МІНІСТЕРСТВО ОСВІТИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСЕТЕТ
«ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра штучного інтелекту

Розрахункова робота

З дисципліни

«Дискретна математика»

Виконав:

Студент групи КН-115

Курило Валентин

Викладач:

Мельникова Н.І.

Львів-2019

Варіант – 17.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Завдання № 1

Виконати наступні операції над графами:

1) знайти доповнення до першого графу.

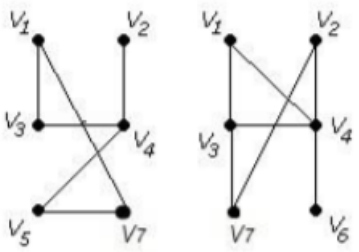
2) об’єднання графів.

3) кільцеву сумму G1 та G2 (G1+G2).

4) розмножити вершину у другому графі.

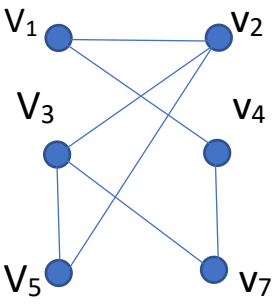
5) виділити підграф А - що скадається з 3-х вершин в G1.

6) добуток графів.

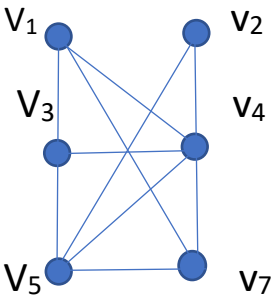


Розв’язання:

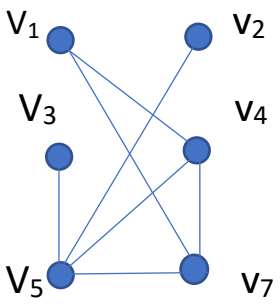
1) знайти доповнення до першого графу:



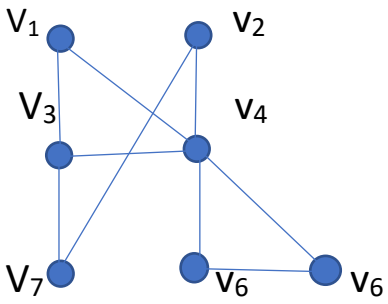
2) об’єднання графів.



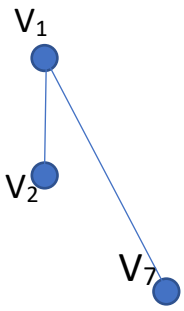
3) кільцеву сумму G1 та G2 (G1+G2).



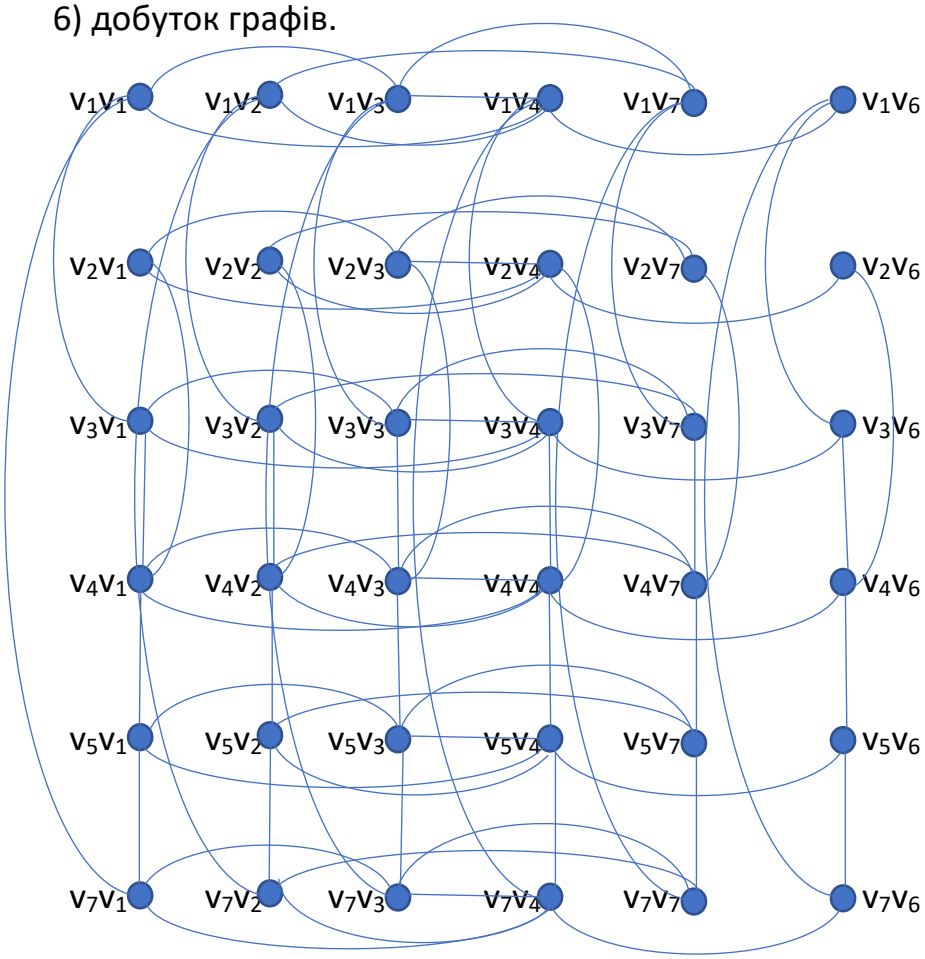
4) розмножити вершину у другому графі.



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

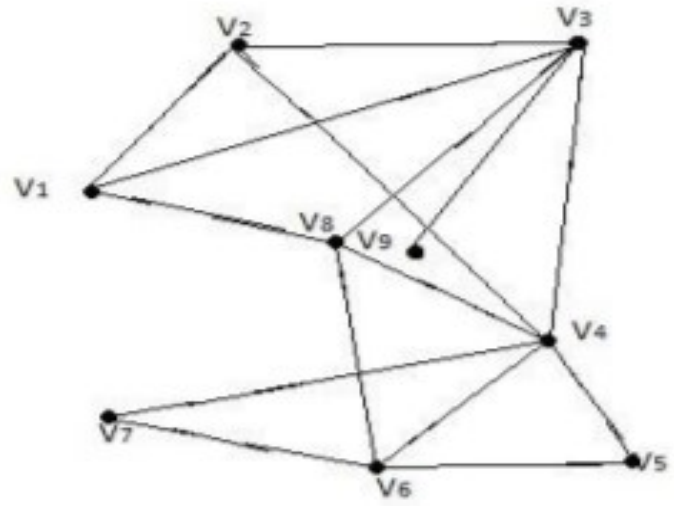


6) добуток графів.



Завдання № 2

Скласти таблицю суміжності для орграфа.



	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_1	0	1	1	0	0	0	0	1	0
V_2	1	0	1	1	0	0	0	0	0
V_3	1	1	0	1	0	0	0	1	1
V_4	0	1	1	0	1	1	1	1	0
V_5	0	0	0	1	0	1	0	0	0
V_6	0	0	0	1	1	0	1	1	0
V_7	0	0	0	1	0	1	0	0	0
V_8	1	0	1	1	0	1	0	0	0
V_9	0	0	1	0	0	0	0	0	0

Завдання № 3

Для графа з другого завдання знайти діаметр.

Діаметр графа 3.

Завдання № 4

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

Розв’язання:

V	N	Стек
V_1	1	V_1
V_2	2	V_1V_2
V_3	3	$V_1V_2V_3$
V_9	4	$V_1V_2V_3V_9$
___	___	$V_1V_2V_3$
V_4	5	$V_1V_2V_3V_4$
V_5	6	$V_1V_2V_3V_4V_5$
V_6	7	$V_1V_2V_3V_4V_5V_6$
V_7	8	$V_1V_2V_3V_4V_5V_6V_7$
___	___	$V_1V_2V_3V_4V_5V_6$
___	___	$V_1V_2V_3V_4V_5$
___	___	$V_1V_2V_3V_4$
V_8	9	$V_1V_2V_3V_4V_8$
___	___	$V_1V_2V_3V_4$
___	___	$V_1V_2V_3$
___	___	V_1V_2
___	___	V_1
___	___	$\varnothing$

Програма:

```
#include <iostream>
#include <list>
using namespace std;
class DFSGraph
{
int V;
list<int> *adjList;
void DFS_util(int v, bool visited[]);
public:
DFSGraph(int V)
   {
 this->V = V;
 adjList = new list<int>[V];
   }
void addEdge(int v, int w)
{
adjList[v].push_back(w);
}
void DFS();
};
void DFSGraph::DFS_util(int v, bool visited[])
{
visited[v] = true;
cout << v << " ";
```

```
list<int>::iterator i;
for(i = adjList[v].begin(); i != adjList[v].end(); ++i)
if(!visited[*i])
DFS_util(*i, visited);

}

void DFSGraph::DFS()

{
bool *visited = new bool[V];
for (int i = 0; i < V; i++)
visited[i] = false;
for (int i = 0; i < V; i++)

if (visited[i] == false)

DFS_util(i, visited);
}
int main()

{
DFSGraph gdfs(9);
gdfs.addEdge(0, 1);
gdfs.addEdge(0, 2);
```

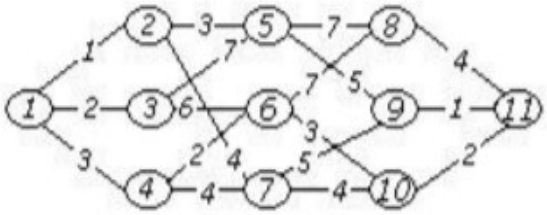
```
gdfs.addEdge(0, 7);

gdfs.addEdge(1, 2);

gdfs.addEdge(1, 3);

gdfs.addEdge(2, 3);

gdfs.addEdge(2, 7);

gdfs.addEdge(2, 8);

gdfs.addEdge(3, 4);

gdfs.addEdge(3, 5);

gdfs.addEdge(3, 6);

gdfs.addEdge(3, 7);

gdfs.addEdge(4, 5);

gdfs.addEdge(5, 6);

gdfs.addEdge(5, 7);

cout << "Depth-first traversal for the given graph:"<<endl;

gdfs.DFS();

return 0;

}
```

Результат:

```
Depth-first traversal for the given graph:
0 1 2 3 4 5 6 7 8
Process returned 0 (0x0)   execution time : 0.199 s
Press any key to continue.
```

Завдання № 5

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

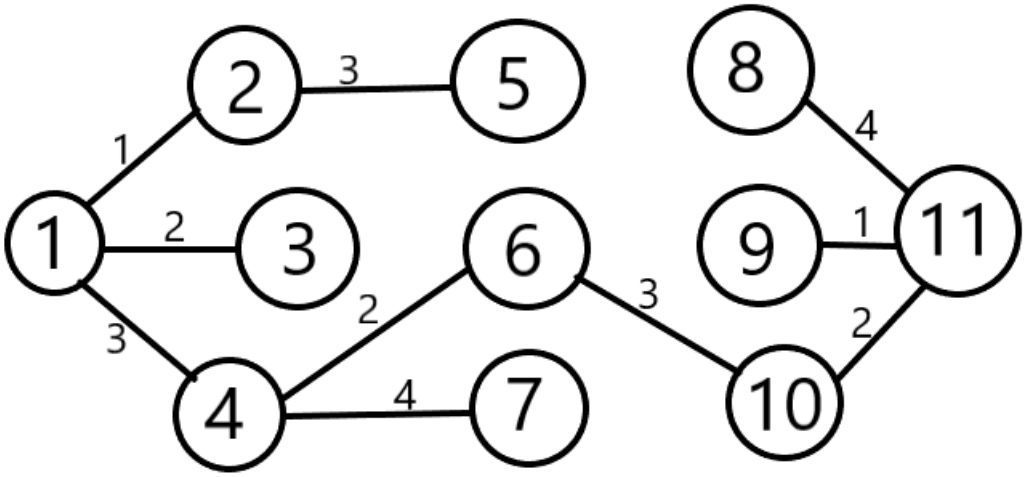


Розв’язання:

Метод Краскала:

V = (1,2,9,11,3,4,6,10,5,7,8).

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



Програма:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
using namespace std;
struct Edge
{
   int src, dest, weight;
};
struct Graph
{
      int V, E;
```

```
        struct Edge* edge;
};
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;
    graph->E = E;
    graph->edge = (struct Edge*) malloc(graph->E * sizeof(struct Edge));
    return graph;
}
struct subset
{
        int parent;
        int rank;
};
int find(struct subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
```

```
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*) a;
    struct Edge* b1 = (struct Edge*) b;
    return a1->weight > b1->weight;
}
void KruskalMST(struct Graph* graph)
{
    int V = graph->V;
    struct Edge result[V];
    int e = 0;
    int i = 0;
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);
    struct subset *subsets = (struct subset*) malloc(V * sizeof(struct subset));
```

```
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    while (e < V - 1)
    {
        struct Edge next_edge = graph->edge[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }
    cout<<"Following are the edges in the constructed MST\n";
    for (i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src, result[i].dest,
                result[i].weight);
    return;
}
int main()
{
    setlocale(LC_CTYPE, "ukr");
```

```
    int x1, x2;
    cout << "Введіть кількість вершин: ";
    cin >> x1;
    cout << "Введіть кількість ребрів: ";
    cin >> x2;
    int V = x1;
    int E = x2;
    int a1, r1, r2;
    struct Graph* graph = createGraph(V, E);
     for(int i = 0; i < x2; i++)
    {
    cout << "Введіть ребро: " ;
    cin >> r1;
    cin >> r2;
    graph->edge[i].src = r1;
    graph->edge[i].dest = r2;
    cout << "Введіть вагу ребра: ";
    cin >> a1;
    graph->edge[i].weight = a1;
    }
    KruskalMST(graph);
    return 0;
}
```

Результат:

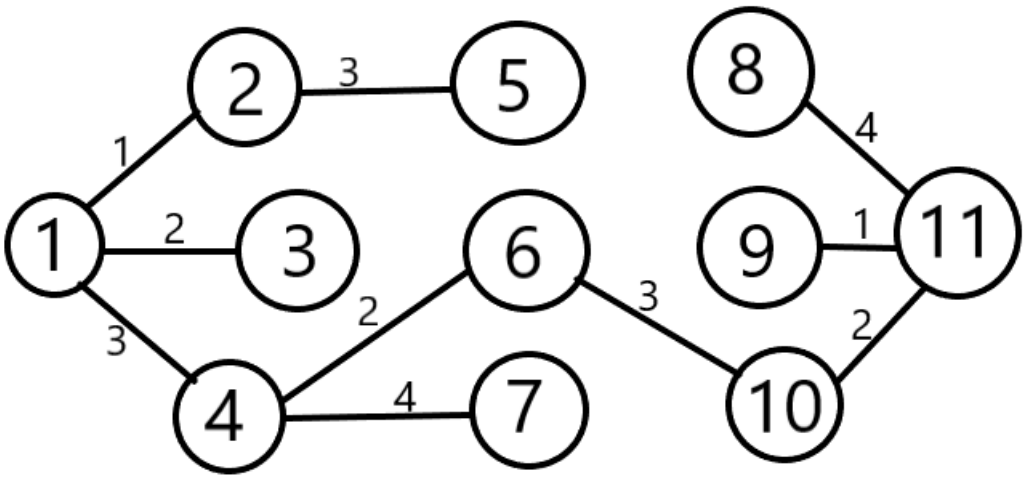
```
Введіть кількість вершин: 11
Введіть кількість ребрів: 18
Введіть ребро: 0 1
Введіть вагу ребра: 1
Введіть ребро: 0 2
Введіть вагу ребра: 2
Введіть ребро: 0 3
Введіть вагу ребра: 3
Введіть ребро: 1 4
Введіть вагу ребра: 3
Введіть ребро: 1 6
Введіть вагу ребра: 4
Введіть ребро: 2 4
Введіть вагу ребра: 7
Введіть ребро: 2 5
Введіть вагу ребра: 6
Введіть ребро: 3 5
Введіть вагу ребра: 2
Введіть ребро: 3 6
Введіть вагу ребра: 4
Введіть ребро: 4 7
Введіть вагу ребра: 7
Введіть ребро: 4 8
Введіть вагу ребра: 5
Введіть ребро: 5 7
Введіть вагу ребра: 7
Введіть ребро: 5 9
Введіть вагу ребра: 3
Введіть ребро: 6 8
Введіть вагу ребра: 5
Введіть ребро: 6 9
Введіть вагу ребра: 4
Введіть ребро: 7 10
Введіть вагу ребра: 4
Введіть ребро: 8 10
Введіть вагу ребра: 1
Введіть ребро: 9 10
Введіть вагу ребра: 2
```

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

Метод Прима:

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

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



Програма:

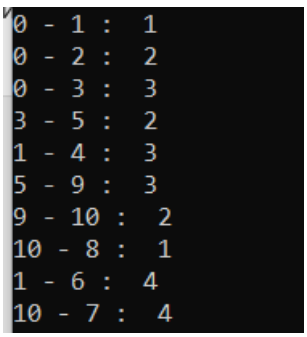
```
#include <iostream>

#include <cstring>

using namespace std;

#define INF 9999

#define V 11

 int G[V][V] = {


    {0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0},

    {1, 0, 0, 0, 3, 0, 4, 0, 0, 0, 0},

    {2, 0, 0, 0, 7, 6, 0, 0, 0, 0, 0},

    {1, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0},

    {0, 3, 7, 0, 0, 0, 0, 7, 5, 0, 0},

    {0, 0, 6, 2, 0, 0, 0, 7, 0, 3, 0},

    {0, 4, 0, 4, 0, 0, 0, 0, 5, 4, 0},
```

```
    {0, 0, 0, 0, 7, 7, 0, 0, 0, 0, 4},
    {0, 0, 0, 0, 5, 0, 5, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 2},
    {0, 0, 0, 0, 0, 0, 0, 4, 1, 2, 0},
   };

int main () {

int num_edge;
  int mst_vertex[V];

  memset (mst_vertex, false, sizeof (mst_vertex));

  num_edge = 0;
  mst_vertex[0] = true;
   int x;
  int y;
  cout<<"The Minimum Spanning Tree as per Prim's Algorithm:"<<endl;
  cout << "Edge" << " : " << "Weight";
  cout << endl;

  while (num_edge < V - 1) {
int min = INF;

    x = 0;
```

```
    y = 0;



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

if (mst_vertex[i]) {

for (int j = 0; j < V; j++) {

if (!mst_vertex[j] && G[i][j]) {

if (min > G[i][j]) {

min = G[i][j];

                    x = i;

                    y = j;

                }

            }

          }

        }
```

```
    }
cout << x <<  " - " << y << " :  " << G[x][y];


cout << endl;


mst_vertex[y] = true;


num_edge++;


   }




return 0;


}
```

Результат:



Завдання № 6

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

17)

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

Розв’язання:

1→5→2→7→4→8→6→3→1

Відповідь: Найкоротший шлях 19.

Програма:

```
#include <bits/stdc++.h>
using namespace std;
#define V 8
int travllingSalesmanProblem(int graph[][V], int s)
{
   vector<int> vertex;
   for (int i = 0; i < V; i++)
      if (i != s)
         vertex.push_back(i);
   int min_path = INT_MAX;
   do {
      int current_pathweight = 0;
```

```
        int k = s;
        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);

    } while (next_permutation(vertex.begin(), vertex.end()));

    return min_path;
}

int main()
{
    int graph[][V] = { { 0, 6, 6, 6, 1, 3, 1, 3 },
                       { 6, 0, 5, 5, 1, 6, 1, 5 },
                       { 6, 5, 0, 7, 7, 7, 7, 5 },
                       { 6, 5, 7, 0, 6, 5, 1, 2 },
                       { 1, 1, 7, 6, 0, 6, 6, 6 },
                       { 3, 6, 7, 5, 6, 0, 1, 2 },
                       { 1, 1, 7, 1, 6, 1, 0, 2 },
                       { 3, 5, 5, 2, 6, 2, 2, 0 }};
    int s = 0;
    cout << travllingSalesmanProblem(graph, s) << endl;
```

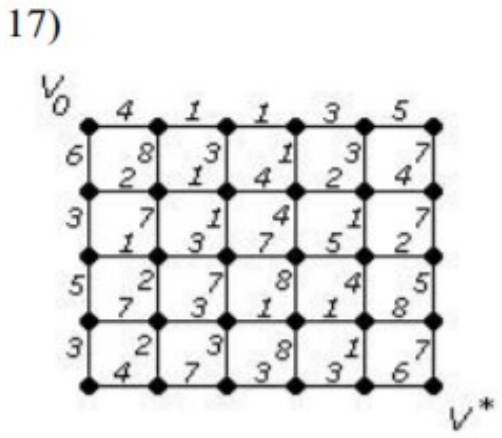
```
    return 0;
}
```

Результат:

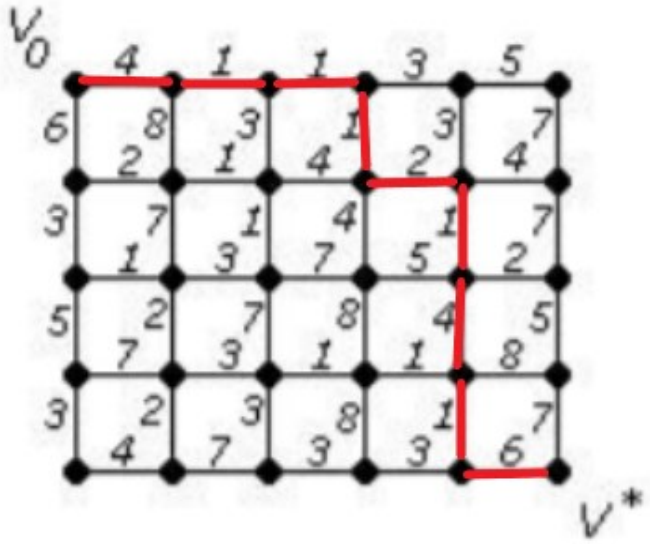
```
19

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

Завдання № 7

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



Розв’язання:



Відповідь: Найкоротший шлях становить 21.

Програма:

```
#include <iostream>
#include <limits.h>
#include <stdio.h>
#define V 30
using namespace std;
int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}

int printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
```

```
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;

        for (int v = 0; v < V; v++)

            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist);
}
```

```
int main()

{

    int graph[V][V] = { { 0, 4, 0, 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, 0},

      { 4, 0, 1, 0, 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, 1, 0, 1, 0, 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, 1, 0, 3, 0, 0, 0, 0, 1, 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, 5, 0, 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, 5, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

      { 6, 0, 0, 0, 0, 0, 0, 2, 0, 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, 0, 2, 0, 1, 0, 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, 0, 1, 0, 4, 0, 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, 0, 4, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 4, 0, 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, 0, 4, 0, 0, 0, 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, 0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 7, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 7, 0, 5, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 7, 0, 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, 0, 7, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 3, 0, 1, 0, 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, 0, 1, 0, 1, 0, 0, 0, 0, 8, 0, 0},

      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 1, 0, 8, 0, 0, 0, 0, 1, 0},
```

```
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 7},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 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, 0, 2, 0, 0, 0, 0, 4, 0, 7, 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, 7, 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, 0, 8, 0, 0, 0, 0, 3, 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, 0, 0, 1, 0, 0, 0, 0, 3, 0, 6},
    { 0, 0, 0, 0, 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, 0, 6, 0},
     };


    dijkstra(graph, 0);


    return 0;
}
```

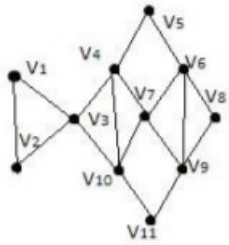
Результат:

Vertex	Distance from Source
0	0
1	4
2	5
3	6
4	9
5	14
6	6
7	8
8	8
9	7
10	9
11	13
12	9
13	10
14	9
15	11
16	10
17	12
18	14
19	12
20	15
21	15
22	14
23	17
24	17
25	14
26	18
27	18
28	15
29	21

Завдання № 8

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

17)



Розв’язання:

Методом Флері:

V1→v2→v3→v4→v5→v6→v7→v4→v10→v7→v9→v6→v8→v9→v11→v10→v3→v2→v1.

Програма:

```
#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()
{
```

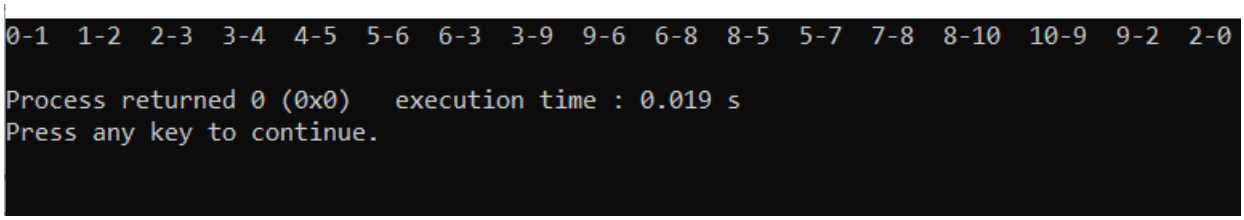
```
  int u = 0;
  for (int i = 0; i < V; i++)
      if (adj[i].size() & 1)
        {   u = i; break;  }
  printEulerUtil(u);
  cout << endl;
}

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()
{
  Graph g1(17);
  g1.addEdge(0, 1);
  g1.addEdge(0, 2);
  g1.addEdge(1, 2);
  g1.addEdge(2, 3);
  g1.addEdge(2, 9);
  g1.addEdge(3, 4);
  g1.addEdge(3, 6);
  g1.addEdge(3, 9);
  g1.addEdge(4, 5);
  g1.addEdge(5, 6);
```

```
 g1.addEdge(5, 7);
 g1.addEdge(5, 8);
 g1.addEdge(6, 8);
 g1.addEdge(6, 9);
 g1.addEdge(7, 8);
 g1.addEdge(8, 10);
 g1.addEdge(9, 10);
 g1.printEulerTour();
 return 0;
}
```

Результат:



Методом елементарних циклів:

V1v2v3→v3v4v7v10v11v9→v7v9v8v6→v7v6v5v4→v4v10v3→v1v2v3.

Програма:

```
#include<iostream>
#include <list>
using namespace std;
class Graph
{
   int V;
   list<int> *adj;
public:
```

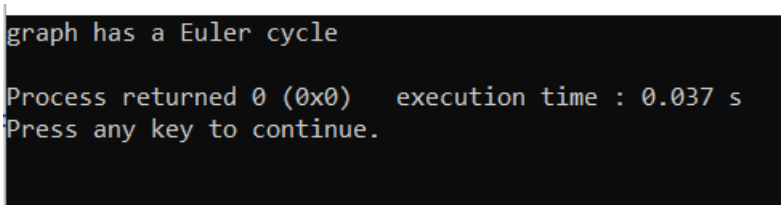
```
    Graph(int V)   {this->V = V; adj = new list<int>[V]; }
    ~Graph() { delete [] adj; }
    void addEdge(int v, int w);
    int isEulerian();
    bool isConnected();
    void DFSUtil(int v, bool visited[]);
};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
    adj[w].push_back(v);
}
void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}
bool Graph::isConnected()
{
    bool visited[V];
    int i;
```

```
    for (i = 0; i < V; i++)
        visited[i] = false;
    for (i = 0; i < V; i++)
        if (adj[i].size() != 0)
            break;
    if (i == V)
        return true;
    DFSUtil(i, visited);
    for (i = 0; i < V; i++)
      if (visited[i] == false && adj[i].size() > 0)
            return false;

    return true;
}
int Graph::isEulerian()
{
    if (isConnected() == false)
        return 0;
    int odd = 0;
    for (int i = 0; i < V; i++)
        if (adj[i].size() & 1)
            odd++;
    if (odd > 2)
        return 0;h
    return (odd)? 1 : 2;
```

```
}
void test(Graph &g)
{
   int res = g.isEulerian();
   if (res == 0)
      cout << "graph is not Eulerian\n";
   else if (res == 1)
      cout << "graph has a Euler path\n";
   else
      cout << "graph has a Euler cycle\n";
}

int main()
{
  Graph g1(17);
  g1.addEdge(0, 1);
  g1.addEdge(0, 2);
  g1.addEdge(1, 2);
  g1.addEdge(2, 3);
  g1.addEdge(2, 9);
  g1.addEdge(3, 4);
  g1.addEdge(3, 6);
  g1.addEdge(3, 9);
  g1.addEdge(4, 5);
  g1.addEdge(5, 6);
```

```
  g1.addEdge(5, 7);

  g1.addEdge(5, 8);

  g1.addEdge(6, 8);

  g1.addEdge(6, 9);

  g1.addEdge(7, 8);

  g1.addEdge(8, 10);

  g1.addEdge(9, 10);

  test(g1);


    return 0;

}
```

Результат:



Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

17. $x\bar{y} \vee \bar{x}\bar{z} \vee yz$

Розв’язання:

(x $\overline{y}$ z)V(x $\overline{y}$ $\overline{z}$)V($\overline{x}$ y $\overline{z}$)V($\overline{x}$ $\overline{y}$ $\overline{z}$)V(x y z)V($\overline{x}$ y z).