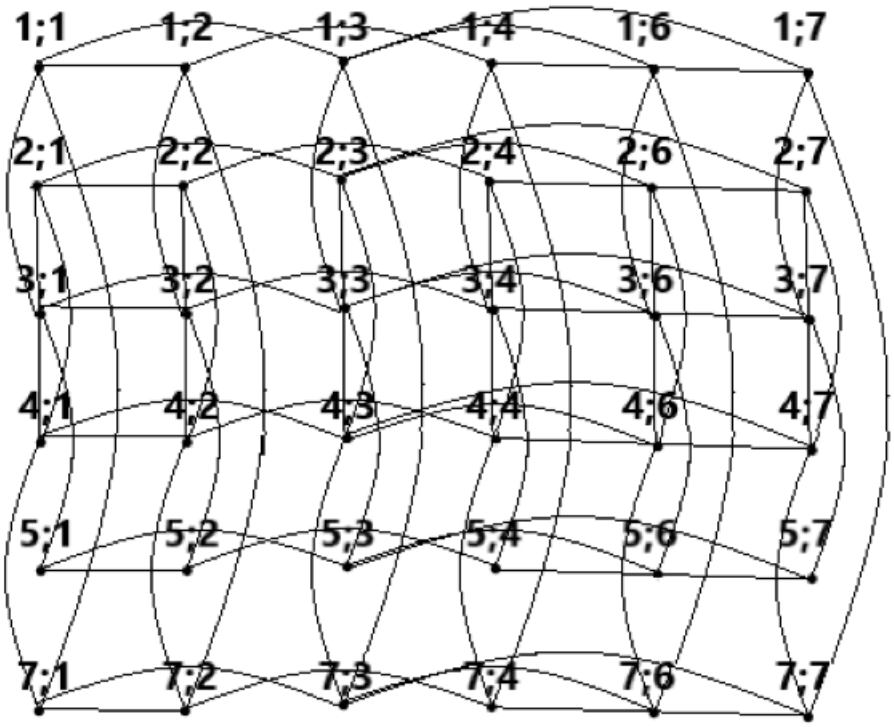
4)



5)



6)

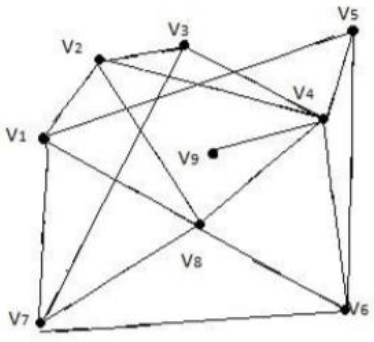


Завдання № 2

Скласти таблицю суміжності для орграфа.

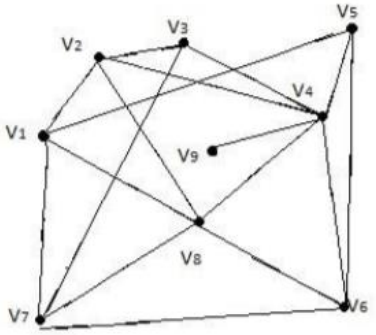
	V1	V2	V3	V4	V5	V6	V7	V8	V9
V1	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**
V2	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
V3	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
V4	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
V5	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**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

23)



Завдання № 3

23)



Для графа з другого завдання знайти діаметр.

Діаметр = V9->V4->V8->V1 = 3

Завдання № 4

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

Вершина	DFS-номер	Вміст стеку
V1	1	V1
V2	2	V1V2
V3	3	V1V2V3
V4	4	V1V2V3V4
V9	5	V1V2V3V4V9
_	_	V1V2V3V4
V5	6	V1V2V3V4V5
V6	7	V1V2V3V4V5V6
V7	8	V1V2V3V4V5V6V7
V8	9	V1V2V3V4V5V6V7V8
_	_	V1V2V3V4V5V6V7
_	_	V1V2V3V4V5V6
_	_	V1V2V3V4V5
_	_	V1V2V3V4
_	_	V1V2V3
_	_	V1V2
_	_	V1
_	_	Ø

Алгоритм для обходу графа вглиб та вшир:

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

struct vershina
{
    bool dfs = false;
};
struct rebro
{
    int v1;
    int v2;
};

int leng(string str)
{
    int i = 0;

    while (str[i] != '\0')
    {
        i++;
    }
    return i;
}
int correct(int m, int n)
{
    int c = 0;
    bool count = false;
    string str;
    stringstream ss;
    while (count == false)
    {
        cin >> str;
        for (int i = 0; i < leng(str); i++)
        {
            if (!isdigit(str[i]))
            {
                if (i == 0 && str[i] == '-')
                {
                    count = true;
                }
                else
                {
                    count = false;
                    break;
                }
            }
            else
            {
                count = true;
            }
        }

        if (str[0] == '0')
        {
            count = false;
        }

        if (count == true)
        {
```

```
            ss << str;
            ss >> c;
            ss.clear();

            if (c < m || c > n)
            {
                count = false;
            }
            else
            {
                count = true;
            }
        }
        if (count == false)
        {
            cout << «Error! Try again!» << endl;
        }
        str = «»;
    }

    return c;
}

void input(rebro *reb, int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        cout << «Enter the first vertex, the incident edge number» << i + 1 << «: «;
        reb[i].v1 = correct(1, m);
        cout << «Enter the first vertex, the incident edge number» << i + 1 << «: «;;
        reb[i].v2 = correct(1, m);
        cout << endl;
    }
}

int main()
{
    int n, m, p;
    int begin;
    int count = 0;
    int t = 0;
    int head = 0;

    cout << «Enter the number of edges in the graph: «;
    n = correct(1, 1000);
    cout << «Enter the number of top in the graph: «;
    m = correct(2, 1000);
    cout << endl;

    int *vec = new int[m];
    rebro *reb = new rebro[n];
    vershina *v = new vershina[m];

    input(reb, n, m);

    cout << «From which top to start the detour? «;
    begin = correct(1, m);

    vec[0] = begin;
    v[begin - 1].dfs = true;
    count++;

    cout << «press 1 to the passage deep 1,\npress 2 to the passage wide: «;
    cout << endl;
```

```
    p = correct(1, 2);
    cout << endl;
    cout << begin << endl;
    switch (p)
    {
        case 1:
        {
            while (count != 0)
            {
                for (int i = 0; i < n; i++)
                {

                    if ((vec[count - 1] == reb[i].v1 && v[reb[i].v2 - 1].dfs == false)
|| (vec[count - 1] == reb[i].v2 && v[reb[i].v1 - 1].dfs == false))
                    {
                        t++;
                    }
                }

                if (t == 0)
                {
                    count--;
                }
                else
                {
                    for (int i = 0; i < n; i++)
                    {
                        if (vec[count - 1] == reb[i].v2 && v[reb[i].v1 - 1].dfs ==
false)
                        {
                            vec[count] = reb[i].v1;
                            v[reb[i].v1 - 1].dfs = true;

                            count++;
                            goto point;
                        }

                        if (vec[count - 1] == reb[i].v1 && v[reb[i].v2 - 1].dfs ==
false)
                        {
                            vec[count] = reb[i].v2;
                            v[reb[i].v2 - 1].dfs = true;

                            count++;
                            goto point;
                        }
                    }
                }
                point:;
                for (int i = 0; i < count; i++)
                {
                    cout << vec[i] << « «;
                }
                if (count != 0)
                {
                    cout << endl;
                }
                t = 0;
            }

            cout << «Empty stack» << endl;
            break;
        }
        case 2:
```

```
        {
            while (head != m)
            {
                for (int i = head; i < n; i++)
                {

                    if ((vec[head] == reb[i].v1 && v[reb[i].v2 - 1].dfs == false) ||
(vec[head] == reb[i].v2 && v[reb[i].v1 - 1].dfs == false))
                    {
                        t++;
                    }
                }

                if (t == 0)
                {
                    head++;
                }
                else
                {
                    for (int i = head; i < n; i++)
                    {
                        if (vec[head] == reb[i].v2 && v[reb[i].v1 - 1].dfs == false)
                        {
                            vec[count] = reb[i].v1;
                            v[reb[i].v1 - 1].dfs = true;

                            count++;
                            goto point1;
                        }

                        if (vec[head] == reb[i].v1 && v[reb[i].v2 - 1].dfs == false)
                        {
                            vec[count] = reb[i].v2;
                            v[reb[i].v2 - 1].dfs = true;

                            count++;
                            goto point1;
                        }
                    }
                }
                point1:;
                for (int i = head; i < count; i++)
                {
                    cout << vec[i] << « «;
                }
                if (head != m)
                {
                    cout << endl;
                }
                t = 0;
            }

            cout << «Empty queue» << endl;
            break;
        }
    }

    return 0;
}
```

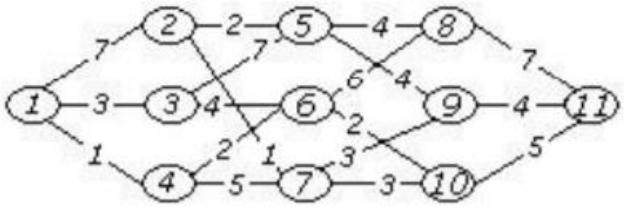
Виведення:

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

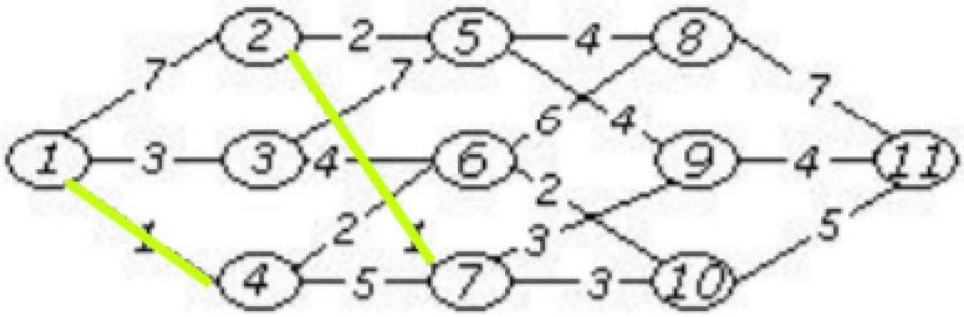
Завдання № 5

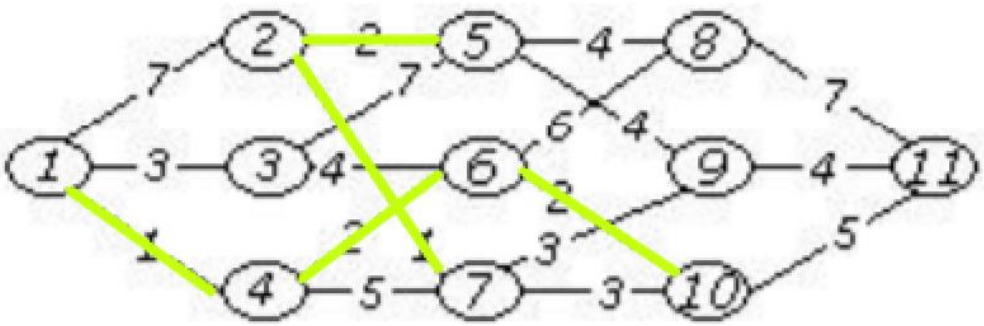
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

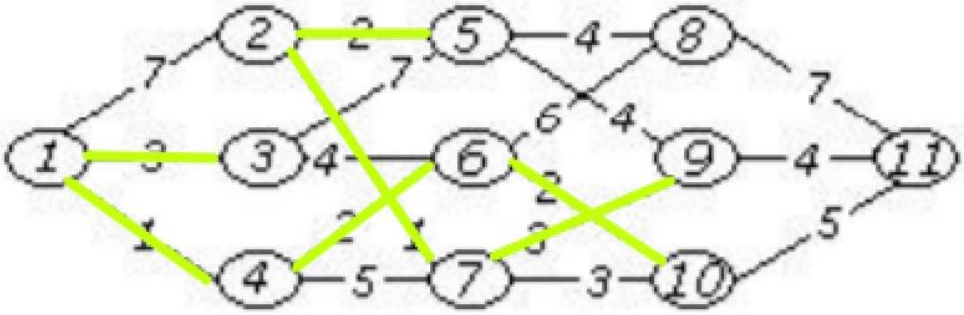
23)

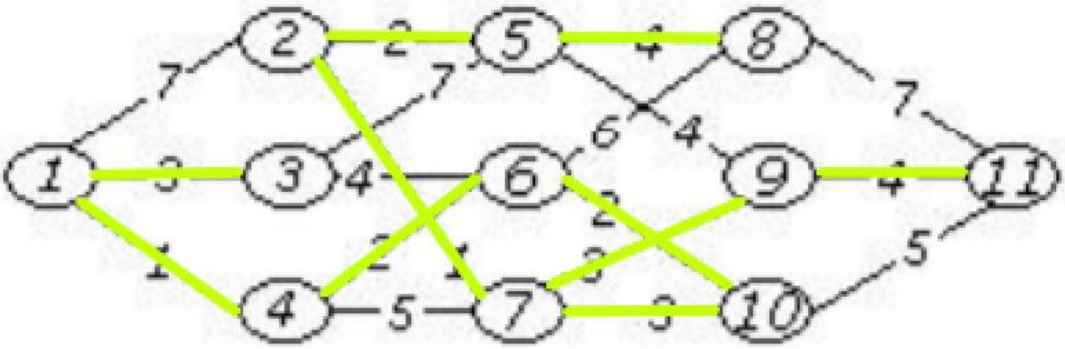


Алгоритм Краскала

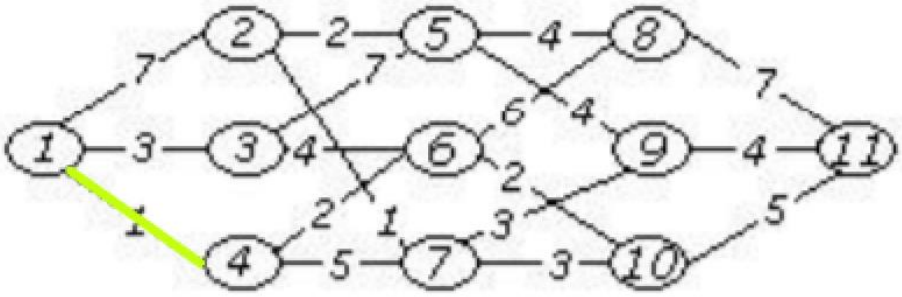


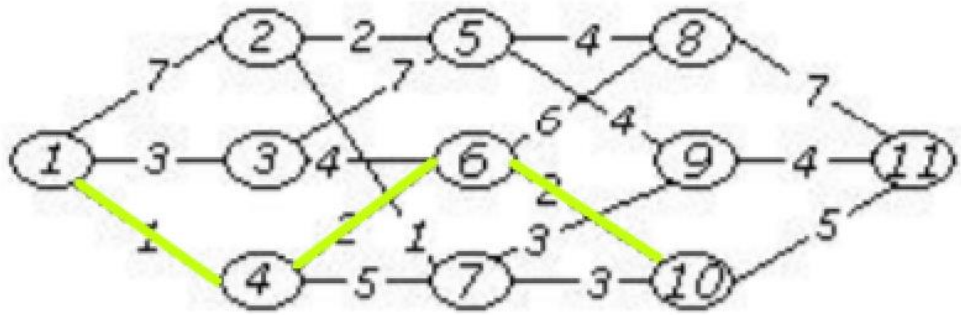
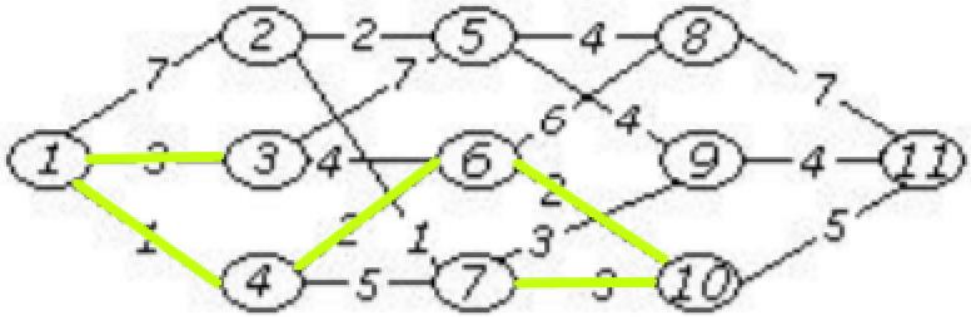
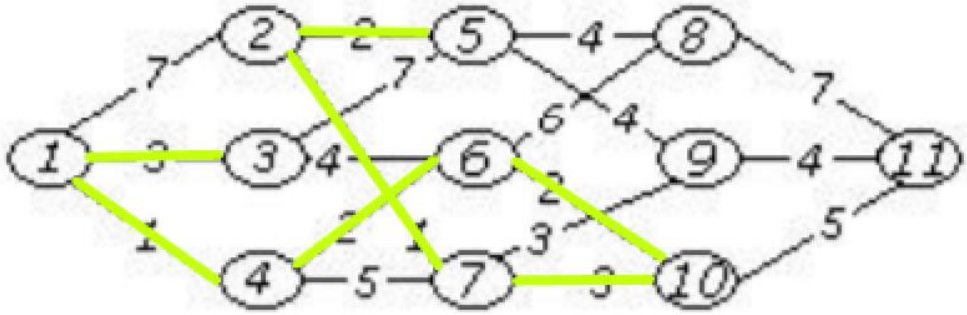
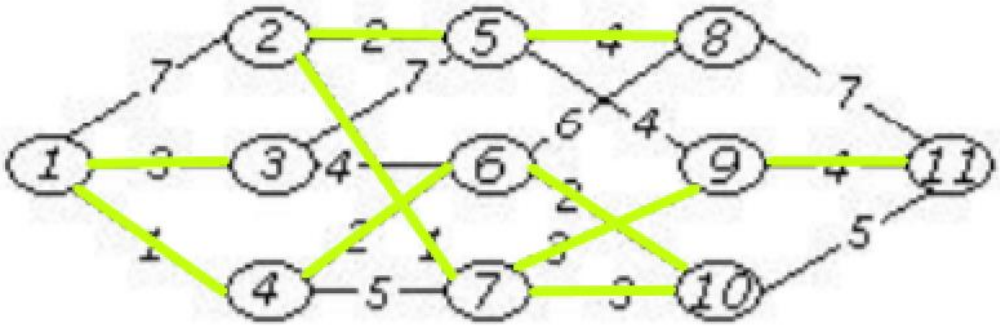






Алгоритм Прима



Алгоритм Краскала для знаходження найменшого остового дерева.

```
#include <iostream>
#include <stdio.h>
using namespace std;
struct rebro {

    int leng;
    int v1;
    int v2;
    bool in = false;
};
struct mas {
```

```
    int arr[100];
    int c = 0;
};

int in(rebro *reb, int n) {
    setlocale(LC_ALL, "Ukrainian");
    for (int i = 0; i < n; i++)
    {
        cout << "Enter a length " << i + 1 << " edge: ";
        cin >> reb[i].leng;
        cout << "Enter the first adjacent vertex with " << i + 1 << " edge: ";
        cin >> reb[i].v1;
        cout << "Enter the first adjacent vertex with " << i + 1 << " edge: ";
        cin >> reb[i].v2;
        cout << endl;
    }
    return 0;
}
int main() {

    int n = 0, x = 100, y = 100;

    cout << "Enter the number of edges in the graph: ";
    cin >> n;
    int z;
    cout << "Enter the number of tops in the graph: ";
    cin >> z;
    cout << endl;

    rebro *reb = new rebro[n];
    mas inn[15];

    for (int i = 0; i < 15; i++)
    {
        for (int j = 0; j < z; j++)
        {
            inn[i].arr[j] = 0;
        }
    }
    in(reb, n);

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - 1; j++)
        {
            if (reb[j].leng > reb[j + 1].leng) { swap(reb[j].leng, reb[j + 1].leng);
swap(reb[j].v1, reb[j + 1].v1); swap(reb[j].v2, reb[j + 1].v2); }
        }
    }

    for (int i = 0; i < n; i++)
        cout << reb[i].leng << " " << reb[i].v1 << " " << reb[i].v2 << endl;


    int c = -1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < 15; j++)
        {
            for (int k = 0; k < z; k++)
            {
                if (reb[i].v1 == inn[j].arr[k]) { x = j; goto point0;; }
            }
        }
```

```
        point0:;

        for (int j = 0; j < 15; j++)
        {
            for (int k = 0; k <z; k++)
            {
                if (reb[i].v2 == inn[j].arr[k]) { y = j; goto point1; }
            }
        }
        point1:;
        if (x != y && x == 100) { inn[y].arr[inn[y].c] = reb[i].v1; inn[y].c++; }

        if (x != y && y == 100) { inn[x].arr[inn[x].c] = reb[i].v2; inn[x].c++; }

        if (x != y && x != 100 && y != 100) {
            if (x < y) {
                for (int l = 0; l < inn[y].c; l++)
                {
                    inn[x].arr[inn[x].c+l] = inn[y].arr[l];
                    inn[y].arr[l] = 0;
                }
                inn[x].c += inn[y].c;
                inn[y].c = 0;
            }

            if (y < x) {
                for (int l = 0; l < inn[x].c; l++)
                {
                    inn[y].arr[inn[y].c+l] = inn[x].arr[l];
                    inn[x].arr[l] = 0;
                }
                inn[y].c += inn[x].c;
                inn[x].c = 0;
            }
        }
        if (x == 100 && y == 100) { c++;  inn[c].arr[inn[c].c] = reb[i].v1;
inn[c].arr[inn[c].c + 1] = reb[i].v2; inn[c].c += 2; }

        reb[i].in = true;

        if (x == y && x != 100) { reb[i].in = false; }

        x = 100; y = 100;

        for (int j = 0; j < 15; j++)
        {
            for (int k = 0; k < 11; k++)
                cout << inn[j].arr[k] << " ";
            cout << endl;
        }
        cout << endl;
    }

    cout << "Edges that must be included in the tree: " << endl;

    int s = 0;;

    for (int i = 0; i < n; i++)
    {
        if (reb[i].in == true) { cout << "edge, that connect the vertices " <<
reb[i].v1 << " " << reb[i].v2 << endl; s += reb[i].leng; }
    }
    cout << "Spanning tree of minimum weight for the given graph: " << s;
    return 0;}
```

Виведення:

```
Edges that must be included in the tree:
edge, that connect the vertices 1 4
edge, that connect the vertices 2 7
edge, that connect the vertices 2 5
edge, that connect the vertices 4 6
edge, that connect the vertices 6 10
edge, that connect the vertices 1 3
edge, that connect the vertices 7 9
edge, that connect the vertices 7 10
edge, that connect the vertices 5 8
edge, that connect the vertices 9 11
Spanning tree of minimum weight for the given graph: 25
```

Алгоритм Прима для знаходження найменшого остового дерева.

```cpp
#include <iostream>

using namespace std;

int main() {

    int tree [11][11] {
            {0,7,3,1,0,0,0,0,0,0,0},
            {7,0,0,0,2,0,1,0,0,0,0},
            {0,0,0,0,7,4,0,0,0,0,0},
            {0,0,0,0,0,2,5,0,0,0,0},
            {0,2,7,0,0,0,0,4,4,0,0},
            {0,0,4,2,0,0,0,6,0,2,0},
            {0,1,0,5,0,0,0,0,3,3,0},
            {0,0,0,0,4,6,0,0,0,0,7},
            {0,0,0,0,4,0,3,0,0,0,4},
            {0,0,0,0,0,2,3,0,0,0,5},
            {0,0,0,0,0,0,0,7,4,5,0},
    };
    // записую матрицю інцидентності

    int min(100), min_e, min_x, count(1);
    int edges[11];
    edges[0] = 0;

    do {

        for(int i = 0; i < count; i++) {
            for (int x = 0; x < 11; x++) {
                if (tree[edges[i]][x] != 0 && tree[edges[i]][x] < min) {
                    min = tree[edges[i]][x];
                    min_e = x;
                    min_x = edges[i];
                }
            }
        }
        //знаходжу мінімальне ребро серед вершин, які відкриті

        edges[count] = min_e;
        // додаю до масиву наступну вершину

        count++;
        min = 10;

        for (int a = 0; a < 11; a++) {
```

```
            tree[a][min_e] = 0;
        }
        // занулюю стовбчик ребер з відкритою вершиною

        cout << min_x+1 << "-->" << min_e+1 << "\t(" << min_e + 1 << ")" << endl;

    }while (count < 11);
}
```

Виведення:

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

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

23)

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

Починаємо з 1 вершини та вибираємо найближчу.

Це 5 вершина, йдемо у неї.

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

Знову вибираємо найближчу і йдемо в неї(4).

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

Повторюємо процедуру, доки не пройдемо всі вершини.

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

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

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

	2	**6**
2	∞	5
6	**5**	∞

Маршрут: 1->5->4->3->7->8->6->2->1

Довжина маршруту: 17

Алгоритм Комівояжера для повного 8-ми вершинного графа:

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <fstream>

using namespace std;
ifstream fin;
string path = "MyFile.txt";

int** input() {
    int count = 8;

    string str;
    str = "";

    fin.open(path);
    int **arr;
    arr = new int*[count];
    for (int i = 0; i < count; i++)
        arr[i] = new int[count];

    for (int i = 0; i < count; i++)
    {
        for (int j = 0; j < count; j++)
            arr[i][j] = 0;
    }
    for (int i = 0; i < count; i++)
    {
        for (int j = i + 1; j < count; j++)
        {
            getline(fin, str);
            arr[i][j] = atoi(str.c_str());
            arr[j][i] = atoi(str.c_str());
        }
    }
    fin.close();
    return arr;
}
bool comp(int* arr, int count)
{
    int* mas = new int[count];
```

```
    mas[0] = 1;

    for (int i = 0; i < count - 1; i++)
    {
        mas[i + 1] = count - i;
    }

    for (int i = 0; i < count; i++)
    {
        if (mas[i] != arr[i])
        {
            return true;
        }
        else
        {
            continue;
        }
    }
    return false;
}
bool povtor(int* mas, int size)
{
    bool k = true;

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (mas[i] == mas[j] && i != j)
            {
                return false;
            }
        }
    }

    return true;
}
int way(int** mat, int* arr)
{
    int count = 0;

    for (int i = 0; i < 7; i++)
    {
        count += mat[arr[i]-1][arr[i+1]-1];
    }

    count += mat[arr[7]-1][arr[0]-1];
    return count;
}

int main() {
    int const count = 8;
    int **arr;
    arr = input();
    int var = count - 1;
    bool k = true;
    int *mas= new int [count];

    int* minmas = new int [9];
    int min = 1000;
    int leng=0;

    for (int i = 0; i < count; i++)
    {
```

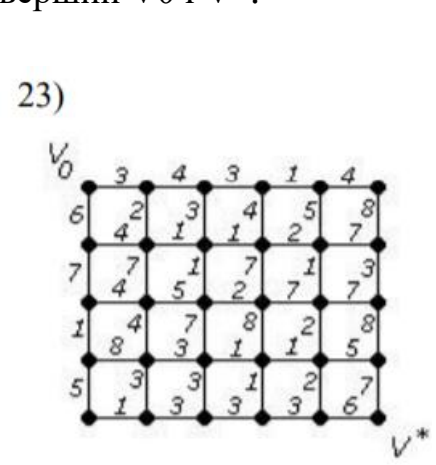
```
        mas[i] = 1;
        minmas[i] = 1;
    }

    while(comp(mas, count))
    {
        while (mas[var] != count)
        {
            mas[var]++;

            if (povtor(mas, count))
            {
                leng= way(arr, mas);

                if (leng < min)
                {
                    min = leng;
                    for (int v = 0; v < count; v++)
                    {
                        minmas[v] = mas[v];
                    }
                    minmas[count] = minmas[0];
                }
            }
        }
        while (mas[var] == count)
        {
            mas[var] = 2;
            var--;
        }
        mas[var]++;

        if (povtor(mas, count))
        {
            leng = way(arr, mas);

            if (leng < min)
            {
                min = leng;
                for (int v = 0; v < count; v++)
                {
                    minmas[v] = mas[v];
                }
                minmas[count] = minmas[0];
            }

        }
        var = count - 1;
    }

    cout << "Way: " << endl;
    for (int i = 0; i <= count; i++)
    {
        if (i != 0)
        {
            cout << "-> ";
        }
        cout << minmas[i] << " ";
    }
    cout << endl << "Leng: " << min;
    cout << endl;

    return 0;
}
```

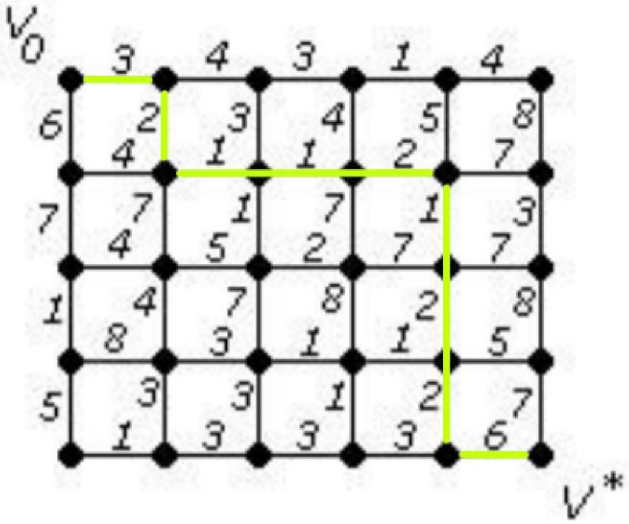
Виведення:

```
Way:
1 -> 2 -> 6 -> 8 -> 7 -> 3 -> 4 -> 5 -> 1
Leng: 17
```

Завдання № 7

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





Алгоритм Дейкстри для знаходження найкоротшого шляху у графі між парою вершин V0 і V*.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <locale>

using namespace std;

int main()
{
    ifstream fin("MyFile.txt");
    int vershina, rebra;
    fin >> vershina >> rebra;
    const int SIZE = 30;
    int matrix[SIZE][SIZE]; // матриця зв'язків
    int distance[SIZE]; // мінімальна відстань
    int visited[SIZE];  // відвідання вершин
    int dis, top, min;
    int begin_index = 0;


    for (int i = 0; i < vershina; i++)     // Ініціалізація матриці зв'язків
    {
        distance[i] = 99999;
        visited[i] = 0;
        for (int j = 0; j < vershina; j++)
        {
            matrix[i][j] = 0;
            matrix[j][i] = 0;
        }
    }
    for (int i = 0; i < rebra; i++) {
        int v1, v2, dis;
        fin >> v1 >> v2 >> dis;
```

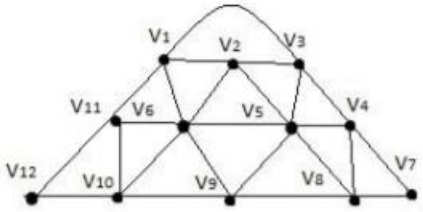
```
        matrix[v1 - 1][v2 - 1] = dis;
        matrix[v2 - 1][v1 - 1] = dis;
    }
    distance[0] = 0;
    do {
        top = 99999;
        min = 99999;
        for (int i = 0; i < vershina; i++)
        {
            if (visited[i] == 0 && distance[i] < min)
            {
                min = distance[i];
                top = i;
            }

        }
        if (top != 99999)
        {
            for (int i = 0; i < vershina; i++)
            {
                if (matrix[top][i] > 0)
                {
                    dis = min + matrix[top][i];
                    if (dis < distance[i])
                    {
                        distance[i] = dis;
                    }
                }
            }
            visited[top] = 1;
        }
    } while (top < 99999);

    int end = vershina - 1;
    int waga = distance[end];
    int way[30];
    way[0] = vershina - 1;
    int k = 1;
    while (end != 0)
    {
        for (int i = 0; i < vershina; i++)
        {
            if (matrix[end][i] > 0)
            {
                if (distance[i] == waga - matrix[end][i])
                {
                    waga = distance[i];
                    way[k] = i;
                    end = i;
                    k++;
                }
            }
        }
    }
    cout << "Smallest path from V0 to V29: ";
    for (int i = k; i > 0; i--)
    {
        if (i - 1 > 0)
            cout << way[i - 1] << " -> ";
        else
            cout << way[i - 1];
    }
    cout << "\nDistanse length: " << distance[vershina - 1] << endl;
}
```

Виведення:

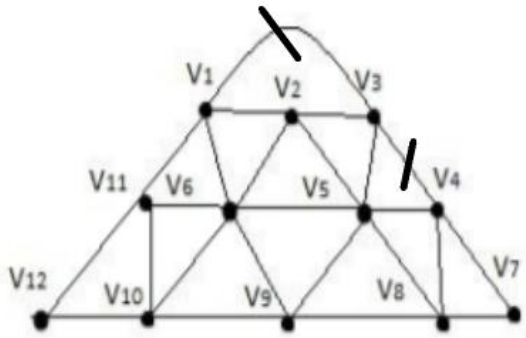
```
Smallest path from V0 to V29: 0 -> 1 -> 7 -> 8 -> 9 -> 10 -> 16 -> 22 -> 28 -> 29
Distanse length: 20
```

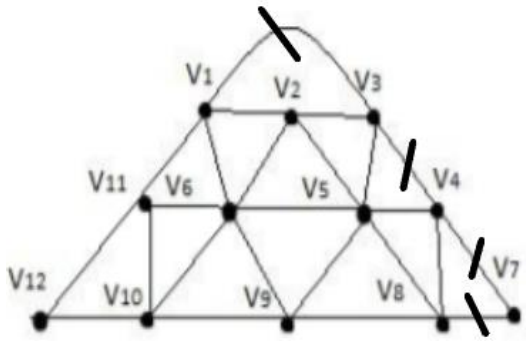
Завдання №8

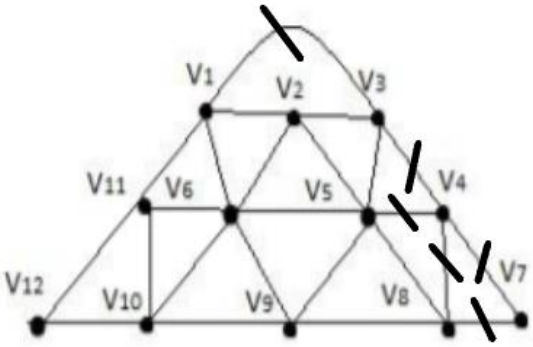
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.

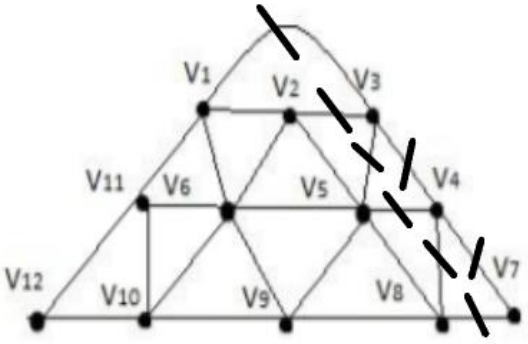


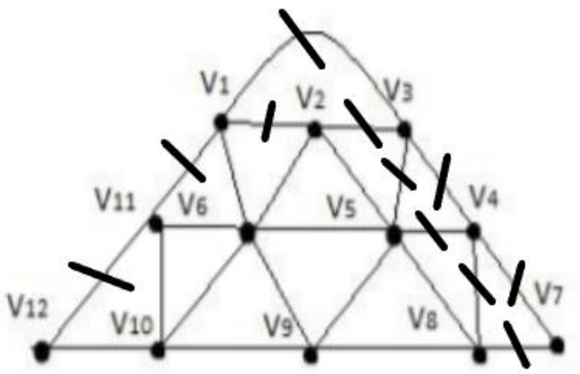
а) Алгоритм Флері

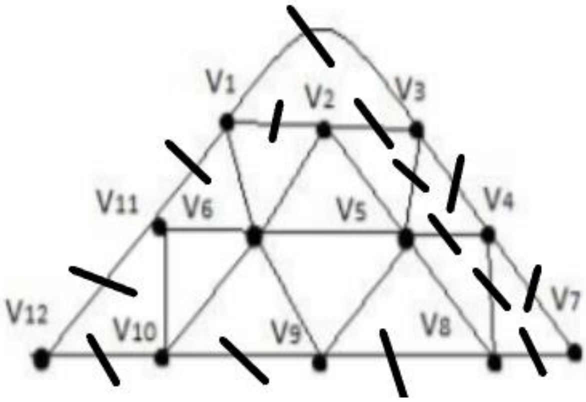


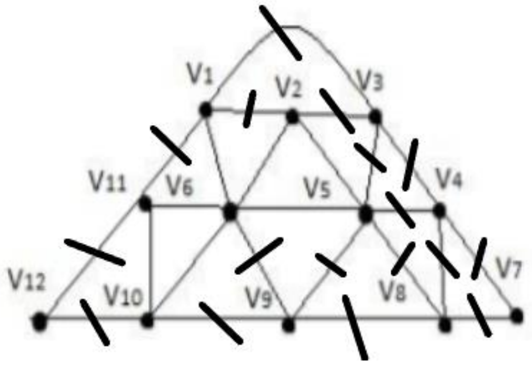


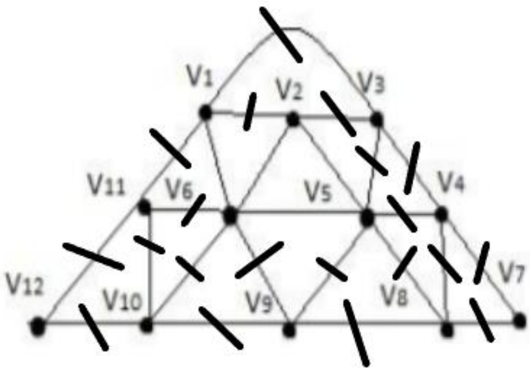


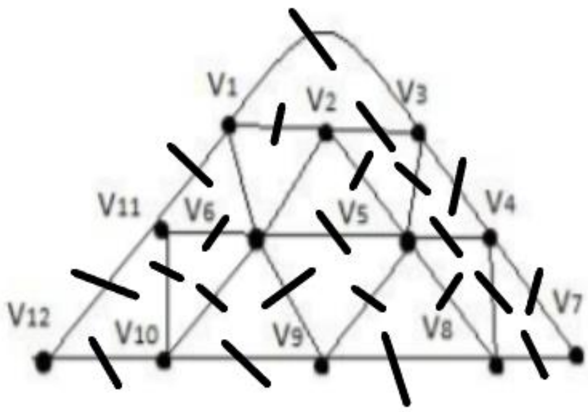


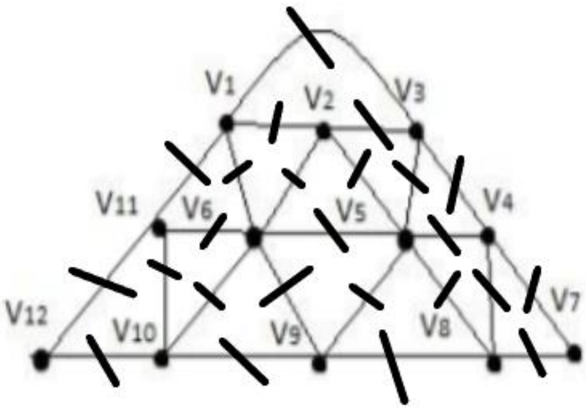












Алгоритм Флері для знаходження ейлерового циклу в ейлеровому графі:

```
#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(12);
    g1.addEdge(0, 10);
    g1.addEdge(0, 11);
    g1.addEdge(1, 11);
    g1.addEdge(1, 6);
    g1.addEdge(1, 2);
    g1.addEdge(1, 3);
    g1.addEdge(2, 6);
    g1.addEdge(2, 3);
    g1.addEdge(2, 5);
    g1.addEdge(3, 5);
    g1.addEdge(3, 4);
    g1.addEdge(4, 5);
    g1.addEdge(4, 7);
    g1.addEdge(4, 8);
    g1.addEdge(5, 8);
    g1.addEdge(5, 6);
    g1.addEdge(5, 9);
    g1.addEdge(6, 10);
    g1.addEdge(6, 9);
    g1.addEdge(6, 11);
    g1.addEdge(7, 8);
    g1.addEdge(8, 9);
    g1.addEdge(9, 10);
    g1.addEdge(10, 11);
    g1.printEulerTour();
    return 0;
}
```

Виведення:

```
0-10 10-6 6-1 1-11 11-6 6-2 2-1 1-3 3-2 2-5 5-3 3-4 4-5 5-8 8-4 4-7 7-8 8-9 9-5 5-6 6-9 9-10 10-11 11-0
```

б) Алгоритм елементарних циклів.

Цикли графа:

12->11->1->3->4->7->8->9->10->12

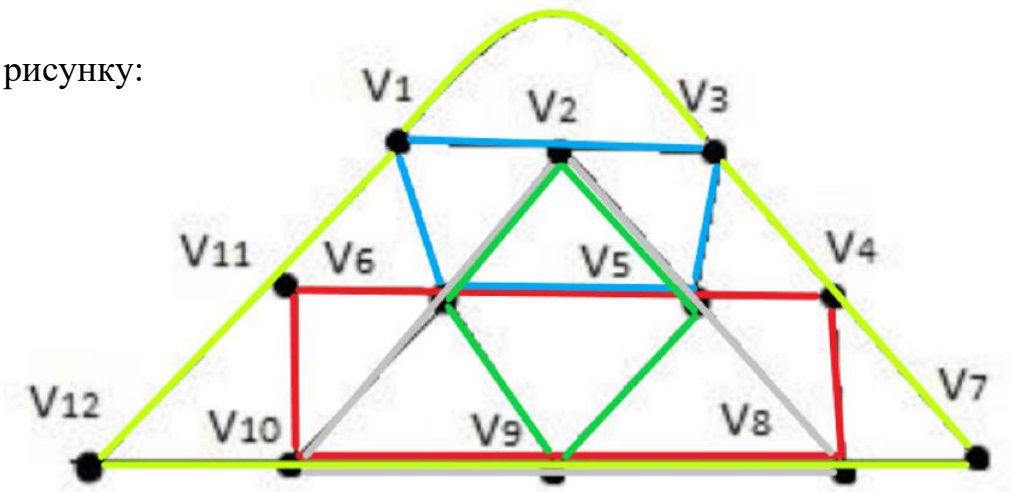
1->2->3->5->6->1

11->6->5->4->8->9->10->6

10->6->2->5->8->9->10

2->5->9->6->2

Зобразимо їх на рисунку:



Алгоритм елементарних циклів для знаходження ейлерового циклу в ейлеровому графі:

```cpp
#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();
        }
    }
}
```

Виведення:

```
1 6 2 5 3 2 1 3 4 8 5 9 6 5 4 7 8 9 10 11 6 10 12 11 1
```

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

23. $(x \vee \bar{y})(\bar{y} \vee \bar{z})$ =

$=\big((x \vee \bar{y}) \wedge \bar{y}\big) \vee \big((x \vee \bar{y}) \wedge \bar{z}\big) = \big((x \wedge \bar{y}) \vee (\bar{y} \wedge \bar{y})\big) \vee \big((x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z})\big) =$

$= (x \wedge \bar{y}) \vee (\bar{y} \wedge \bar{y}) \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z}) = (x \wedge \bar{y}) \vee (\bar{y} \wedge \bar{y}) \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z}) =$

$= (x \wedge \bar{y}) \vee \bar{y} \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z}) = \bar{y} \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z}) = \bar{y} \vee (x \wedge \bar{z})$

$\bar{y} \vee (x \wedge \bar{z})$