**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.

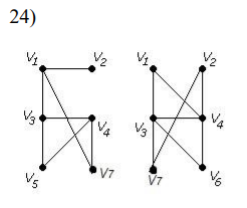
Львів – 2019 р.

**ІНДИВІДУАЛЬНІ ЗАВДАННЯ**

**24 варіант**

**Завдання № 1**

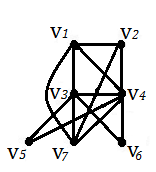
Виконати наступні операції над графами: 1) знайти доповнення до першого графу, 2) об’єднання графів, 3) кільцеву сумму G1 та G2 (G1+G2), 4) розмножити вершину у другому графі, 5) виділити підграф А - що скадається з 3-х вершин в G1 6) добуток графів.

****

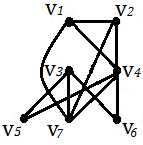
1) Знайти доповнення до першого графу.



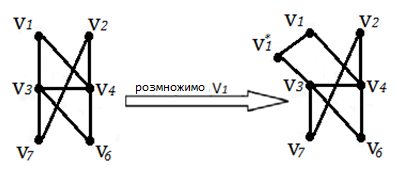
2) Об’єднання графів.



3) Кільцеву суму G1 та G2 (G1+G2).

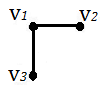


4) Розмножити вершину у другому графі

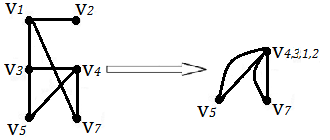


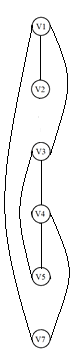
5) Виділити підграф А, що складається з 3-х вершин в G1 і знайти стягнення А в G1 (G1\ A).

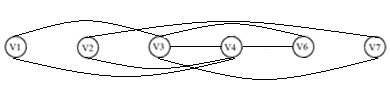
Підграф A:



Cтягнення А в G1:



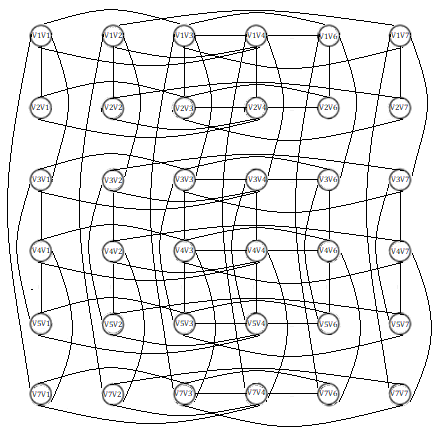
6) Добуток графів.

****



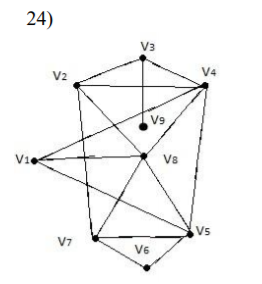
G2

G1



**Завдання № 2**

Скласти таблицю суміжності для неографа.



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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

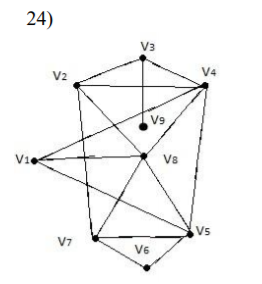
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 |
| V1 | 0 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 3 |
| V2 |  | 0 | 1 | 1 | 2 | 3 | 1 | 1 | 2 |
| V3 |  |  | 0 | 1 | 2 | 3 | 2 | 2 | 1 |
| V4 |  |  |  | 0 | 1 | 2 | 2 | 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).



|  |  |  |
| --- | --- | --- |
| Вершина | BFS-номер | Вміст черги |
|  | 1 |  |
|  | 2 |  |
|  | 3 |  |
|  | 4 |  |
| **-** | **-** |  |
|  | 5 |  |
|  | 6 |  |
|  | 7 |  |
| **-** | **-** |  |
| **-** | **-** |  |
|  | 8 |  |
| **-** | **-** |  |
|  | 9 |  |
| **-** | **-** |  |
| **-** | **-** |  |
| **-** | **-** |  |
| **-** | **-** |  |
| **-** | **-** | ∅ |

**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);

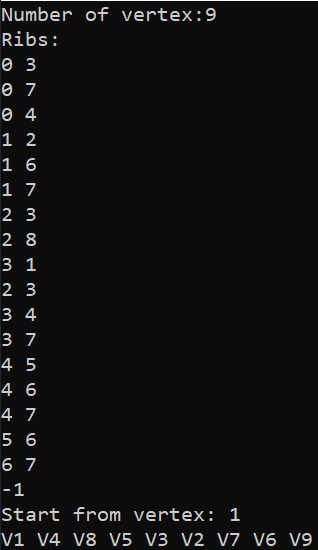
  }

  g.BFS(0);

  return 0;

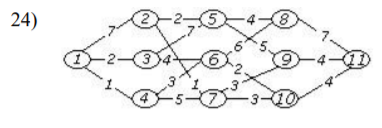
}

**Результат:**



**Завдання № 5**

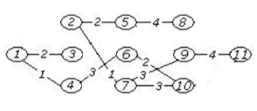
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.



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)}

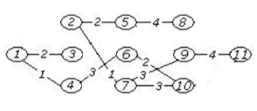


Методом Краскала:

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 << "Кiлькiсть 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] << " ), ";

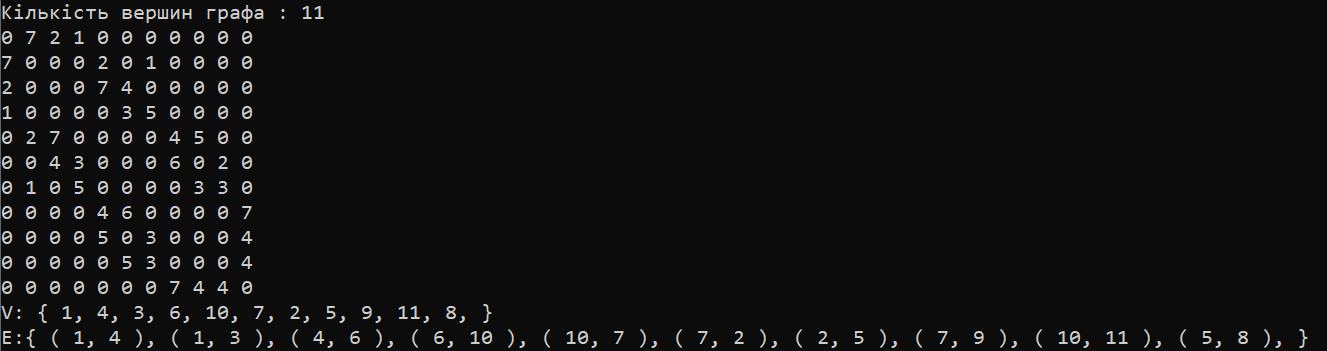
  }

  cout << "}";

  return 0;

}

**Результат:**



**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);

  }

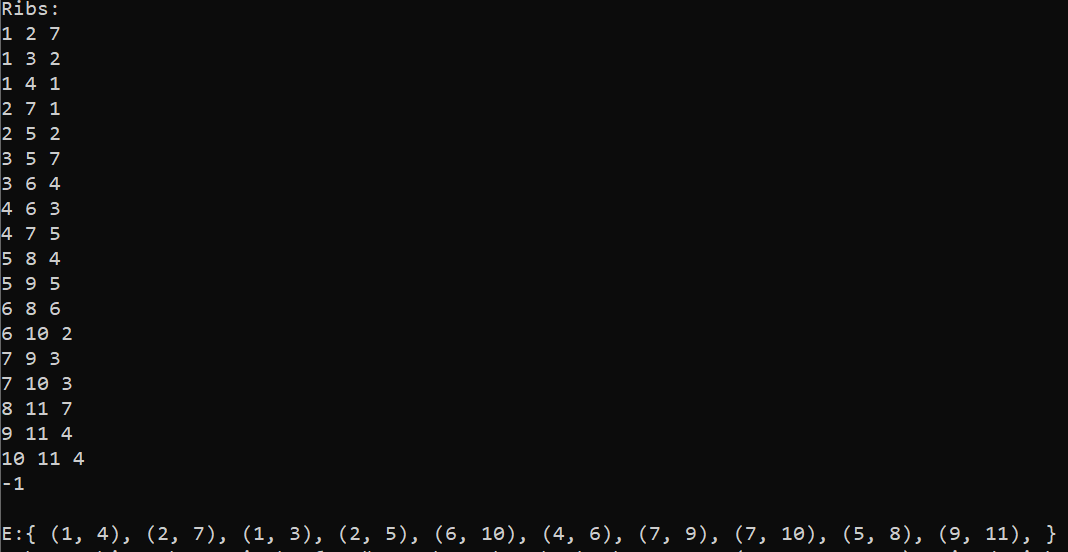
  g.kruskal\_algorithm();

  g.display\_mst();

  return 0;

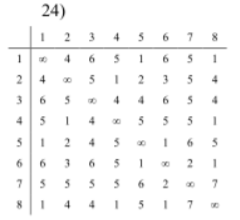
}

**Результат:**

****

**Завдання № 6**

Розв’язати задачу комiвояжера для повного 8-ми вершинного графа методом «iди у найближчий», матриця вагiв якого має вигляд:



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 << "Кiлькiсть 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 << "Оптимальнi 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;

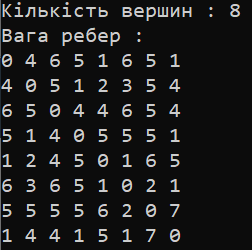
            }

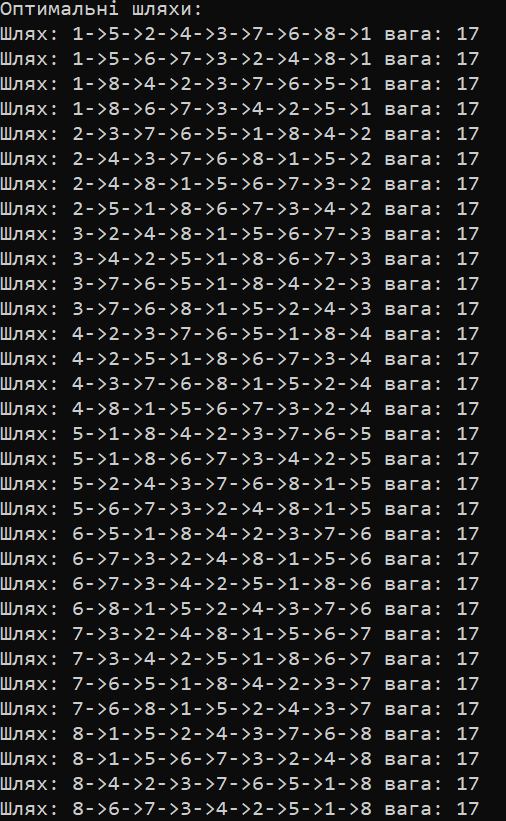
        }

    return 0;

}

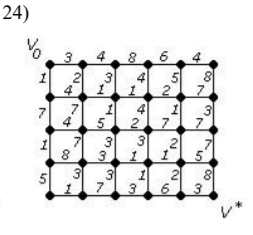
**Результат:**

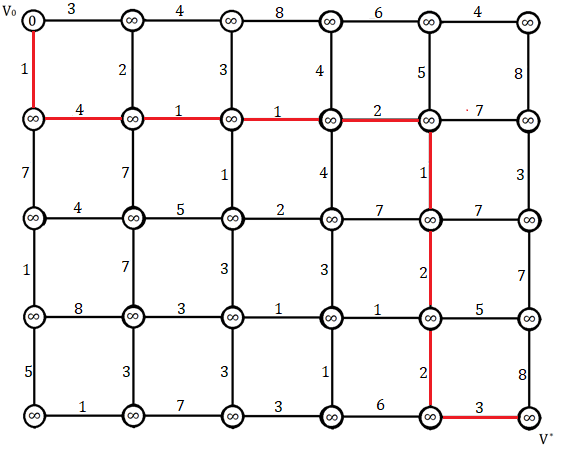




**Завдання № 7**

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





**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 << "Кiлькiсть 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 << "Введiть 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 << "Вихiдна 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 << "Введiть потрiбну вершину: ";

    int k; cin >> k;

    cout << "Мiнiмальний 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];

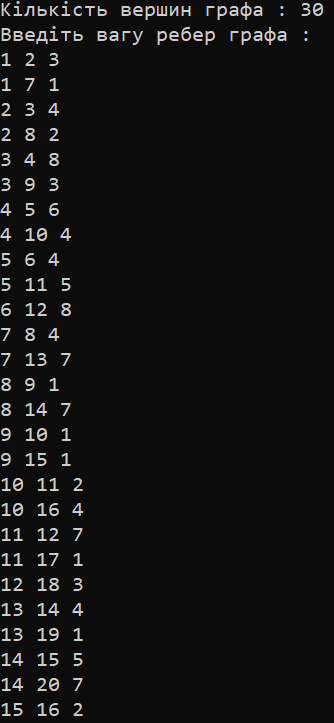
    }

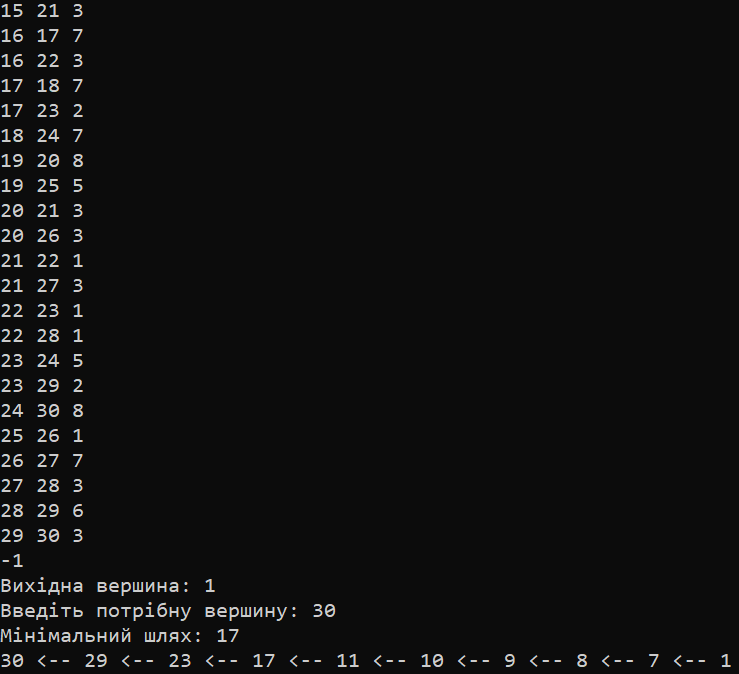
    cout << p << endl;

    return 0;

}

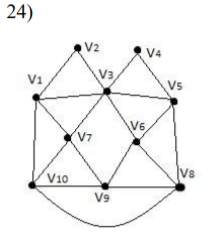
**Результат:**





**Завдання № 8**

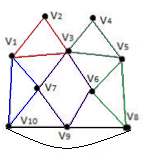
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.



1. **→ → → → → → → → → → → → → → → → → → →**

**б)**

* **→ → →**
* **→ → → → → →**
* **→ → → → → → → →→→**
* **→ → → → → → → →→→ → → →**
* **→ → → → → → → →→→ → → → → → →**
* **→ → → → → → → →→→ → → → → → → → → →**

****

**Реалізація алгоритму Флері**

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

int main()

{

    setlocale(LC\_ALL, "Ukrainian");

    int v = 0;

    cout << "Кiлькiсть вершин графа : ";

    cin >> v;

    cout << "Введiть матрицю сумiжностi:"<<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] <<"->";

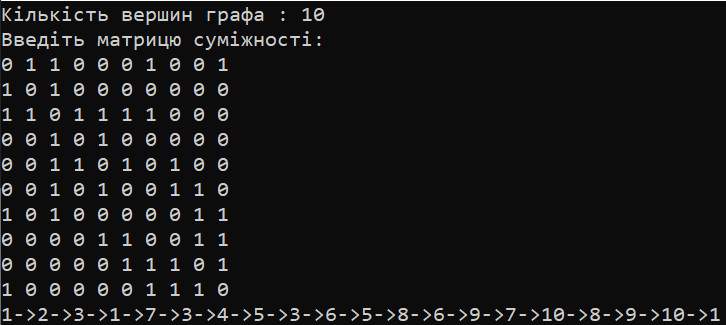
    }

    cout << path[0];

    return 0;

}

**Результат:**

****

**Завдання №9**

Спростити формули (привести їх до скороченої ДНФ)



Отримаємо ДДНФ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | y | z |  | y | x | y∨ x | x(y∨ x) |  |
| 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: yyz

Знайти скорочену ДНФ функції:

yyz = () y() xz() = y xz = xz = z

**Відповідь:** z