МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Розрахункова робота

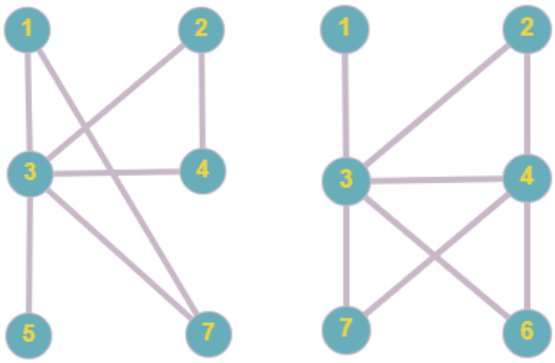
з дисципліни
«Дискретна математика»

Виконала:
студентка групи КН-114
Кмитюк Катерина
Перевірила:
Мельникова Н. І.

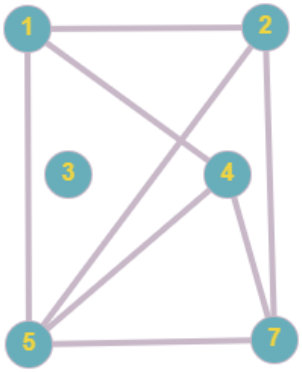
Львів – 2019

Варіант 7

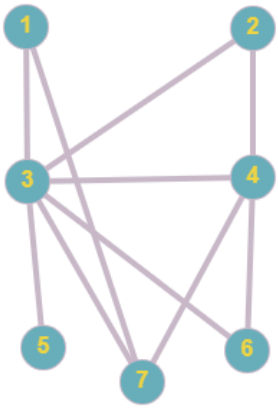
1. Виконати наступні операції над графами:



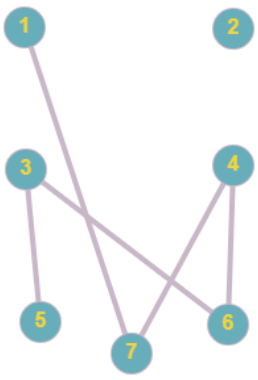
1) знайти доповнення до першого графу



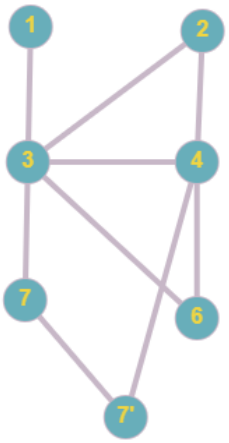
2) об’єднання графів



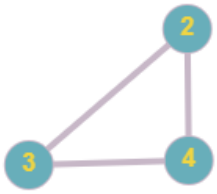
3) кільцеву сумму G1 та G2 (G1+G2)



4) розмножити вершину у другому графі

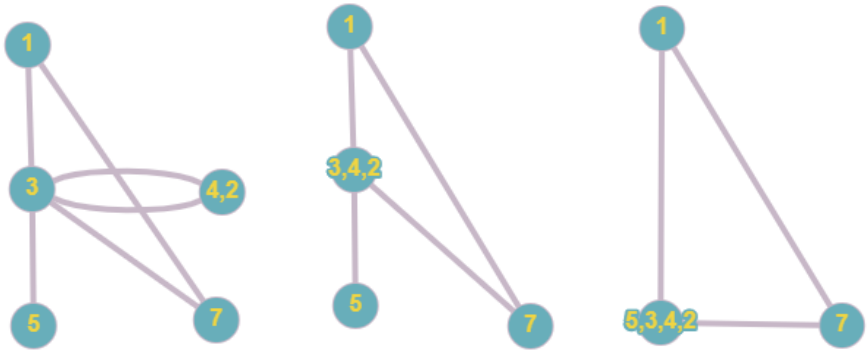


5) виділити підграф А - що скадається з 3-х вершин в G1

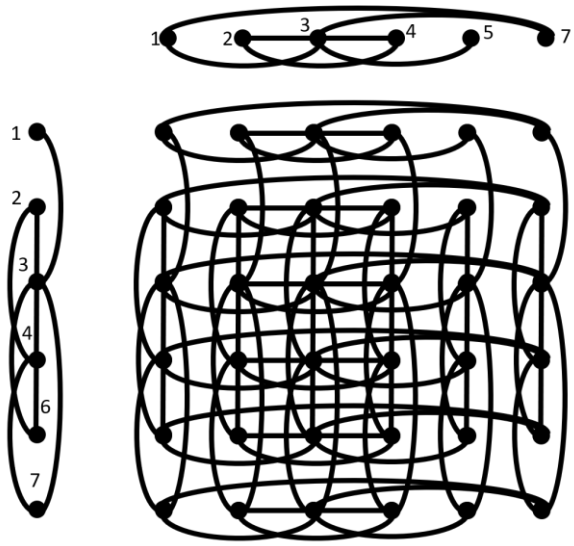


Граф А:

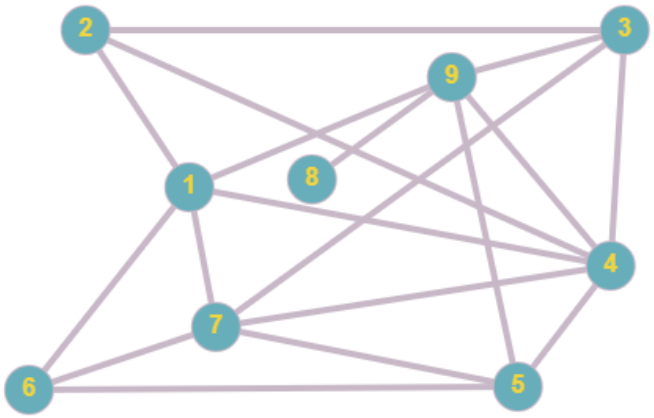
Стягування графу G1:



6) добуток графів



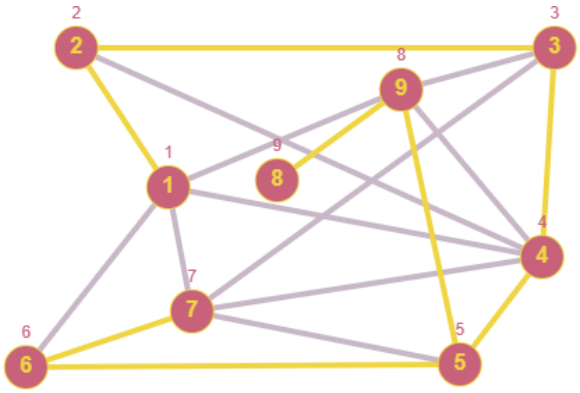
2. Скласти таблицю суміжності для графа:



	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	1	1	0	1
2	1	0	1	1	0	0	0	0	0
3	0	1	0	1	0	0	1	0	1
4	1	1	1	0	1	0	1	0	1
5	0	0	0	1	0	1	1	0	1
6	1	0	0	0	1	0	1	0	0
7	1	0	1	1	1	1	0	0	0
8	0	0	0	0	0	0	0	0	1
9	1	0	1	1	1	0	0	1	0

3. Діаметр графа = 3. Адже саме за 3 кроки можна дістатися з вершини 8 до 2, 7, 6.
4. Обхід вглиб:

№	Стек	Порядок додавання вершин
1	1	1
2	12	2
3	123	3
4	1234	4
5	12345	5
6	123456	6
7	1234567	7
8	123456	-
9	12345	-
10	123459	9
11	1234598	8
12	123459	-
13	12345	-
14	1234	-
15	123	-
16	12	-
17	1	-
18	∅	-



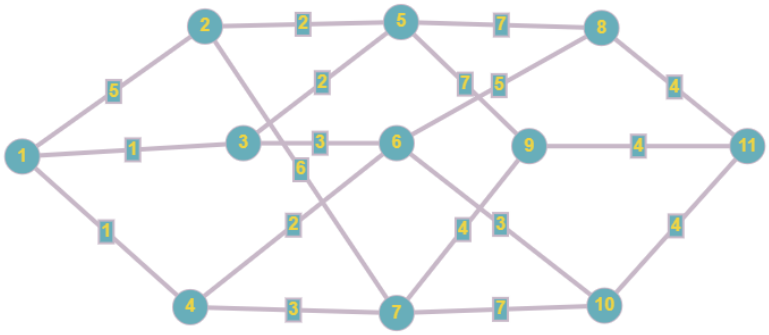
Програмна реалізація:

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

void DFS(int st, int n, int **v, bool *visited)
{
    cout<<st+1<<" ";
    visited[st]=1;
    for (int i=0; i<=n; i++)
    if ((v[st][i]!=0) && (!visited[i]))
    DFS(i, n, v, visited);
}

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

    int **v;
    int n, temp;
    int start;

    cout << "Введіть кількість вершин" << endl;
    cin >> n;
    v = (int **) calloc(n, sizeof(int*));
    for (int i=0; i<n; i++)
        v[i] = (int *) calloc(n, sizeof(int));

    bool *visited = (bool *) calloc(n, sizeof(bool));

    cout << "Введіть 1 (є ребро) або 0 (немає ребра)" << endl;
    cout << "Таблиця суміжності:" << endl;

    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            cin >> temp;
            if (temp == 1)
                v[i][j]=temp;
            else v[i][j]=0;
        }
        visited[i]=0;
    }

    cout << "Введіть початкову вершину" << endl;
    cin >> start;

    cout << "Порядок обходу:" << endl;
    DFS(start-1, n, v, visited);
    cout << endl;

    return 0;
}
```

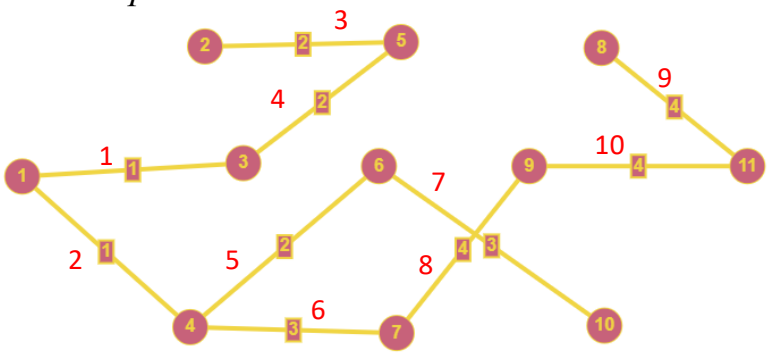
Приклад виконання програми:

```
Введіть кількість вершин
9
Введіть 1 (є ребро) або 0 (немає ребра)
Таблиця суміжності:
0 1 0 1 0 1 1 0 1
1 0 1 1 0 0 0 0 0
0 1 0 1 0 0 1 0 1
1 1 1 0 1 0 1 0 1
0 0 0 1 0 1 1 0 1
1 0 0 0 1 0 1 0 0
1 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1
1 0 1 1 1 0 0 1 0
Введіть початкову вершину
1
Порядок обходу:
1 2 3 4 5 6 7 9 8
```

5. Знайти мінімальне остове дерево методами Краскала та Прима



Метод Краскала:



Вибираємо найменше ребро та додаємо його до остового дерева, якщо є кілька ребер однакової ваги, обираємо довільне, слідкуємо, щоб не утворювався цикл. Спочатку розглядаємо ребро вагою 1 із суміжними вершинами 1 та 3, додаємо його в остове дерево, продовжуємо шукати найменші ребра, доки усі вершини не будуть додані, а кожне ребро перевірене, чи не буде при його додаванні утворюватися цикл.

Програмна реалізація:

```
#include <iostream>

using namespace std;

int count_edge (int n, int **a, int j)
{
    int k=0;
    for (int i=0; i<n; i++)
        if (a[j][i]==1) k++;
    return k;
}

bool cir (int n, int **a)
{
    int k=0;
    int **b = (int **)calloc(n, sizeof(int *));
    for (int i=0; i<n; i++)
        b[i] = (int *)calloc(n, sizeof(int));

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
                b[i][j]=a[i][j];

    int l=0;
    for (int i=0; i<n; i++)
        l+=count_edge(n,a,i);
    for (int t=0; t<l; t++)
    for (int i=0; i<n; i++)
    {
        if (count_edge(n,b,i)<=1)
            for (int j=0; j<n; j++)
            {
                b[i][j]=0;
                b[j][i]=0;
            }
    }

    for (int i=0; i<n; i++)
        k+=count_edge(n,b,i);

    if (k%2 == 0 && k!=0) return 1;
    else return 0;
}


int main()
{
    int *v,*e;
    int k;
    int **w, **temp;
    bool cheki=1, chekj=1;

    cout << "Input count of vertices" << endl;
    cin >> k;

    w=(int **)calloc(k, sizeof(int *));
    temp=(int **)calloc(k,sizeof(int *));
    for (int i=0; i<k; i++)
    {
        w[i]=(int *)calloc(k+1, sizeof(int));
        temp[i]=(int *)calloc(k+1, sizeof(int));
    }

    v = (int *)calloc((k*k), sizeof(int));
    e = (int *)calloc((k*k), sizeof(int));

    for (int i=0; i<(k*k); i++)
    {
        e[i]=0;
        v[i]=0;
    }

    cout << "Input weight of edges" << endl;
    for (int i=0; i<k; i++)
```

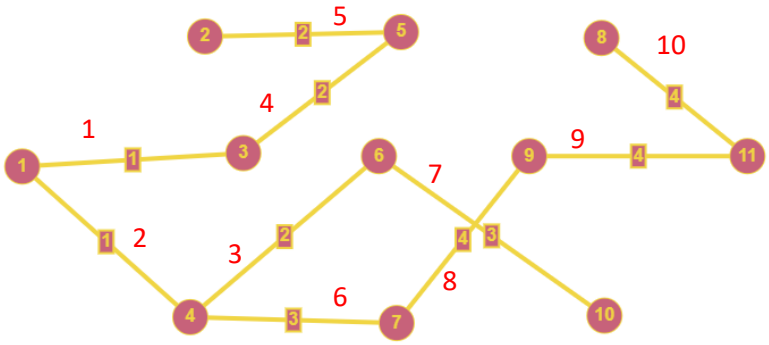
```
    {
        for (int j=0; j<k; j++)
        {
            if (j>i)
            {
                cout << i+1 << " - " << j+1 << " ";
                cin >> w[i][j];
            }
            else w[i][j]=0;
        }
        cout << endl;
    }
int C=k;
int p,q;
int minim=INT_MAX;
    for (int l=0; l<C; l++)
    {
        for (int i=0; i<k; i++)
            for (int j=0; j<k; j++)
            if (w[i][j]<minim && w[i][j]!=0)
            {
                p=i; q=j;
                minim=w[i][j];
            }
        temp[p][q]=1;
        temp[q][p]=1;
        for (int t=0; t<C; t++)
        {
            if (v[t]!=(p+1)) cheki*=1;
            else cheki*=0;
            if (v[t]!=(q+1)) chekj*=1;
            else chekj*=0;
        }
        if ((cheki+chekj)==2)
        {
            e[l]=p+1;
            v[l]=p+1;
            l++;
            C++;
            e[l]=q+1;
            v[l]=q+1;
        }
        else{
        if (cheki==1)
        {
            v[l]=p+1;
            e[l]=p+1;
            l++;
            C++;
            e[l]=q+1;
        }
        else{
        if (chekj==1)
        {
            v[l]=q+1;
            e[l]=p+1;
            l++;
            C++;
            e[l]=q+1;
        }
        else{
        if (cheki+chekj==0 && cir(k,temp)==0)
        {
            e[l]=p+1;
            l++;
            C++;
            e[l]=q+1;
        }
        else {temp[p][q]=0; temp[q][p]=0;}}}}
        minim=INT_MAX;
        w[p][q]=0;
        cheki=1;
        chekj=1;
    }
```

```
149
150     int news=0;
151     for (int i=0; i<C; i++)
152         if (e[i]!=0)
153         {
154             e[news]=e[i];
155             news++;
156         }
157     cout << "E: { ";
158     for (int i=0; i<news; i++)
159     {
160         cout << "(" << e[i] << ", " << e[i+1] << ") ";
161         i++;
162     }
163     cout << "}\n";
164
165     news=0;
166     for (int i=0; i<C; i++)
167         if (v[i]!=0)
168         {
169             v[news]=v[i];
170             news++;
171         }
172     cout << "V: { ";
173     for (int i=0; i<news; i++)
174         cout << v[i] << " ";
175     cout << "}\n";
176     return 0;
177 }
178
```

Приклад виконання програми:

```
Input count of vertices
11
Input weight of edges
1 - 2 6
1 - 3 1
1 - 4 1
1 - 5 0
1 - 6 0
1 - 7 0
1 - 8 0
1 - 9 0
1 - 10 0
1 - 11 0

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

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

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

5 - 6 0
5 - 7 0
5 - 8 7
5 - 9 5
5 - 10 0
5 - 11 0

6 - 7 0
6 - 8 5
6 - 9 0
6 - 10 3
6 - 11 0

7 - 8 0
7 - 9 4
7 - 10 7
7 - 11 0

8 - 9 0
8 - 10 0
8 - 11 4

9 - 10 0
9 - 11 4

10 - 11 4


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

Метод Прима:



Для отримання мінімального остового дерева методом Прима обираємо довільну вершину та шукаємо найменше ребро інцидентне їй, додаємо його у дерево разом із суміжною вершиною. Далі серед вже доданих вершин шукаємо найменше ребро та додаємо його. Слідкуємо, щоб не утворювався цикл, для цього потрібно додавати ребра, якщо вже у дереві наявна лише одна з вершин, інцидентних їй.

Програмна реалізація:

```
#include<conio.h>
#include<iostream>
using namespace std;

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

    int a,b,u,v,n;
    int *visited, **cost;
    int mins;

    int *ver;
    int ver_index=0;

    cout << "Введіть кількість вершин  "; cin >> n;
    cost = (int **) calloc(n,sizeof(int *));
    visited = (int *) calloc(n,sizeof(int ));
    ver = (int *)calloc(n*n, sizeof(int));
    for (int i=1; i<=n; i++)
        cost[i] = (int *) calloc(n,sizeof(int ));

    cout << "Введіть матрицю суміжності" << endl;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if (j>i)
            {
                cout << i << " - " << j << "  ";
                cin >> cost[i][j];
            }
            else cost[i][j]=0;
        }
        cout << endl;
    }
```

```
    visited[1]=1;
    cout << endl;

    int ne=1;
    while(ne < n)
    {
        for(int i=1,mins=INT_MAX; i<=n; i++)
            for(int j=1; j<=n; j++)
                if(cost[i][j]< mins && cost[i][j]!=0)
                    if(visited[i]!=0)
                    {
                        mins=cost[i][j];
                        a=u=i;
                        b=v=j;
                    }
        if(visited[u]==0 || visited[v]==0)
        {
            ver[ver_index]=b;
            ver_index++;
            ne++;
            visited[u]=1;
            visited[v]=1;

        }
        cost[a][b]=cost[b][a]=0;
    }


    cout << endl;

    cout << "V = { " << 1 << " ";
    for (int i=0; i<ver_index; i++)
      cout << ver[i] << " ";

    cout << "}" << endl;

    return 0;
```

Приклад виконання програми:

```
Введіть кількість вершин  11
Введіть матрицю суміжності
1 - 2  6
1 - 3  1
1 - 4  1
1 - 5  0
1 - 6  0
1 - 7  0
1 - 8  0
1 - 9  0
1 - 10  0
1 - 11  0

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

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

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

5 - 6  0
5 - 7  0
5 - 8  7
5 - 9  5
5 - 10  0
5 - 11  0

6 - 7  0
6 - 8  5
6 - 9  0
6 - 10  3
6 - 11  0

7 - 8  0
7 - 9  4
7 - 10  7
7 - 11  0

8 - 9  0
8 - 10  0
8 - 11  4

9 - 10  0
9 - 11  4

10 - 11  4



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

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

	2	3	4	51	6	7	8
2	∞	7	3	3	5	4	5
3	7	∞	4	5	6	4	5
4	3	4	∞	1	2	5	1
51	3	5	1	∞	5	1	1
6	5	6	2	5	∞	2	3
7	4	4	5	1	2	∞	5
8	5	5	1	1	3	5	∞

	2	3	451	6	7	8
2	∞	7	3	5	4	5
3	7	∞	4	6	4	5
451	3	4	∞	2	5	1
6	5	6	2	∞	2	3
7	4	4	5	2	∞	5
8	5	5	1	3	5	∞

	2	3	6	7	8451
2	∞	7	5	4	5
3	7	∞	6	4	5
6	5	6	∞	2	3
7	4	4	2	∞	5
8451	5	5	3	5	∞

	2	3	68451	7
2	∞	7	5	4
3	7	∞	6	4
68451	5	6	∞	2
7	4	4	2	∞

	2	3	768451

2	∞	7	4
3	7	∞	4
768451	4	4	∞

	3	2768451
3	∞	7
2768451	7	∞

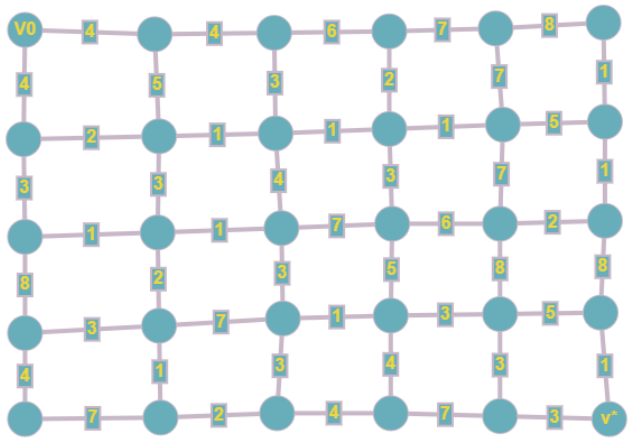
Програмна реалізація:

```
#include <iostream>

using namespace std;

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

    int n;
    int **w, *vis;
    int start, temp;
    cout << "Введіть кількість вершин" << endl;
    cin >> n;
    w = (int **) calloc(n, sizeof(int*));
    for (int i=0; i<n; i++)
        w[i] = (int *)calloc(n, sizeof(int));

    vis = (int *)calloc (n, sizeof(int));

    cout << "Введіть вагу ребра між вершинами" << endl;
    cout << "Таблиця суміжності:" << endl;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            cin >> w[i][j];
    cout << endl;

    cout << "Введіть початкову вершину" << endl;
    cin >> start;

    cout << "Порядок обходу:" << endl;
    int st = start-1;

    int min = INT_MAX;
    for (int i=0; i<n; i++)
    {
        vis[i]=st+1;
        cout << vis[i] << "\t";
        for (int j=0; j<n; j++)
        {
            if(w[st][j]<min && w[st][j]!=0)
            {
                min = w[st][j];
                temp=j;
            }
        }
        min = INT_MAX;
        for (int k=0; k<n; k++)
        {
            w[st][k]=0;
            w[k][st]=0;
        }
        st=temp;

    }
    return 0;
}
```

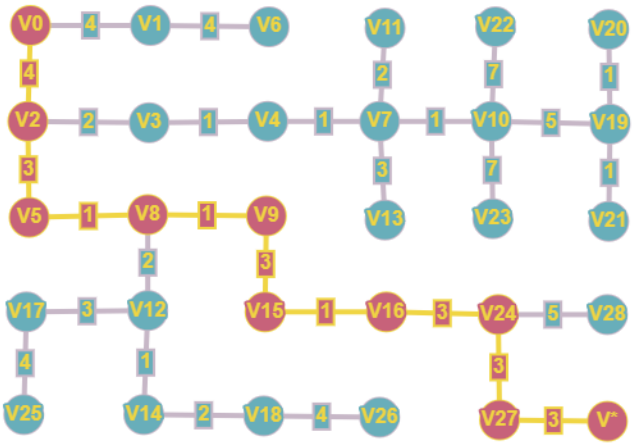
Приклад виконання програми:

```
Введіть кількість вершин
8
Введіть вагу ребра між вершинами
Таблиця суміжності:
0 5 5 5 4 6 5 5
5 0 7 3 3 5 4 5
5 7 0 4 5 6 4 5
5 3 4 0 1 2 5 1
4 3 5 1 0 5 1 1
6 5 6 2 5 0 2 3
5 4 4 5 1 2 0 5
5 5 5 1 1 3 5 0

Введіть початкову вершину
1
Порядок обходу:
1       5       4       8       6       7       2       3
```

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



Для побудови найкоротшого шляху від V0 дод V* додаємо нумеруючи вершини, що є найближчими до V0. Отримаємо наступне кістякове дерево:



Множина найближчих вершин V={V0, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, V28, V*} Відповідне зростаюче дерево D={4, 4, 2, 1, 3, 4, 1, 1, 1, 1, 2, 2, 3, 1, 3, 1, 3, 2, 5, 1, 1, 7, 7, 3, 4, 4, 3, 5, 3} Довжина шляху V0 – V* дорівнює 22.

Програмна реалізація:

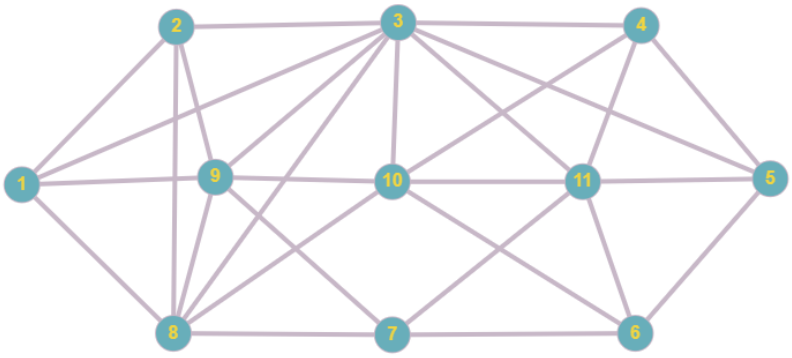
```cpp
#include <iostream>

using namespace std;

int main()
{
    int mind, min;

    int *v, *d;
    int m,n,k,p;
    int **w;
    int r, temp;
    bool *visit;

    cout << "Input count of vertices" << endl;
    cin >> m >> n;
    k=m*n;

    w=(int **)calloc(k, sizeof(int *));
    for (int i=0; i<k; i++)
        w[i]=(int *)calloc(k+1, sizeof(int));


    v = (int *)calloc(k, sizeof(int));
    visit = (bool *)calloc (k, sizeof(bool));
    d = (int *)calloc(k,sizeof (int));

    r=1;
    cout << "Input weight of edges" << endl;
    for (int i=0; i<k; i++)
    {
        for (int j=0; j<k; j++)
        {
            if (j==i+m)
            {
                cout << i+1 << " - " << j+1 << "  ";
                cin >> w[i][j];
            }
            else
            if (j==i+1)
            {
                if ((i+1)!=(m*r))
                {
                    cout << i+1 << " - " << j+1 << "  ";
                    cin >> w[i][j];
                }
                else
                {
                    w[i][j]=w[j][i];
                    r++;
                }
            }
            else w[i][j]=w[j][i];

        }
        cout << endl;
        visit [i]=0;
    }

    cout << "Input last vertic:"; cin >> p;

    for (int i = 0; i<k; i++)
    {
        d[i] = 10000;
        v[i] = 1;
    }
    d[0] = 0;

    do
    {
```

```
    mind = 10000;
    min = 10000;
    for (int i = 0; i<k; i++)
    { // Если вершину ещё не обошли и вес меньше min
      if ((v[i] == 1) && (d[i]<min))
      { // Переприсваиваем значения
        min = d[i];
        mind = i;
      }
    }
    // Добавляем найденный минимальный вес к текущему весу вершины и сравниваем с текущим минимальным весом вершины
    if (mind != 10000)
    {
      for (int i = 0; i<k; i++)
      {
        if (w[mind][i] > 0)
        {
          temp = min + w[mind][i];
          if (temp < d[i])
          {
            d[i] = temp;
          }
        }
      }
      v[mind] = 0;
    }
    } while (mind < 10000);

    int q=p-1;
    // Вывод кратчайших расстояний до вершин
    cout << "The shortest distance from " <<1<<" to "<<p<< endl;
    cout << d[q]<<endl;

    int *ver;
    ver = (int *)calloc(k, sizeof(int)); // массив пройденных вершин
    ver[0] = p; // начальный элемент - конечная вершина
    r = 1; // индекс предыдущей вершины
    int weight = d[q];
    while (q != 0) // пока не дошли до начальной вершины
    {
        for (int i = 0; i<k; i++) // просматриваем все вершины
        if (w[q][i] != 0)    // если связь есть
        {
            temp = weight - w[q][i]; // определяем вес пути из предыдущей вершины
            if (temp == d[i]) // если вес совпал с рассчитанным
            {                 // значит из этой вершины и был переход
            weight = temp; // сохраняем новый вес
            q = i;       // определяем номер этой вершины
            ver[r] = i + 1; // и записываем ее в массив
            r++;
            }
        }
  }
    cout << "The vertics from " <<1<<" to "<<p<< endl;
    for (int i = r - 1; i >= 0; i--)
        cout << ver[i]<<"\t";

    return 0;
}
```

Приклад виконання програми:

```
Input count of vertices
6
5
Input weight of edges
1 - 2  4
1 - 7  4

2 - 3  4
2 - 8  5

3 - 4  6
3 - 9  3

4 - 5  7
4 - 10  2

5 - 6  8
5 - 11  7

6 - 12  1

7 - 8  2
7 - 13  3

8 - 9  1
8 - 14  3

9 - 10  1
9 - 15  4

10 - 11  1
10 - 16  3

11 - 12  5
11 - 17  7

12 - 18  1

13 - 14  1
13 - 19  8

14 - 15  1
14 - 20  2

15 - 16  7
15 - 21  3

16 - 17  6
16 - 22  5
17 - 18  2
17 - 23  8

18 - 24  8

19 - 20  3
19 - 25  4

20 - 21  7
20 - 26  1

21 - 22  1
21 - 27  3

22 - 23  3
22 - 28  4

23 - 24  5
23 - 29  3

24 - 30  1

25 - 26  7

26 - 27  2

27 - 28  4

28 - 29  7

29 - 30  3


Input last vertic:30
The shortest distance from 1 to 30
22
The vertics from 1 to 30
1       7       13      14      15      21      22      23      24      30
```

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



а) Для пошуку ейлерового циклу перш за все має виконуватися умова – степені усіх вершин парні. Після перевірки обирається довільна вершина, видаляється ребро, інцидентне їй і відбувається перехід до наступної вершини, яка була поєднана з початковою видаленим ребром. Процедура повторюється, доки не будуть видалені усі ребра. Також потрібно слідкувати, щоб при видаленні ребра не утворювався дводольний граф.

Послідовність ейлерового циклу методом Флері:
1⇒2⇒3⇒1⇒8⇒2⇒9⇒3⇒4⇒5⇒3⇒8⇒7⇒6⇒5⇒11⇒3⇒10⇒4⇒11⇒6⇒10⇒8⇒9⇒7⇒11⇒10⇒9⇒1

б) Для пошуку ейлерового циклу методом елементарних циклів знаходять всі прості цикли (до яких кожна вершина входить лише раз), потім шукають їх перетин.

Алгоритм Флері. **Програмна реалізація:**

```cpp
#include <iostream>

using namespace std;

bool step (int n, int **w)
{
    int k=0;
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            if (w[i][j]==1) k++;
        }
        if (k%2 != 0) return 0;
    }
    return 1;
}


int count_edge (int n, int **w)
{
    int k=0;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            if (w[i][j]==1)
            k++;
    return k;
}

bool bridge (int n, int **w, int j)
{
    int k=0;
    for (int i=0; i<n; i++)
        if (w[j][i]==1)
            k++;
    if (k>1) return 0;
    return 1;
}

int fleri (int n, int **w, int start)
{
    int i=0;
    int k=count_edge(n,w);
    for (i=0; i<n; i++)
    {
        if (w[start][i]==1)
        if (k<=1 || bridge(n,w,i)==0)
        {
            cout << start+1 << " - " << i+1 << ", ";
            w[i][start]=0;
            w[start][i]=0;
            k--;
            fleri(n,w,i);
        }
    }
    return i;
}

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

    int **w;
    int n, temp, start;

    cout << "Введіть кількість вершин" << endl;
    cin >> n;
    w = (int **) calloc(n, sizeof(int*));
    for (int i=0; i<n; i++)
        w[i] = (int *) calloc(n, sizeof(int));

    cout << "Введіть 1 (є ребро) або 0 (немає ребра)" << endl;
    cout << "Таблиця суміжності:" << endl;

```

```
75      for (int i=0; i<n; i++)
76      {
77          for (int j=0; j<n; j++)
78          {
79              cin >> temp;
80              if (temp == 1)
81                  w[i][j]=temp;
82              else w[i][j]=0;
83          }
84      }
85
86      if (step(n,w)==0) cout << "У графі немає ейлерового циклу" << endl;
87      else
88      {
89          cout << "Введіть початкову вершину" << endl;
90          cin >> start;
91
92          cout << "Послідовність видалення вершин за алгоритмом Флері" << endl;
93          cout << fleri(n,w,(start-1)) << " - " << start;
94      }
95
96      delete []w;
97      return 0;
98  }
99
```

Приклад виконання програми:

```
Введіть кількість вершин
11
Введіть 1 (є ребро) або 0 (немає ребра)
Таблиця суміжності:
0 1 1 0 0 0 0 1 1 0 0
1 0 1 0 0 0 0 1 1 0 0
1 1 0 1 1 0 0 1 1 1 1
0 0 1 0 1 0 0 0 0 1 1
0 0 1 1 0 1 0 0 0 0 1
0 0 0 0 1 0 1 0 0 1 1
0 0 0 0 0 1 0 1 1 0 1
1 1 1 0 0 0 1 0 1 1 0
1 1 1 0 0 0 1 1 0 1 0
0 0 1 1 0 1 0 1 1 0 1
0 0 1 1 1 1 1 0 0 1 0
Введіть початкову вершину
1
Послідовність видалення вершин за алгоритмом Флері
1 - 2, 2 - 3, 3 - 1, 1 - 8, 8 - 2, 2 - 9, 9 - 3, 3 - 4, 4 - 5, 5 - 3, 3 - 8, 8 - 7, 7 - 6, 6 - 5, 5 - 11, 11 - 3, 3 - 10,
10 - 4, 4 - 11, 11 - 6, 6 - 10, 10 - 8, 8 - 9, 9 - 7, 7 - 11, 11 - 10, 10 - 9, 11 - 1
```

Метод елементарних циклів. **Програмна реалізація:**

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

int n, v;
vector <vector <int> > M;
vector <int> visited;
vector <int> colours;
vector <int> cyclicnodes;

bool dfs (int v)
{
    colours[v] = 2;
    visited[v]++;
    for (int i = 0; i < M[v].size(); i++)
    {
        if (colours[M[v][i]] != 3 and visited[M[v][i]] != 2)
            {
                if (colours[M[v][i]] == 1)
                    if (dfs(M[v][i])) return true;
                else if (colours[M[v][i]] == 2) return true;
            }
        if (colours[v] == 3) return false;
        if (visited[v] == 2) return false;
    }
    colours[v] = 3;
    return false;
}

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

    cout << "Введіть кількість вершин та ребер" << endl;
    cin >> n >> v;
    M.resize(n);
    visited.resize(n);
    colours.resize(n);
    int v1, v2;
    colours.assign(n, 1);
    visited.assign(n, 0);
    cout << "Введіть вершини, між якими є ребро" << endl;
    for (int i = 0; i < v; i++)
    {
        cin >> v1 >> v2;
        M[v1 - 1].push_back(v2 - 1);
        i++;
        v++;
        M[v2 - 1].push_back(v1 - 1);
    }
    int t = 0;
    for (int i = 0; i < n; i++)
        if (dfs(i)==1) t = 1;
    if (t == 0) cout << "Немає циклів у графі";
    else cout << "У графі є елементарні цикли" << endl;
    return 0;
}
```

Приклад виконання програми:

```
Введіть кількість вершин та ребер
11 28
Введіть вершини, між якими є ребро
1 2
1 3
1 8
1 9
2 3
2 8
2 9
3 4
3 5
3 8
3 9
3 10
3 11
4 5
4 10
4 11
5 6
5 11
6 7
6 10
6 11
7 8
7 9
7 11
8 9
8 10
9 10
10 11
У графі є елементарні цикли
```

9. Спростити формули (привести їх до скороченої ДНФ).

$\overline{X\ X\ \overline{X} \rightarrow Y\ \overline{Y}} \rightarrow Z$

X	Y	Z	$\overline{X}$	$\overline{Y}$	$X\ \overline{X}$	$Y\ \overline{Y}$	$\overline{X\ \overline{X} \rightarrow Y\ \overline{Y}}$	$\overline{X\ X\ \overline{X} \rightarrow Y\ \overline{Y}}$	$\overline{X\ X\ \overline{X} \rightarrow Y\ \overline{Y}} \rightarrow Z$
0	0	0	1	1	0	0	0	0	1
0	0	1	1	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1
0	1	1	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0	0

$F = \overline{X}\ \overline{Y}\ \overline{Z}\ \vee\ \overline{X}\ Y\ \overline{Z}\ \vee\ X\ \overline{Y}\ \overline{Z}\ \vee\ X\ Y\ \overline{Z}$

$\overline{X}\ \overline{Y}\ \overline{Z}$ $\quad$ $\overline{X}\ \overline{Z}$

$\overline{\overline{X}\ Y\ \overline{Z}}$ $\quad$ $\overline{Y}\ \overline{Z}$

$X\ \overline{Y}\ \overline{Z}$ $\quad$ $Y\ \overline{Z}$

$\overline{X\ Y\ \overline{Z}}$ $\quad$ $X\ \overline{Z}$

$F = \overline{X}\ \overline{Z}\ \vee\ \overline{Y}\ \overline{Z}\ \vee\ Y\ \overline{Z}\ \vee\ X\ \overline{Z}$