МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота(21 варіант)

З дисципліни:

Дискретна математика

Виконав:

Студент групи КН-113

Черній Юрій Миколайович

Викладач:

Мельникова Н.І.

Завдання № 1

Виконати наступні операції над графами:

1) знайти доповнення до першого графу,

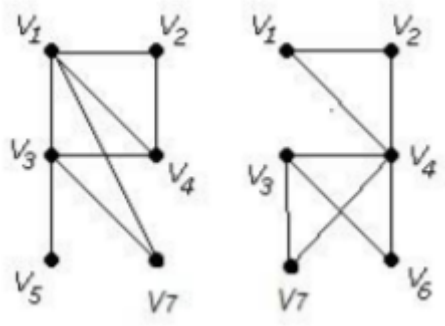
2) об’єднання графів,

3) кільцеву суму G1 та G2 (G1+G2),

4) розмножити вершину у другому графі,

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

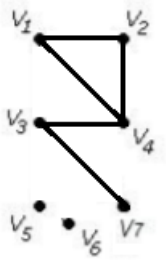
6) добуток графів.



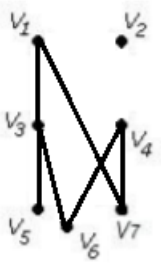
Розв’язання.

1)

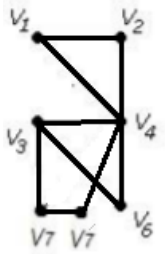
2)



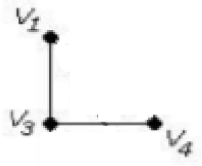
3)



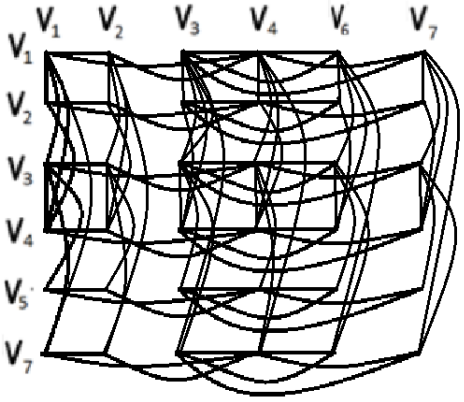
4)



5)

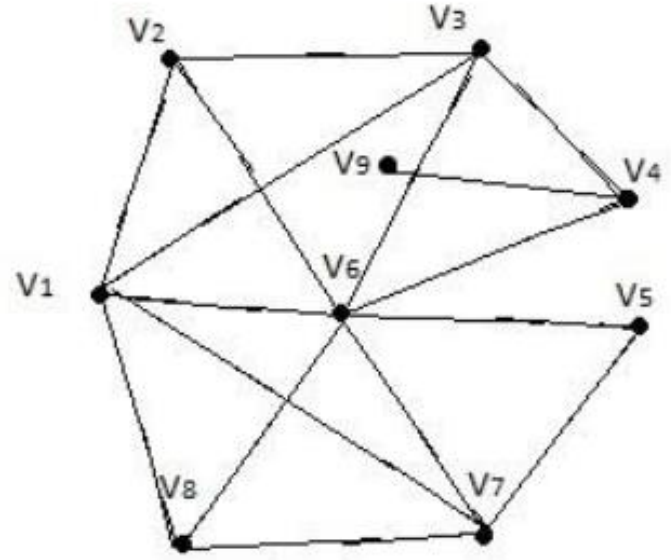


6)



Завдання № 2

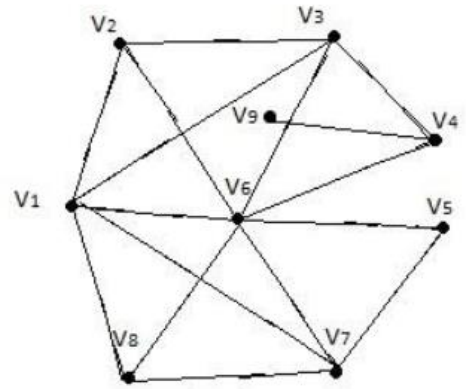
Скласти таблицю суміжності для графа.



Розв’язання.

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_1	0	1	1	0	0	0	1	1	0
V_2	1	0	1	0	0	1	0	0	0
V_3	1	1	0	1	0	1	0	0	0
V_4	0	0	1	0	0	1	0	0	1
V_5	0	0	0	0	0	1	1	0	0
V_6	0	1	1	1	1	0	1	1	0
V_7	1	0	0	0	1	1	0	1	0
V_8	1	0	0	0	0	1	1	0	0
V_9	0	0	0	1	0	0	0	0	0

Завдання № 3
Для графа з другого завдання знайти діаметр.

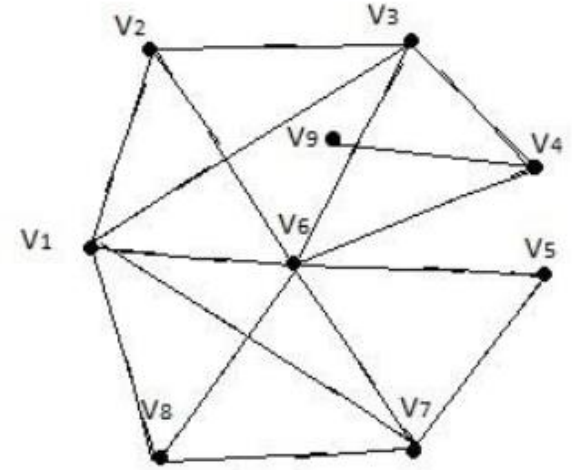


Розв’язання.

Діаметр даного графа дорівнює 3, оскільки цьому дорівнює максимальна відстань найкоротшого шлях між двома вершинами (у цьому випадку V_8 і V_9).

Завдання № 4
Для графа з другого завдання виконати обхід дерева вглиб (варіант закінчується на непарне число) або вшир (закінчується на парне число).

21 варіант - обхід дерева вглиб.



Розв’язання.

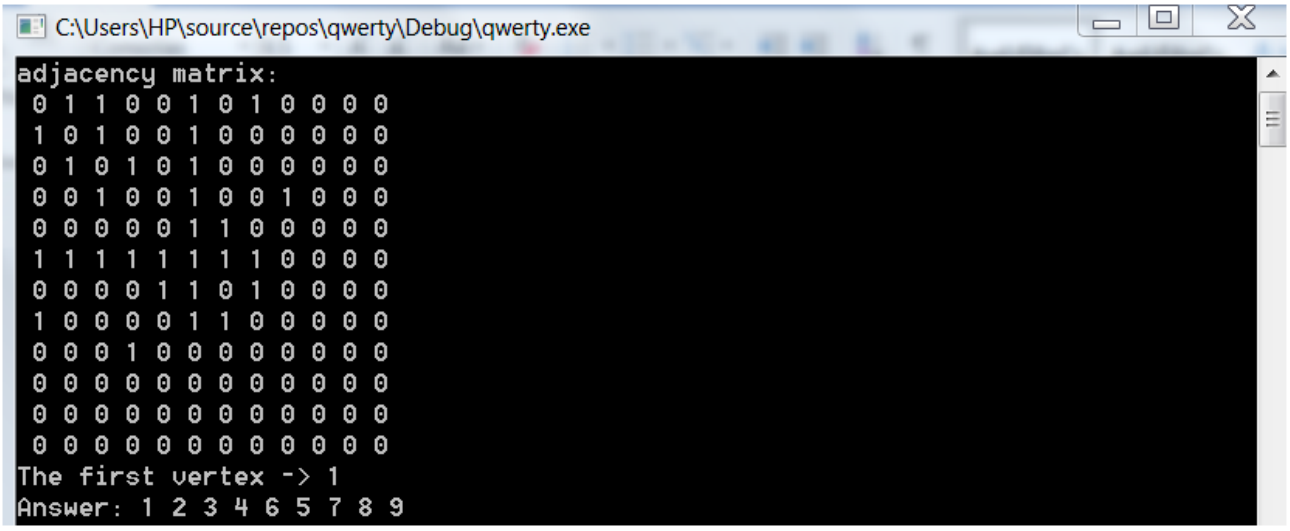
Вершина	DFS-номери	Вміст стеку
1	1	1
2	2	12
3	3	123
4	4	1234
6	5	12346
5	6	123465
7	7	1234657
8	8	12346578
-	-	1234657
-	-	123465
-	-	12346
-	-	1234
9	9	12349
-	-	1234
-	-	123
-	-	12
-	-	1
-	-	∅

Код програми:

```
#include <iostream>
#include <fstream>
using namespace std;
const int n = 12;
int i, j;
bool* visited = new bool[n];
int graph[n][n] =
{
{0,1,1,0,0,1,0,1,0},
{1,0,1,0,0,1,0,0,0},
{0,1,0,1,0,1,0,0,0},
{0,0,1,0,0,1,0,0,1},
{0,0,0,0,0,1,1,0,0},
{1,1,1,1,1,1,1,1,0},
{0,0,0,0,1,1,0,1,0},
{1,0,0,0,0,1,1,0,0},
{0,0,0,1,0,0,0,0,0},
};
void DFS(int st)
{
	int r;
	cout << st + 1 << " ";
	visited[st] = true;
	for (int r = 0; r <= n; r++)
		if ((graph[st][r] != 0) && (!visited[r]))
			DFS(r);


	}
int main()
{
	setlocale(LC_ALL, "Ukr");
```

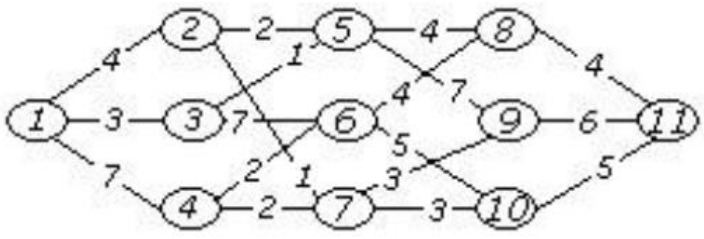
```
        int start;
        cout << "adjacency matrix: " << endl;
        for (int i = 0; i < n; i++)
        {
                visited[i] = false;
                for (int j = 0; j < n; j++)

                        cout << " " << graph[i][j];
                cout << endl;

        }
        cout << "The first vertex -> ";cin >> start;
        bool* vis = new bool[n];
        cout << "Answer: ";
        DFS(start - 1);
        delete[]visited;
        system("pause>>void");
}
```

Результат:



Завдання № 5

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



Розв'язання.

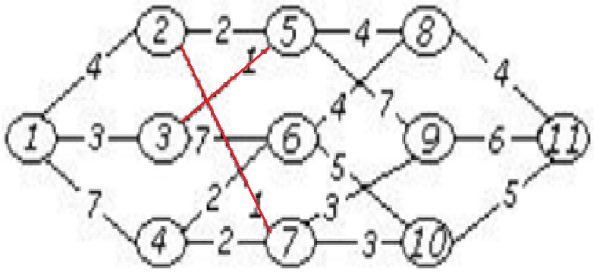
• **Метод Краскала**:

Етап 1: упорядковуємо для даного графа ребра у порядку спадання ваг цих ребер.
(3, 5) - 1;
(2, 7) - 1;
(2, 5) - 2;
(4,7) - 2;
(4, 6) - 2;
(1, 3) - 3;

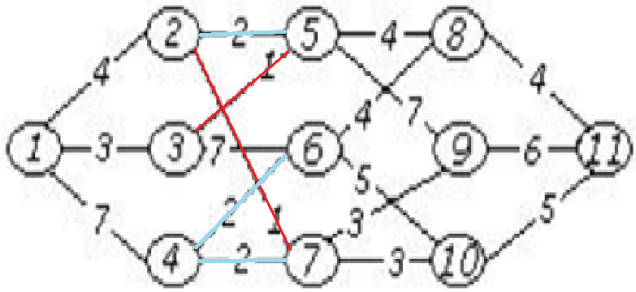
(7, 10) - 3;
(7, 9) - 3;
(1, 2) - 4;
(8, 11) - 4;
(6, 8) - 4;
(5, 8) - 4;
(6, 10) - 5;
(11, 10) - 5;
(9, 11) - 6;
(5, 9) - 7;
(3, 6) - 7;

(1, 4) - 7;

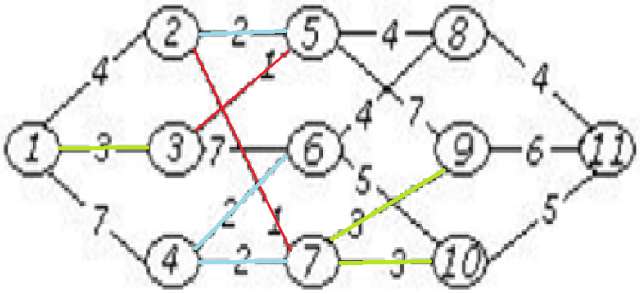
Етап 2: вибираємо ребро графа з найменшою вагою і додаємо до дерева. В даному випадку це ребра (2, 7) і (3, 5) з вагою 1.



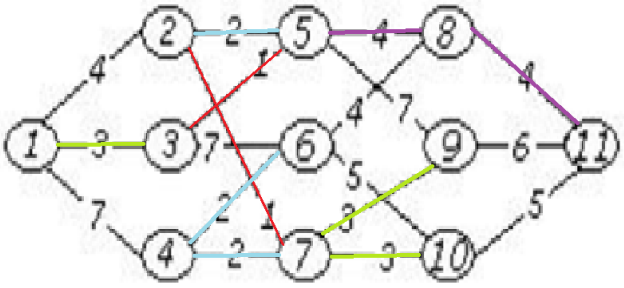
Етап 3: беремо ребро з наступною найменшою вагою. Це ребра (4, 6), (7, 4), (2, 5) з вагою 2. Додаємо до дерева.



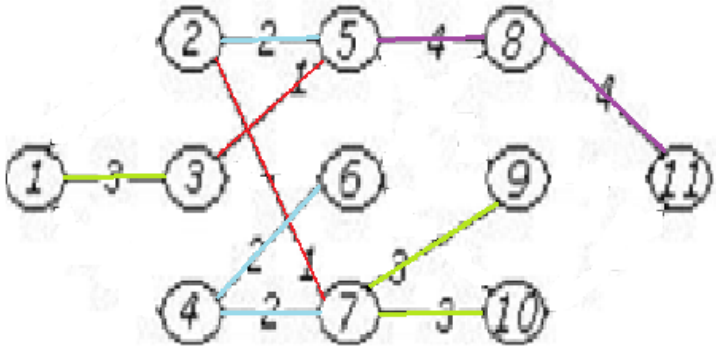
Етап 4: беремо ребра з наступною найменшою вагою - (1, 3), (7, 10), (7, 9) з вагою 3. Додаємо до кістякового дерева.



Етап 5: беремо ребра з наступною найменшою вагою - (1, 2), (8, 11), (5, 8), (6, 8) з вагою 4. Не додаємо ребра (1, 2) і (6, 8), оскільки вони утворять цикл.



Отже, мінімальне остове дерево за алгоритмом Краскала буде виглядати так:



Код програми (метод Краскала):

```
#include <iostream>
#include <fstream>
using namespace std;
const int q = 11;
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;
					}
				}
			}
		}
	}
}
```

```
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 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 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];
}

int main()
{
		ifstream in("matrix.txt");
		int A[11][11];
		for (int i = 0; i < 11; i++)
		{
			for (int k = 0; k < 11; k++)
			{
				in >> A[i][k];
			}
		}

   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;
}
```

Вхідні дані:

0 4 3 7 0 0 0 0 0 0

4 0 0 0 2 0 1 0 0 0 0

3 0 0 0 1 7 0 0 0 0 0

7 0 0 0 0 2 2 0 0 0 0

0 2 1 0 0 0 0 4 7 0 0

0 0 7 2 0 0 0 4 0 5 0

0 1 0 2 0 0 0 0 3 3 0

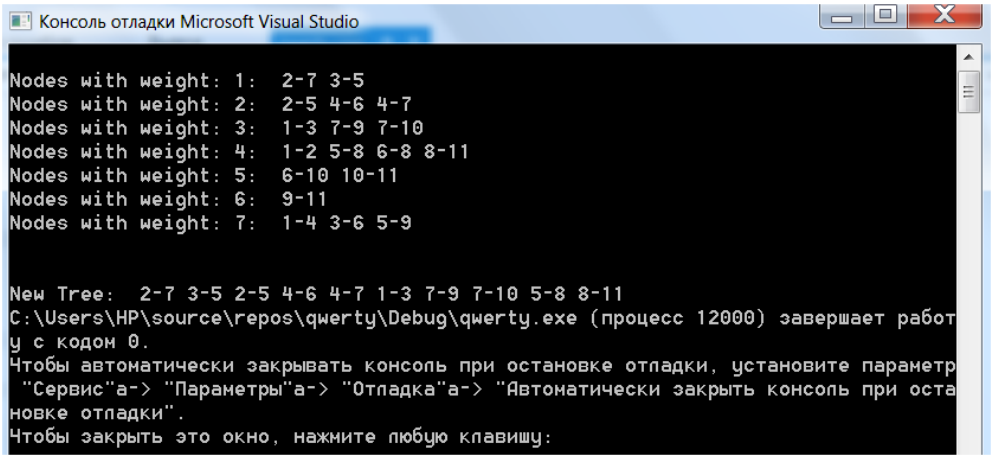
0 0 0 0 4 4 0 0 0 0 4

0 0 0 0 7 0 3 0 0 0 6

0 0 0 0 0 5 3 0 0 0 5

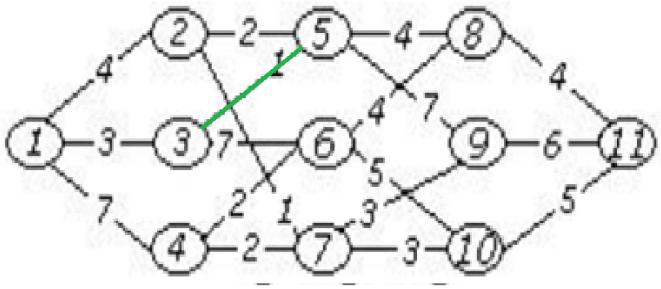
0 0 0 0 0 0 0 4 6 5 0

Результат:

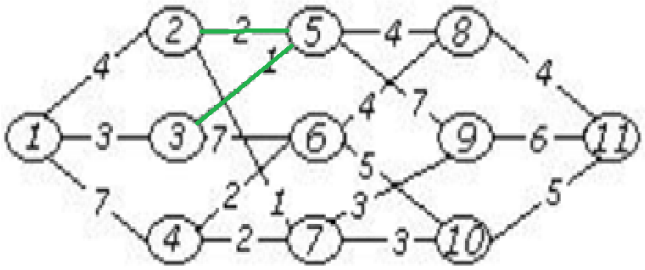


• Метод Прима:

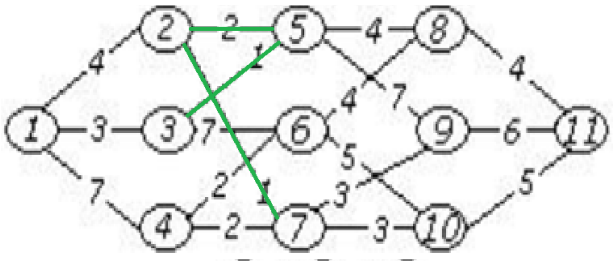
Етап 1: спочатку фіксуємо довільну вершину, з якої розпочнемо пошук мінімального дерева. У даному випадку це вершина 5. Знаходимо ребро, інцидентне цій вершині з найменшою вагою і це (3, 5) з вагою 1. Додаємо до кістякового дерева ребро (3, 5).



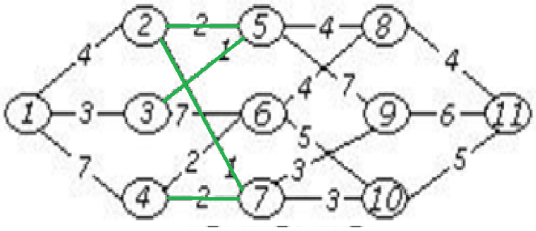
Етап 2: розглядаємо ребра, інцидентні вершинам 3 і 5, і вибираємо те, яке має найменшу вагу. І це (2, 5) з вагою 2. Додаємо його до кістякового дерева.



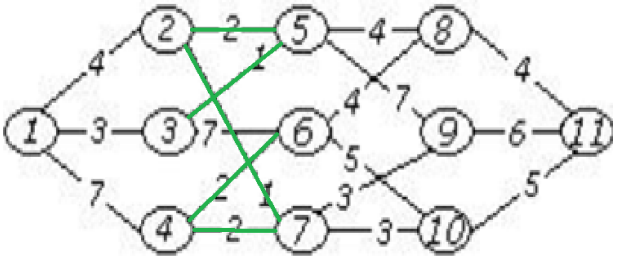
Етап 3: розглядаємо ребра, інцидентні вершинам 2, 3, 5 і вибираємо з найменшою вагою - і це (7, 2) з вагою 1. Додаємо до дерева.



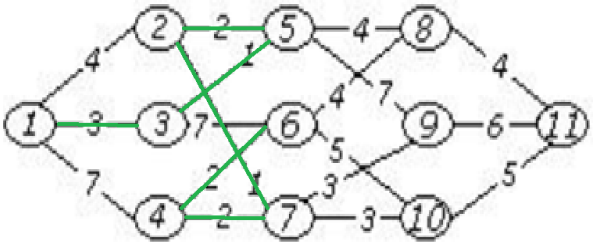
Етап 4: розглядаємо ребра, інцидентні вершинам 2, 3, 5, 7 і вибираємо з найменшою вагою - і це (7, 4) з вагою 2. Додаємо до дерева.



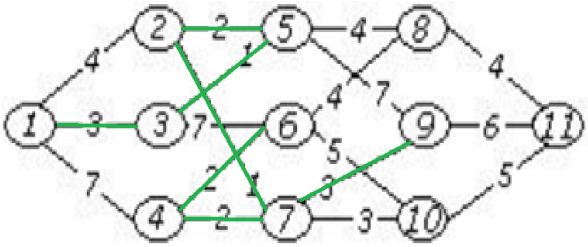
Етап 5: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 3, 4 і вибираємо з найменшою вагою - і це (4, 6) з вагою 2. Додаємо до кістякового дерева.



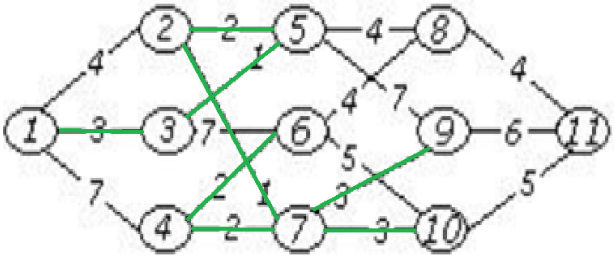
Етап 6: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 3, 4, 6 і вибираємо з найменшою вагою - і це (1, 3) з вагою 3. Додаємо до кістякового дерева.



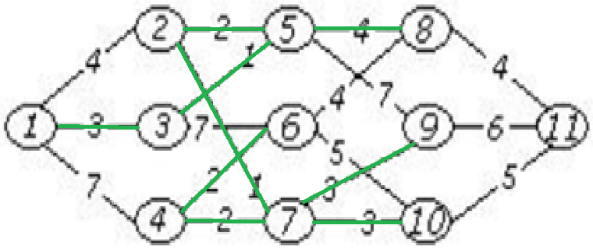
Етап 7: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3 і вибираємо з найменшою вагою - і це (7, 9) з вагою 3. Додаємо до кістякового дерева.



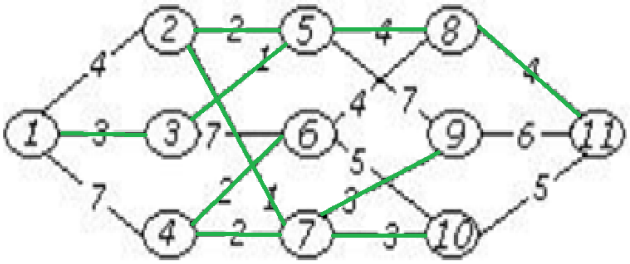
Етап 8: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 9 і вибираємо з найменшою вагою - і це (10, 7) з вагою 3. Додаємо до кістякового дерева.



Етап 9: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 9, 10 і вибираємо з найменшою вагою - і це (5, 8) з вагою 4. Додаємо до дерева.



Етап 10: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 9, 10, 8 і вибираємо з найменшою вагою - і це (11, 8) з вагою 4. Додаємо до дерева.



Усі точки пройшли. Отже, мінімальне остове дерево за алгоритмом Прима знайдене.

Код програми(метод Прима):

```
#include <iostream>
#include <fstream>
using namespace std;
const int gSize = 11;
int n;
ofstream fout;
int matrix[gSize][gSize];
int usedDots[gSize];
void init() {
	ifstream fin("matrix.txt");
	for (int i = 0;i < gSize;i++) {
		usedDots[i] = 0;
		for (int j = 0;j < gSize;j++)
			fin >> matrix[i][j];
```

```
	}
	fin.close();
}
void transform()
{
	for (int q = 0; q < gSize; q++)
	{
		for (int i = 0; i < gSize; i++)
		{
			if (matrix[q][i] == 0)
			{
				matrix[q][i] = 999;
			}
		}
	}
}
void print() {
	for (int i = 0;i < gSize;i++) {
		for (int j = 0;j < gSize;j++)
			cout << matrix[i][j];
		cout << endl;
	}
}
void tree(int dot) {
	while (dot != -1) {
		usedDots[dot] = 1;
		int enterDot;
		int minDot = -1;
		int min = 999;
		for (int i = 0;i < gSize;i++)
			if (usedDots[i])
				for (int j = 0;j < gSize;j++)
					if (matrix[i][j] < min && !usedDots[j]) {
						min = matrix[i][j];
						minDot = j;
						enterDot = i;
					}
		dot = minDot;
		if (dot != -1)
			cout << "(" << enterDot+1 << ";" << dot+1 << ") ";
	}
}
int main()
{
	init();
	print();
	transform();
	tree(0);
	return 0;
}
```

Вхідні дані:

0 4 3 7 0 0 0 0 0 0 0

4 0 0 0 2 0 1 0 0 0 0

3 0 0 0 1 7 0 0 0 0 0

7 0 0 0 0 2 2 0 0 0 0

0 2 1 0 0 0 0 4 7 0 0

0 0 7 2 0 0 0 4 0 5 0

0 1 0 2 0 0 0 0 3 3 0

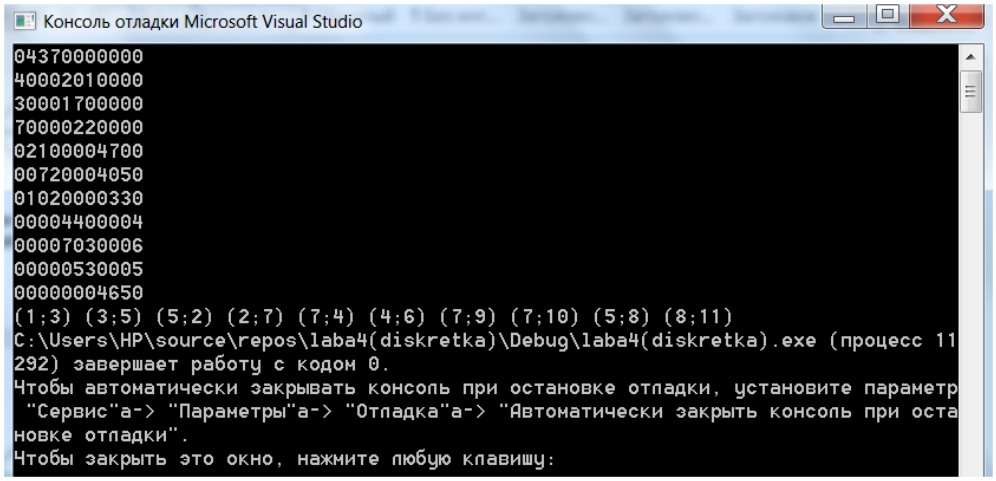
0 0 0 0 4 4 0 0 0 0 4

0 0 0 0 7 0 3 0 0 0 6

0 0 0 0 0 5 3 0 0 0 5

0 0 0 0 0 0 0 4 6 5 0

Результат:



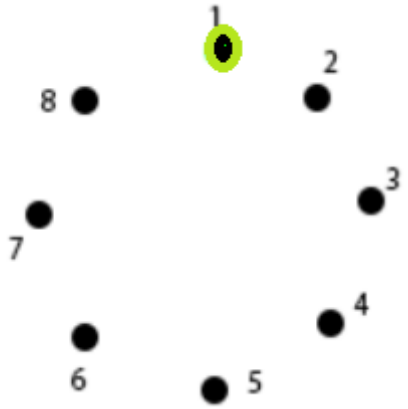
Завдання № 6

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

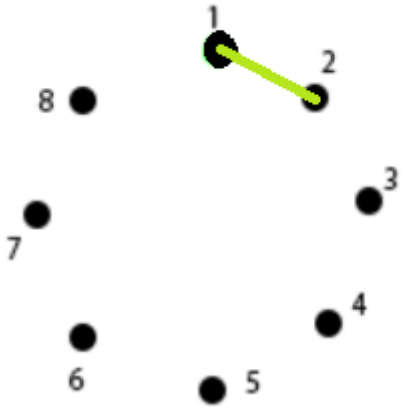
	1	2	3	4	5	6	7	8
1	∞	1	3	5	1	5	3	2
2	1	∞	6	6	6	1	5	5
3	3	6	∞	7	3	5	4	1
4	5	6	7	∞	5	5	5	1
5	1	6	3	5	∞	6	6	6
6	5	1	5	5	6	∞	5	2
7	3	5	4	5	6	5	∞	2
8	2	5	1	1	6	2	2	∞

Розв’язання.

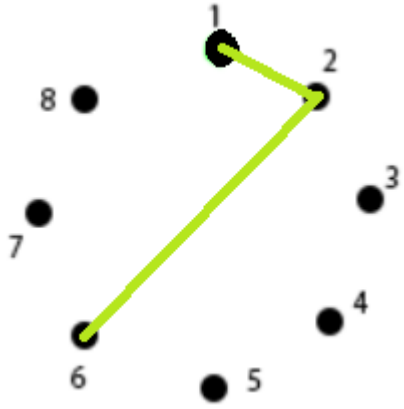
Отже нехай початкова точка буде 1.



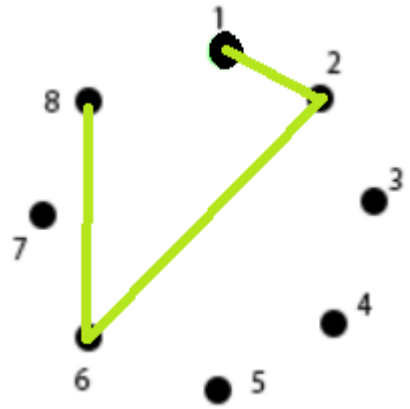
Найближчою є точка 2.



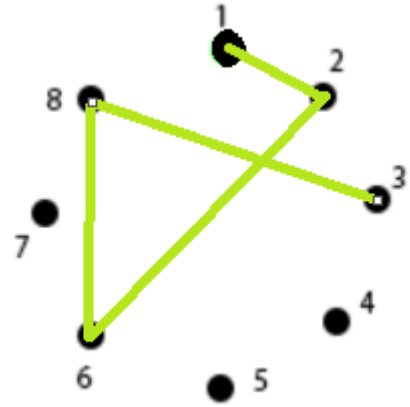
Найближчою є точка 6.



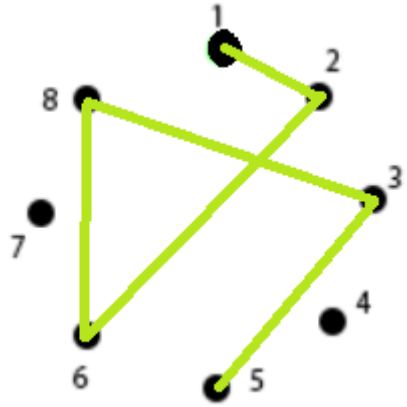
Вибираємо наступну найближчу точку: 8.



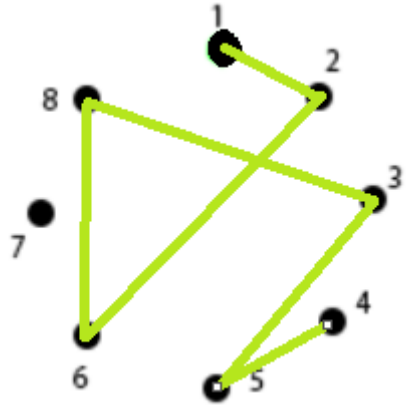
Далі 3.



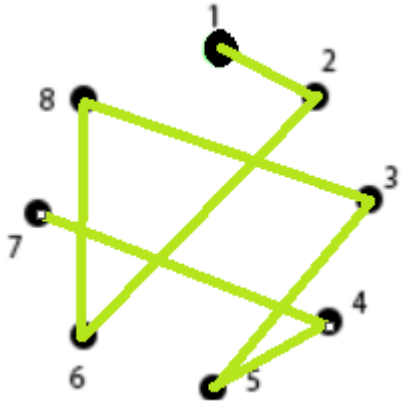
Наступна 5.



Найближчою точкою до 5 буде 4.



Далі очевидно, що точка 7.



Сума рівна 18. Перевіримо чи це найменший шлях з усіх інших.

Код програми:

```
#include <iostream>
#include <fstream>
using namespace std;

const int gSize = 8;

int matrix[gSize][gSize];
bool visited[gSize];

int next(int dot) {
	int min = 99999999;
	int minDot = -1;
	for (int i = 0;i < gSize;i++)
		if (matrix[dot][i] < min && !visited[i] && matrix[dot][i]) {
			min = matrix[dot][i];
			minDot = i;
		}
	return minDot;
}

void findOst(int dot) {
	while (dot != -1) {
		visited[dot] = true;
		int prevDot = dot;
		dot = next(prevDot);
		if (dot != -1)
			cout << "(" << prevDot + 1 << ";" << dot + 1 << ") ";
	}
}

int main()
{
	ifstream f1("input.txt");
	for (int i = 0;i < gSize;i++) {
		visited[i] = false;
		for (int j = 0;j < gSize;j++)
			f1 >> matrix[i][j];
	}
	findOst(0);


	return 0;
}
```

Вхідні дані:

0 1 3 5 1 5 3 2

1 0 6 6 6 1 5 5

3 6 0 7 3 5 4 1

5 6 7 0 5 5 5 1

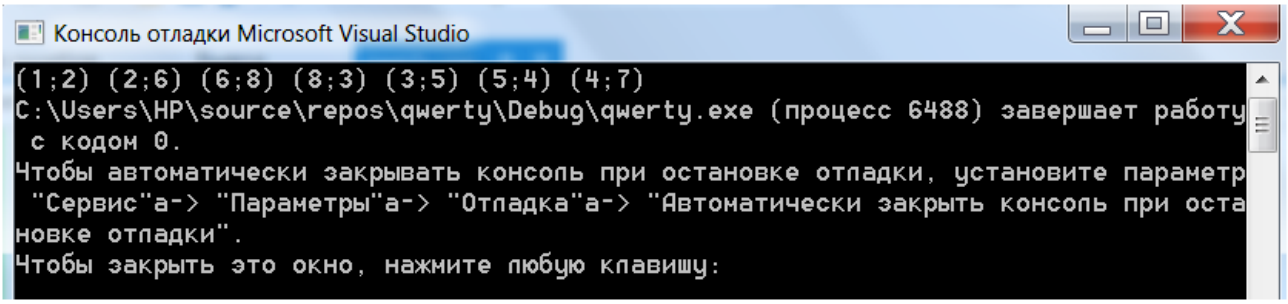
1 6 3 5 0 6 6 6

5 1 5 5 6 0 5 2

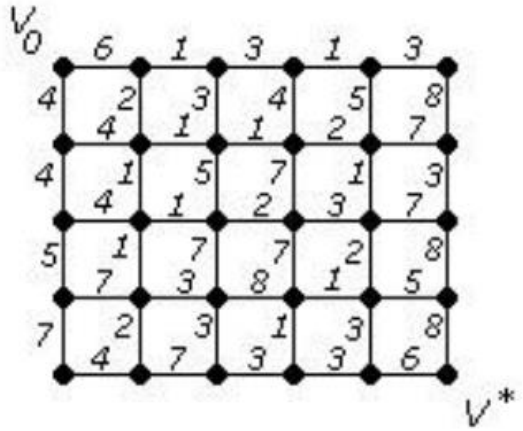
3 5 4 5 6 5 0 2

2 5 1 1 6 2 2 0

Результат:



Завдання № 7
За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V0 і V* .



Розв’язання.

Відстань до всіх вершин графа рівне нескінченності. Відстань до $V_0 = 0$. Жодна вершина графа ще не опрацьована.

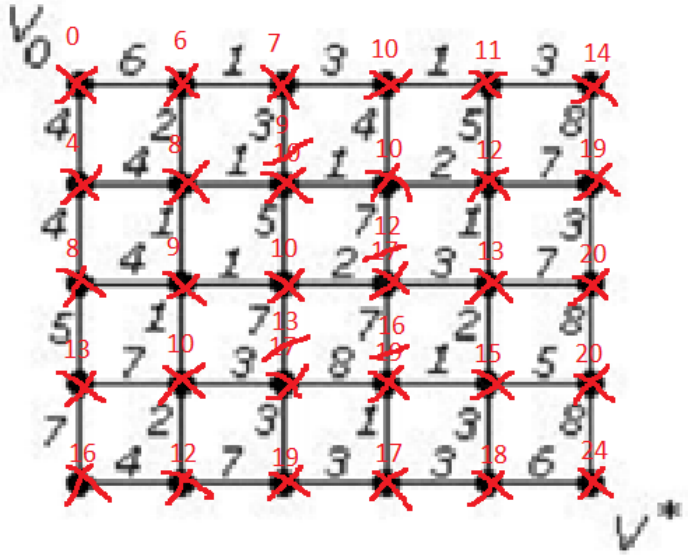
Знаходимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина V_0. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через V_0менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.

Отже запишемо, що шлях від V_0 до $V_{0\,1}$ рівне 6, а від V_0 до $V_{1\,0}$ рівне 4. Виберемо мінімальний. Ним є 4 – шлях від V_0 до $V_{1\,0}$.

Всі сусіди вершини V_0 перевірені. Поточна мінімальна відстань до вершини V_0 вважається остаточною і обговоренню не підлягає. Тому точку V_0 може викреслити.

Знову знаходимо найближчу невикреслену вершину з точок $V_{1\,0}$ зменшуючи відстань до усіх сусідніх вершин.

Продовжуємо виконувати ці дії допоки не викреслені всі вершини.



1->2->8->9->10->11->17->23->29->30.

Найкоротший шлях: 24.

Код програми:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	setlocale(LC_CTYPE, "ukr");
	bool visited[30];int tmp, l;
	for (int i = 0; i < 30; i++)
	{
		visited[i] = false;
	}
	int mas1[30][30];
	int min[30],min1[30], min2[30];
	for (int i = 0; i < 30; i++)
	{
		min[i] = 999;
	}
	ifstream in("Input.txt");
	for (int i = 0; i < 30; i++)
	{
		for (int k = 0; k < 30; k++)
		{
			in >> mas1[i][k];
			if (mas1[i][k] == 0)
			{
				mas1[i][k] = 999;
			}
		}
	}
	int e, e1;
```

```
cout << "Введіть початкову точку: ";
cin >> e;
l = e-1;
cout << "Введіть кінцеву точку: ";
cin >> e1;
min[l] = 0;
mas1[l][l] = 0;
for (int u = 0; u < 30; u++)
{
	for (int i = 0; i < 30; i++)
	{
		if ((min[i] > mas1[l][i]) && (visited[i] == false))
		{
			if (min[i] > (mas1[l][i] + min[l]))
			{
				min[i] = mas1[l][i] + min[l];
			}
		}
	}
	visited[l] = true;
	for (int i = 0; i < 30; i++)
	{
		if (visited[i] == true)
		{
			min2[i] = 999;
		}
		else { min2[i] = min[i]; }
	}
	for (int j = 0; j < 29; j++) {
		for (int i = 0; i < 29; i++) {
			if (min2[i] > min2[i + 1])
			{
				tmp = min2[i];
				min2[i] = min2[i + 1];
				min2[i + 1] = tmp;
			}
		}
	}
	l = 0;
	for (int i = 0; i < 30; i++)
	{
		if (visited[i] == false)
		{
			min1[i] = min[i];
		}
		else min1[i] = 999;
	}
	while (min2[0] != min1[l])
	{
		l++;
	}
}
cout << "Найкоротший шлях: " << min[e1-1];
int way[30], temp= min[e1 - 1];
int t = 0,t1 = (e1 - 1);
for (int i = 0; i < 30; i++)
{
	way[i] = 0;
}
```

```
	for (int i = 0; i < 900; i++)
	{
		if ((temp - mas1[t1][t]) == min[t])
		{
			way[t] = t + 1;temp = min[t];t1 = t;t = 0;

			continue;
		}
		t++;
		if (t == 30)break;
	}
		cout << endl;
	for (int i = 0; i < 30; i++)
	{
		if (way[i]!= 0)cout << way[i] << "->";
	}
	cout << e1;
}
```

Вхідні дані:

0 6 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

6 0 1 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 1 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 1 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 1 0 3 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 0 0 0 3 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

4 0 0 0 0 0 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 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 5 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 5 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 8 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 4 0 0 0 0 0 0 4 0 0 0 0 5 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 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 5 0 0 0 0 1 0 2 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 2 0 3 0 0 0 0 7 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 3 0 7 0 0 0 0 2 0 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 0 8 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 7 0 0 0 0 7 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 7 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 7 0 0 0 0 3 0 8 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 8 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 2 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 8 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 7 0 0 0 0 0 4 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 4 0 7 0 0 0

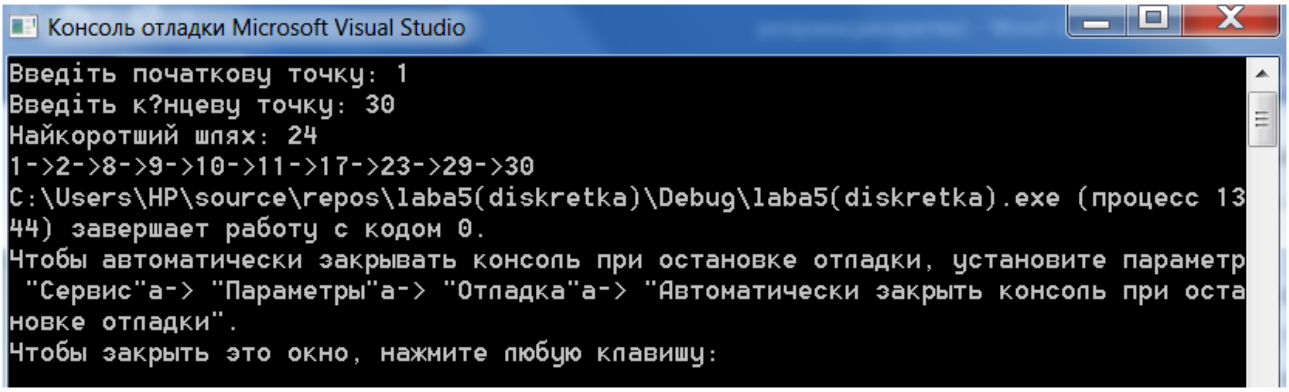
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 7 0 3 0 0

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 0 3 0 3 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 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 8 0 0 0 0 6 0

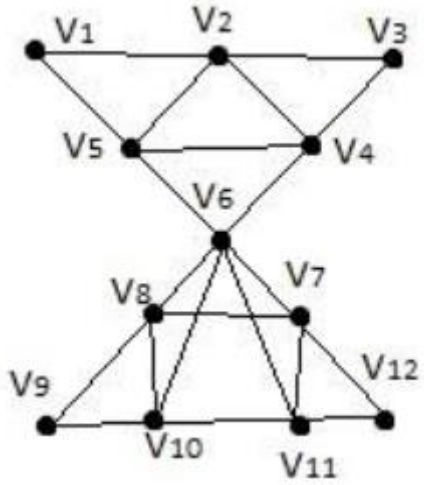
Результат:



Завдання № 8
Знайти ейлеровий цикл в ейлеровому графі двома методами:

а) Флері.

б) елементарних циклів.



Розв’язання.

а) Метод Флері.

Отже, **Ейлеровим циклом** називається замкнутий маршрут, в якому кожне ребро графа зустрічається точно один раз. Для існування такого маршруту в зв'язному графові необхідно і достатньо, щоб степені всіх його вершин були парними.

Основна суть даного алгоритму полягає в наступному: починаючи з будь-якої вершини деякого неорієнтованого графа G, довільним чином йдемо по суміжних ребрах, видаляючи пройдене ребро і

вершину, що стала після видалення ребра ізольованою (вершина, степінь якої дорівнює нулю). Не проходимо по ребру, якщо після його видалення граф стає незв'язним.

Виберемо вершину 1. Нехай наступною вершиною буде 2. Спираючись на вище описаний алгоритм у нас утворилася ось така послідовність обходу графа:

V_1-> V_2-> V_3->V_4->V_2->V_5->V_4->V_6->V_7->V_8->V_6->V_{10}->V_8->V_9->V_{10}->V_{11}->V_7->V_{12}->V_{11}->V_6->V_5->V_1.

Усі ребра із точками видалені, отже наш Ейлеровий цикл знайдений.

Код програми:

```
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;

int matr[1000][1000];
int n, m;
vector<int> path;
bool stop = false;

void print() {
	for (int i = 0;i < path.size();i++)
		cout << path[i] + 1 << " ";
	cout << endl;
}

void dfs(int v) {
	path.push_back(v);
	//cout << v << endl;
	if (!n) {
		print();
		stop = true;
	}
	for (int i = 0;i < m;i++) {
		if (matr[v][i]) {

			n--;
			matr[v][i] = 0;
			matr[i][v] = 0;
			dfs(i);
			if (stop)
				return;
			n++;
			matr[v][i] = 1;
			matr[i][v] = 1;
			path.erase(path.begin() + path.size() - 1);
			//cout << "erase " << path[path.size()] << endl;
		}
	}
}

int main()
{
	ifstream cin("input.txt");
	cin >> n >> m;
	for (int i = 0;i < m;i++)
		for (int j = 0;j < m;j++)
			matr[i][j] = 0;
```

```
	for (int i = 0;i < n;i++) {
		int a, b;
		cin >> a >> b;
		matr[a - 1][b - 1] = 1;
		matr[b - 1][a - 1] = 1;
	}
	dfs(0);
	return 0;
}
```

Вхідні дані:

21 12 - кількість ребер та то точок відповідно.

1 5 - ребра

5 6

6 8

8 9

9 10

10 6

6 11

11 7

7 8

8 10

10 11

11 12

12 7

7 6

6 4

4 5

5 2

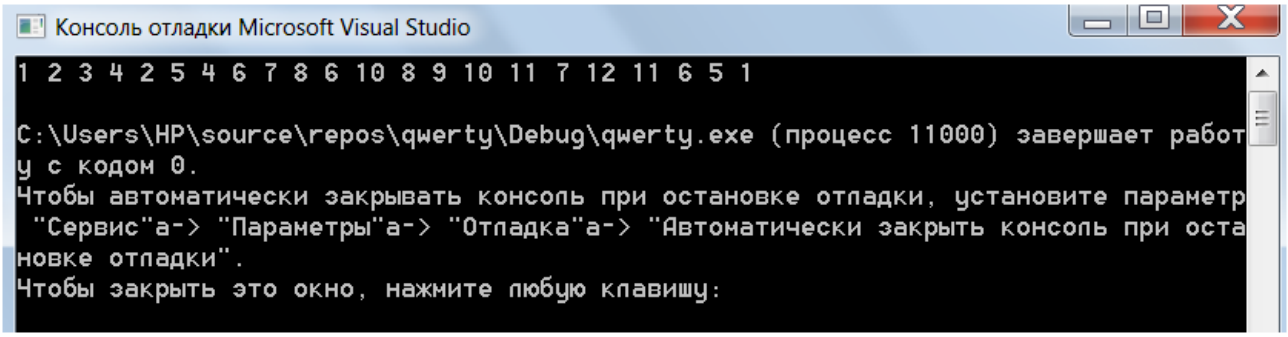
2 4

4 3

3 2

2 1

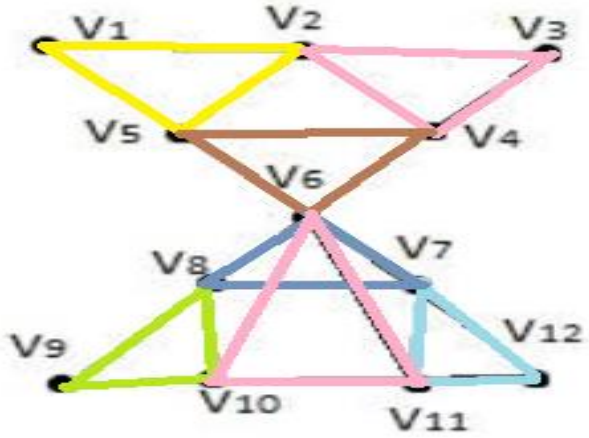
Результат:



б) Метод елементарних циклів.

Отже, для побудови Ейлерового циклу довільно виберемо одну з вершин графа G. На наступному кроці, виберемо будь-яке з інцидентних даній вершині ребер і з його допомогою перейдемо у відповідну суміжну вершину. Це ребро після виконання цих дій, вважатиметься пройденим. Після цього, повторюємо цю операцію, щоразу обираючи нове ребро, поки не опинимося в вихідній вершині і не замкнемо цикл.

Позначимо знайдений таким чином цикл як C_1. Якщо C_1 включає всі ребра графа G, то Ейлерів цикл побудовано. В іншому випадку, переходимо до розгляду графа G_1, який отримуємо шляхом видалення з графа G всіх ребер, що входять до графу C_1. Цей граф може бути як зв'язним, так і не зв'язним, але будь-яка його компонента в силу зв'язності G має хоча б одну загальну вершину з графом C_1. Так як кожна вершина в G і C_1 має парну степінь, то і в G_1 степінь всіх вершин повинна бути парною.
Якщо C_1 і наступний цикл включає всі ребра графа G, то Ейлерів цикл побудовано. Якшо ні, то шукаємо наступні цикли. Після цього, об'єднаємо цикли.
Отже розфарбуємо наші елементарні цикли різними кольорами.



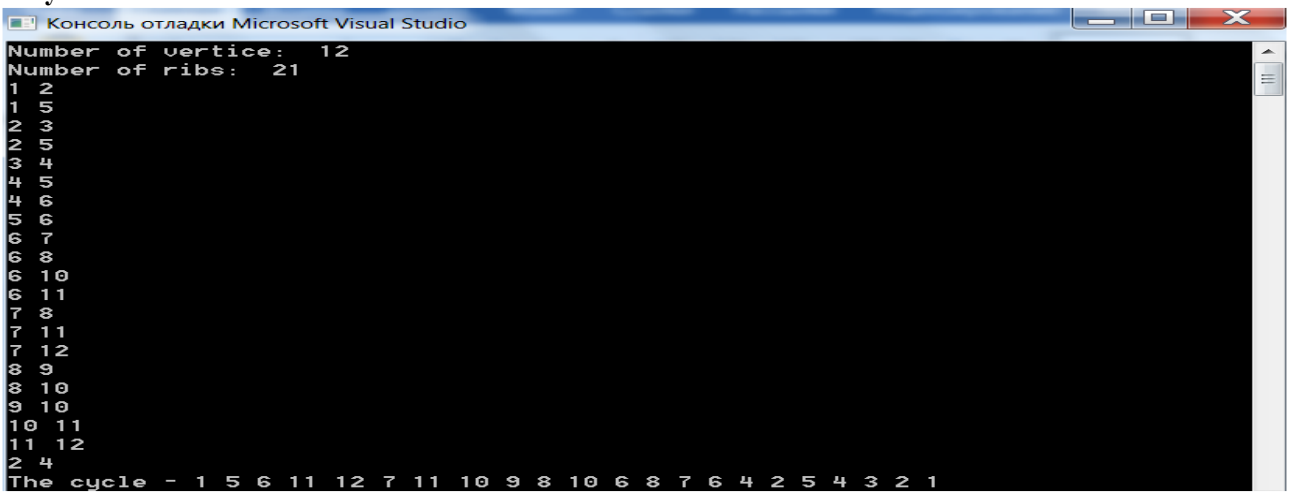
Та об’єднаємо ефективно їх в один цикл:
V_1-> V_5-> V_6->V_{11}->V_{12}->V_7->V_{11}->V_{10}->V_9->V_8->V_{10}->V_6->V_8->V_7->V_6->V_4->V_2->V_5->V_4->V_3->V_2->V_1.

Код програми:

```
#include <iostream>
#include <stack>
#include <vector>
#include <list>
#include <algorithm>
using namespace std;
vector <list<int>>graf;
vector<int>step;
stack<int> head, tail;
int main()
{
	int n, a, x, y;
	cout << "Number of vertice: " << " ";
```

```
	cin >> n;
	cout << "Number of ribs: " << " ";
	cin >> a;
	graf.resize(n + 1);
	step.resize(n + 1);
	for (;a--;)
	{
		cin >> x >> y;
		graf[x].push_back(y);
		graf[y].push_back(x);
		++step[x];
		++step[y];
	}
	if (any_of(step.begin() + 1, step.end(), [](int i) {return i & 1;}))
		cout << "-1";
	else {
		head.push(1);
		while (!head.empty())
		{
			while (step[head.top()])
			{
				int v = graf[head.top()].back();
				graf[head.top()].pop_back();
				graf[v].remove(head.top());
				--step[head.top()];
				head.push(v);
				--step[v];
			}
			while (!head.empty() && !step[head.top()])
			{
				tail.push(head.top());
				head.pop();
			}
		}
		cout << "The cycle - ";
		while (!tail.empty())
		{
			cout << tail.top() << ' ';
			tail.pop();
		}
	}

}
```

Результат:



Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

Перший спосіб.

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

Тоді за законом дистрибутивності:

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

За законом суперечності:

$(x \wedge \bar{x}) \vee (\bar{y} \wedge \bar{x}) \vee (x \wedge y) \vee (\bar{y} \wedge y) \vee \bar{z}\,\bar{x} \vee \bar{z}\,y = 0 \vee (\bar{y} \wedge \bar{x}) \vee (x \wedge y) \vee 0 \vee \bar{z}\,\bar{x} \vee \bar{z}\,y = (\bar{y} \wedge \bar{x}) \vee (x \wedge y) \vee \bar{z}\,\bar{x} \vee \bar{z}\,y.$

Другий спосіб.

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

Складемо табличку:

x	y	z	T or F
0	0	0	1
1	0	0	0
1	1	0	1
1	1	1	1
0	0	1	1
0	1	1	0
0	1	0	1
1	0	1	0

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