МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ
“ЛЬВІВСЬКА ПОЛІТЕХНІКА

Кафедра систем штучного інтелекту

Розрахункова робота

З дисципліни

“Дискретна математика”

Виконав:

студент групи КН-112

Думич Іван

Викладач:

Мельникова Н.І.

Львів – 2019 р.

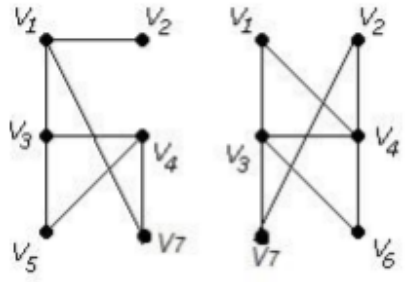
ІНДИВІДУАЛЬНІ ЗАВДАННЯ

24 варіант

Завдання № 1

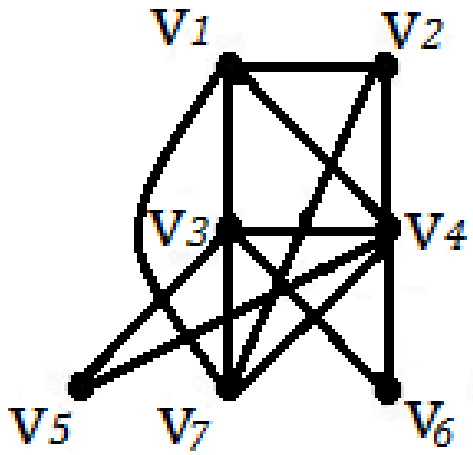
Виконати наступні операції над графами: 1) знайти доповнення до першого графу, 2) об’єднання графів, 3) кільцеву сумму G1 та G2 (G1+G2), 4) розмножити вершину у другому графі, 5) виділити підграф А - що скадається з 3-х вершин в G1 6) добуток графів.

24)

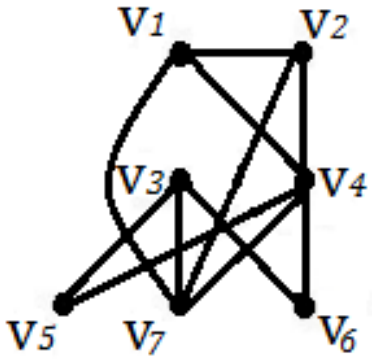


1) Знайти доповнення до першого графу.

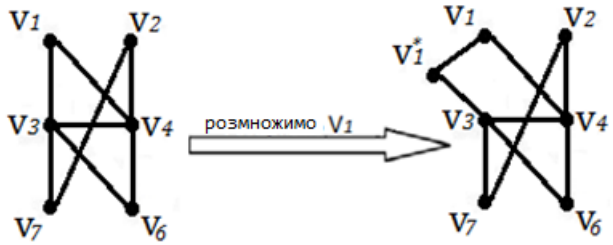
2) Об’єднання графів.



3) Кільцеву суму G1 та G2 (G1+G2).

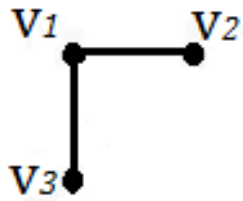


4) Розмножити вершину у другому графі

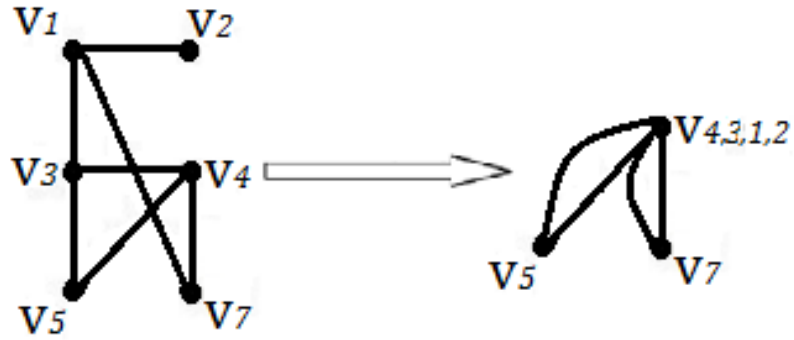


5) Виділити підграф A, що складається з 3-х вершин в G1 і знайти стягнення A в G1 (G1\ A).

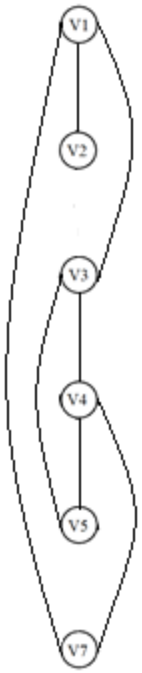
Підграф A:



Стягнення A в G1:

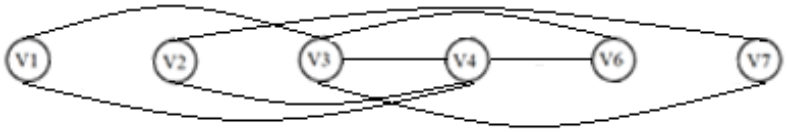


6) Добуток графів.

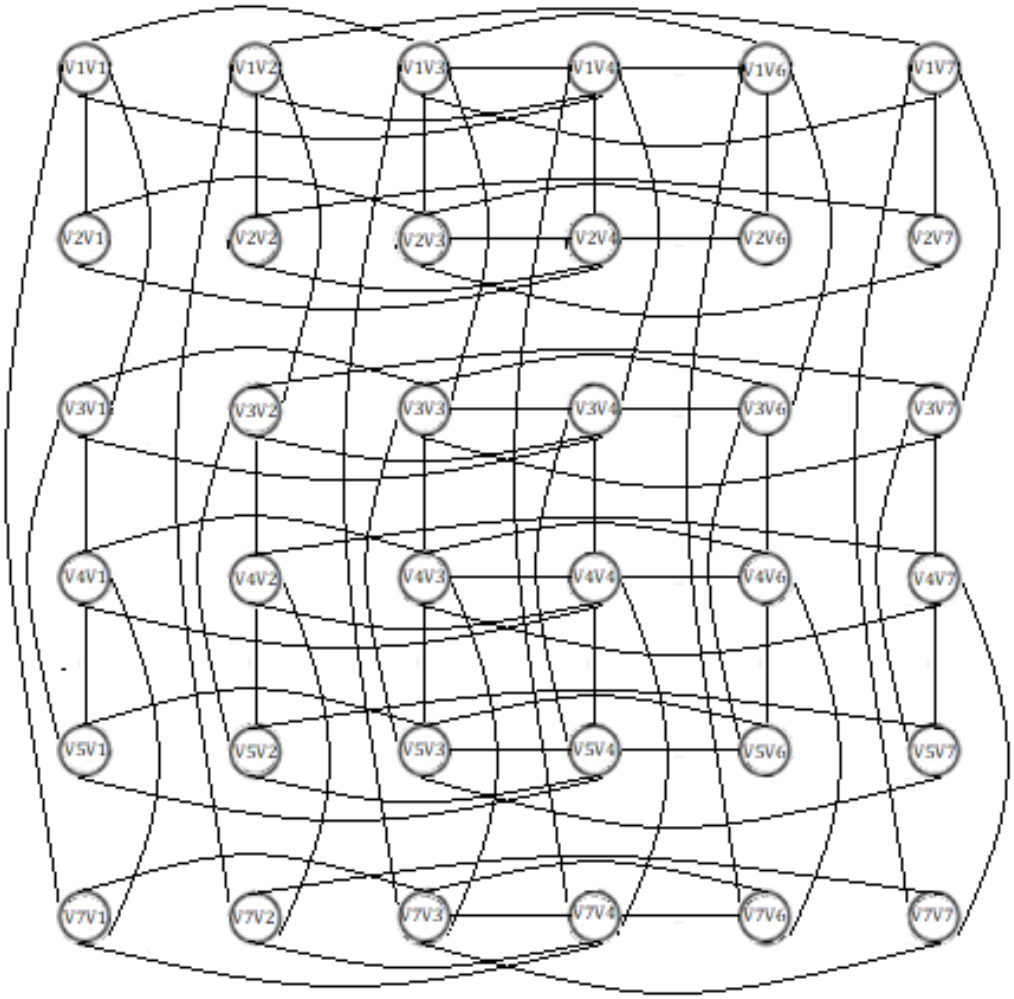


G1

×



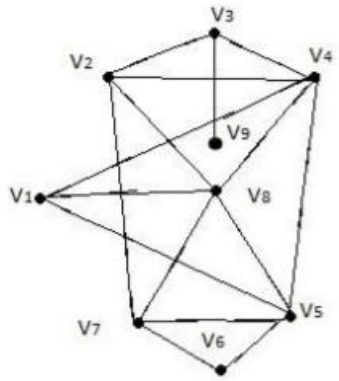
G2



Завдання № 2

Скласти таблицю суміжності для неографа.

24)



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

Завдання № 3

Для графа з другого завдання знайти діаметр.

Побудуємо матрицю відстаней між вершинами

	V1	V2	V3	V4	V5	V6	V7	V8	V9
V1	0	1	2	1	1	2	2	1	3
V2		0	1	1	2	3	1	1	2
V3			0	1	2	3	2	2	1
V4				0	1	2	2	1	2
V5					0	1	1	1	3
V6						0	1	2	4
V7							0	1	3
V8								0	3
V9									0

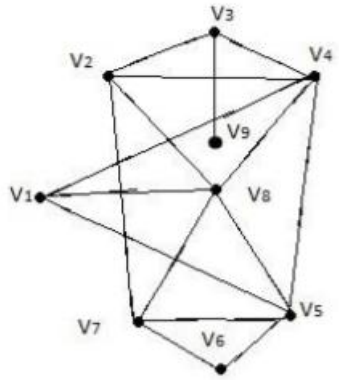
Максимальне число 4. Отже Діаметр графа дорівнює 4.

Відповідь: 4

Завдання № 4

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

24)



Вершина	BFS-номер	Вміст черги
V_1	1	V_1
V_4	2	V_1V_4
V_8	3	$V_1V_4V_8$
V_5	4	$V_1V_4V_8V_5$
-	-	$V_4V_8V_5$
V_3	5	$V_4V_8V_5V_3$
V_2	6	$V_4V_8V_5V_3V_2$
V_7	7	$V_4V_8V_5V_3V_2V_7$
-	-	$V_8V_5V_3V_2V_7$
-	-	$V_5V_3V_2V_7$
V_6	8	$V_5V_3V_2V_7V_6$
-	-	$V_3V_2V_7V_6$
V_9	9	$V_3V_2V_7V_6V_9$
-	-	$V_2V_7V_6V_9$
-	-	$V_7V_6V_9$
-	-	V_6V_9
-	-	V_9
-	-	$\emptyset$

Програмна реалізація:

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

class Graph
{
  int V;
  list<int>* adj;

public:

  Graph(int V);
  void addEdge(int v, int w);
  void BFS(int s);
};

Graph::Graph(int V)
{
  this->V = V;
  adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
  adj[v].push_back(w);
  adj[w].push_back(v);
}

void Graph::BFS(int s)
{
  bool* visited = new bool[V];
  for (int i = 0; i < V; i++)
    visited[i] = false;

  list<int> queue;

  visited[s] = true;
  queue.push_back(s);

  list<int>::iterator i;

  while (!queue.empty())
  {
    s = queue.front();
    cout <<"V" <<s+1 << " ";
    queue.pop_front();

    for (i = adj[s].begin(); i != adj[s].end(); ++i)
    {
```

```
      if (!visited[*i])
      {
        visited[*i] = true;
        queue.push_back(*i);
      }
    }
  }
}
int main()
{
  int a, b, c;
  cout << "Number of vertex:";
  cin >> c;
  cout << "Ribs:"<<endl;
  Graph g(c);
  while (true) {
    cin >> a;
    if (a == -1) { break; }
    cin >> b;
    g.addEdge(a, b);
  }


  g.BFS(0);

  return 0;
}
```

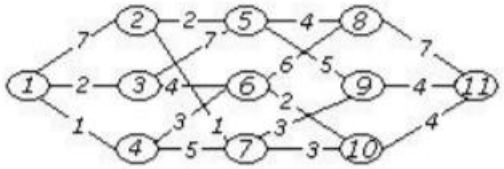
Результат:

```
Number of vertex:9
Ribs:
0 3
0 7
0 4
1 2
1 6
1 7
2 3
2 8
3 1
2 3
3 4
3 7
4 5
4 6
4 7
5 6
6 7
-1
Start from vertex: 1
V1 V4 V8 V5 V3 V2 V7 V6 V9
```

Завдання № 5

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

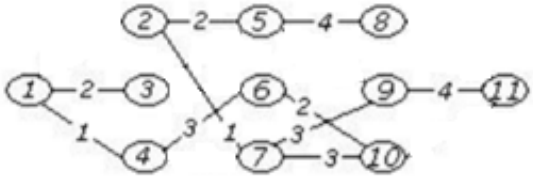
24)



Методом Прима:

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

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

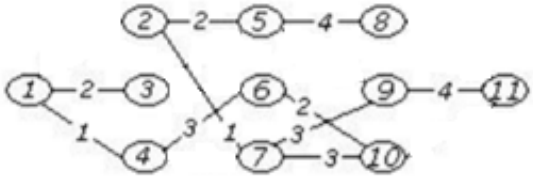


Методом Краскала:

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

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

Мінімальне остове дерево має вигляд:



Програмна реалізація алгоритму Прима:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{
  setlocale(LC_ALL, "Ukrainian");
```

```
int v, count = 0, min = 0, k, t;
bool check = false;
cout << "Кількість вершин графа : ";
cin >> v;
int* tops = new int[v];
int** graph = new int* [v];
int** ribs = new int* [v - 1];

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

for (int j = 0; j < v - 1; j++) {
  ribs[j] = new int[2];
}

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

  }
}
//////Будуємо дерево, що включає в себе одну вершину

tops[count] = 1;
count++;

///Знаходження мінімального кістякового дерева
for (int i = 0; count < v; i++) {
  for (int j = 0; j < count; j++) {
    for (int a = 0; a < v; a++) {
      for (int m = 0; m < count; m++) {
        if (tops[m] == a + 1) {
          check = true;
        }
      }
      if (check) { check = false; continue; }
      if (min == 0 && graph[tops[j] - 1][a] > 0) {
        min = graph[tops[j] - 1][a];
        k = ribs[count - 1][0] = tops[j]; t = ribs[count - 1][1] = a + 1;
        continue;
      }
      if (graph[tops[j] - 1][a] > 0 && graph[tops[j] - 1][a] < min) {
        min = graph[tops[j] - 1][a];
        k = ribs[count - 1][0] = tops[j]; t = ribs[count - 1][1] = a + 1;
      }
    }


  }

  graph[k - 1][t - 1] = 0; graph[t - 1][k - 1] = 0;
```

```
    tops[count] = t;
    count++;
    min = 0;


  }
  /////Результат
  cout << "V: { ";
  for (int j = 0; j < v; j++) {
    cout << tops[j] << ", ";
  }
  cout << "}";
  cout << endl << "E:{ ";
  for (int j = 0; j < v - 1; j++) {
    cout << "( " << ribs[j][0] << ", " << ribs[j][1] << " ), ";
  }
  cout << "}";

  return 0;
}
```

Результат:

```
Кількість вершин графа : 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 5 3 0 0 0 4
0 0 0 0 0 0 0 7 4 4 0
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 ), ( 10, 11 ), ( 5, 8 ), }
```

Програмна реалізація Краскала:

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

#define graph_edge pair<int,int>

class Graph {
private:
  int V;
  vector<pair<int, graph_edge>> G;
  vector<pair<int, graph_edge>> T;
  int* parent;

public:
```

```
  Graph(int V);
  void AddEdge(int u, int v, int wt);
  int find_set(int i);
  void union_set(int u, int v);
  void kruskal_algorithm();
  void display_mst();
};
Graph::Graph(int V) {
  parent = new int[V];

  for (int i = 0; i < V; i++)
    parent[i] = i;

  G.clear();
  T.clear();
}
void Graph::AddEdge(int u, int v, int wt) {
  G.push_back(make_pair(wt, graph_edge(u, v)));
}
int Graph::find_set(int i) {
  if (i == parent[i])
    return i;
  else
    return find_set(parent[i]);
}
void Graph::union_set(int u, int v) {
  parent[u] = parent[v];
}
void Graph::kruskal_algorithm() {
  int i, uSt, vEd;
  sort(G.begin(), G.end());
  for (i = 0; i < G.size(); i++) {
    uSt = find_set(G[i].second.first);
    vEd = find_set(G[i].second.second);
    if (uSt != vEd) {
      T.push_back(G[i]);
      union_set(uSt, vEd);
    }
  }
}
void Graph::display_mst() {

  cout << endl << "E:{ ";
  for (int i = 0; i < T.size(); i++) {
    cout <<"(" <<T[i].second.first+1 << ", " << T[i].second.second+1 << "), ";

  }
  cout << "}";

}
int main() {

  int a, b, c, w;
```

```
  cout << "Number of vertex:";
  cin >> c;
  cout << "Ribs:" << endl;
  Graph g(c);
  while (true) {
    cin >> a;
    if (a == -1) { break; }
    cin >> b;
    cin >> w;
    g.AddEdge(a-1, b-1, w);
  }

  g.kruskal_algorithm();
  g.display_mst();
  return 0;
}
```

Результат:

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

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

Завдання № 6

Розв’язати задачу комівояжера для повного 8-ми вершинного 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)Вихідна вершина: 1

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

Довжина шляху : 20

2)Вихідна вершина: 2

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

Довжина шляху : 17

3)Вихідна вершина: 3

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

Довжина шляху : 17

4)Вихідна вершина: 4

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

Довжина шляху : 17

5)Вихідна вершина: 5

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

Довжина шляху : 18

6)Вихідна вершина: 6

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

Довжина шляху : 17

7)Вихідна вершина: 7

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

Довжина шляху : 17

8)Вихідна вершина: 8

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

Довжина шляху : 19

Відповідь:

Оптимальні шляхи:

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

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

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

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

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

Програмна реалізація:

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

using namespace std;

class Komivoiser {

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

        for (int j = 0; j < v; j++) {
```

```
            graph[j] = new int[v];
        }

        cout << "Вага ребер : " << endl;

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

            }
        }


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

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

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

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

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

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

    return 0;
}
```

Результат:

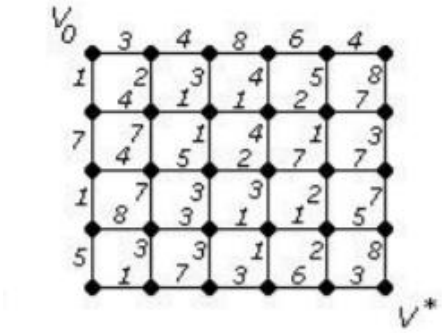
```
Кількість вершин : 8
Вага ребер :
0 4 6 5 1 6 5 1
4 0 5 1 2 3 5 4
6 5 0 4 4 6 5 4
5 1 4 0 5 5 5 1
1 2 4 5 0 1 6 5
6 3 6 5 1 0 2 1
5 5 5 5 6 2 0 7
1 4 4 1 5 1 7 0
```

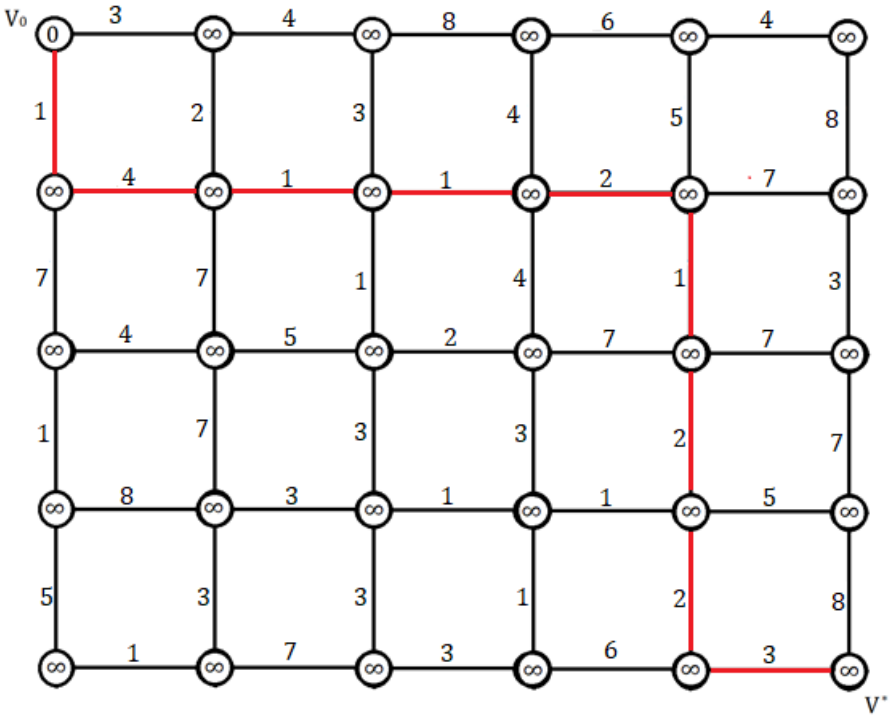
```
Оптимальні шляхи:
Шлях: 1->5->2->4->3->7->6->8->1 вага: 17
Шлях: 1->5->6->7->3->2->4->8->1 вага: 17
Шлях: 1->8->4->2->3->7->6->5->1 вага: 17
Шлях: 1->8->6->7->3->4->2->5->1 вага: 17
Шлях: 2->3->7->6->5->1->8->4->2 вага: 17
Шлях: 2->4->3->7->6->8->1->5->2 вага: 17
Шлях: 2->4->8->1->5->6->7->3->2 вага: 17
Шлях: 2->5->1->8->6->7->3->4->2 вага: 17
Шлях: 3->2->4->8->1->5->6->7->3 вага: 17
Шлях: 3->4->2->5->1->8->6->7->3 вага: 17
Шлях: 3->7->6->5->1->8->4->2->3 вага: 17
Шлях: 3->7->6->8->1->5->2->4->3 вага: 17
Шлях: 4->2->3->7->6->5->1->8->4 вага: 17
Шлях: 4->2->5->1->8->6->7->3->4 вага: 17
Шлях: 4->3->7->6->8->1->5->2->4 вага: 17
Шлях: 4->8->1->5->6->7->3->2->4 вага: 17
Шлях: 5->1->8->4->2->3->7->6->5 вага: 17
Шлях: 5->1->8->6->7->3->4->2->5 вага: 17
Шлях: 5->2->4->3->7->6->8->1->5 вага: 17
Шлях: 5->6->7->3->2->4->8->1->5 вага: 17
Шлях: 6->5->1->8->4->2->3->7->6 вага: 17
Шлях: 6->7->3->2->4->8->1->5->6 вага: 17
Шлях: 6->7->3->4->2->5->1->8->6 вага: 17
Шлях: 6->8->1->5->2->4->3->7->6 вага: 17
Шлях: 7->3->2->4->8->1->5->6->7 вага: 17
Шлях: 7->3->4->2->5->1->8->6->7 вага: 17
Шлях: 7->6->5->1->8->4->2->3->7 вага: 17
Шлях: 7->6->8->1->5->2->4->3->7 вага: 17
Шлях: 8->1->5->2->4->3->7->6->8 вага: 17
Шлях: 8->1->5->6->7->3->2->4->8 вага: 17
Шлях: 8->4->2->3->7->6->5->1->8 вага: 17
Шлях: 8->6->7->3->4->2->5->1->8 вага: 17
```

Завдання № 7

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

24)





Програмна реалізація:

```
#include <iostream>
#define inf 1000000
using namespace std;
int min_top(int** arr,int v) {
    int m=0;
    for (int i = 0; i < v; i++) {
        if (arr[i][1]) {
            m = i; break;
        }
    }

    for (int i = 1; i < v; i++) {
        if (arr[m][0] >= arr[i][0] && arr[i][1]==1) {
            m = i;
        }
    }
    return m;
}
```

```
int main()
{
    setlocale(LC_ALL, "Ukrainian");

    int a, b, c;
    int v = 0;
    cout << "Кількість вершин графа : ";
    cin >> v;
    int** graph = new int* [v];

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

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

        }
    }
    cout << "Введіть вагу ребер графа : " << endl;

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



    int p;
    int** tops = new int*[v];
    for (int j = 0; j < v; j++) {
        tops[j] = new int[2];
    }
    int* tops_path = new int[v];

    cout << "Вихідна вершина: ";
    cin >> p;

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

```
    int m;

    for (int i = 0; i < v; i++) {
        m = min_top(tops, v);
        for (int j = 0; j < v; j++) {
            if (graph[m][j]) {
                if (tops[j][0] > tops[m][0] + (graph[m][j])) {
                    tops[j][0] = tops[m][0] + (graph[m][j]);
                    tops_path[j] = m;
                }
            }

        }
        tops[m][1] = 0;

    }

    ////шлях
    cout << "Введіть потрібну вершину: ";
    int k; cin >> k;
    cout << "Мінімальний шлях: ";
    cout << tops[k-1][0];
    cout << endl << k <<" <-- ";
    k--;
    for (int a = 0; tops_path[k] != p-1; a++) {
        cout << tops_path[k]+1 <<" <-- ";
        k = tops_path[k];
    }
    cout << p << endl;


    return 0;
}
```

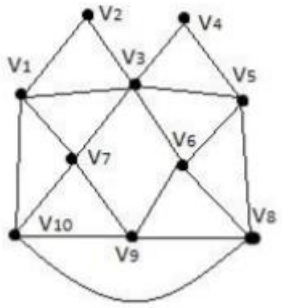
Результат:

```
Кількість вершин графа : 30
Введіть вагу ребер графа :
1 2 3
1 7 1
2 3 4
2 8 2
3 4 8
3 9 3
4 5 6
4 10 4
5 6 4
5 11 5
6 12 8
7 8 4
7 13 7
8 9 1
8 14 7
9 10 1
9 15 1
10 11 2
10 16 4
11 12 7
11 17 1
12 18 3
13 14 4
13 19 1
14 15 5
14 20 7
15 16 2
15 21 3
16 17 7
16 22 3
17 18 7
17 23 2
18 24 7
19 20 8
19 25 5
20 21 3
20 26 3
21 22 1
21 27 3
22 23 1
22 28 1
23 24 5
23 29 2
24 30 8
25 26 1
26 27 7
27 28 3
28 29 6
29 30 3
-1
Вихідна вершина: 1
Введіть потрібну вершину: 30
Мінімальний шлях: 17
30 <-- 29 <-- 23 <-- 17 <-- 11 <-- 10 <-- 9 <-- 8 <-- 7 <-- 1
```

Завдання № 8

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

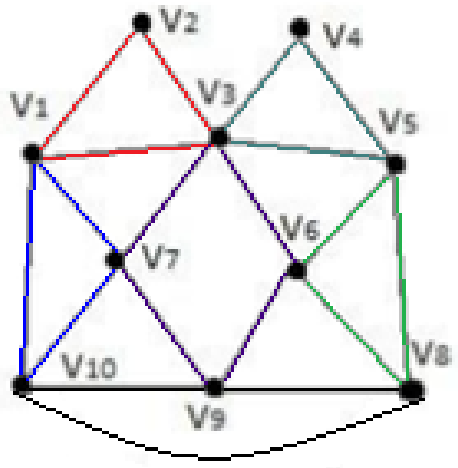
24)



а) $V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_3 \rightarrow V_1 \rightarrow V_7 \rightarrow V_3 \rightarrow V_6 \rightarrow V_5 \rightarrow V_8 \rightarrow V_6 \rightarrow V_9 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_9 \rightarrow V_8 \rightarrow V_{10} \rightarrow V_1$

б)

- $V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_1$
- $V_1 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_1$
- $V_1 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_7 \rightarrow V_9 \rightarrow V_6 \rightarrow V_3 \rightarrow V_1$
- $V_1 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_7 \rightarrow V_9 \rightarrow V_6 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_3 \rightarrow V_1$
- $V_1 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_7 \rightarrow V_9 \rightarrow V_6 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6 \rightarrow V_8 \rightarrow V_5 \rightarrow V_3 \rightarrow V_1$
- $V_1 \rightarrow V_7 \rightarrow V_{10} \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_7 \rightarrow V_9 \rightarrow V_6 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6 \rightarrow V_8 \rightarrow V_9 \rightarrow V_{10} \rightarrow V_8 \rightarrow V_5 \rightarrow V_3 \rightarrow V_1$



Реалізація алгоритму Флері

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

```cpp
#include <stack>

using namespace std;

int main()
{
    setlocale(LC_ALL, "Ukrainian");

    int v = 0;
    cout << "Кількість вершин графа : ";
    cin >> v;
    cout << "Введіть матрицю суміжності:"<<endl;
    int** graph = new int* [v];

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

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

    int m, ver;
    Stack.push_back(1);
    while (!Stack.empty()) {
        m = 0;
        ver = Stack[Stack.size() - 1];
        for (int i = 0; i < v; i++) {
            if (graph[ver-1][i]) {
                m = i+1;
                graph[ver - 1][i] = 0;
                graph[i][ver - 1] = 0;
                Stack.push_back(m);
                break;
            }
        }
        if (m == 0 ) {
            path.push_back(ver);
            Stack.pop_back();
        }


    }
    for (int i = path.size() - 1; i > 0; i--) {

        cout << path[i] <<"->";
    }
    cout << path[0];
```

```
    return 0;
}
```

Результат:

```
Кількість вершин графа : 10
Введіть матрицю суміжності:
0 1 1 0 0 0 1 0 0 1
1 0 1 0 0 0 0 0 0 0
1 1 0 1 1 1 1 0 0 0
0 0 1 0 1 0 0 0 0 0
0 0 1 1 0 1 0 1 0 0
0 0 1 0 1 0 0 1 1 0
1 0 1 0 0 0 0 0 1 1
0 0 0 0 1 1 0 0 1 1
0 0 0 0 0 1 1 1 0 1
1 0 0 0 0 0 1 1 1 0
1->2->3->1->7->3->4->5->3->6->5->8->6->9->7->10->8->9->10->1
```

Завдання №9

Спростити формули (привести їх до скороченої ДНФ)

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

Отримаємо ДДНФ $\overline{(x(y\bar{z} \vee x\bar{z}))}$

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

Знайти скорочену ДНФ функції:
$\overline{xyz} \vee \overline{xy}z \vee \bar{x}y\bar{z} \vee \bar{x}yz \vee x\bar{y}z \vee xyz = \overline{xy}(\bar{z} \vee z) \vee \bar{x}y(\bar{z} \vee z) \vee xz(\bar{y} \vee y) = \overline{xy} \vee \bar{x}y \vee xz = \bar{x} \vee xz = \bar{x} \vee z$

Відповідь: $\bar{x} \vee z$