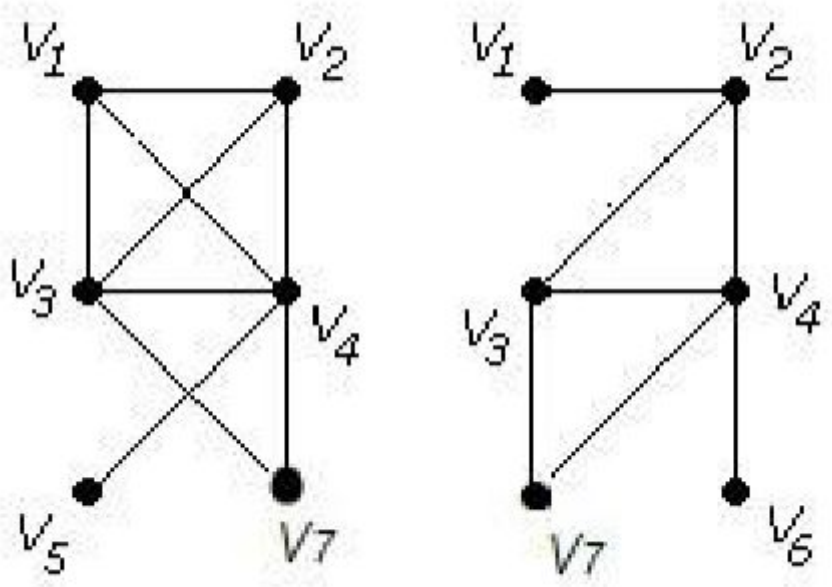
Варіант 15

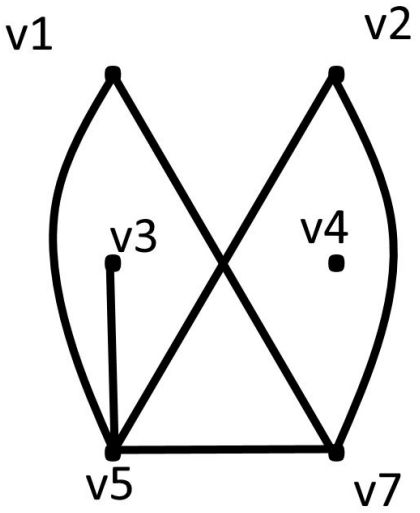
Завдання № 1

Виконати наступні операції над графами:

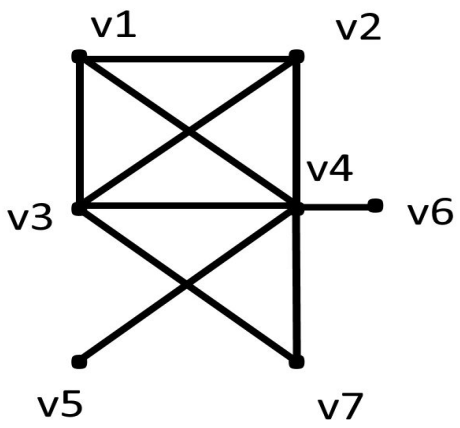
1) знайти доповнення до першого графу,
2) об’єднання графів,
3) кільцеву суму G1 та G2 (G1+G2),
4) розмножити вершину у другому графі,
5) виділити підграф А - що складається з 3-х вершин в G1 6) добуток графів.



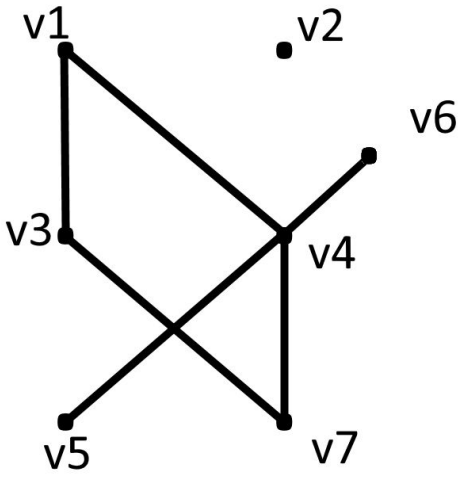
1)



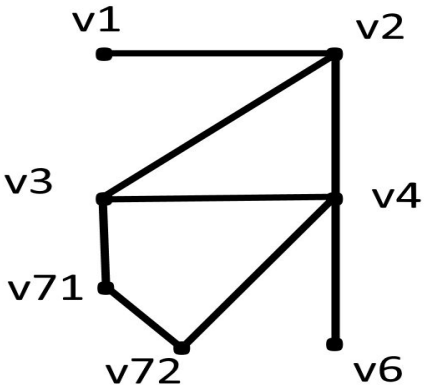
2)



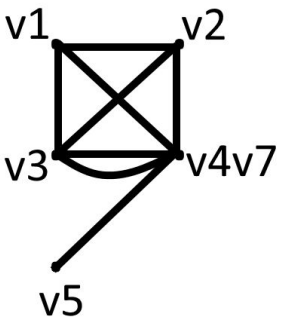
3)

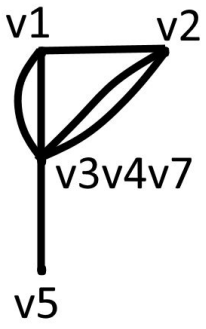


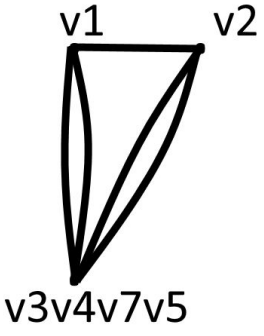
4)

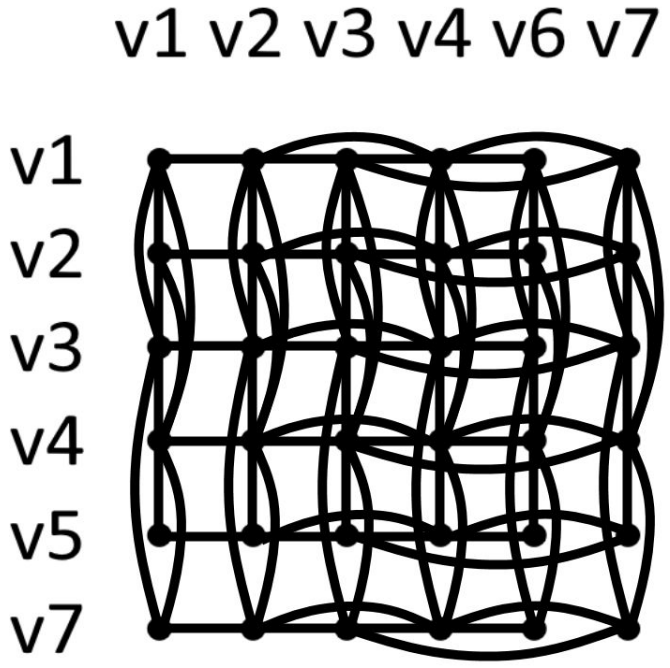


5)

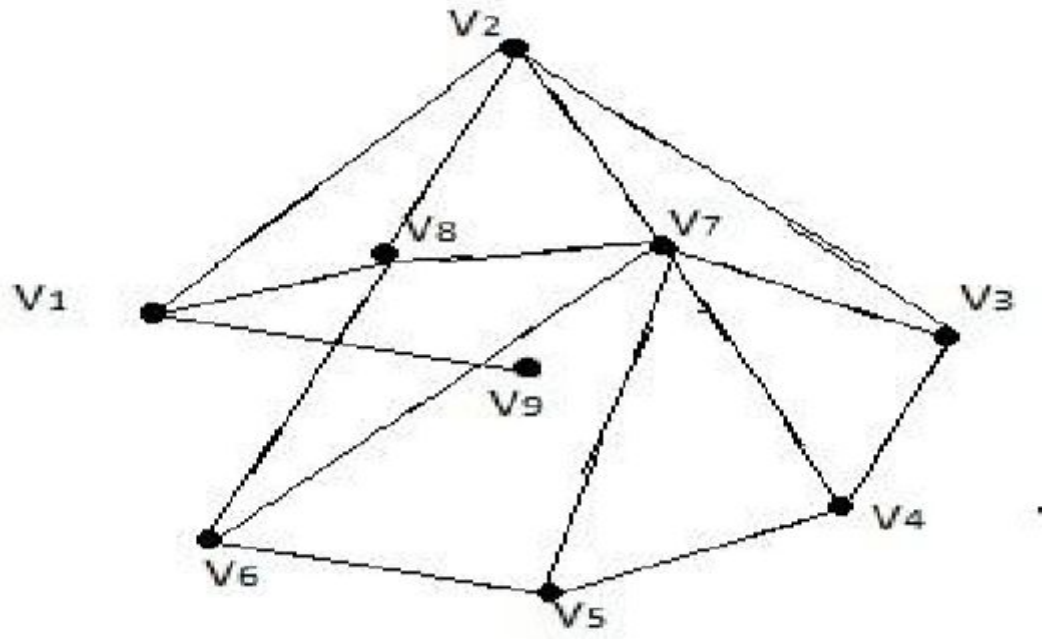

v1
v2
v3
v4v7
v5


v1
v2
v3v4v7
v5


v1
v2
v3v4v7v5


v1 v2 v3 v4 v6 v7
v1
v2
v3
v4
v5
v7

Завдання № 2
Скласти таблицю суміжності для орграфа.



	v1	v2	v3	v4	v5	v6	v7	v8	v9
v1	0	1	0	0	0	0	0	1	1
v2	1	0	1	0	0	0	1	1	0
v3	0	1	0	1	0	0	1	0	0
v4	0	0	1	0	1	0	1	0	0
v5	0	0	0	1	0	1	1	0	0
v6	0	0	0	0	1	0	1	1	0
v7	0	1	1	1	1	1	0	1	0
v8	1	1	0	0	0	1	1	0	0
v9	1	0	0	0	0	0	0	0	0

Завдання № 3
Для графа з другого завдання знайти діаметр.
Діаметр = 4 (v5 -> v9)

Завдання № 4

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

Код:

```
#include <iostream>
#define SIZE 9
using namespace std;
int i, j;
bool* visited = new bool[SIZE];
int graph[SIZE][SIZE] =
{
{0,1,0,0,0,0,0,1,1},
{1,0,1,0,0,0,1,1,0},
{0,1,0,1,0,0,1,0,0},
{0,0,1,0,1,0,1,0,0},
{0,0,0,1,0,1,1,0,0},
{0,0,0,0,1,0,1,1,0},
{0,1,1,1,1,1,0,1,0},
{1,1,0,0,0,1,1,0,0},
{1,0,0,0,0,0,0,0,0}
};

void dfs(int start)
{
 int temp;
 cout << start + 1 << " ";
 visited[start] = true;
 for (temp = 0; temp <= SIZE; temp++)
  if ((graph[start][temp] != 0) && (!visited[temp]))
   dfs(temp);
}
int main()
{
 int start;
 cout << "Matrix : " << endl;
 for (i = 0; i < SIZE; i++)
 {
  visited[i] = false;
  for (j = 0; j < SIZE; j++)
   cout << " " << graph[i][j];
  cout << endl;
 }
 cout << "start = "; cin >> start;
 bool* vis = new bool[SIZE];
 cout << "order: ";
 dfs(start - 1);
```

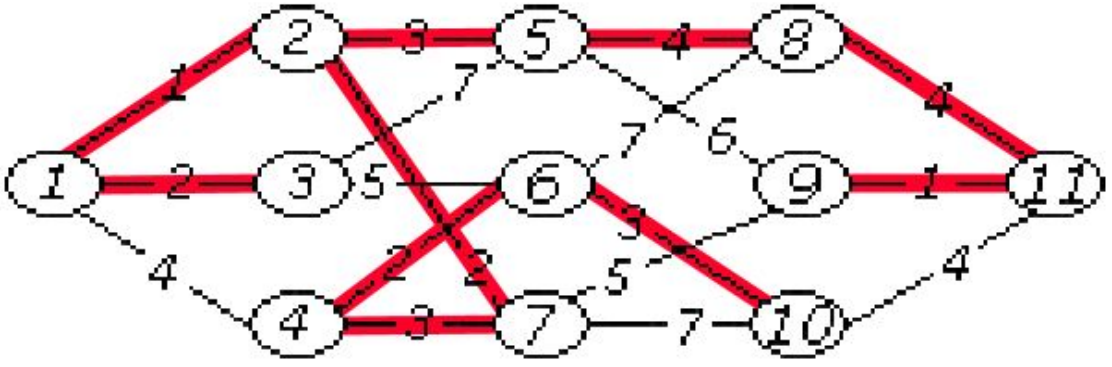
```
 delete[]visited;
 return 0;
}
```

Результат:

```
Matrix :
 0 1 0 0 0 0 0 1 1
 1 0 1 0 0 0 1 1 0
 0 1 0 1 0 0 1 0 0
 0 0 1 0 1 0 1 0 0
 0 0 0 1 0 1 1 0 0
 0 0 0 0 1 0 1 1 0
 0 1 1 1 1 1 0 1 0
 1 1 0 0 0 1 1 0 0
 1 0 0 0 0 0 0 0 0
start = 6
order: 6 5 4 3 2 1 8 7 9
```

Завдання № 5

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



Код алгоритму Краскала:

```
#include <iostream>
#include<algorithm>
#include <vector>
#include<locale>
using namespace std;
int main()
{
 setlocale(LC_ALL, "ukr");
 int n,m,weight,x,y;
 cout << "кількість вершин: ";cin >> n;
 cout << "кількість ребер:";cin >> m;
 vector < pair < int, pair<int, int> > > g;
 for (int i = 0; i < m; i++) {
  cout << "Ребро[" << i << "] = "<<endl;
  cout << "Вершина 1: ";cin >> x;
```

```
 cout << "Вершина 2: ";cin >> y;
 cout << "Вага: ";cin >> weight;
 g.push_back({weight, {--x,--y}});
}
vector < pair<int, int> > res;
sort(g.begin(), g.end());
vector<int> tree_id(n);
for (int i = 0; i < n; ++i)
 tree_id[i] = i;
for (int i = 0; i < m; ++i)
{
 int a = g[i].second.first, b = g[i].second.second;
 if (tree_id[a] != tree_id[b])
 {
  res.push_back(make_pair(a, b));
  int old_id = tree_id[b], new_id = tree_id[a];
  for (int j = 0; j < n; ++j)
   if (tree_id[j] == old_id)
    tree_id[j] = new_id;
 }
}
for (auto index : res) {
 cout << index.first + 1 << " - " << index.second + 1<< endl;
}
 return 0;
}
```

Код алгоритму Прима:

```
#include <iostream>
#include<vector>
using namespace std;
#define INF 1000000000
int main()
{
 int n;
 cout << "vertex: ";cin >> n;
 int m;
 cout << "edges: ";cin >> m;
 int** graph;
 graph = new int* [n];
 for (int i = 0; i < n;i++) {
  graph[i] = new int[n];
 }
 for (int i = 0; i < m;i++) {
  int ver1, ver2, weight;
```

```
 cout << "vertex 1: ";
 cin >> ver1;
 cout << "vertex 2: ";
 cin >> ver2;
 cout << "weight: ";
 cin >> weight;
 graph[--ver1][--ver2] = weight;
}
for (int i = 0; i < n; i++) {
 for (int j = 0; j < n;j++) {
  if (graph[i][j] < 0) {
   graph[i][j] = INF;
  }
 }
 cout << endl;
}
for (int i = 0; i < n; i++) {
 for (int j = 0; j < n;j++) {
  cout << graph[i][j] << " ";
 }
 cout << endl;
}

vector<bool> used(n);
vector<int> min_e(n, INF), sel_e(n, -1);
min_e[0] = 0;
for (int i = 0; i < n; ++i) {
 int v = -1;
 for (int j = 0; j < n; ++j)
  if (!used[j] && (v == -1 || min_e[j] < min_e[v]))
   v = j;
 if (min_e[v] == INF) {
  cout << "No MST!";
  exit(0);
 }
 used[v] = true;
 if (sel_e[v] != -1)
  cout << v << " " << sel_e[v] << endl;
 for (int to = 0; to < n; ++to)
  if (graph[v][to] < min_e[to]) {
   min_e[to] = graph[v][to];
   sel_e[to] = v;
  }
}
return 0;
}
```

Результат кодів:

```
кількість вершин: 11
кількість ребер:18
Ребро[0] =
Вершина 1: 1
Вершина 2: 2
Вага: 1
Ребро[1] =
Вершина 1: 1
Вершина 2: 3
Вага: 2
Ребро[2] =
Вершина 1: 1
Вершина 2: 4
Вага: 4
Ребро[3] =
Вершина 1: 2
Вершина 2: 5
Вага: 3
Ребро[4] =
Вершина 1: 2
Вершина 2: 7
Вага: 2
Ребро[5] =
Вершина 1: 3
Вершина 2: 5
Вага: 7
Ребро[6] =
Вершина 1: 3
Вершина 2: 6
Вага: 5
Ребро[7] =
Вершина 1: 4
Вершина 2: 6
Вага: 2
Ребро[8] =
Вершина 1: 4
Вершина 2: 7
Вага: 3
Ребро[9] =
Вершина 1: 5
Вершина 2: 8
Вага: 4
```

```
Ребро[10] =
Вершина 1: 5
Вершина 2: 9
Вага: 6
Ребро[11] =
Вершина 1: 6
Вершина 2: 8
Вага: 7
Ребро[12] =
Вершина 1: 6
Вершина 2: 10
Вага: 3
Ребро[13] =
Вершина 1: 7
Вершина 2: 9
Вага: 5
Ребро[14] =
Вершина 1: 7
Вершина 2: 10
Вага: 7
Ребро[15] =
Вершина 1: 8
Вершина 2: 11
Вага: 4
Ребро[16] =
Вершина 1: 9
Вершина 2: 11
Вага: 1
Ребро[17] =
Вершина 1: 10
Вершина 2: 11
Вага: 4
```

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

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

При стартовій вершині 1: 1 -> 4 -> 6 -> 3 -> 5 -> 7 -> 2 -> 8
Вага: 10
Код:

```
#include<iostream>
#include<vector>
using namespace std;
const int INF = INT_MAX;
const int n = 8;
vector<int>visited;
int way = 0;
int graph[n][n] =
{
 {INF,3,2,1,2,2,3,2},
 {3,INF,6,5,4,5,1,2},
 {2,6,INF,3,2,1,3,3},
 {1,5,3,INF,5,1,5,1},
 {2,4,2,5,INF,2,2,2},
 {2,5,1,1,2,INF,7,5},
 {3,1,3,5,2,7,INF,5},
 {2,2,3,1,2,5,5,INF}
};
bool checkVis(int value) {
 bool flag = false;
 for (auto v : visited) if (v == value)flag = true;
 return flag;
}
```

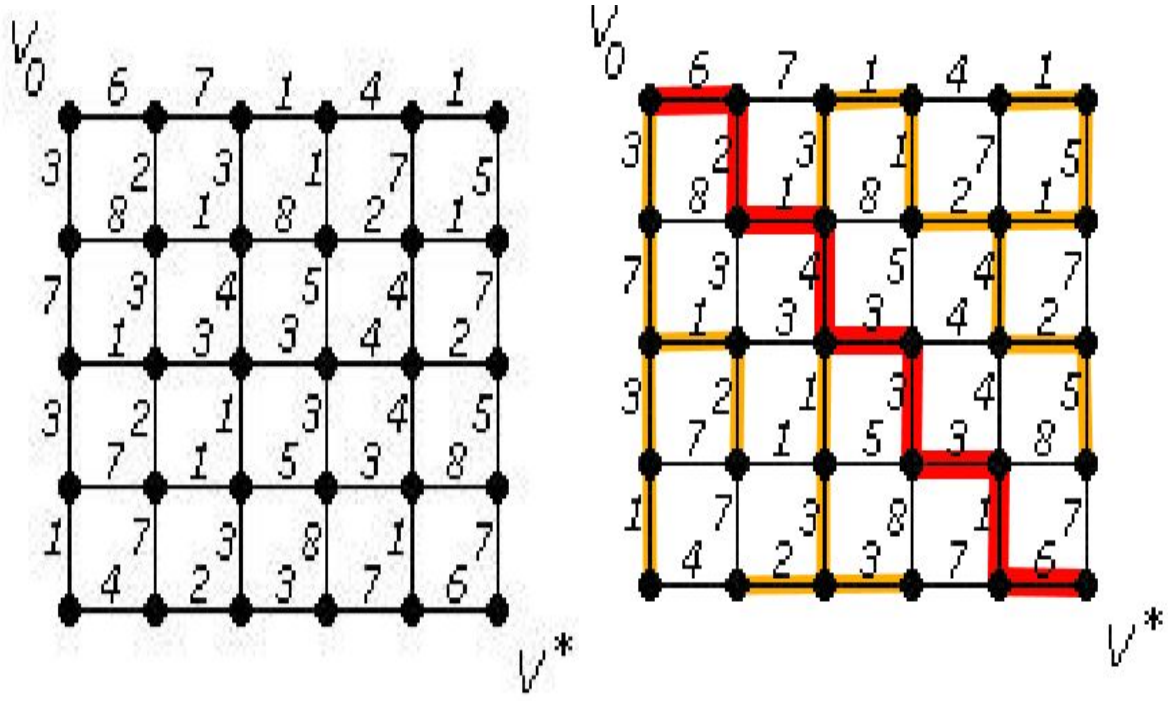
```
void findMin() {
 int ver = visited.at(visited.size() - 1);
 cout << "Current vertex : " << ver+1 << endl;
 int index;
 int min = INF;
 for (int i = 0; i < n; i++)
  if (graph[i][ver] < min && !checkVis(i)) {
   min = graph[i][ver];
   index = i;
  }

 if (visited.size() != n) {//остання вершина
  way += min;
 }
 visited.push_back(index);
}
int main() {
 visited.push_back(0);
 while (visited.size() <= n) {
  findMin();
 }

 cout << "way weight = "<<way;
  return 0;
}
```

Результат:

```
Current vertex : 1
Current vertex : 4
Current vertex : 6
Current vertex : 3
Current vertex : 5
Current vertex : 7
Current vertex : 2
Current vertex : 8
way weight = 10
```

Завдання № 7

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



Код:

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
using namespace std;
int main()
{
 int vertex,edges;//к-сть вершин, ребер
 cout << "Count of vertex: ";
 cin >> vertex;
 cout << "Count of edges: ";
 cin >> edges;
 int **a=new int*[vertex]; // матриця суміжності
 int *d=new int[vertex]; // мінімальна відстань
 int *visited=new int[vertex]; // відвідані вершини
 int temp, minindex, min;
 //виділення пам'яті для матриці
 for (int i = 0; i < vertex; i++)
 {
  a[i] = new int [vertex];
 }
 //заповнення матриці 0
 for (int i = 0; i < vertex; i++)
 {
  for (int j = 0; j < vertex; j++) {
   a[i][j] = 0;
  }
 }
 //ініціалізація матриці суміжності
```

```
for (int i = 0; i < edges; i++)
{
 int x, y,weight;
 cout << "Edge[" << i << "]=" << endl;
 cout << "Vertex1: ";
 cin >> x;
 cout << "Vertex2: ";
 cin >> y;
 cout << "Weight: ";
 cin >> weight;
 a[--x][--y] =weight;
 a[y][x] = weight;
}

// вивід матриці суміжності
for (int i = 0; i < vertex; i++)
{
 for (int j = 0; j < vertex; j++)
  cout<< a[i][j]<<" ";
 cout << endl;
}
//ініціалізація вершин та відстаней
for (int i = 0; i < vertex; i++)
{
 d[i] = 10000;
 visited[i] = 1;
}
int start, finish;//індекс вершини з якої ідем та вершини в яку ідем
cout << "From vertex :";
cin >> start;
start--;
cout << "To: ";
cin >> finish;
finish--;
int begin_index = start;
d[begin_index] = 0;
// Крок алгоритму
do {
 minindex = 10000;
 min = 10000;
 for (int i = 0; i < vertex; i++)
 {
  if ((visited[i] == 1) && (d[i] < min))
  {
   min = d[i];
   minindex = i;
```

```
 }
}
if (minindex != 10000)
{
 for (int i = 0; i < vertex; i++)
 {
  if (a[minindex][i] > 0)
  {
   temp = min + a[minindex][i];
   if (temp < d[i])
   {
    d[i] = temp;
   }
  }
 }
 visited[minindex] = 0;
}
} while (minindex < 10000);
// вивід найкоротших відстаней до вершини
cout<<"Minimal ways to vertex: "<<endl;
for (int i = 0; i < vertex; i++)cout << d[i] << " ";
//перевірка чи шлях існує
bool flag = false;
for (int i = 0; i < vertex; i++)if(d[i]!=0&&d[i]!=10000)flag = true;
if (flag) {
 // відновлення шляху
 int* ver = new int[vertex]; // масив відвіданих вершин
 int end = finish; // індекс кінцевої вершини
 ver[0] = end + 1; // початковий елем - кінцева вершина
 int k = 1; // індекс попередньої вершини
 int weight = d[end]; // вага кінцевої вершини
 while (end != begin_index) // поки не дійшли до початкової вершини
 {
  for (int i = 0; i < vertex; i++) //переглядаєм всі вершини
   if (a[end][i] != 0)   // якщо є зв'язок
   {
    int temp = weight - a[end][i]; //визначаєм вагу з попередньої вершини
    if (temp == d[i]) //якщо вага співпала з порахованою
    {                 //значить з цієї вершини і був перехід
     weight = temp; // зберігаєм вагу
     end = i;      // зберігаєм попередню вершину
     ver[k] = i + 1; // і записуєм її в масив
     k++;
    }
   }
 }
```

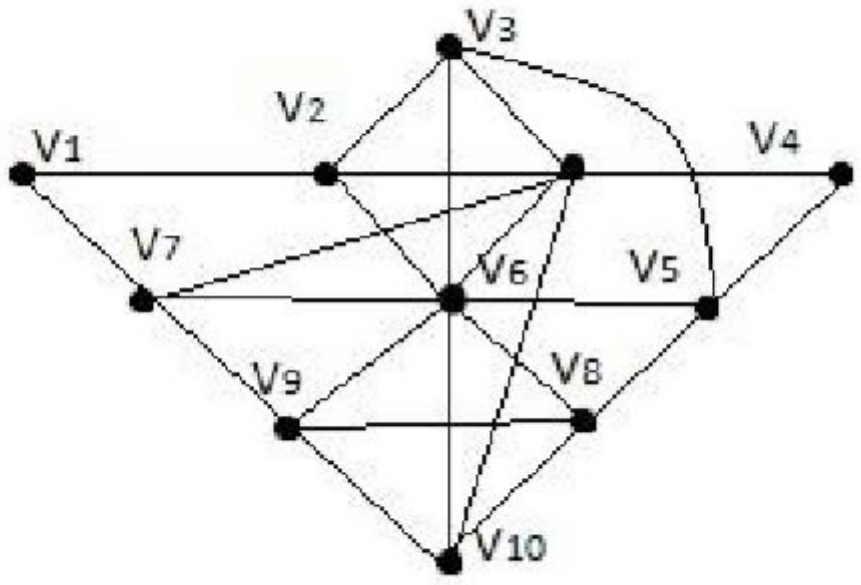
```
  // Вивід шляху (початкова вершина в кінці масиву з к елементів)
  cout << endl << "Print minimal way" << endl;
  for (int i = k - 1; i >= 0; i--)cout << ver[i] << " ";

 }
 else {
  cout << "There isnt such way";
 }
 return 0;
}
```

Завдання № 8

Знайти ейлеровий цикл в ейлеровому графі двома методами:

а) Флері;

б) елементарних циклів.



а)

Код:

```
#include<iostream>
#include<vector>
using namespace std;
const int n = 11;
int graph[n][n] = {
 {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
 {1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1},
 {0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1},
 {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1},
 {0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0},
```

```
 {0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1},
 {1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1},
 {0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0},
 {0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0},
 {0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1},
 {0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0}
};
int oddGraph[n][n];
int startVertex() {
 for (int i = 0; i < n; i++) {
  int deg = 0;
  for (int j = 0; j < n; j++) {
   if (oddGraph[i][j])
    deg++;
  }
  if (deg % 2 != 0)
   return i;
 }
 return 0;
}
bool check(int u, int v) {
 int deg = 0;
 for (int i = 0; i < n; i++)
  if (oddGraph[v][i])
   deg++;
 if (deg > 1) {
  return false;
 }
 return true;
}
int edgeCount() {
 int count = 0;
 for (int i = 0; i < n; i++)
  for (int j = i; j < n; j++)
   if (oddGraph[i][j])
    count++;
 return count;
}
void func(int start) {
 static int edge = edgeCount();
 for (int v = 0; v < n; v++) {
  if (oddGraph[start][v]) {
   if (edge <= 1 || !check(start, v)) {
    cout << start << "-" << v <<endl;
    oddGraph[start][v] = oddGraph[v][start] = 0;
    edge--;
```

```
    func(v);
   }
  }
 }
}
int main() {
 for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
   oddGraph[i][j] = graph[i][j];
 cout << "Result: "<<endl;
 func(startVertex());
return 0;
}
```

Результат:

```
Result:
0-1
1-2
2-4
4-3
3-10
10-1
1-5
5-2
2-10
10-5
5-4
4-7
7-5
5-6
6-8
8-5
5-9
9-7
7-8
8-9
9-10
10-6
6-0
```

б)

Код:

```
#include <iostream>
#define N 11
#define STACK_SIZE 100
using namespace std;
int G[N][N] =
{
{0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
{1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1},
{0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1},
```

```
{0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0},
{0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1},
{1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1},
{0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0},
{0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0},
{0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1},
{0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 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 p, q, s;
 int j, vv;
 k = -1;
 cout<< "start vertex: ";
 cin>>vv;

  Search(vv);
  for (j = 0; j <= k; j++)
   if (j % 2 == 0) {
    cout<< Stack[j]<<"-";
   }
   else {
    cout << Stack[j] << endl;
   }
  for (j = 1; j < k; j++)
   if (j % 2 == 0) {
    cout << Stack[j] << endl;
   }
   else {
    cout << Stack[j] << "-";
   }
```

```
 return 0;
}
```

Результат:

```
start vertex: 1
1-10
9-8
7-9
5-10
6-8
5-7
4-5
2-10
3-4
2-1
5-6
0-1
10-9
8-7
9-5
10-6
8-5
7-4
5-2
10-3
4-2
1-5
6-0
```

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

 - скорочена ДНФ