МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота

з дисципліни
«Дискретна математика»

Виконав:
студент групи КН-113
Костів Богдан

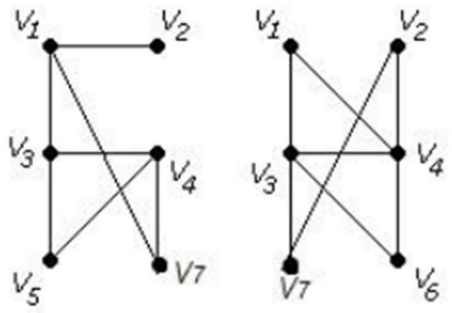
Викладач:
Мельникова Н.І.

Львів – 2019 р.

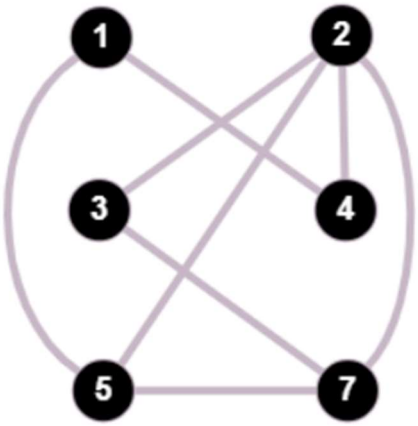
ІНДИВІДУАЛЬНІ ЗАВДАННЯ
Варіант№24
Завдання № 1

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.

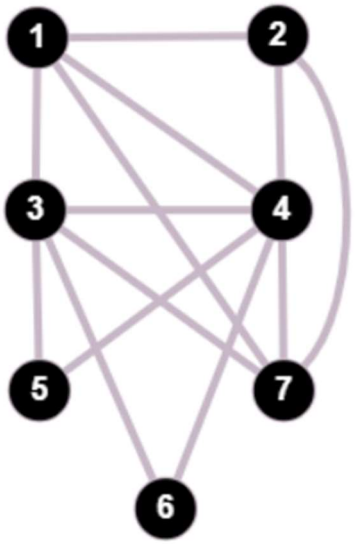
24)



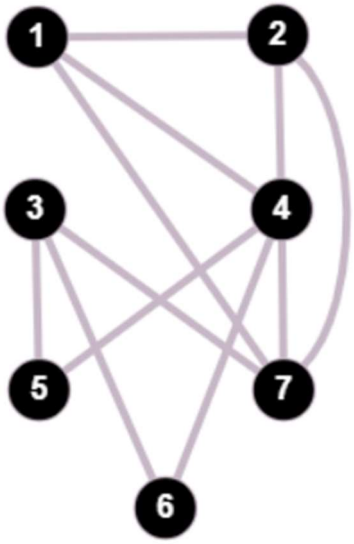
1) знайти доповнення до першого графу



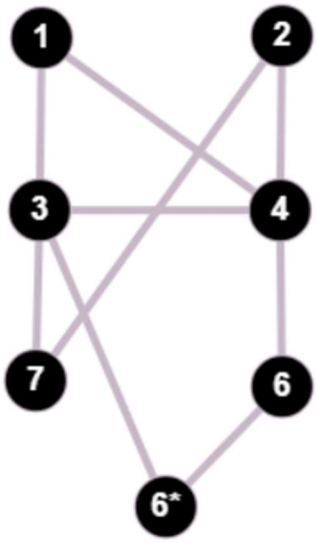
2) об’єднання графів



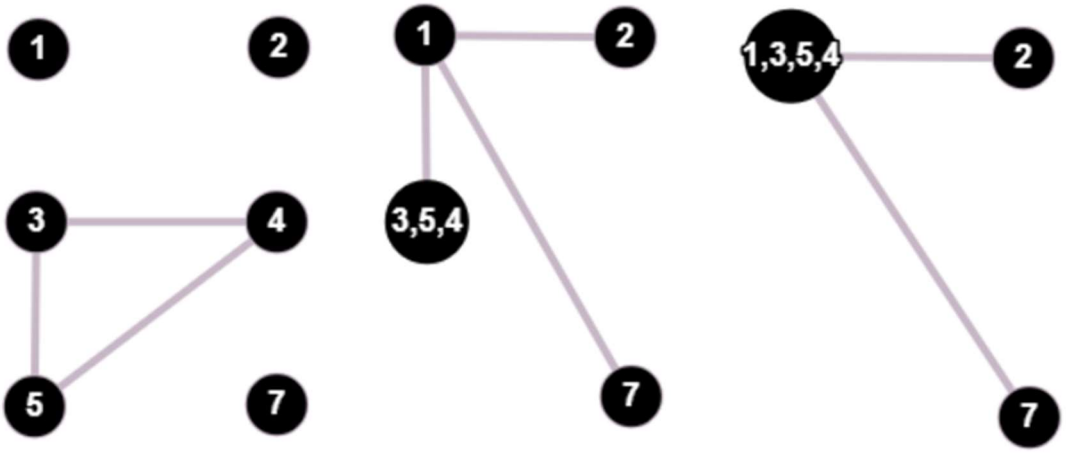
3) кільцеву сумму G1 та G2



4) розмножити вершину у другому графі

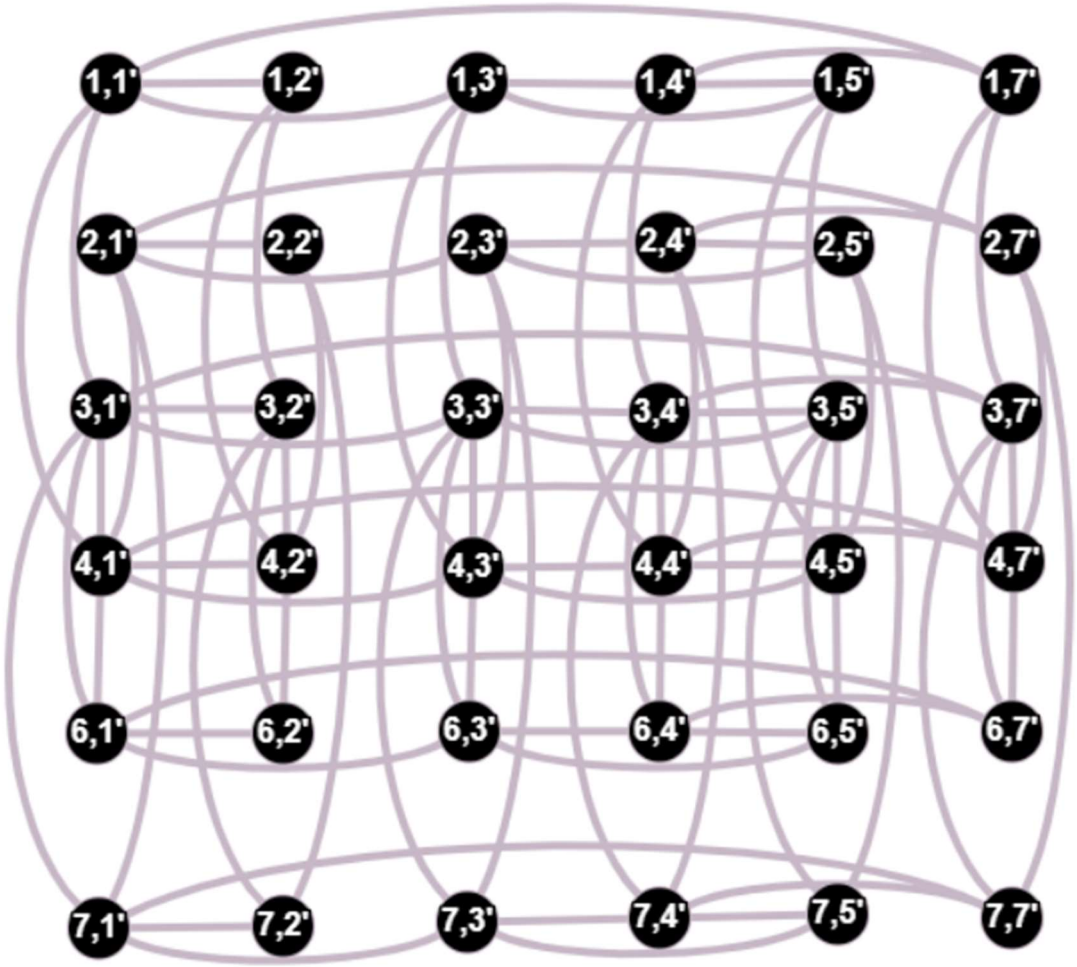


5) виділити підграф А - що скадається з 3-х вершин в G1



Підграф А Стягнення в G1

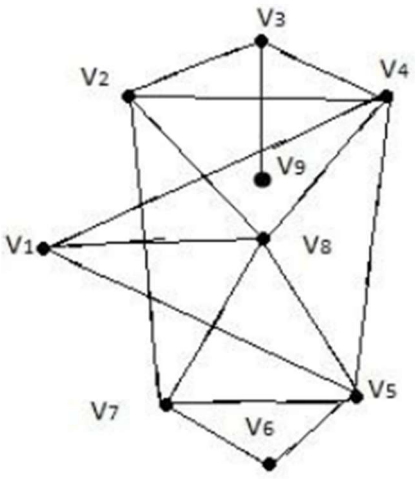
6) добуток графів.



Завдання № 2

Скласти таблицю суміжності для неорграфа.

24)



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

Порожні клітинки заповнюються нулями.

Завдання № 3

Для графа з другого завдання знайти діаметр.

d = 4 (V6-V7-V2-V3-V9)

Завдання № 4

Для графа з другого завдання виконати обхід дерева вшир

Вершина	BFS номер	Вміст черги
V1	1	V1
V4	2	V1V4
V8	3	V1V4V8
V5	4	V1V4V8V5
_	_	V4V8V5
V3	5	V4V8V5V3
V2	6	V4V8V5V3V2
_	_	V8V5V3V2
V7	7	V8V5V3V2V7
_	_	V5V3V2V7
V6	8	V5V3V2V7V6
_	_	V3V2V7V6
V9	9	V3V2V7V6V9
_	_	V2V7V6V9
_	_	V7V6V9
_	_	V6V9
_	_	V9
_	_	Ø

Також зробимо це програмно

```
//#include "pch.h"
#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 (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 << "Введіть першу вершину, інцидентну ребру №" << i + 1 << ": ";
        reb[i].v1 = correct(1, m);
        cout << "Введіть другу вершину, інцидентну ребру №" << i + 1 << ": ";;
        reb[i].v2 = correct(1, m);
        cout << endl;
    }
}

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

    cout << "Введіть кількість ребер у графі: ";
    n = correct(1, 1000);
    cout << "Введіть кількість вершин у графі: ";
    m = correct(1, 1000);
    cout << endl;

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

```
	vershina* v = new vershina[m];

	input(reb, n, m);

	cout << "З якої вершини почати обхід? ";
	begin = correct(1, m);

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

	cout << "Якщо ви хочете зробити обхід вглиб натисніть 1, обхід вшир - натисніть 2: ";
	p = correct(1, 2);

	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 << "Стек пустий" << endl;
		break;
```

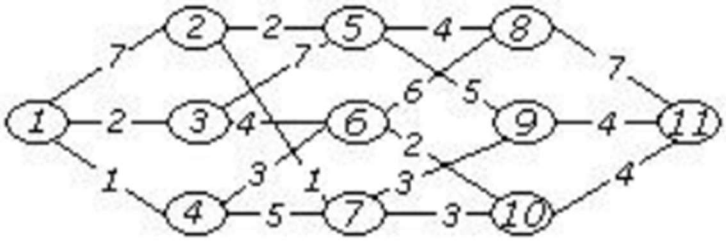
```
	}
	case 2:
	{
		while (head != m)
		{
			for (int i = head; i < n; i++)
			{

				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 << "Черга порожня" << endl;
		break;
	}
	}
	return 0;
}
```

```
Введіть кількість ребер у графі: 16
Введіть кількість вершин у графі: 9

1 4
1 4 5
1 4 5 8
4 5 8
4 5 8 3
4 5 8 3 2
5 8 3 2
5 8 3 2 7
5 8 3 2 7 6
8 3 2 7 6
3 2 7 6
3 2 7 6 9
2 7 6 9
7 6 9
6 9
9
Черга порожня
```

Завдання № 5

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

24)



Краскала:

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

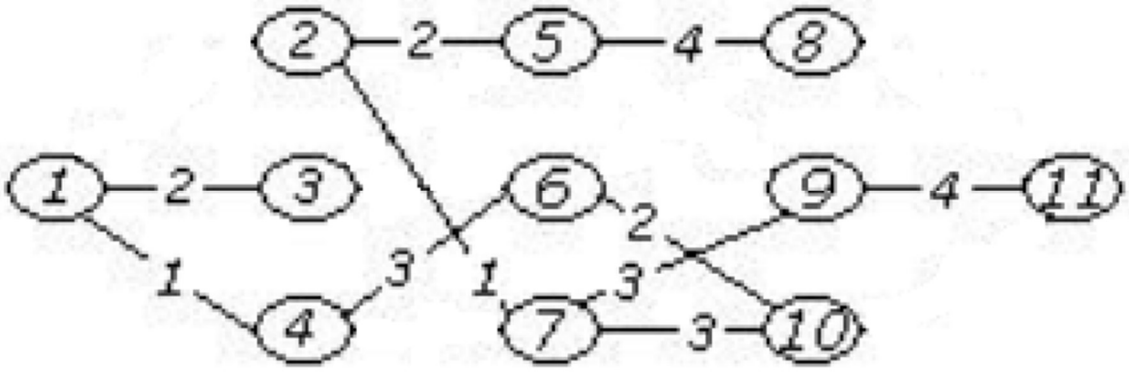
E = {(2,7),(1,4),(2,5),(6,10),(1,3),(4,6),(7,10),(7,9),(9,11),(5,8)}

Прима:

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

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

Результат:



Також зробимо це програмно

Краскала:

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
const int q = 11;
int MakeTrees(int n, int A[q][q]);
void RemoveRepeated(int n, int A[q][q]);
```

```
int AreInDifferentTrees(int n, int A[q][q], int first, int second);
void AddToTree(int n, int A[q][q],int first, int second);

int main()
{
   int A[11][11] =
      {0,7,2,1,0,0,0,0,0,0,0,
       7,0,0,0,2,0,1,0,0,0,0,
       2,0,0,0,7,4,0,0,0,0,0,
       1,0,0,0,0,3,5,0,0,0,0,
       0,2,7,0,0,0,0,4,5,0,0,
       0,0,4,3,0,0,0,6,0,2,0,
       0,1,0,5,0,0,0,0,3,3,0,
       0,0,0,0,4,6,0,0,0,0,7,
       0,0,0,0,5,0,3,0,0,0,4,
       0,0,0,0,0,2,3,0,0,0,4,
       0,0,0,0,0,0,0,7,4,4,0};
   RemoveRepeated(11, A);
   for (int i =1 ; i <= 7; i++)
   {
      std::cout<<"\nNodes with weight: "<< i<<": ";
      for (int j=1;j <=11;j++)
      {
         for (int k=1; k<=11; k++)
         {
            if (A[j-1][k-1] == i)
            {
               std::cout<<" "<<j<<"-"<<k;;
            }
         }
      }
   }
   std::cout<<"\n";
   //Check sorted nodes and add to the tree

   int B[11][11];
   MakeTrees(11,B);
   std::cout<<"\n\nNew Tree: ";//weight 7 is max weight
   for (int i = 1; i<=7;i++)
   {
      //first node´
      for (int j = 1; j<=11; j++)
      {
         //second node
         for (int k = 1 ; k<=11; k++)
         {
            if (A[j-1][k-1] == i && AreInDifferentTrees(11, B, j ,k))
            {
               AddToTree(11, B,j,k);
               std::cout<<" "<<j <<"-"<<k;
            }
         }
      }
```

```
    }
    return 0;
}

int MakeTrees(int n, int A[q][q])
{
    for (int i=0; i<n; i++)
    {
        for (int j = 0; j<n; j++)
        {
            A[i][j] = 0;
        }
    }
    for (int i = 0; i<n; i++)
    {
        A[i][i] = i+1;
    }
    return A[n][n];
}

void RemoveRepeated(int n, int A[q][q])
{
    for(int i = 0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
        {
            if (j < i)
            {
                A[i][j] = 0;
            }
        }
    }
}

int AreInDifferentTrees(int n, int A[q][q], int first, int second)
{
    int temp1, temp2;
    //Line
    for (int i = 0; i < n; i++)
    {
        temp1 = 0;
        temp2 = 0;
        //first element
        for (int j = 0; j < n; j++)
        {
            if (A[i][j] == first)
            {
                temp1 = 1;
            }
        }
        //second element
        for (int k = 0; k < n; k++)
        {
```

```
            if (A[i][k] == second)
            {
               temp2 = 1;
            }
         }
         if (temp1 && temp2)
         {
            return 0;
         }
      }
      return 1;
}

void AddToTree(int n, int A[q][q], int first, int second)
{
   int scndLine;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         if (A[i][j] == second)
         {
            scndLine = i;
         }
      }
   }
   for (int i = 0; i < n; i++)
   {
      for ( int j = 0; j < n; j++)
      {
         if (A[i][j] == first)
         {
            for (int k = 0; k < n; k++)
            {
               if (A[scndLine][k])
               {
                  A[i][k] = A[scndLine][k];
                  A[scndLine][k] = 0;
               }
            }
         }
      }
   }
}
```

```
Nodes with weight: 1:  1-4 2-7
Nodes with weight: 2:  1-3 2-5 6-10
Nodes with weight: 3:  4-6 7-9 7-10
Nodes with weight: 4:  3-6 5-8 9-11 10-11
Nodes with weight: 5:  4-7 5-9
Nodes with weight: 6:  6-8
Nodes with weight: 7:  1-2 3-5 8-11


New Tree:  1-4 2-7 1-3 2-5 6-10 4-6 7-9 7-10 5-8 9-11
```

Прима:

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

using namespace std;

void output(int size, int** adjacencyarr) {
	cout << "  ";
	for (int i = 0; i < size; i++) { cout << setw(3) << i + 1; }
	for (int i = 0; i < size; i++) {
		cout << endl << setw(3) << i + 1;
		for (int j = 0; j < i; j++) {
			cout << setw(3) << adjacencyarr[j][i];
		}
		cout << "  -";
		for (int j = i + 1; j < size; j++) {
			cout << setw(3) << adjacencyarr[i][j];
		}
	}
}

void main()
{
	int size;
	cout << "Enter amount of vertices ";
	cin >> size;
	int** adjacencyarr = new int* [size];
	bool* vertex = new bool[size];
	cout << "Enter adjacency matrix \n";
	cout << "  ";
	for (int i = 0; i < size; i++) { cout << setw(3) << i + 1; adjacencyarr[i] = new int[size];
vertex[i] = 0; }
	vertex[0] = 1;
	cout << endl;
	for (int i = 0; i < size; i++) {
		cout << setw(2) << i + 1;
		for (int j = 0; j < i; j++) { cout << setw(3) << adjacencyarr[j][i]; }
		cout << "  -";
		for (int j = i + 1; j < size; j++) { cin >> adjacencyarr[i][j]; }
	}
	cout << endl;
	int min, minI, minJ;
	for (int n = 0; n < size - 1; n++) {
		min = 100, minI = 0, minJ = 0;
		for (int i = 0; i < size; i++) {
			for (int j = i + 1; j < size; j++) {
				if (adjacencyarr[i][j] < min && adjacencyarr[i][j] != 0 && vertex[i] !=
vertex[j]) {
					min = adjacencyarr[i][j];
					minI = i, minJ = j;
				}
			}
		}
		vertex[minI] = 1; vertex[minJ] = 1;
		cout << n + 1 << ")conected " << minI + 1 << "-" << minJ + 1 << endl;
	}
}
```

Результат після введення користувачем всіх необхідних данних:

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

Завдання № 6

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

24)

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

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

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

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

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

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

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

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

Довжина цього шляху:15

Обійдемо ще двічі:

1-8-4-2-5-6-7-3 , довжина: 13

2-4-8-6-5-1-7-3 , довжина: 15

Також зробимо це програмно:

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

using namespace std;
ifstream fin;
string path = "File1.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))
		{
```

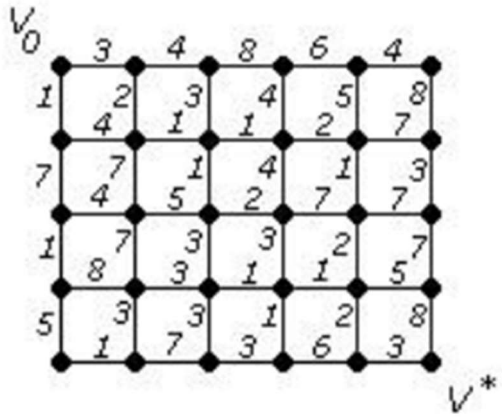
```
                leng = way(arr, mas);

                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;
}
```

```
4 -> 2 -> 5 -> 1 -> 8 -> 6 -> 7 -> 3 -> 4
4 -> 3 -> 7 -> 6 -> 8 -> 1 -> 5 -> 2 -> 4
4 -> 8 -> 1 -> 5 -> 6 -> 7 -> 3 -> 2 -> 4
5 -> 1 -> 8 -> 4 -> 2 -> 3 -> 7 -> 6 -> 5
5 -> 1 -> 8 -> 6 -> 7 -> 3 -> 4 -> 2 -> 5
5 -> 2 -> 4 -> 3 -> 7 -> 6 -> 8 -> 1 -> 5
5 -> 6 -> 7 -> 3 -> 2 -> 4 -> 8 -> 1 -> 5
6 -> 5 -> 1 -> 8 -> 4 -> 2 -> 3 -> 7 -> 6
6 -> 7 -> 3 -> 2 -> 4 -> 8 -> 1 -> 5 -> 6
6 -> 7 -> 3 -> 4 -> 2 -> 5 -> 1 -> 8 -> 6
6 -> 8 -> 1 -> 5 -> 2 -> 4 -> 3 -> 7 -> 6
7 -> 3 -> 2 -> 4 -> 8 -> 1 -> 5 -> 6 -> 7
7 -> 3 -> 4 -> 2 -> 5 -> 1 -> 8 -> 6 -> 7
7 -> 6 -> 5 -> 1 -> 8 -> 4 -> 2 -> 3 -> 7
7 -> 6 -> 8 -> 1 -> 5 -> 2 -> 4 -> 3 -> 7
8 -> 1 -> 5 -> 2 -> 4 -> 3 -> 7 -> 6 -> 8
8 -> 1 -> 5 -> 6 -> 7 -> 3 -> 2 -> 4 -> 8
8 -> 4 -> 2 -> 3 -> 7 -> 6 -> 5 -> 1 -> 8
Minimal leng = 17
```

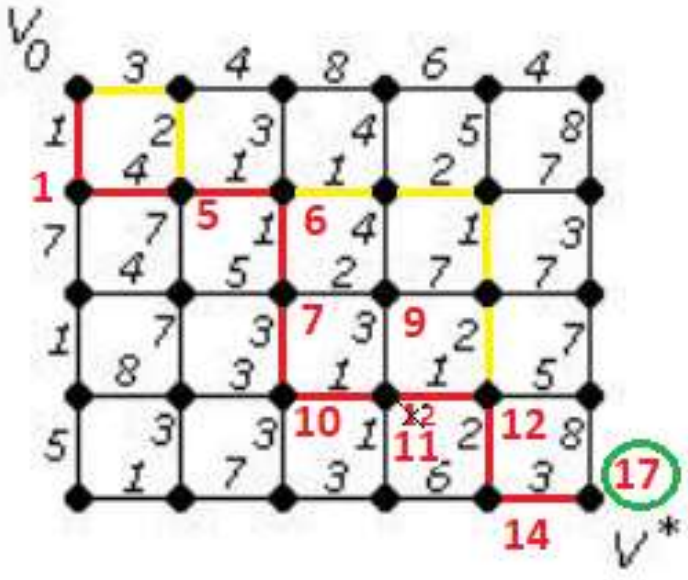
Завдання № 7

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

24)



У цьому графі є декілька шляхів:



Також зробимо це програмно

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

using namespace std;

int main()
{
	ifstream fin("Cheburek.txt");
	setlocale(LC_ALL, "Ukrainian");
	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];  // чи відвідані вершни
	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;
	}
	cout << "DIJKSTRA 
ALGORITHM\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n";

	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 << "\nSmallest 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 << "\n\nDistanse length: " << distance[vershina - 1] << endl;
```

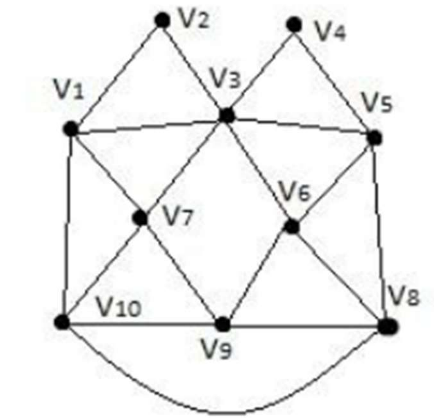
```
        cout <<
"\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n";
}
```

```
DIJKSTRA ALGORITHM
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


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

Distanse length: 17


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Завдання № 8

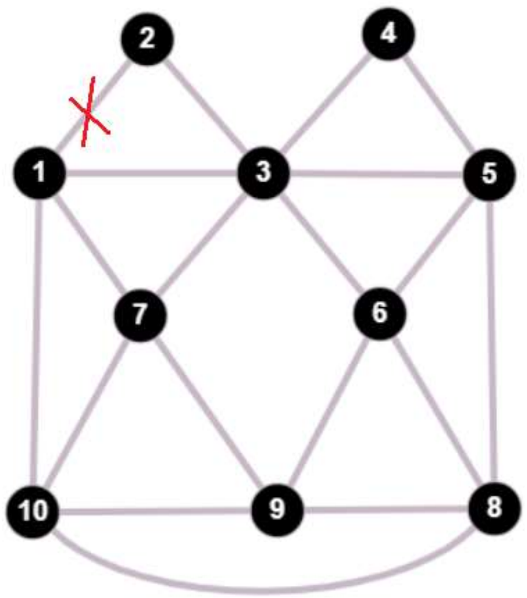
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.

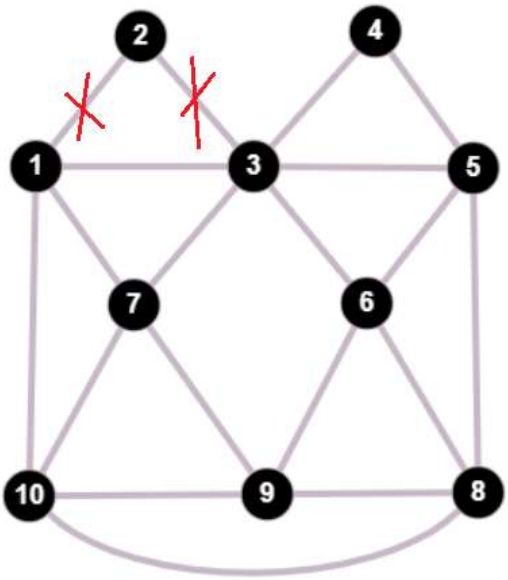
24)

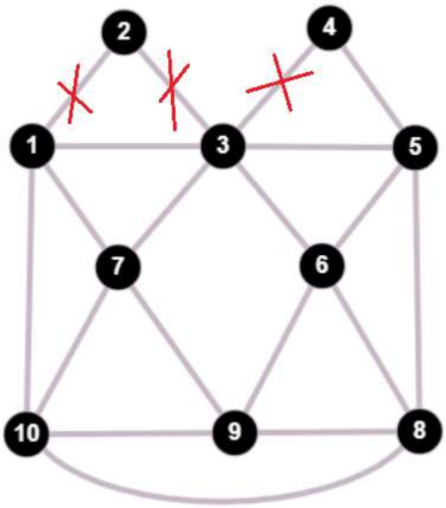


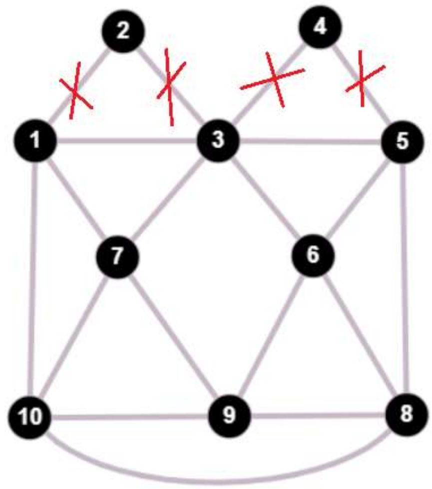
Флері: (Видалення ребер я зробив методом їхнього послідовного закреслення)

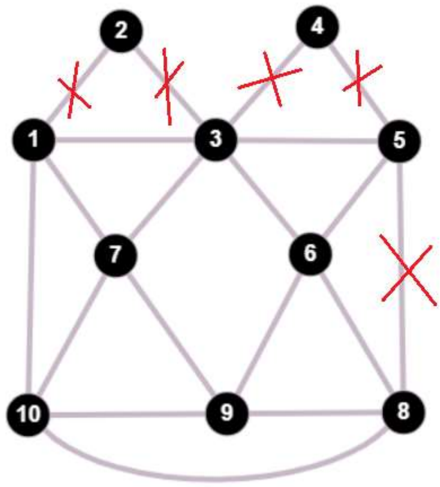
Починаємо обхід з вершини 1:

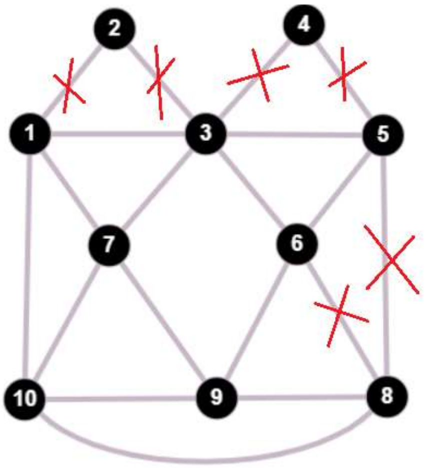


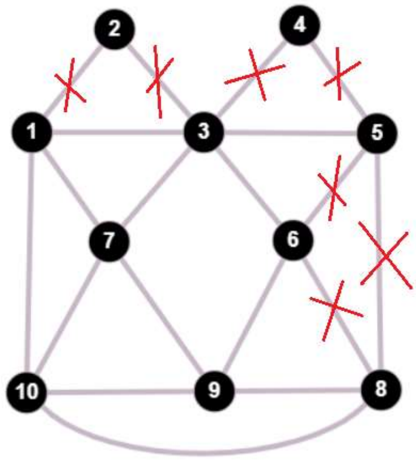


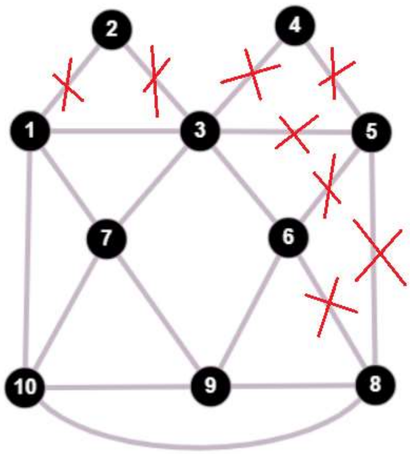

2
4
1
3
5
7
6
10
9
8

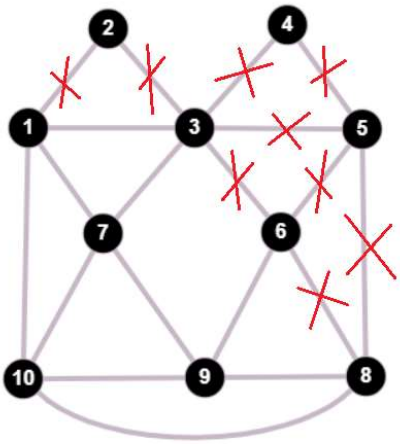

2
4
1
3
5
7
6
10
9
8

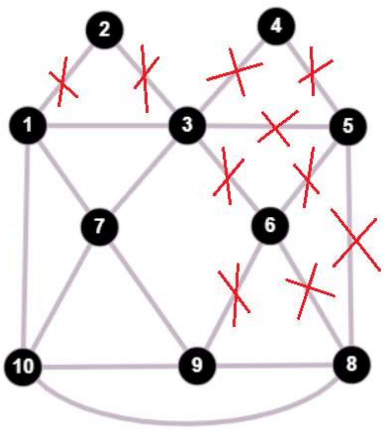

2
4
1
3
5
7
6
10
9
8

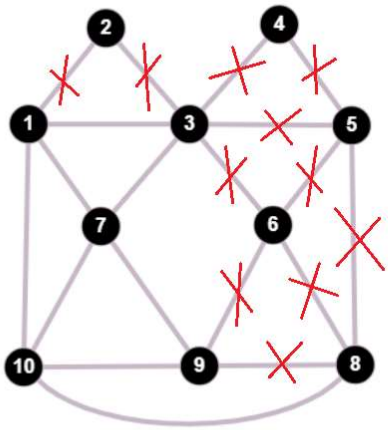

2
4
1
3
5
7
6
10
9
8

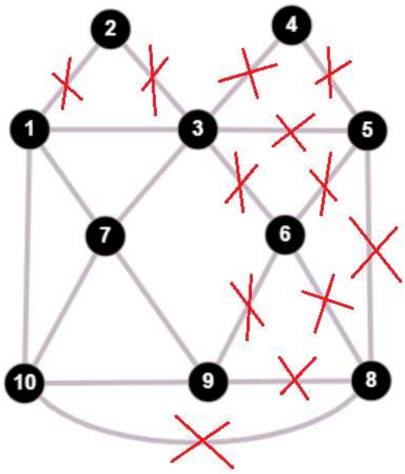

2
4
1
3
5
7
6
10
9
8

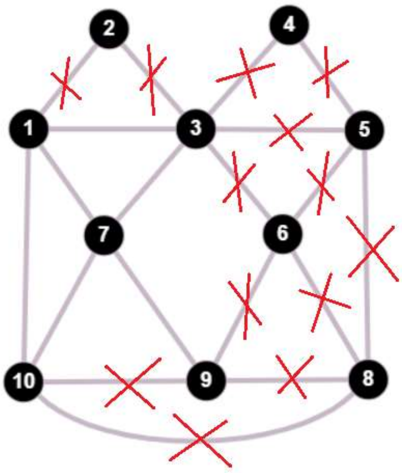

2
4
1
3
5
7
6
10
9
8

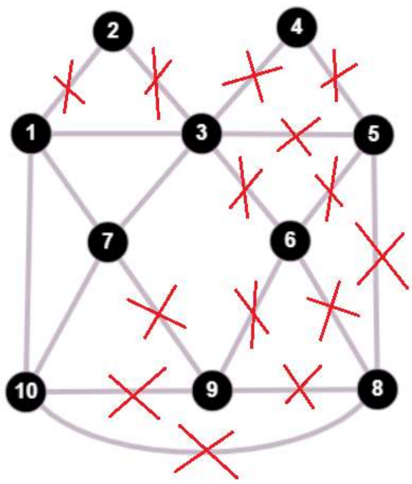

2
4
1
3
5
7
6
10
9
8

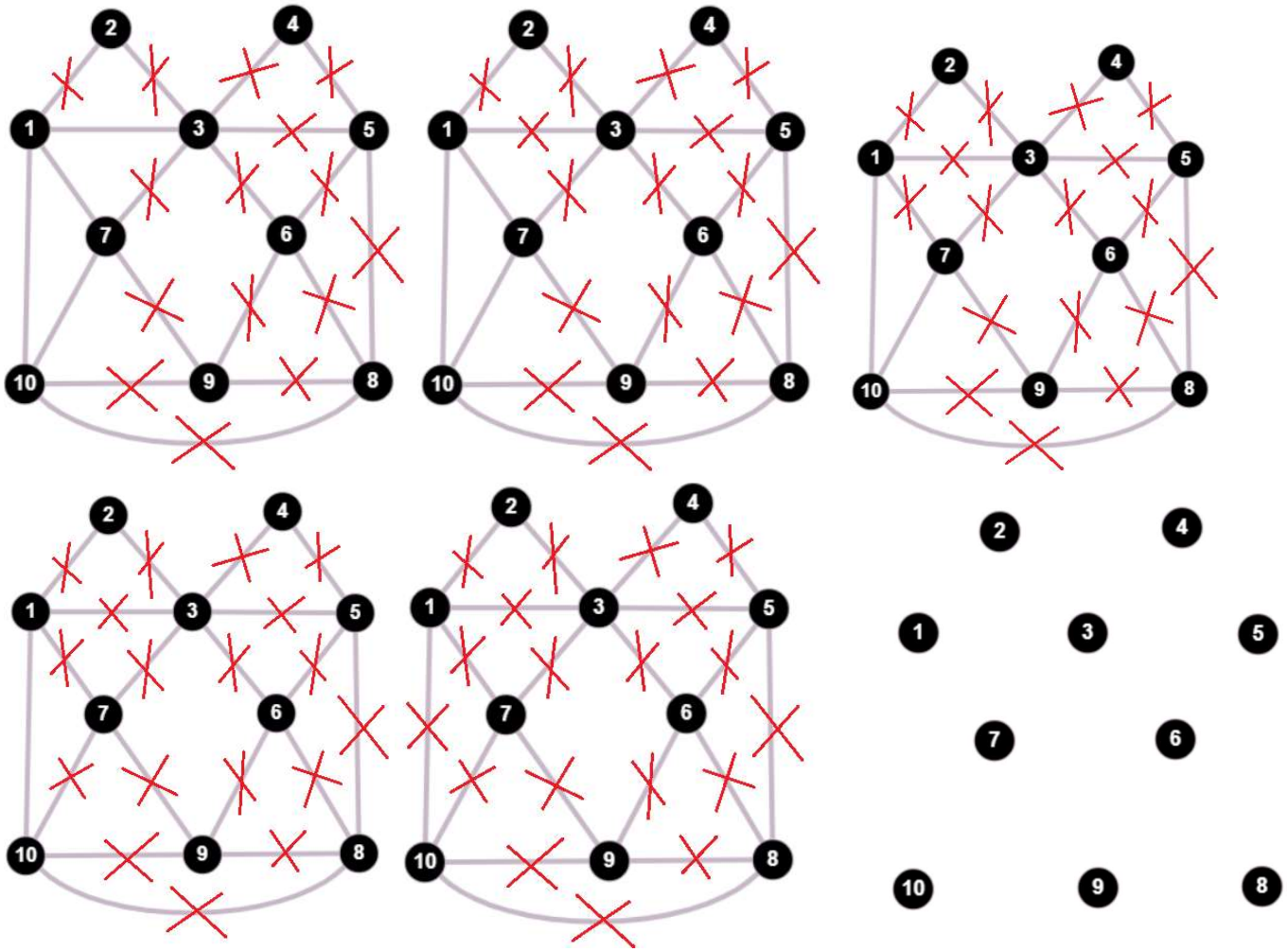

2
4
1
3
5
7
6
10
9
8


2
4
1
3
5
7
6
10
9
8


2
4
1
3
5
7
6
10
9
8


2
4
1
3
5
7
6
10
9
8


2
4
1
3
5
7
6
10
9
8



Також зробимо це програмно:

```
// A C++ program print Eulerian Trail in a given Eulerian or Semi-Eulerian Graph
#include <iostream>
#include <string.h>
#include <algorithm>
#include <list>
using namespace std;

// A class that represents an undirected graph
class Graph
{
int V; // No. of vertices
list<int> *adj; // A dynamic array of adjacency lists
public:
        // Constructor and destructor
Graph(int V) { this->V = V; adj = new list<int>[V]; }
~Graph()        { delete [] adj; }

// functions to add and remove edge
void addEdge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }
void rmvEdge(int u, int v);

// Methods to print Eulerian tour
void printEulerTour();
void printEulerUtil(int s);

// This function returns count of vertices reachable from v. It does DFS
int DFSCount(int v, bool visited[]);
```

```
// Utility function to check if edge u-v is a valid next edge in
// Eulerian trail or circuit
bool isValidNextEdge(int u, int v);
};

/* The main function that print Eulerian Trail. It first finds an odd
degree vertex (if there is any) and then calls printEulerUtil()
to print the path */
void Graph::printEulerTour()
{
// Find a vertex with odd degree
int u = 0;
for (int i = 0; i < V; i++)
        if (adj[i].size() & 1)
                { u = i; break; }

// Print tour starting from oddv
printEulerUtil(u);
cout << endl;
}

// Print Euler tour starting from vertex u
void Graph::printEulerUtil(int u)
{
// Recur for all the vertices adjacent to this vertex
list<int>::iterator i;
for (i = adj[u].begin(); i != adj[u].end(); ++i)
{
        int v = *i;

        // If edge u-v is not removed and it's a a valid next edge
        if (v != -1 && isValidNextEdge(u, v))
        {
                cout << u << "-" << v << " ";
                rmvEdge(u, v);
                printEulerUtil(v);
        }
}
}

// The function to check if edge u-v can be considered as next edge in
// Euler Tout
bool Graph::isValidNextEdge(int u, int v)
{
// The edge u-v is valid in one of the following two cases:

// 1) If v is the only adjacent vertex of u
int count = 0; // To store count of adjacent vertices
list<int>::iterator i;
for (i = adj[u].begin(); i != adj[u].end(); ++i)
        if (*i != -1)
                count++;
```

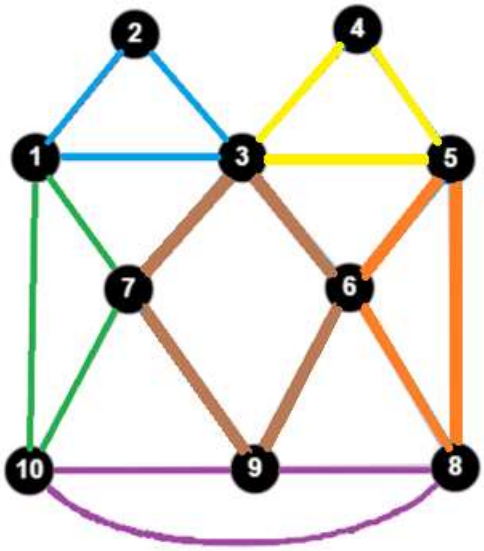
```
if (count == 1)
        return true;


// 2) If there are multiple adjacents, then u-v is not a bridge
// Do following steps to check if u-v is a bridge

// 2.a) count of vertices reachable from u
bool visited[V];
memset(visited, false, V);
int count1 = DFSCount(u, visited);

// 2.b) Remove edge (u, v) and after removing the edge, count
// vertices reachable from u
rmvEdge(u, v);
memset(visited, false, V);
int count2 = DFSCount(u, visited);

// 2.c) Add the edge back to the graph
addEdge(u, v);

// 2.d) If count1 is greater, then edge (u, v) is a bridge
return (count1 > count2)? false: true;
}

// This function removes edge u-v from graph. It removes the edge by
// replacing adjcent vertex value with -1.
void Graph::rmvEdge(int u, int v)
{
// Find v in adjacency list of u and replace it with -1
list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);
*iv = -1;

// Find u in adjacency list of v and replace it with -1
list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);
*iu = -1;
}

// A DFS based function to count reachable vertices from v
int Graph::DFSCount(int v, bool visited[])
{
// Mark the current node as visited
visited[v] = true;
int count = 1;

// Recur for all vertices adjacent to this vertex
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;
}
```

```
// Driver program to test above function
int main()
{
// Let us first create and test graphs shown in above figure
Graph g1(10);
g1.addEdge(0, 1);
        g1.addEdge(0, 2);
        g1.addEdge(0, 6);
        g1.addEdge(0, 9);
        g1.addEdge(1, 2);
        g1.addEdge(2, 3);
        g1.addEdge(2, 4);
        g1.addEdge(2, 5);
        g1.addEdge(2, 6);
        g1.addEdge(3, 4);
        g1.addEdge(4, 5);
        g1.addEdge(4, 7);
        g1.addEdge(5, 7);
        g1.addEdge(5, 8);
        g1.addEdge(6, 8);
        g1.addEdge(6, 9);
        g1.addEdge(7, 8);
        g1.addEdge(7, 9);
        g1.addEdge(8, 9);
        g1.printEulerTour();


return 0;
}
```

Номери вершин є від 0 до 9.

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

Елементарні цикли:



Є 6 підграфів:
1,10,7,1 – зелений;
3,7,9,6,3 - коричневий;
9,10,8,9 – пурпуровий;
5,6,8,5 – оранжевий;
1,3,2,1 – голубий.
3,5,4,3 - жовтий

Також зробимо це програмно:

```
#include <iostream>
#include <vector>
#include <stack>
```

```cpp
#include <algorithm>
#include <list>
using namespace std;

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


int main()
{
	/** n is the number of vertices
	 *  a is the number od edges
	 *  deg is the degree of a given vertex
	 */

	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";		/**< no euler cycle exists (all degrees must be even) */

	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();




			}



		}
		/**< tail is the eulerian cycle */
		while (!tail.empty())
		{
			cout << tail.top() << ' ';
			tail.pop();
		}
```

```
    }


}
```

Результат виводу програми, після введення користувачем всіх необхідних данних:

```
1 10 7 1 3 7 9 10 8 9 6 3 5 6 8 5 4 3 2 1
C:\Users\PC\Desktop\Visualka   0w0\proga\Debug\proga.exe (process 18456) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

24. $\overline{x(y\bar{z} \vee x\bar{z})}$

x	y	z	$\bar{z}$	$y\bar{z}$	$x\bar{z}$	$y\bar{z} \vee x\bar{z}$	$x(y\bar{z} \vee x\bar{z})$	$\overline{x(y\bar{z} \vee x\bar{z})}$
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	0	1
0	1	1	0	0	0	0	0	1
1	0	0	1	0	1	1	1	0
1	0	1	0	0	0	0	0	1
1	1	0	1	1	1	1	1	0
1	1	1	0	0	0	0	0	1

ДДНФ: $\overline{xyz} \vee \overline{xy}z \vee \bar{x}y\bar{z} \vee \bar{x}yz \vee x\bar{y}z \vee xyz$

$\overline{xyz} \vee \overline{xy}z \vee \bar{x}y\bar{z} \vee \bar{x}yz \vee x\bar{y}z \vee xyz = \overline{xy}(\bar{z} \vee z) \vee \bar{x}y(\bar{z} \vee z) \vee xz(\bar{y} \vee y) =$

$= \overline{xy} \vee \bar{x}y \vee xz = \bar{x}(\bar{y} \vee y) \vee xz = \bar{x} \vee xz = \bar{x} \vee z$