МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота

з дисципліни
«Дискретна математика»

Виконала:
студент групи КН-114
Ярка Ірина
Викладач:
Мельникова Н.І.

Львів – 2019 р.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Варіант 19

Завдання № 1

Виконати наступні операції над графами:

1) знайти доповнення до першого графу,

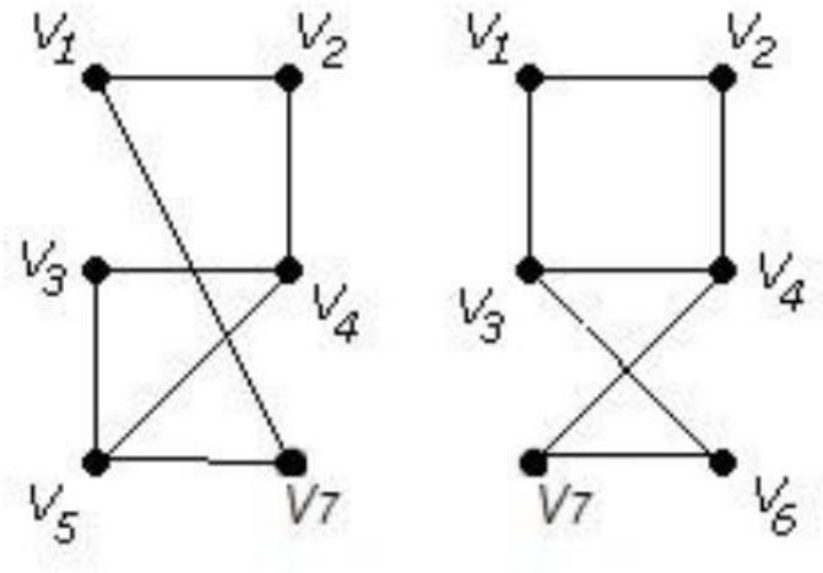
2) об’єднання графів,

3) кільцеву суму G_1 та G_2 (G_1+G_2),

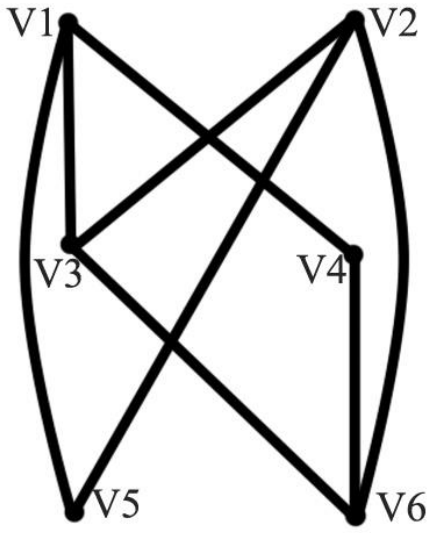
4) розмножити вершину у другому графі,

5) виділити підграф А - що складається з 3-х вершин в G_1,

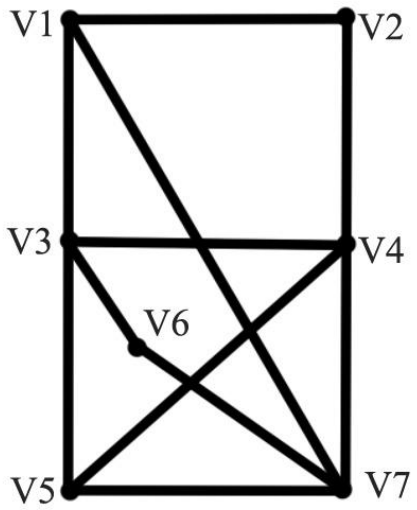
6) добуток графів.



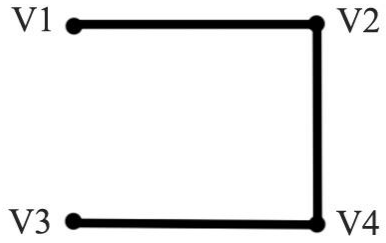
1)Доповнення до першого графа



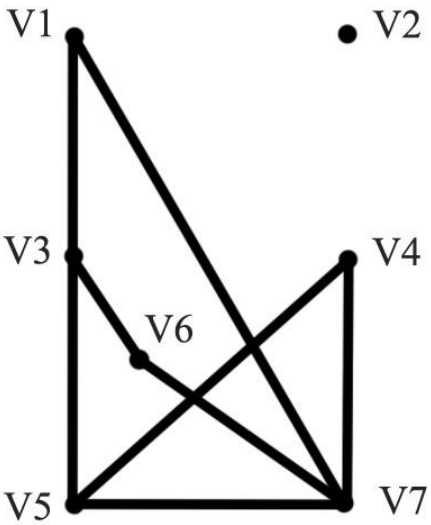
2) об’єднання графів



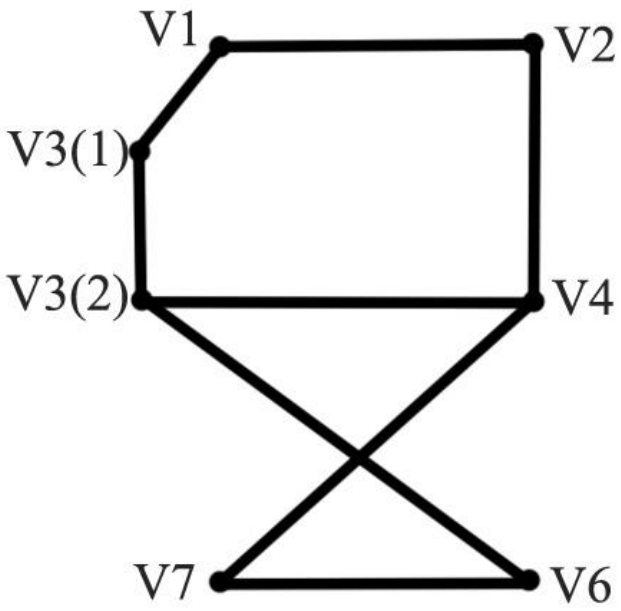
3) кільцеву сумму G_1 та G_2 (G_1+G_2)
Оскільки $G1+G2 = (G1 \cup G2)\setminus(G1 \cap G2)$, то
Перетин $(G1 \cap G2)$:



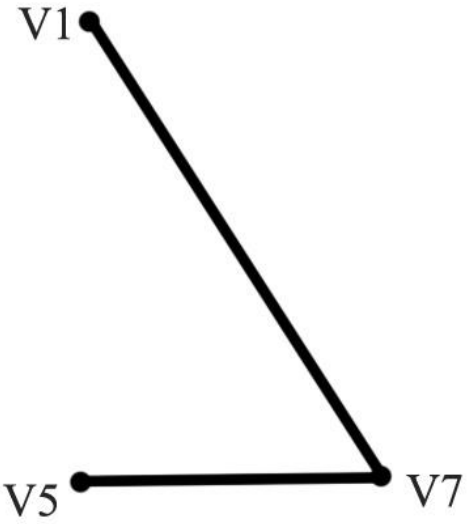
Кільцева сума G1+G2:



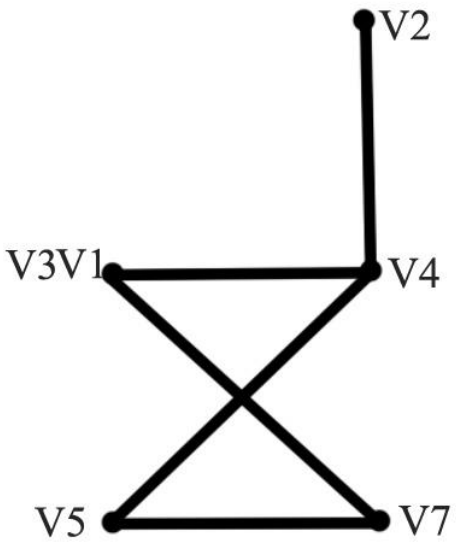
4) розмножити вершину у другому графі (розмноження V3):



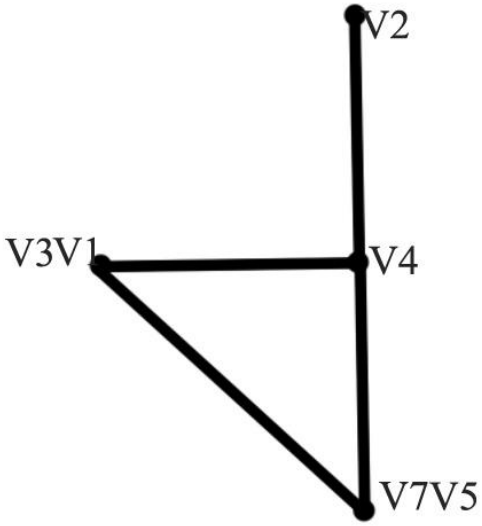
5) виділити підграф А - що скадається з 3-х вершин в G1
Підграф А



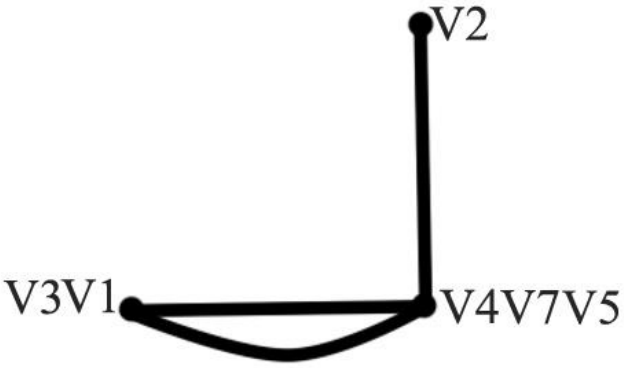
Крок 1: Стягуємо V1 та V3



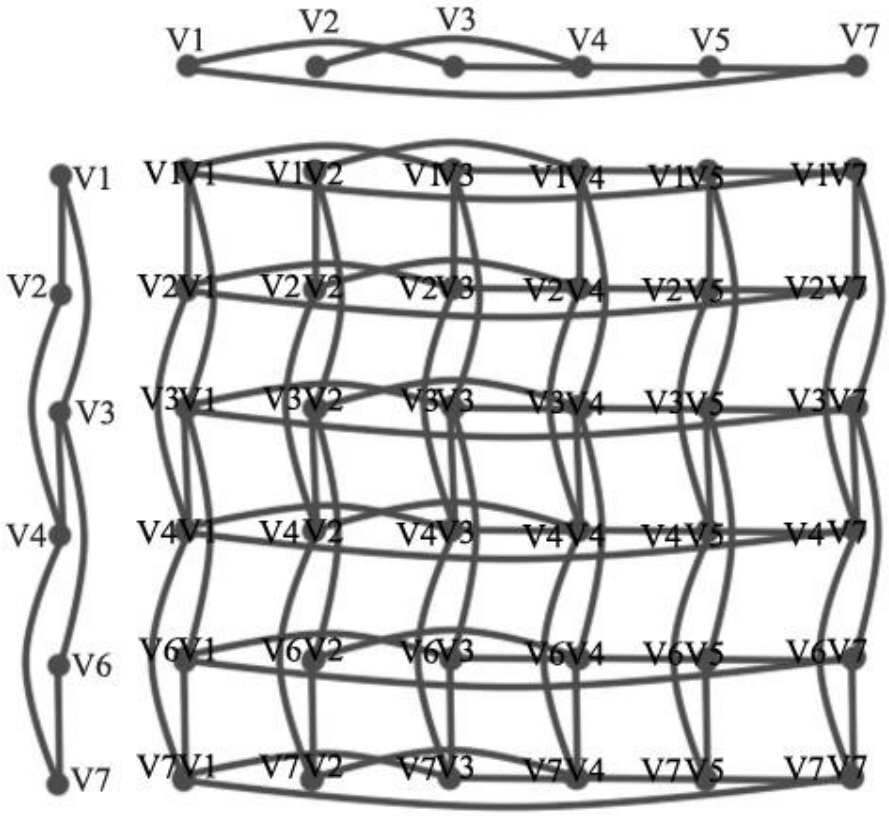
Крок 2: Стягуємо V5 та V7



Крок 3: Стягуємо V4 та V7V5

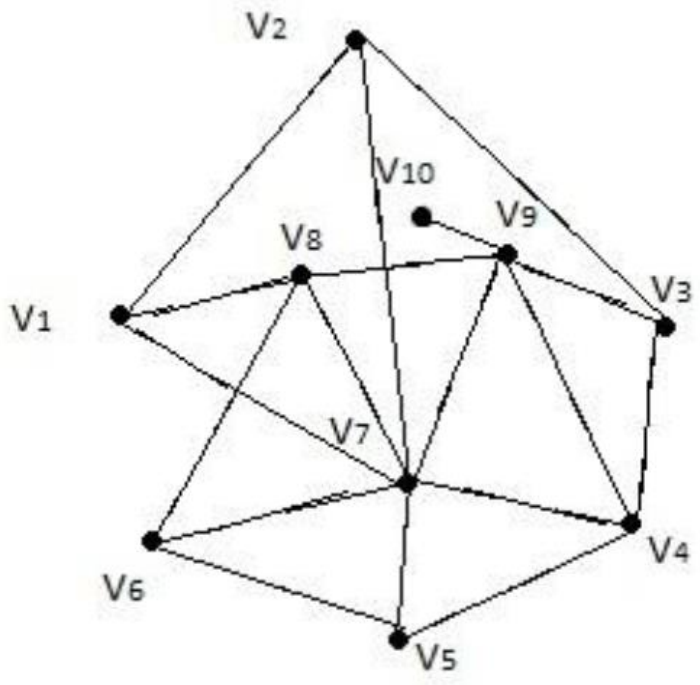


6) добуток графів



Завдання №2

Скласти таблицю суміжності для неографа.



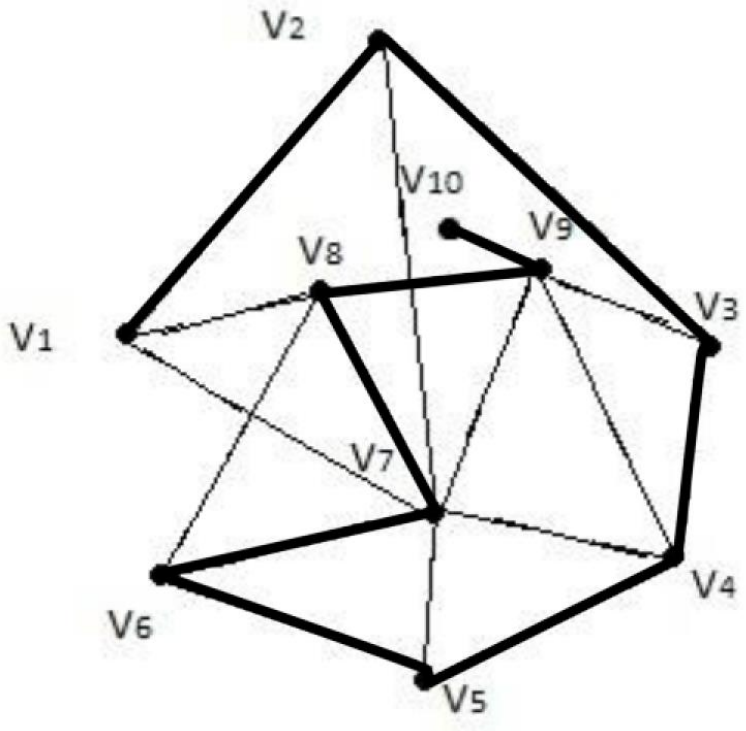
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
V1	0	1	0	0	0	0	1	1	0	0
V2	1	0	1	0	0	0	1	0	0	0
V3	0	1	0	1	0	0	0	0	1	0
V4	0	0	1	0	1	0	0	0	1	0
V5	0	0	0	1	0	1	1	0	0	0
V6	0	0	0	0	1	0	1	1	0	0
V7	1	1	0	0	1	1	0	1	1	0
V8	1	0	0	0	0	1	1	0	1	0
V9	0	0	1	1	0	0	1	1	0	1
V10	0	0	0	0	0	0	0	0	1	0

Завдання №3

Для графа з другого завдання знайти діаметр.
Діаметр: 1-> 7 -> 9 -> 10 (3).

Завдання №4

Для графа з другого завдання виконати обхід дерева вглиб.



Вершина	DFS-номер	Вміст стеку
V1	1	V1
V2	2	V1V2
V3	3	V1V2V3
V4	4	V1V2V3V4
V5	5	V1V2V3V4V5
V6	6	V1V2V3V4V5V6
V7	7	V1V2V3V4V5V6V7
V8	8	V1V2V3V4V5V6V7V8
V9	9	V1V2V3V4V5V6V7V8V9
V10	10	V1V2V3V4V5V6V7V8V9V10
-	-	V1V2V3V4V5V6V7V8V9
-	-	V1V2V3V4V5V6V7V8
-	-	V1V2V3V4V5V6V7
-	-	V1V2V3V4V5V6
-	-	V1V2V3V4V5
-	-	V1V2V3V4
-	-	V1V2V3
-	-	V1V2
-	-	V1
-	-	Ø

Скріншоти коду :

```
#include <bits/stdc++.h>
using namespace std;

class Graph {
    int v;
    int e;
    int** adj;

public:
    Graph(int v, int e);
    void addEdge(int start, int e);
    void DFS(int start, vector<bool>& visited);
};

Graph::Graph(int v, int e)
{
    this->v = v;
    this->e = e;
    adj = new int*[v];
    for (int row = 0; row < v; row++) {
        adj[row] = new int[v];
        for (int column = 0; column < v; column++) {
```

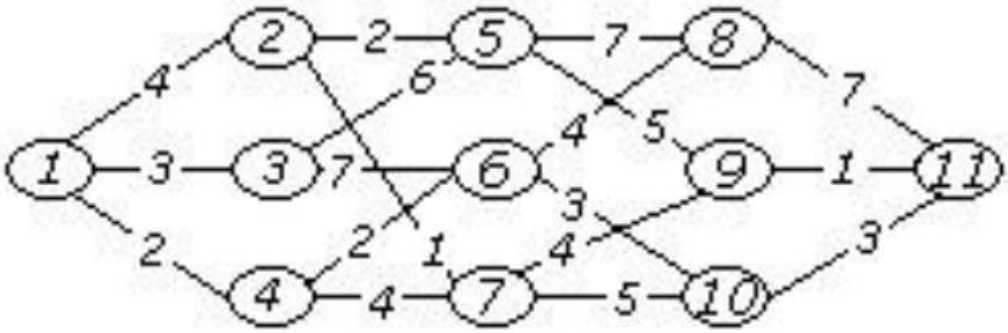
```
            adj[row][column] = 0;
        }
    }
}

void Graph::addEdge(int start, int e)
{
    adj[start][e] = 1;
    adj[e][start] = 1;
}

void Graph::DFS(int start, vector<bool>& visited)
{
    cout << start << " ";
    visited[start] = true;

    for (int i = 0; i < v; i++) {

        if (adj[start][i] == 1 && (!visited[i])) {
            DFS(i, visited);
        }
    }
}

int main()
{
    int v, e, v1, v2;
    Graph G(10, 18);
    cout<<"Num of vertices n edges";
    cin>>v>>e;

    cout<<"Enter the ways";
    for(int i = 0; i<e; i++){
        cin>>v1>>v2;
        G.addEdge(v1, v2);
    }
    vector<bool> visited(v, false);

    G.DFS(1, visited);
}
```

Скріншоти результату:

```
Num of vertices n edges10 18
Enter the ways1 2
1 8
1 7
2 3
2 7
3 4
3 9
4 9
4 5
4 7
5 6
5 7
6 7
6 8
7 8
7 9
8 9
9 10
1 2 3 4 5 6 7 8 9 10
```

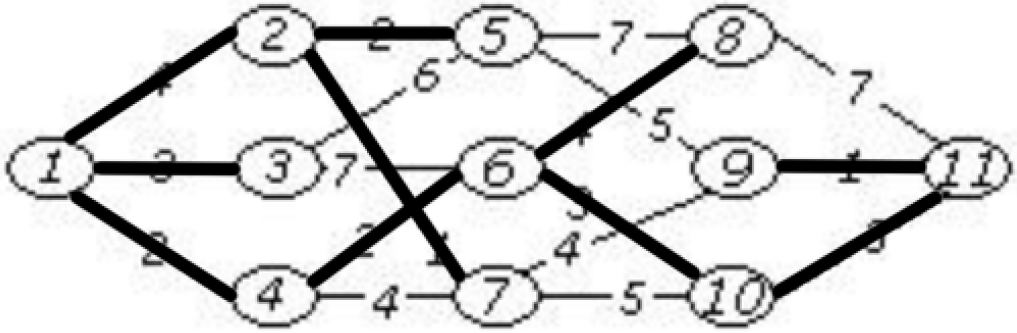
Завдання № 5

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



Алгоритм Краскала:

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



Скріншоти коду:

```
#include <iostream>

using namespace std;

int create(int n, int A[11][11]) {
    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 11; j++) {
            A[i][j] = 0;
        }
    }
    for (int i = 0; i < 11; i++) {
        A[i][i] = i+1;
    }
    return A[11][11];
}

void dubl(int n, int A[11][11]) {
    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 11; j++) {
            if (j < i) {
                A[i][j] = 0;
            }
        }
    }
}

int no(int n, int A[11][11], int t, int r) {
    int tm, tm2;
```

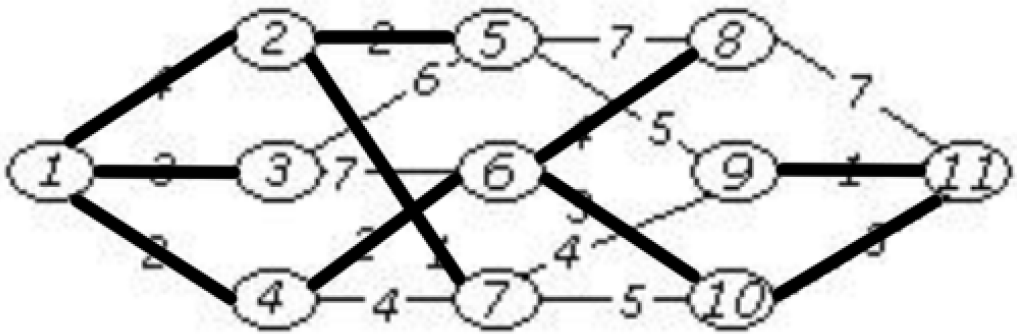
```
    for (int i = 0; i < 11; i++) {
        tm = tm2 = 0;
        for (int j = 0; j < 11; j++) {
            if (A[i][j] == t) {
                tm = 1;
            }
        }
        for (int f = 0; f < 11; f++) {
            if (A[i][f] == r) {
                tm2 = 1;
            }
        }
        if (tm && tm2) {
            return 0;
        }
    }
    return 1;
}

void add(int n, int A[11][11], int t, int r) {
    int scn;
    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 11; j++) {
            if (A[i][j] = r) {
                scn = i;
            }
        }
    }
    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 11; j++) {
            if (A[i][j] == t) {
                for (int k = 0; k < 11; k++) {
                    A[i][k] = A[scn][k];
                    A[scn][k] = 0;
                }
            }
        }
    }
}
int main() {
    int MS[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},
            {2,0,0,0,0,2,4,0,0,0,0},
            {0,2,6,0,0,0,0,7,5,0,0},
            {0,0,7,2,0,0,0,4,0,3,0},
            {0,1,0,4,0,0,0,0,4,5,0},
            {0,0,0,0,7,4,0,0,0,0,7},
            {0,0,0,0,5,0,4,0,0,0,1},
            {0,0,0,0,0,3,5,0,0,0,3},
            {0,0,0,0,0,0,0,7,1,3,0}
    };
    dubl(11, MS);
    for (int i = 1; i <= 7; i++) {
        cout << endl<<"Edges with weight: " << i << "    ";
        for (int j = 1; j <= 11; j++) {
            for (int k = 1; k <= 11; k++) {
                if (MS[j - 1][k - 1] == i) {
                    cout << " " << j << "," << k << " ";
```

```
                }

            }
        }
    }

    int B[11][11];
    create(11, B);
    cout << endl << "New Tree: ";
    for (int i = 1; i <= 7; i++) {
        for (int j = 1; j <= 11; j++) {
            for (int k = 1; k <= 11; k++) {
                if (MS[j - 1][k - 1] == i && no(11, B, j, k)) {
                    add(11, B, j, k);
                    cout << " " << j << "," << k << " ";
                }
            }
        }
    }
    cout << endl;
    return 0;
}
```

Скріншоти результатів:

```
Edges with weight: 1    2,7  9,11
Edges with weight: 2    1,4  2,5  4,6
Edges with weight: 3    1,3  6,10  10,11
Edges with weight: 4    1,2  4,7  6,8  7,9
Edges with weight: 5    5,9  7,10
Edges with weight: 6    3,5
Edges with weight: 7    3,6  5,8  8,11
New Tree:  2,7  9,11  1,4  2,5  4,6  1,3  6,10  10,11  1,2  6,8

Process finished with exit code 0
```

Алгоритм Прима:
Починаємо з вершини 1
1-4, 4-6, 1-3, 6-10, 6-8, 10-11, 11-9, 1-2, 2-7, 2-5.



Скріншоти коду:

```cpp
#include<bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

void addEdge(vector <pair<int, int> > adj[], int u,
             int v, int wt)
{
    adj[u].emplace_back(v, wt);
    adj[v].emplace_back(u, wt);
}

void primMST(vector<pair<int,int> > adj[], int V)
{
    priority_queue< iPair, vector <iPair> , greater<iPair> > pq;

    int src = 0;
    vector<int> key(V, INF);
    vector<int> parent(V, -1);
    vector<bool> inMST(V, false);
    pq.push(make_pair(0, src));
    key[src] = 0;
    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        inMST[u] = true;
        for (auto x : adj[u])
        {
            int v = x.first;
            int weight = x.second;
            if (inMST[v] == false && key[v] > weight)
            {
                key[v] = weight;
                pq.push(make_pair(key[v], v));
                parent[v] = u;
            }
        }
    }
    for (int i = 1; i < V; ++i)
        printf("%d - %d\n", parent[i]+1, i+1);
}

int main()
{
    int V = 11;
    vector<iPair > adj[V];
    addEdge(adj, 0, 1, 4);
    addEdge(adj, 0, 2, 3);
    addEdge(adj, 0, 3, 2);
    addEdge(adj, 1, 4, 2);
    addEdge(adj, 1, 6, 1);
    addEdge(adj, 2, 4, 6);
    addEdge(adj, 2, 5, 7);
    addEdge(adj, 3, 5, 2);
    addEdge(adj, 3, 6, 4);
    addEdge(adj, 4, 7, 7);
```

```
    addEdge(adj, 4, 8, 5);
    addEdge(adj, 5, 7, 4);
    addEdge(adj, 5, 9, 3);
    addEdge(adj, 6, 8, 4);
    addEdge(adj, 6, 9, 5);
    addEdge(adj, 7, 10, 7);
    addEdge(adj, 8, 10, 1);
    addEdge(adj, 9, 10, 3);

    primMST(adj, V);

    return 0;
}
```

Скріншоти результатів:

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

Process finished with exit code 0
```

Завдання № 6

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

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

	12	3	4	5	6	7	8
12	∞	5	1	2	3	2	4
3	5	∞	6	6	5	1	5
4	1	6	∞	6	6	6	6
5	2	6	6	∞	5	1	5
6	3	5	6	5	∞	2	1
7	2	1	6	1	2	∞	5
8	4	5	6	5	1	5	∞

	3	5	124	6	7	8
3	∞	6	6	5	1	5
5	6	∞	6	6	6	6
124	6	6	∞	5	1	5
6	5	6	5	∞	2	1
7	1	6	1	2	∞	5
8	5	6	5	1	5	∞

	3	5	6	1247	8
3	∞	6	5	1	5
5	6	∞	6	6	6
6	5	6	∞	2	1
1247	1	6	2	∞	5
8	5	6	1	5	∞

	12473	5	6	8
12473	∞	6	5	5
5	6	∞	6	6
6	5	6	∞	1
8	5	6	1	∞

	124738	5	6
124738	∞	6	6
5	6	∞	1
6	6	1	∞

	1247386	5
1247386	∞	6
5	6	∞

Скріншоти коду:

```
#include <bits/stdc++.h>
using namespace std;
const int N = 8;
int final_path[N+1];
bool visited[N];
int final_res = INT_MAX;

void copyToFinal(int curr_path[]){
    for (int i=0; i<N; i++)
        final_path[i] = curr_path[i];
    final_path[N] = curr_path[0];
}

int firstMin(int adj[N][N], int i){
    int min = INT_MAX;
    for (int k=0; k<N; k++)
        if (adj[i][k]<min && i != k)
            min = adj[i][k];
    return min;
}

int secondMin(int adj[N][N], int i){
    int first = INT_MAX, second = INT_MAX;
    for (int j=0; j<N; j++){
        if (i == j)
            continue;

        if (adj[i][j] <= first){
            second = first;
            first = adj[i][j];
        }
        else if (adj[i][j] <= second &&
                 adj[i][j] != first)
            second = adj[i][j];
    }
    return second;
}

void TSPRec(int adj[N][N], int curr_bound, int curr_weight, int level, int curr_path[]){
    if (level==N){
        if (adj[curr_path[level-1]][curr_path[0]] != 0){
            int curr_res = curr_weight + adj[curr_path[level-1]][curr_path[0]];

            if (curr_res < final_res){
                copyToFinal(curr_path);
                final_res = curr_res;
            }
        }
        return;
    }
    for (int i=0; i<N; i++){
        if (adj[curr_path[level-1]][i] != 0 &&
            visited[i] == false){
            int temp = curr_bound;
            curr_weight += adj[curr_path[level-1]][i];
            if (level==1)
                curr_bound -= ((firstMin(adj, curr_path[level-1]) + firstMin(adj,
i))/2);
```

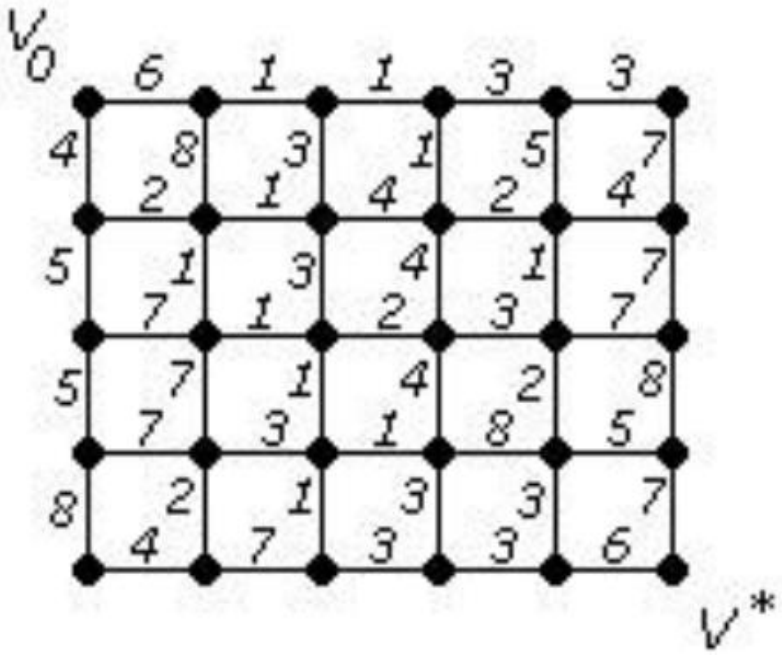
```
            else
                curr_bound -= ((secondMin(adj, curr_path[level-1]) + firstMin(adj,
i))/2);
            if (curr_bound + curr_weight < final_res){
                curr_path[level] = i;
                visited[i] = true;
                TSPRec(adj, curr_bound, curr_weight, level+1,
                       curr_path);
            }
            curr_weight -= adj[curr_path[level-1]][i];
            curr_bound = temp;
            memset(visited, false, sizeof(visited));
            for (int j=0; j<=level-1; j++)
                visited[curr_path[j]] = true;
        }
    }
}

void TSP(int adj[N][N]){
    int curr_path[N+1];
    int curr_bound = 0;
    memset(curr_path, -1, sizeof(curr_path));
    memset(visited, 0, sizeof(curr_path));
    for (int i=0; i<N; i++)
        curr_bound += (firstMin(adj, i) + secondMin(adj, i));
    curr_bound = (curr_bound&1)? curr_bound/2 + 1 : curr_bound/2;
    visited[0] = true;
    curr_path[0] = 0;
    TSPRec(adj, curr_bound, 0, 1, curr_path);
}

int main()
{
    int adj[N][N] = { {0, 2, 2, 2, 2, 3, 2, 2},
                      {2, 0, 5, 1, 2, 3, 2, 4},
                      {2, 5, 0, 6, 6, 5, 1, 5},
                      {2, 1, 6, 0, 6, 6, 6, 6},
                      {2, 2, 6, 6, 0, 5, 1, 5},
                      {3, 3, 5, 6, 5, 0, 2, 1},
                      {2, 2, 1, 6, 1, 2, 0, 5},
                      {2, 4, 5, 6, 5, 1, 5, 0}
    };

    TSP(adj);

    printf("Minimum cost : %d\n", final_res);
    printf("Path Taken : ");
    for (int i=0; i<=N; i++)
        printf("%d ", final_path[i]+1);

    return 0;
}
```

Скріншоти результатів:

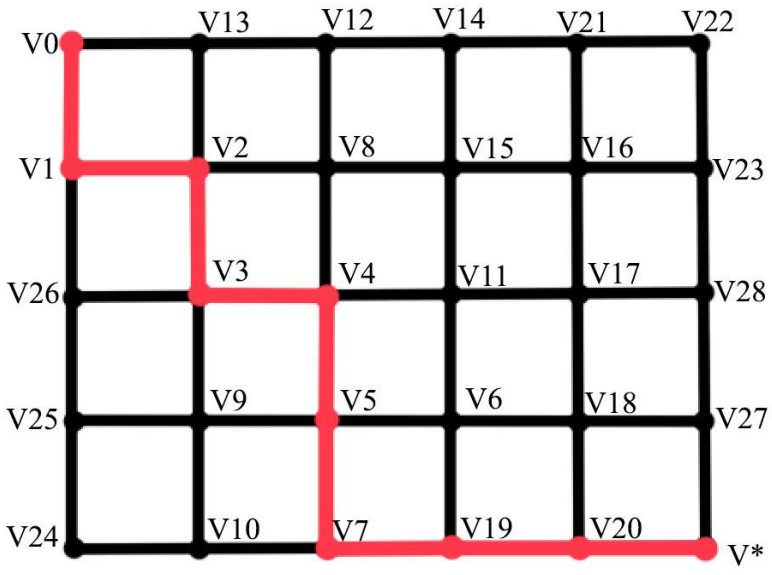
```
Minimum cost : 17
Path Taken : 1 3 4 2 5 7 6 8 1
Process finished with exit code 0
```

Завдання № 7

За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V_0 і V^*.



Шлях: V0 -> V1 -> V2 -> V3 -> V4 -> V5 -> V7 -> V19 -> V20 -> V*



Скріншоти коду:

```cpp
#include <iostream>
using namespace std;
int n, i, j;
int dist[30];
bool visited[30];
int pred[30];

void creating(int c[30][30]){
    int width, height, temp;
    cout << "Enter the number of vertices: ";
    cin >> n;
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            c[i][j] = 0;
        }
    }
    cout << "Enter the width and height of graph: ";
    cin >> width >> height;
    cout << "length between vertices : ";
    for(i = 0; i < n; i++){
        for(j = i+1; j < n; j++){
            if(j==i+1 || j == i+width){
                cin >> temp;
                c[i][j] = temp;
            }
            else{
                c[i][j] = 0;
            }
        }
    }
}

int minimumDist(){
    int minimum = 10000, minDist;
    for(int v = 0; v < n; v++){
        if(!visited[v] && dist[v]<=minimum){
            minimum = dist[v];
            minDist = v;
        }
    }
    return minDist;
}

void printPath(int j){
    if(pred[j] == -1){
        return;
    }
    printPath(pred[j]);
    cout << "V" << j + 1<< " -> ";
}

void deykstra(int c[30][30]){
    int start;
    cout << "Enter the first node: ";
    cin >> start;
    for(i = 0; i < n; i++){
        pred[0] = -1;
        dist[i] = 10000;
```

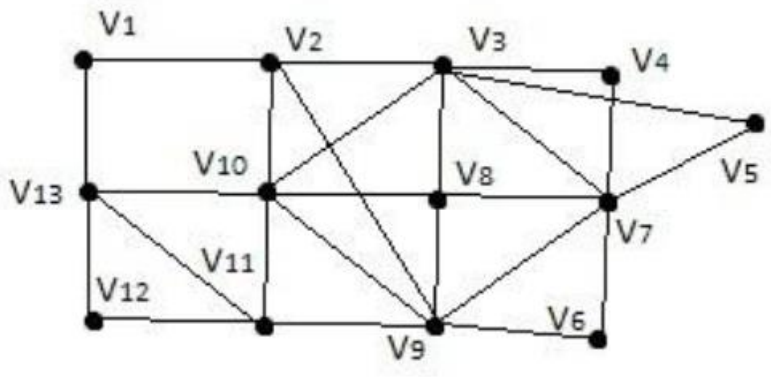
```
        visited[i] = false;
    }
    dist[start-1] = 0;
    for(int count = 0; count < n-1; count++){
        int u = minimumDist();
        visited[u] = true;
        for(int v = 0; v < n; v++){
            if(!visited[v] && c[u][v] && dist[u] + c[u][v] < dist[v]){
                pred[v] = u;
                dist[v] = dist[u] + c[u][v];
            }
        }
    }
    cout << "The least way is: " << dist[29] << endl;
    cout << "The way is: V1 -> ";
    printPath(29);
}

int main() {
    int c[30][30];
    creating(c);
    deykstra(c);
    return 0;
}
```

Скріншоти результатів:

```
Enter the number of vertices:30
 Enter the width and height of graph:6 5
 length between vertices :6 4 1 8 1 3 3 1 3 5 0 7 2 5 1 1 4 3 2 4 4 1 0 7 7 5 1 7 2 1 3 4 7 2 0 8 7 8 3 2 1 1 8 3 5 3 0 7 4 7 3 3 6
 Enter the first node:1
 The l
east way is: 22
The way is: V1 -> V7 -> V8 -> V14 -> V15 -> V21 -> V27 -> V28 -> V29 -> V30 ->
Process finished with exit code 0
```

Завдання № 8

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



а) V1-V2; V2-V9; V9-10; V10-V2; V2-V3; V3-V10; V10-V13; V13-V11; V11-V10; V10V8; V8-V7; V7-V9; V9-V8; V8-V3; V3-V4; V4-V7; V7-V3; V3-V5; V5-V7; V7-V6; V6-V9; V9-V11; V11-V12; V12-V13; V13-V1.

Скріншоти коду:

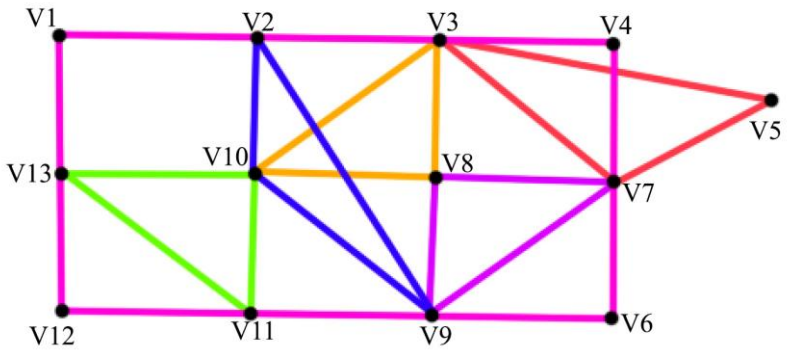
```cpp
#include <iostream>
#include <string.h>
#include <algorithm>
#include <list>
using namespace std;
class Graph
{
  int V;
  list<int> *adj;
public:
  Graph(int V)  { this->V = V;  adj = new list<int>[V];  }
  ~Graph()      { delete [] adj;  }
  void addEdge(int u, int v) {  adj[u].push_back(v); adj[v].push_back(u); }
  void rmvEdge(int u, int v);
  void printEulerTour();
  void printEulerUtil(int s);
  int DFSCount(int v, bool visited[]);
  bool isValidNextEdge(int u, int v);
};

void Graph::printEulerTour(){
  int u = 0;
  for (int i = 0; i < V; i++)
      if (adj[i].size() & 1)
        {   u = i; break;  }
  printEulerUtil(u);
  cout << endl;
}

void Graph::printEulerUtil(int u){
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i){
      int v = *i;
      if (v != -1 && isValidNextEdge(u, v)){
          cout << u << "-" << v << "   ";
          rmvEdge(u, v);
          printEulerUtil(v);
      }
  }
}

bool Graph::isValidNextEdge(int u, int v){
  int count = 0;
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i)
     if (*i != -1)
        count++;
  if (count == 1)
    return true;
  bool visited[V];
  memset(visited, false, V);
  int count1 = DFSCount(u, visited);
  rmvEdge(u, v);
  memset(visited, false, V);
  int count2 = DFSCount(u, visited);
  addEdge(u, v);
  return (count1 > count2)? false: true;
}
```

```
void Graph::rmvEdge(int u, int v){
  list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);
  *iv = -1;
  list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);
  *iu = -1;
}

int Graph::DFSCount(int v, bool visited[]){
  visited[v] = true;
  int count = 1;
  list<int>::iterator i;
  for (i = adj[v].begin(); i != adj[v].end(); ++i)
      if (*i != -1 && !visited[*i])
          count += DFSCount(*i, visited);
  return count;
}

int main()
{
  int V, E, v1, v2;
  cout << "Enter the number of vertices and edges: ";
  cin >> V >> E;
  Graph g1(V);
  for(int i = 0; i < E; i++){
      cin >> v1 >> v2;
      g1.addEdge(v1,v2);
  }

  g1.printEulerTour();

  return 0;
}
```

Скріншоти результату:

```
0 1
0 12
1 9
1 8
1 2
2 9
2 7
2 6
2 4
2 3
3 6
4 6
5 6
5 8
6 7
6 8
7 8
7 9
8 9
8 10
9 10
9 12
10 12
10 11
11 12
0-1  1-9  9-2  2-1  1-8  8-5  5-6  6-2  2-7  7-6  6-3  3-2  2-4  4-6  6-8  8-7  7-9  9-8  8-10  10-9  9-12  12-10  10-11
  11-12  12-0
```

б)Елементарні цикли:
V2-V9-V10;
V3-V5-V7;
V3-V8-V10;
V7-V8-V9;
V10-V11-V13, V1-V2-V3-V4-V7-V6-V9-V11-V12-V13-V1.



Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

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

Побудуємо таблицю істинності:

X	Y	Z	XY	$\overline{XY}$	$\bar{Y}$	$\bar{XY}$	$\bar{XYZ}$	$\bar{X}$	$\bar{XY}$	$\bar{XYZ} \vee \bar{XY}$	$\overline{XY}\,\bar{(XYZ \vee XY)}$	$\overline{\overline{XY}\,\bar{(XYZ \vee XY)}}$
0	0	0	0	1	1	0	0	1	0	0	0	1
0	0	1	0	1	1	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0	1	1	1	1	0
0	1	1	0	1	0	0	0	1	1	1	1	0
1	0	0	0	1	1	1	0	0	0	0	0	1
1	0	1	0	1	1	1	1	0	0	1	1	0
1	1	0	1	0	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	0	0	0	0	1

Заповнимо карту Карно:

X\YZ	**00**	**01**	**11**	**10**
0	1	1	0	0
1	1	0	1	1

Після об’єднання мінімізуємо функцію:

$\overline{X}\,\overline{Y} \vee X\overline{Y} \vee X\overline{Z}$