### **Задачі**

[AФлойд - 1](https://www.e-olymp.com/uk/contests/16976/problems/175737)

[BФлойд](https://www.e-olymp.com/uk/contests/16976/problems/175738)

[CФлойд - існування](https://www.e-olymp.com/uk/contests/16976/problems/175739)

[DТранзитивное замыкание](https://www.e-olymp.com/uk/contests/16976/problems/175740)

[EТранзитивність графа](https://www.e-olymp.com/uk/contests/16976/problems/175741)

[FСоревнование коров](https://www.e-olymp.com/uk/contests/16976/problems/175742)

[GДіаметр графа](https://www.e-olymp.com/uk/contests/16976/problems/175743)

[HБікфордів шнур](https://www.e-olymp.com/uk/contests/16976/problems/175744)

[IНумерація шляхів](https://www.e-olymp.com/uk/contests/16976/problems/175745)

[JПідрахунок шляхів](https://www.e-olymp.com/uk/contests/16976/problems/175746)

[KБатарея Бони](https://www.e-olymp.com/uk/contests/16976/problems/175747)

# Алгоритм Флойда - Воршелла.

Назву алгоритм отримав на честь двох американських дослідників Роберта Флойда і Стівена Воршелла, які одночасно відкрили його у 1962 році. Інколи зустрічаються інші варіанти найменувань: алгоритм Рой - Воршелла або алгоритм Рой - Флойда. Рой - прізвище професора, який розробив аналогічний алгоритм на 3 роки раніше колег (в 1959 р), але це його відкриття залишилося невідомим.

Алгоритм Флойда - Воршелла - динамічний алгоритм обчислення значень найкоротших шляхів для кожної пари вершин графа. Метод працює на зважених графах, з позитивними і негативними вагами ребер, але без негативних циклів, будучи, таким чином, більш загальним в порівнянні з алгоритмом Дейкстри, в силу того, що останній не працює з негативною вагою ребер, і до того ж класична його реалізація має на меті визначення оптимальних відстаней від однієї вершини до всіх інших.

Для реалізації алгоритму Флойда - Воршелла сформуємо матрицю суміжності **D [ ] [ ]** графа **G = (V, E)** , в якому кожна вершина пронумерована від **1** до **| V |**. Ця матриця має розмір **| V | \* | V |**, І кожному її елементу **D [i] [j]** присвоєно вагу ребра, що йде з вершини **i** в вершину **j** . По ходу виконання алгоритму, дана матриця буде заповнюватись: в кожну з її комірок записуватиметься значення, що визначає оптимальну довжину шляху з вершини **i** до вершини **j** (відмова від виділення спеціального масиву для цієї мети збереже пам'ять і час).

Тепер, перед складанням основної частини алгоритму, необхідно розібратися з вмістом матриці найкоротших шляхів. Оскільки кожен її елемент **D [i] [j]** повинен містити найменший з наявних маршрутів, то відразу можна сказати, що для одиничної вершини він дорівнює нулю, навіть якщо вона має петлю (негативні цикли не розглядаються), отже, всі елементи головної діагоналі (**D [i] [i]**) потрібно обнулити.

А щоб нульові недіагональні елементи (матриця суміжності могла мати нулі в тих місцях, де немає безпосереднього ребра між вершинами **i** та **j** ) змінили по можливості своє значення, визначимо їх рівними нескінченності, яка в програмі може бути, наприклад, максимально можливої довгою шляху в графі, або просто - великим числом.

**Ключова частина алгоритму, складаючись з трьох циклів, вирази і умовного оператора, записується досить компактно:**

**Для k від 1 до | V | виконувати**  
 **Для i від 1 до | V | виконувати**  
 **Для j від 1 до | V | виконувати**  
 **Якщо D [i] [k] + D [k] [j] <D [i] [j] то D [i] [j] ← D [i] [k] + D [k] [j]**

Найкоротший шлях з вершини **i** в вершину **j** може проходити, як лише через них самих, так і через безліч інших вершин **k∈ (1, ..., | V |)** . Оптимальним з **i** в **j** буде шлях або що не проходить через **k** , або проходить. Зробити висновок про друге, означає встановити, що такий шлях йде з **i** до **k**, а потім з **k** до **j**, тому замінюємо, значення найкоротшого шляху **D [i] [j]** сумою **D [i] [k] + D [ k] [j]** .

Розглянемо повний код алгоритму Флойда - Воршелла на C ++, Паскалі та Python, а потім детально розберемо послідовність виконуваних ним дій.

### Код програми на C ++:

#include <fstream>

#include <iostream>

const int16\_t MAXN = 100;

int64\_t D[MAXN][MAXN];

int64\_t V;

//алгоритм Флойда-Воршелла

void FU()

{

    for (int64\_t i=0; i<V; i++) D[i][i]=0;

    for (int64\_t k=0; k<V; k++)

        for (int64\_t i=0; i<V; i++)

            for (int64\_t j=0; j<V; j++)

                if (D[i][k] && D[k][j] && i!=j)

                    if (D[i][k]+D[k][j]<D[i][j] || D[i][j]==0)

                        D[i][j]=D[i][k]+D[k][j];

}

void ReadData(){

    using namespace std;

    int64\_t m, v1, v2, v3;

    ifstream fin("input.txt");

    fin >> V;

    for (int64\_t i=1; i<=V; i++){

        fin >> v1 >> v2 >> v3;

        D[i-1][0] = v1;

        D[i-1][1] = v2;

        D[i-1][2] = v3;

    }

    fin.close();

}

void WriteData(){

    std::ofstream fout("output.txt");

    for (int64\_t i=0; i<V; i++){

        for (int64\_t j=0; j<V; j++){

            fout<<D[i][j]<<" ";

        }

        fout<<"\n";

    }

    fout.close();

}

//Головна функція

int main(){

    ReadData();

    FU();

    WriteData();

    return 0;

}

### Код програми на Pascal:

program floyd;

const maxV=1000;

var D:array[1..maxV, 1..maxV] of longint;

    V: longint;

{алгоритм Флойда-Воршелла}

Procedure FU;

var i, j, k, inf: longint;

begin

    inf:=1000000;

    for i:=1 to V do D[i, i]:=0;

    for k:=1 to V do

        for i:=1 to V do

            for j:=1 to V do

                if (D[i, k]<>0) and (D[k, j]<>0) and (i<>j) then

                    if (D[i, k]+D[k, j]<D[i, j]) or (D[i, j]=0) then

                        D[i, j]:=D[i, k]+D[k, j];

    for i:=1 to V do

    begin

        for j:=1 to V do

            write(D[i, j]:4);

        writeln;

    end;

end;

procedure ReadData;

var

    i: longint;

    v1,v2,v3: longint;

    f\_in: text;

begin

     Assign(f\_in,'input.txt');

     reset(f\_in);

     readln(f\_in, V);

     for i:=1 to V do begin

         readln(f\_in, v1, v2, v3);

         D[i,1]:=v1;

         D[i,2]:=v2;

         D[i,3]:=v3;

     end;

     close(f\_in);

end;

procedure WriteData;

var

    i, j: longint;

    f\_out: text;

begin

     Assign(f\_out,'output.txt');

     rewrite(f\_out);

     for i:=1 to V do begin

         for j:=1 to V do

           write(f\_out, D[i,j], ' ');

           writeln(f\_out);

     end;

     close(f\_out);

end;

{main block}

begin

     ReadData;

     FU;

     WriteData;

end.

Код програми на Python

def Floyd(v, D):

    for k in range(0, v):

        for i in range(0, v):

            for j in range(0, v):

                if (D[i][k] != 0) and (D[k][j] != 0) and (i != j):

                    if (D[i][k]+D[k][j] < D[i][j]) or (D[i][j] == 0):

                        D[i][j] = D[i][k] + D[k][j]

    return v, D

def ReadData():

    f = open('input.txt', 'r')

    l = [line.strip() for line in f]

    v = int(l[0])

    D=[]

    for i in range(1, v+1):

        print (l[i])

        sD=[]

        v1, v2, v3 = map(int, l[i].split())

        sD.append(v1)

        sD.append(v2)

        sD.append(v3)

        D.append(sD)

    f.close

    return v,  D

def WriteData(v, D):

    f = open('output.txt', 'w')

    for i in range(0, v):

        for j in range(0, v):

            f.write(str(D[i][j])+" ")

        f.write('\n')

    f.close

# main block

v, D = ReadData()

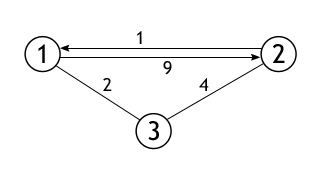
v, D = Floyd(v, D)

WriteData(v, D)

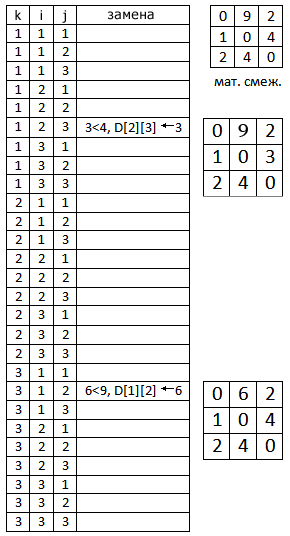
Нехай у якості матриці суміжності, кожен елемент якої зберігає вагу деякого ребра, була задана наступна матриця:

|  |  |  |
| --- | --- | --- |
| 0 | 9 | 2 |
| 1 | 0 | 4 |
| 2 | 4 | 0 |

Кількість вершин в графі, поданням якого є дана матриця, дорівнює 3, і, причому між кожними двома вершинами існує ребро. Ось власне сам цей граф:



Завдання алгоритму: перезаписати дану матрицю так, щоб кожна клітинка замість ваги ребра з i в j, містила найкоротший шлях з i в j. Для прикладу взято зовсім маленький граф, і тому можливий варіант, що матриця збереже свій початковий стан. Але результат тестування програми показує заміну двох значень в ній. Наступна схема допоможе з аналізом цього конкретного прикладу.



В даній таблиці показані 27 кроків виконання основної частини алгоритму. Їх стільки з тієї причини, що час виконання методу дорівнює O (|V|3). Наш граф має 3 вершини, а 33= 27. Перша заміна відбувається на ітерації, при якій k = 1, i = 2, а j = 3. В той момент D [2] [1] = 1, D [1] [3] = 2, D [2] [3] = 4. Умова істинна, тобто D [1] [3] + D [3] [2] = 3, а 3 <4, отже, елемент матриці D [2] [3] отримує нове значення. Наступний крок, коли умова також істинна вносить зміни в елемент, розташований на перетині другого рядка і третього стовпця.

Джерело: <https://kvodo.ru/algoritm-floyda-uorshella.html>