МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахунково-графічна робота

з дисципліни
«Дискретна математика»

Виконала:
студентка групи КН-113
Зварич Адріана
Викладач: Мельникова Н.І.

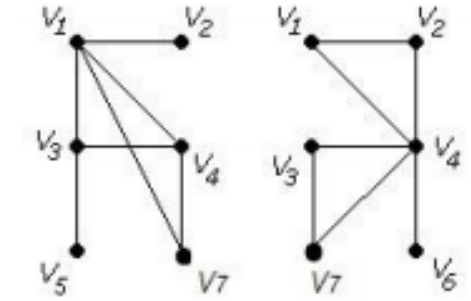
Львів – 2019 р.

Варіант 20

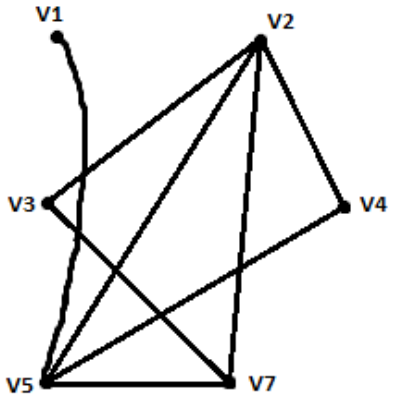
Завдання № 1

Виконати наступні операції над графами:

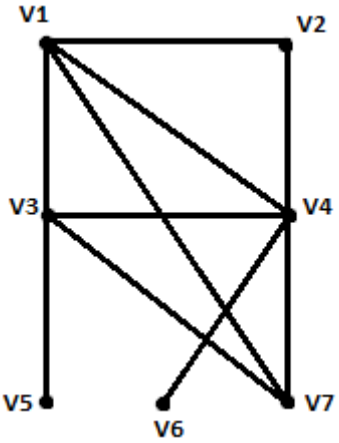
20)



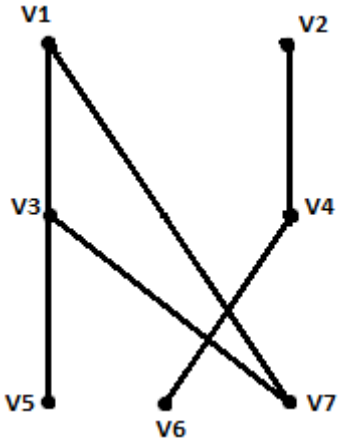
1) знайти доповнення до першого графу;



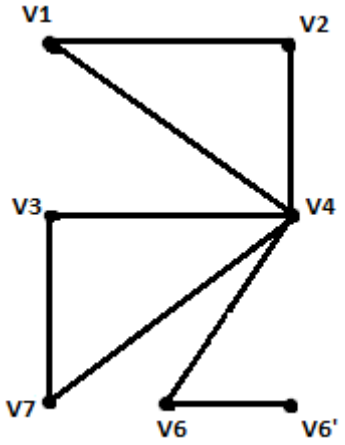
2) об’єднання графів;



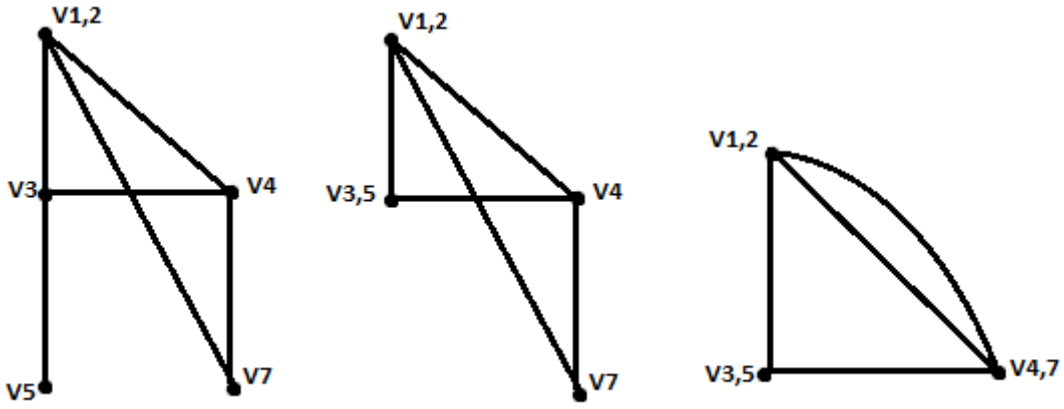
3) кільцеву сумму G1 та G2 (G1+G2);



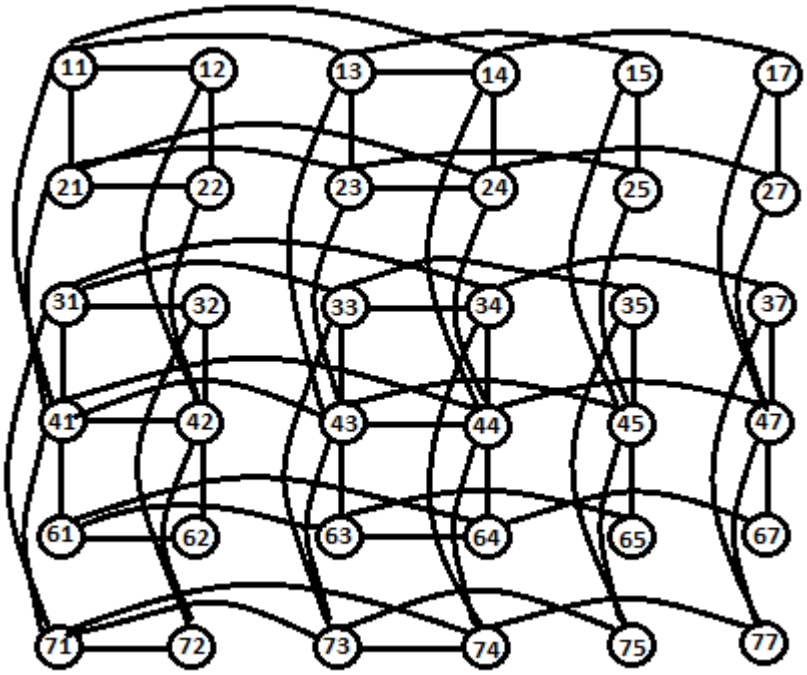
4) розмножити вершину у другому графі;



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



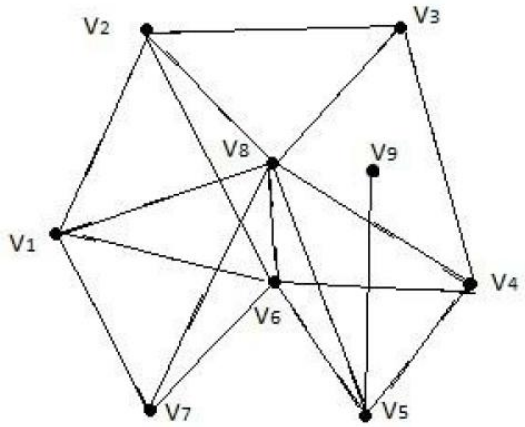
6) добуток графів.



Завдання № 2

Скласти таблицю суміжності для графа.

20)



Таблиця суміжності виглядатиме так:

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

Завдання №3

Для графа з другого завдання знайти діаметр.

Діаметр графа дорівнює 3.

Завдання №4

Для графа з другого завдання виконати обхід дерева вшир.

1)	1	1	10)678345	5
2)	12	2	11)78345	-
3)	126	6	12)8345	-
4)	1267	7	13)345	-
5)	12678	8	14)45	-
6)	2678	-	15)5	-
7)	26783	3	16)59	9
8)	6783	-	17)9	-
9)	67834	4	18)	-

Порядок обходу 126783459.

Код програми:

```
#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;
```

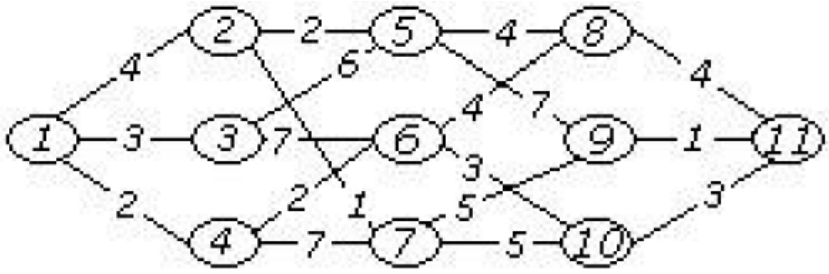
```
                }

                cout << "Черга порожня" << endl;
                break;
            }
        }

        return 0;
}
```

Результати програми:

```
3 якої вершини почати обхід? 1
Якщо ви хочете зробити обхід вглиб натисніть 1, обхід вшир - натисніть 2: 2
1 2
1 2 7
1 2 7 8
1 2 7 8 6
2 7 8 6
2 7 8 6 3
7 8 6 3
8 6 3
8 6 3 4
8 6 3 4 5
6 3 4 5
3 4 5
4 5
5
5 9
9
Черга порожня
```

Завдання №5

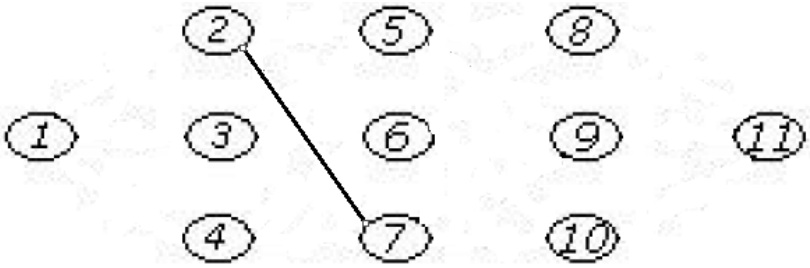
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

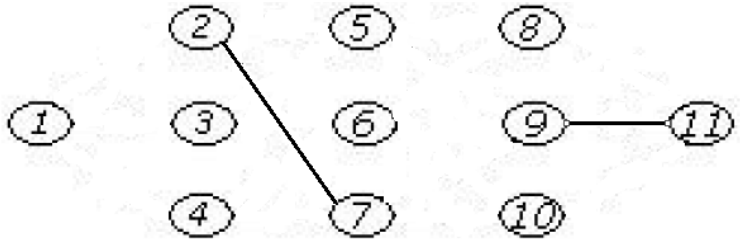


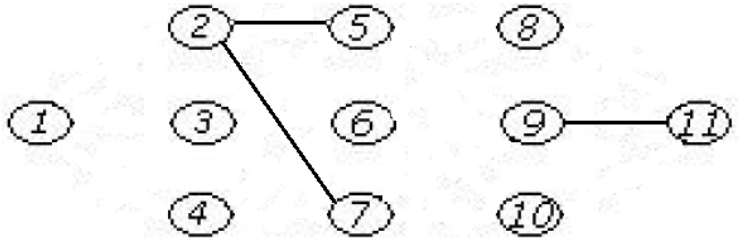
Метод Краскала:

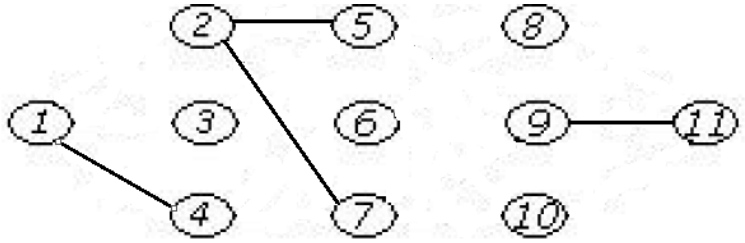
V(G) = {2,7,9,11,5,1,4,6,10,3,8};

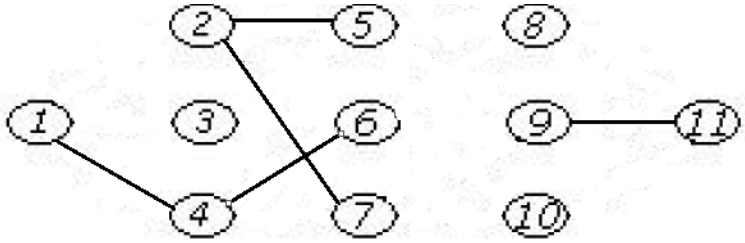
E(G) = {(2,7),(9,11),(2,5),(1,4),(4,6),(6,10),(10,11),(1,3),(1,2),(5,8)};

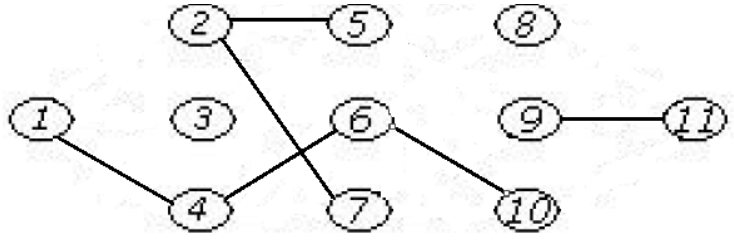


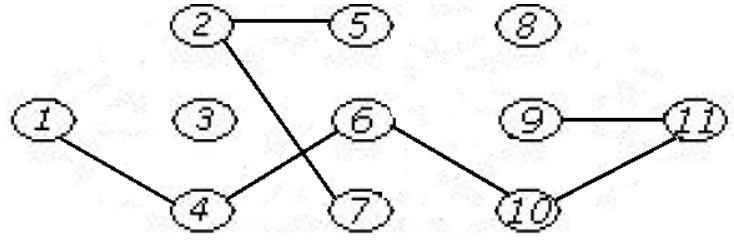

2
5
8
1
3
6
9
11
4
7
10

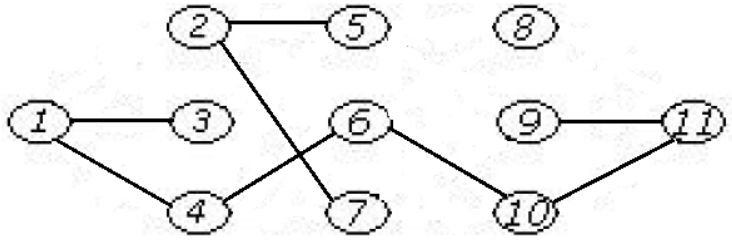

2
5
8
1
3
6
9
11
4
7
10

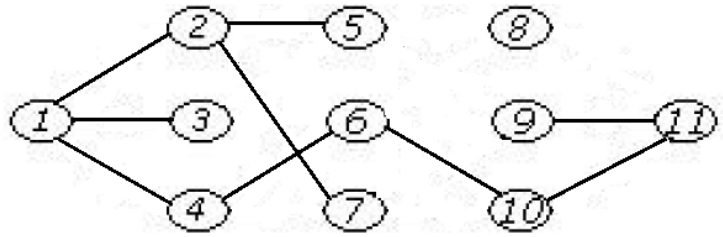

2
5
8
1
3
6
9
11
4
7
10

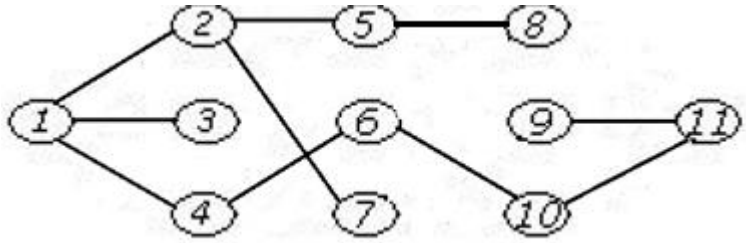

2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10







Код програми:

```
#include <stdio.h>
#define MAX 30
#define VERT 11
typedef struct edge
{
 int u,v,w;
}edge;
typedef struct edgelist
{
 edge data[MAX];
 int n;
}edgelist;
edgelist elist;
//matrix of adjacency
int G[11][11] = { {0,4,3,2,0,0,0,0,0,0,0},
 {4,0,0,0,2,0,1,0,0,0,0},
 {3,0,0,0,6,7,0,0,0,0,0},
 {2,0,0,0,0,2,7,0,0,0,0},
 {0,2,6,0,0,0,0,4,7,0,0},
 {0,0,7,2,0,0,0,4,0,3,0},
 {0,1,0,7,0,0,0,0,5,5,0},
 {0,0,0,0,4,4,0,0,0,0,4},
 {0,0,0,0,7,0,5,0,0,0,1},
```

```
 {0,0,0,0,0,3,5,0,0,0,3},
 {0,0,0,0,0,0,0,4,1,3,0}};
int n;
edgelist spanlist;
void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();
int main()
{
 int i,j,total_cost;
 printf(" Kruskal Algorithm \n");
 n = VERT;
 kruskal();
 print();
 }
void kruskal()
{
 int belongs[MAX],i,j,cno1,cno2;
 elist.n=0;
 for(i=1;i<n;i++)
 for(j=0;j<i;j++)
 {
 if(G[i][j]!= 0)
 {
 elist.data[elist.n].u=i;
 elist.data[elist.n].v=j;
 elist.data[elist.n].w=G[i][j];
 elist.n++;
 }
 }
 sort();
 for(i=0;i<n;i++)
 belongs[i]=i;
 spanlist.n=0;
 for(i=0;i<elist.n;i++)
 {
 cno1=find(belongs,elist.data[i].u);
 cno2=find(belongs,elist.data[i].v);
 if(cno1!=cno2)
 {
 spanlist.data[spanlist.n]=elist.data[i];
```

```
 spanlist.n=spanlist.n+1;
 union1(belongs,cno1,cno2);
 }
 }
}
int find(int belongs[],int vertexno)
{
 return(belongs[vertexno]);
}
void union1(int belongs[],int c1,int c2)
{
 int i;
 for(i=0;i<n;i++)
 if(belongs[i]==c2)
 belongs[i]=c1;
}
void sort()
{
 int i,j;
 edge temp;
 for(i=1;i<elist.n;i++)
 for(j=0;j<elist.n-1;j++)
 if(elist.data[j].w>elist.data[j+1].w)
 {
 temp=elist.data[j];
 elist.data[j]=elist.data[j+1];
 elist.data[j+1]=temp;
 }
}
void print()
{
 int i,cost=0;
 for(i=0;i<spanlist.n;i++)
 {
 printf("\n[%d]--->[%d] =%d",spanlist.data[i].u+1,spanlist.data[i].v+1,spanlist.data[i]
.w);
 cost=cost+spanlist.data[i].w;
 }
 printf("\n\nCost of the spanning tree=%d",cost);
}
```

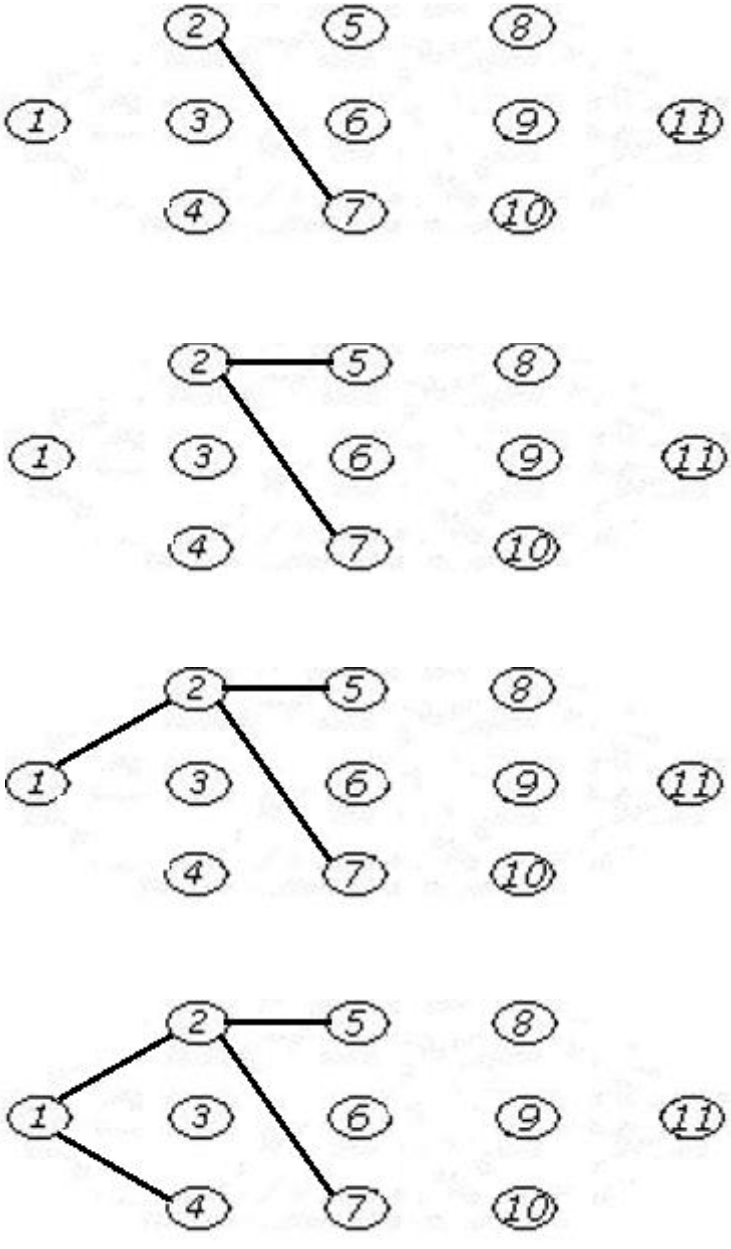
Результати програми:

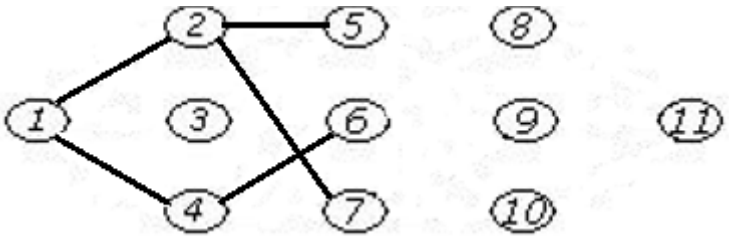
```
 Kruskal Algorithm

[7]--->[2] =1
[11]--->[9] =1
[4]--->[1] =2
[5]--->[2] =2
[6]--->[4] =2
[3]--->[1] =3
[10]--->[6] =3
[11]--->[10] =3
[2]--->[1] =4
[8]--->[5] =4

Cost of the spanning tree=25
```

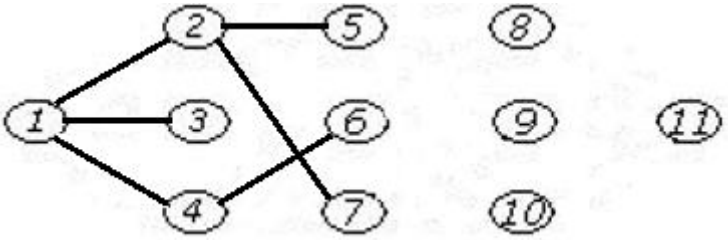
Алгоритм Прима:

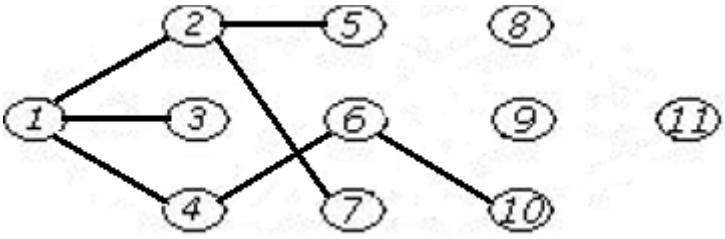
V(G) = {2,7,5,1,4,6,3,10,11,9,8};

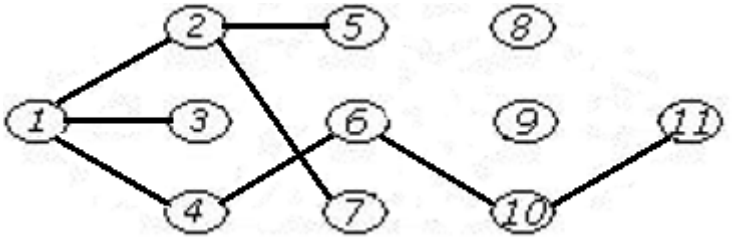
E(G) = {(2,7),(2,5),(2,1),(1,4),(4,6),(1,3),(6,10),(10,11),(11,9),(8,5)};

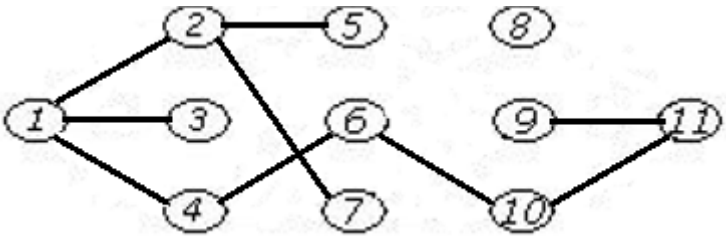


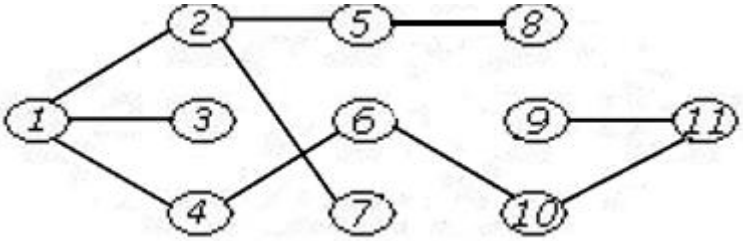

2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10


2
5
8
1
3
6
9
11
4
7
10

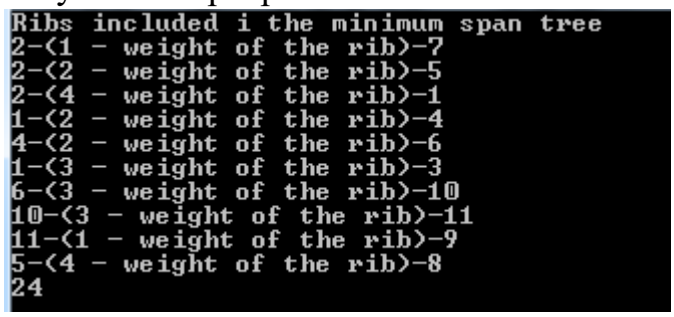

2
5
8
1
3
6
9
11
4
7
10

Код програми:

```
#include<stdio.h>
#include<stdlib.h>
#define size 11
int main()
{
int wght =0;
int graph[size][size] = {{0,4,3,2,0,0,0,0,0,0,0},
 {4,0,0,0,2,0,1,0,0,0,0},
 {3,0,0,0,6,7,0,0,0,0,0},
 {2,0,0,0,0,2,7,0,0,0,0},
 {0,2,6,0,0,0,0,4,7,0,0},
 {0,0,7,2,0,0,0,4,0,3,0},
 {0,1,0,7,0,0,0,0,5,5,0},
 {0,0,0,0,4,4,0,0,0,0,4},
 {0,0,0,0,7,0,5,0,0,0,1},
 {0,0,0,0,0,3,5,0,0,0,3},
 {0,0,0,0,0,0,0,4,1,3,0}
};
int visit[size] = {0};
int i, j, p=0, q=0;
int arr[size]={1,2,3,4,5,6,7,8,9,10,11};
int min;
int flag=0;
for(i=0;i<size;i++){
for(j=0;j<size;j++){
if(flag ==0 && graph[i][j]!=0){
flag=1;
p=i;
q=j;
min=graph[p][q];
}
else if(flag == 1 && graph[i][j]<min && graph[i][j]!=0){
        p=i;
q=j;
min = graph[i][j];
}
}
}
visit[p]=1;
visit[q]=1;
int flag1=0;
int p1,q1,min1,qq=0;
```

```
printf("Ribs included i the minimum span tree \n");
printf("%d-(%d - weight of the rib)-%d ", arr[p], graph[p][q],
arr[q]);
do{
for(i=0;i<size;i++){
for(j=0;j<size;j++){
if(visit[i]==1 && visit[j]==0 && graph[i][j]!=0){
if(flag1==0){
flag1 = 1;
p1 = i;
q1 = j;
min1=graph[i][j];
}else if(flag1 == 1 && graph[i][j]< min1){
p1 = i;
q1 = j;
min1 = graph[i][j];
}
}
}
}
visit[q1]=1;
flag1=0;
printf("\n%d-(%d - weight of the rib)-%d", arr[p1],
graph[p1][q1], arr[q1]);
qq++;
wght = wght+graph[p1][q1];
}
while(qq < size-2);
printf("\n%d\n", wght);
return 0;
}
```

Результати програми:



Завдання №6

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

20)

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

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

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

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

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

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

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

Мінімальна вага 15.

Код програми:

```
#include "pch.h"
#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;
}
```

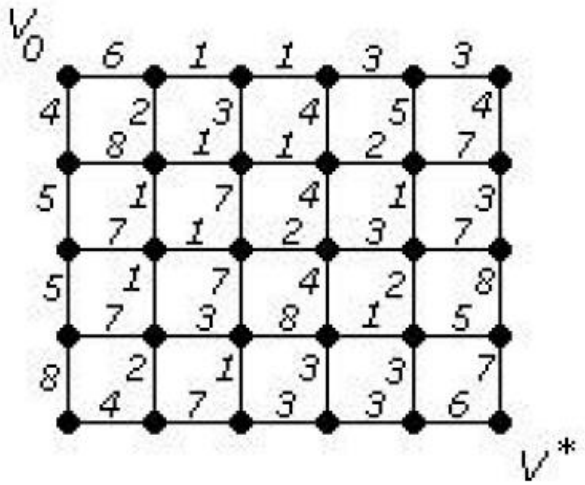
```
cout << "Way: " << endl;
for (int i = 0; i <= count; i++)
{
if (i != 0)
{
cout << "-> ";
}
cout << minmas[i] << " ";
}
cout << endl << "Leng: " << min;
cout << endl;

return 0;
}
```

Результати програми:

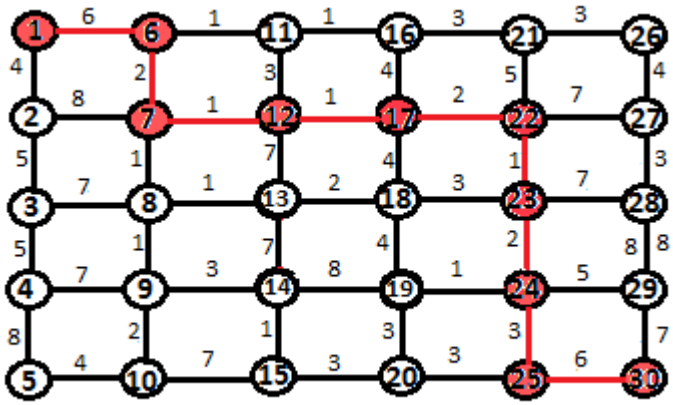
```
Way:
1 -> 5 -> 7 -> 8 -> 2 -> 4 -> 3 -> 6 -> 1
Leng: 15
```

Завдання №7

За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V 0 і V * .



За алгоритмом починаючи з вершини V0, вибираємо такі найближчі вершини, щоб довжина ланцюга до них була мінімальною. Результатом буде граф G:



Маршрут: V1⇨V6⇨V7⇨V12⇨V17⇨V22⇨V23⇨V24⇨V25⇨V30
Його ціна рівна 24.

V1=0	V11=7	V21=11
V2=4	V12=9	V22=12
V3=9	V13=10	V23=13
V4=14	V14=13	V24=15
V5=16	V15=14	V25=18
V6=6	V16=8	V26=14
V7=8	V17=10	V27=18
V8=9	V18=12	V28=20
V9=10	V19=16	V29=20
V10=12	V20=17	V30=24

Код програми:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 30              // Кількість вершин, розмір матриці

int Array[SIZE][SIZE];       // Масив матриці графа
int distance[SIZE];          // Масив мінімальних відстаней
int vertex[SIZE];            // Масив вершин

// Заповнення матриці нулями
void zeroArray()
{
   for(int i=0; i<SIZE; i++)
      for(int j=0; j<SIZE; j++)
         Array[i][j]=0;
}

// Запис дуг в матрицю
void enterEdges()
```

```
{
   printf("Enter edges:\n");
   int r, c, n;
   for(int i=0; i<49; i++)
   {
      scanf("%d %d %d", &r, &c, &n);
      Array[r-1][c-1]=n;
      Array[c-1][r-1]=n;
   }
}
// Вивід матриці графа
void printArray()
{
   for(int i=0; i<SIZE; i++)
   {
      for(int j=0; j<SIZE; j++)
      {
         printf("%d ", Array[i][j]);
      }
      printf("\n");
   } }
// Ініціалізація масивів відстаней та вершин
void initArray()
{
   for(int i=0; i<SIZE; i++)
   {
      distance[i]=10000;       // Всі відстані поки що є невизначеними, тому
10000
      vertex[i]=1;             // Всі вершини є необійденими, мають значення 1
   }
   distance[0]= 0;              // Відстань до першої вершини 0
}

// Вивід найкоротших відстаней до вершин 1-30
void printDistance()
{
   printf("\nShortest path to every vertex(1-30): \n");
      for(int i=0; i<SIZE; i++)
      {
         printf("%d) %d;  ", i+1, distance[i]);
         if((i+1)%5==0 && i!=0)
               printf("\n");
      }
```

```
}

// Головна функція
int main(void)
{
   int temp;                 // Тимчасова змінна для запису матриці графа
   int minindex;             // Змінна для ітерацій циклу основного алгоритму
   int min;

   zeroArray();              // Заповнення матриці нулями
   enterEdges();             // Ввід ребер та їхньої ваги
   printArray();             // Вивід матриці
   initArray();              // Ініціалізація масивів відстаней та вершин

   // Основний алгоритм
   do
   {
      minindex=10000;
      min=10000;
      for (int i=0; i<SIZE; i++)
      {
         if ((vertex[i]==1) && (distance[i]<min))        // Якщо вершину ще не
обійшли і вага менша за min
         {
            min=distance[i];  // Встановлюємо змінній min мінімальне
значення
            minindex=i;                                                    //
Визначаємо позицію мінімальної відстані
         }
      }

      if(min!=10000)                  // Якщо попередня умова виконалась
      {
         for(int i=0; i<SIZE; i++)
         {
            if(Array[minindex][i]>0)
            {
               temp=min+Array[minindex][i];
               if(temp<distance[i])
               {
                  distance[i]=temp; // Відстань записуємо в масив відстаней
               }
            }
```

```
            }
            vertex[minindex]=0;         // Помічаємо вершину пройденою
        }
    }
    while(minindex < 10000);       // Допоки не знайдемо всіх відстаней

    printDistance();                          // Вивід найкоротших відстаней

        // Відновлення шляху
    int ver[SIZE];                          // Масив відвіданих вершин
    int end = 29;                           // Індекс кінцевої вершини 30-1=29
    ver[0] = end + 1;                       // Перший елемент -- остання вершина
    int k = 1;                              // Індекс попередньої вершини
    int weight = distance[end];            // Вага кінцевої вершини

    while (end > 0) // Поки не дійшли до початкової вершини
    {
        for(int i=0; i<SIZE; i++) // Переглядаємо всі вершини
            if (Array[end][i] != 0)   // Якщо зв'язок є
            {
                int temp = weight - Array[end][i]; // Визначаємо вагу шляху з
попередньої вершини
                if (temp == distance[i]) // Якщо вага співпала з вирахуваною
                {                           // Значить, з цієї вершини був здійснений
перехід
                    weight = temp;       // Зберігаємо нову вагу
                    end = i;         // Зберігаємо попередню вершину
                    ver[k] = i + 1;         // І записуємо її в масив
                    k++;
                }
            }
    }

    // Вивід шляху (початкова вершина опинилась в кінці масиву з k
елементів)
    printf("\nOutput of the shortest path:\n");
    for(int i = k-1; i>=0; i--)
        printf("%3d ", ver[i]);

    scanf("%d", &temp);
    return 0;
}
```

Результати програми:

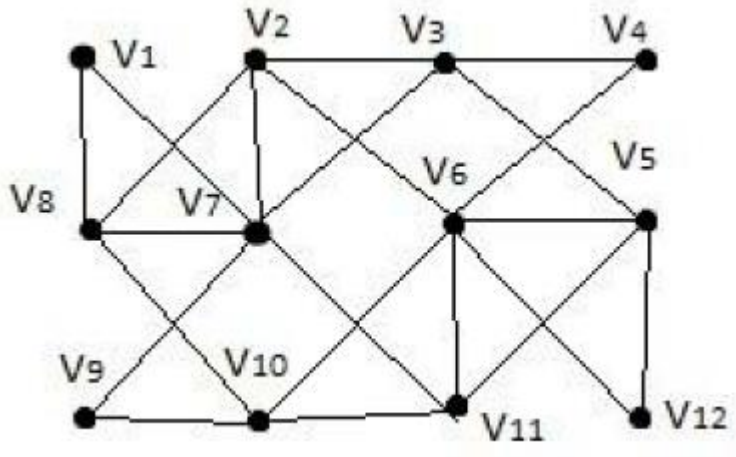
```
0 4 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 5 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 5 0 5 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 5 0 8 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 8 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 2 0 0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

Shortest path to every vertex(1-30):
1) 0;  2) 4;  3) 9;  4) 14;  5) 16;
6) 6;  7) 8;  8) 9;  9) 10;  10) 12;
11) 7;  12) 9;  13) 10;  14) 13;  15) 14;
16) 8;  17) 10;  18) 12;  19) 16;  20) 17;
21) 11;  22) 12;  23) 13;  24) 15;  25) 18;
26) 14;  27) 18;  28) 20;  29) 20;  30) 24;

Output of the shortest path:
  1   6   7  12  17  22  23  24  25  30
```

Завдання №8

Знайти ейлеровий цикл в ейлеровому графі двома методами:

а) Флері;

б) елементарних циклів**.**



б) Виділимо прості цикли у графі

2 8 10 6

7 3 5 11

1 8 7

2 8 7

2 4 6
3 2 7
7 9 11
10 11 6
5 6 12
5 6 11
Починаємо об’єднувати цикли.
Шуканий Ейлеровий цикл: 1 7 11 5 3 4 6 5 12 6 11 10 6 2 7 8 10 9 7 3 2 8 1
б) Флері
1 7 9 10 5 11 4 10 6 8 9 5 4 2 6 7 11 6 1 5 3 2 1
Код програми:

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

#define N 13
#define STACK_SIZE 200


int G[N][N] =
{
   {0,0,0,0,0,0,1,1,0,0,0,0},
   {0,0,1,0,0,1,1,1,0,0,0,0},
   {0,1,0,1,1,0,1,0,0,0,0,0},
   {0,0,1,0,0,1,0,0,0,0,0,0},
   {0,0,1,0,0,1,0,0,0,0,1,1},
   {0,1,0,1,1,0,0,0,0,1,1,1},
   {1,1,1,0,0,0,0,1,1,0,1,0},
   {1,1,0,0,0,0,1,0,0,1,0,0},
   {0,0,0,0,0,0,1,0,0,1,0,0},
   {0,0,0,0,0,1,0,1,1,0,1,0},
   {0,0,0,0,1,1,1,0,0,1,0,0},
   {0,0,0,0,1,1,0,0,0,0,0,0}
};

int k;
int Stack[STACK_SIZE];

void Search(int v)
{
   int i;
   for(i = 0; i < N; i++)
      if(G[v][i])
      {
```

```
            G[v][i] = G[i][v] = 0;
            Search(i);
        }
    Stack[++k] = v;
}



int main()
{
    int T, p, q, s;
    int j, vv;

    T = 1;
    for(p = 0; p < N; p++)
    {
        s = 0;
        for(q = 0; q < N; q++)
        {
            s += G[p][q];
        }
        if(s%2) T = 0;
    }
    k = -1;
    printf("start vertex: "); scanf("%d", &vv);
    if(T)
    {
        Search (vv);
        for(j = 0; j <= k; j++)
            printf("%d ", Stack[j]);
    }
    else
        printf("not Eulerian graph\n");
    return 0;
}
```

Результати програми:

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

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

((x∨¬ z)^(¬ y))∨((x∨¬ z)^z)

(¬(¬x^z)^ (¬ y))∨(x^z)∨(¬z^z)

¬(¬xzy)∨xz

x¬z¬y∨xz