Алгоритм Дейкстри.

Алгоритм голландського вченого Едсгера Дейкстри знаходить всі найкоротші шляхи з однієї наперед заданої вершини графа до всіх інших. З його допомогою, при наявності всієї необхідної інформації, можна, наприклад, дізнатися яку послідовність доріг краще використовувати, щоб дістатися з одного міста до іншого.

Мінусом даного методу є неможливість обробки графів, в яких є ребра з від'ємною вагою.

Для програмної реалізації алгоритму знадобитися два масиви: логічний **visited** - для зберігання інформації про відвідані вершинах і чисельний **distance** , в який будуть заноситися знайдені найкоротші шляхи. Отже, є граф **G = (V, E)** . Кожна з вершин входять в множину **V** , спочатку відзначена як не відвідана, тобто елементам масиву **visited** присвоєно значення **false** .

Оскільки найвигідніші шляхи тільки належить знайти, в кожен елемент вектора **distance** записується таке число, яке свідомо більше будь-якого потенційного шляху (зазвичай це число називають нескінченністю, але в програмі використовують, наприклад максимальне значення конкретного типу даних).

В якості вихідного пункту вибирається вершина **s** і йому приписується нульовий шлях: **distance [s] = 0** , т. К. Немає ребра з **s** в **s** (метод не передбачає петель).

Далі знаходяться всі сусідні вершини (в які є ребро з **s**) [нехай такими будуть **t** і **u**] і по черзі досліджуються, а саме обчислюється вага маршруту з **s** по черзі в кожну з них:

- **distance [t] = distance [s] + вага інцідентного(спільного) s і t ребра;**
- **distance [u] = distance [s] + вага інцідентного(спільного)s і u ребра.**

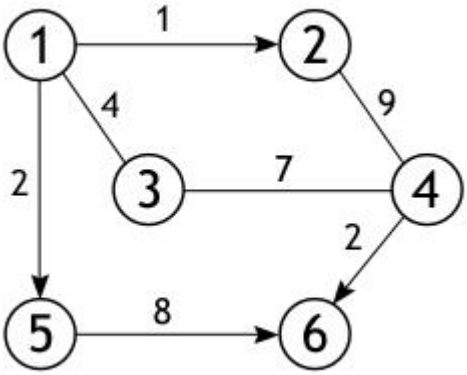
Але цілком ймовірно, що в ту чи іншу вершину з **s** існує кілька шляхів, тому ціну шляху в таку вершину в масиві **distance** доведеться переглядати, тоді найбільше (неоптимальний) значення ігнорується, а найменше ставитися у відповідність вершині.

Після обробки суміжних з **s** вершин вона позначається як відвідана: **isited [s] = true** , і активною стає та вершина, шлях з **s** в яку мінімальний.

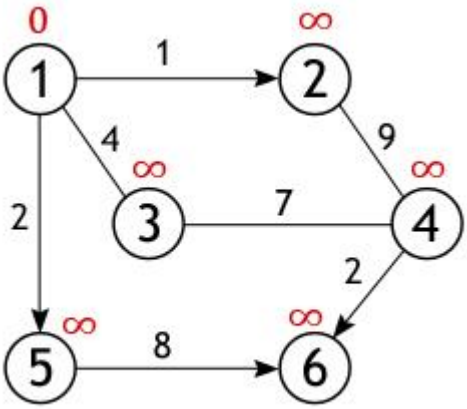
Припустимо, шлях з **s** в **u** коротше, ніж з **s** в **t** , отже, вершина **u** стає активною і описаним чином досліджуються її сусіди, за винятком вершини **s** .

Далі, **u** позначається як пройдена: **visited [u] = true** , активної стає вершина **t** , і вся процедура повторюється для неї. Алгоритм Дейкстри триває до тих пір, поки всі доступні з **s** вершини не будуть досліджені.

Тепер на конкретному графі простежимо роботу алгоритму, знайдемо всі найкоротші шляхи між початковою і всіма іншими вершинами. Розмір (кількість ребер) зображеного нижче графа дорівнює **7 (| E | = 7)** , а порядок (кількість вершин) - **6 (| V | = 6)** . Це зважений граф, кожному з його ребер поставлено у відповідність деяке числове значення, тому вага маршруту необов'язково визначається числом ребер, що лежать між парою вершин.



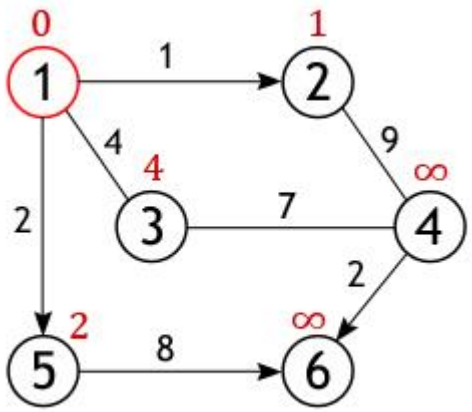
З усіх вершин входять в множину **V** виберемо одну, ту, від якої необхідно знайти найкоротші шляхи до інших доступних вершин. Нехай такий буде вершина **1** . Довжина шляху до всіх вершин, крім першої, спочатку дорівнює нескінченності, а до неї - **0** , т. К. Граф не має петель.



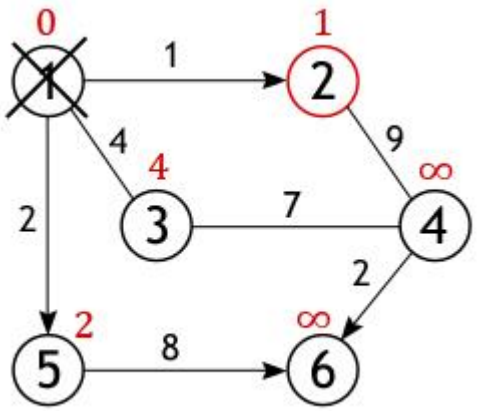
У вершини 1 рівно 3 сусіда (вершини 2, 3, 5), і щоб обчислити довжину шляху до них потрібно скласти вагу дуг, що лежать між вершинами: 1 і 2, 1 і 3, 1 і 5 зі значенням першої вершини (з нулем) :

2 ← 1 + 0
3 ← 4 + 0
5 ← 2 + 0

Як уже зазначалося, отримані значення присвоюються вершинам лише в тому випадку якщо вони «краще» (менше) тих які є на на даний момент. А так як кожне з трьох чисел менше нескінченності, вони стають новими величинами, визначальними довжину шляху з вершини 1 до вершин 2, 3 і 5.



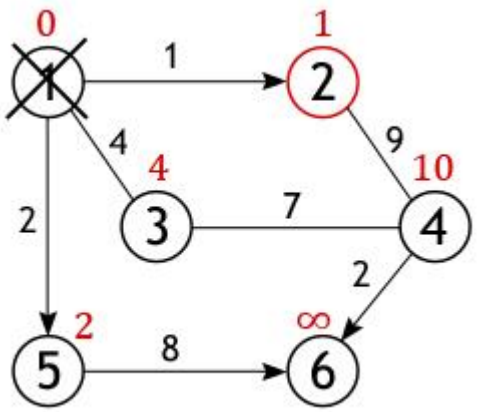
Далі, активна вершина позначається як відвідана, статус «активної» (червоне коло) переходить до однієї з її сусідок, а саме до вершини 2, оскільки вона найближча до раніше активної вершині.



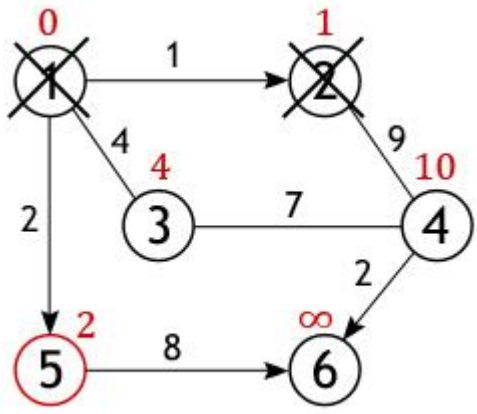
У вершини 2 всього один не розглянутий сусід (вершина 1 позначена як відвідана), відстань до якого з неї дорівнює 9, але нам необхідно обчислити довжину шляху з стартової вершини, для чого потрібно скласти величину прописану до вершині 2 з вагою дуги з неї в вершину 4

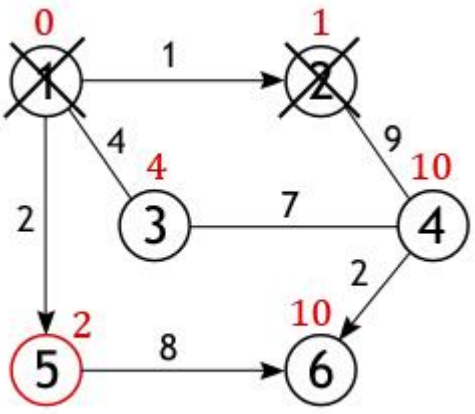
4 ← 1 + 9

Умова «мінімальності» (10 <∞) виконується, отже, вершина 4 отримує нове значення довжини шляху.



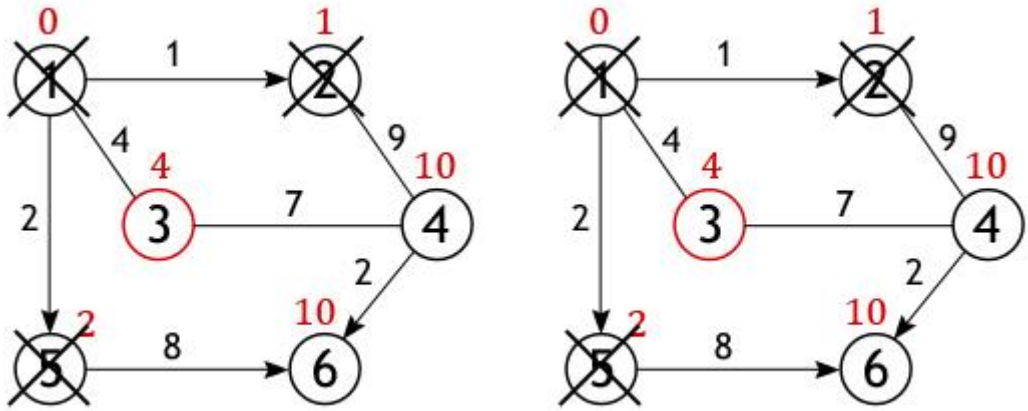
Вершина 2 перестає бути активною, також як і вершина 1 видаляється зі списку **НЕ відвіданих**. Тепер тим же способом досліджуються сусіди вершини 5, і обчислюється відстань до них.



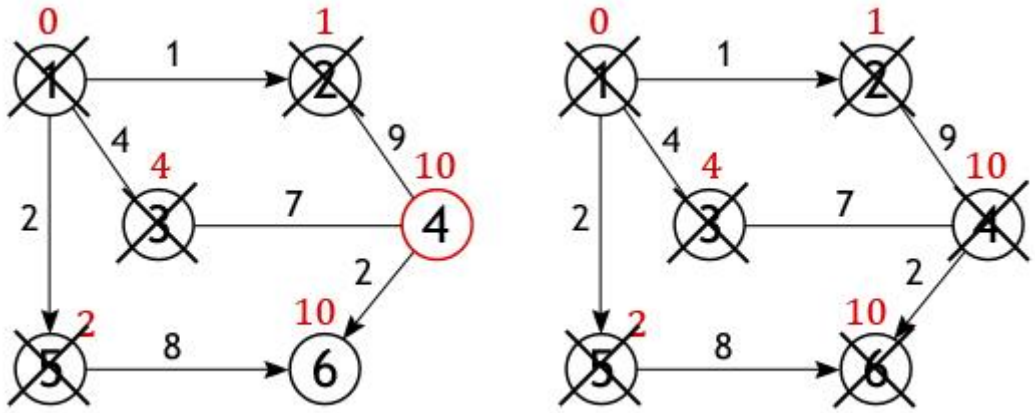


Коли справа доходить до огляду сусідів вершини 3, то тут важливо не помилитися, так як вершина 4 вже була досліджена і відстань одного з можливих шляхів з стартової до неї обчислено. Якщо рухатися в неї через

вершину 3, то шлях складе 4 + 7 = 11, а 11> 10, тому нове значення ігнорується, старе залишається.



Аналогічна ситуація з вершиною 6. Значення найближчого шляху до неї з вершини 1 дорівнює 10, а воно виходить тільки в тому випадку, якщо йти через вершину 5.



Коли всі вершини графа, або все ті, що доступні з стартової, будуть позначені як відвідані, тоді робота алгоритму Дейкстри завершиться, і всі знайдені шляхи будуть найкоротшими. Так, наприклад, буде виглядати список найоптимальніших відстаней лежать між вершиною 1 і всіма іншими вершинами, розглянутого графа:

1 → 1 = 0
1 → 2 = 1
1 → 3 = 4
1 → 4 = 10

1 → 5 = 2
1 → 6 = 10

У програмі, що знаходить найближчі шляху між вершинами за допомогою методу Дейкстри, граф буде представлений в вигляді не бінарної матриці суміжності. Замість одиниць в ній будуть виставлена вага ребер, функція нулів залишиться колишньою: показувати, між якими вершинами немає ребер.

Можливий варіант коду на C++. Тут матриця суміжності задана масивом констант, вам варто передбачити читання з файлу. Дана реалізація представлена як зразок. Вам варто самостійно подбати про більш вдалу структуру даних

```
#include "stdafx.h"
#include <iostream>
using namespace std;
const int V=6;
//алгоритм Дейкстры
void Dijkstra(int GR[V][V], int st)
{
int distance[V], count, index, i, u, m=st+1;
bool visited[V];
for (i=0; i<V; i++)
{
distance[i]=INT_MAX; visited[i]=false;
}
distance[st]=0;
for (count=0; count<V-1; count++)
{
int min=INT_MAX;
for (i=0; i<V; i++)
if (!visited[i] && distance[i]<=min)
{
min=distance[i]; index=i;
}
u=index;
visited[u]=true;
for (i=0; i<V; i++)
if (!visited[i] && GR[u][i] && distance[u]!=INT_MAX &&
distance[u]+GR[u][i]<distance[i])
distance[i]=distance[u]+GR[u][i];
```

```
29 }
30 cout<<"Стоимость пути из начальной вершины до остальных:\t\n";
31 for (i=0; i<V; i++) if (distance[i]!=INT_MAX)
32 cout<<m<<" > "<<i+1<<" = "<<distance[i]<<endl;
33 else cout<<m<<" > "<<i+1<<" = "<<"маршрут недоступен"<<endl;
34 }
35 //главная функция
36 void main()
37 {
38 setlocale(LC_ALL, "Rus");
39 int start, GR[V][V]={
40 {0, 1, 4, 0, 2, 0},
41 {0, 0, 0, 9, 0, 0},
42 {4, 0, 0, 7, 0, 0},
43 {0, 9, 7, 0, 0, 2},
44 {0, 0, 0, 0, 0, 8},
45 {0, 0, 0, 0, 0, 0}};
46 cout<<"Начальная вершина >> "; cin>>start;
47 Dijkstra(GR, start-1);
48 system("pause>>void");
49 }
```

Можливий варіант коду на Pascal. Тут матриця суміжності задана масивом констант, вам варто передбачити читання з файлу. Дана реалізація представлена як зразок. Вам варто самостійно подбати про більш вдалу структуру даних

```
 1 program DijkstraAlgorithm;
 2 const V=6; inf=100000;
 3 type vektor=array[1..V] of integer;
 4 var start: integer;
 5 const GR: array[1..V, 1..V] of integer=(
 6 (0, 1, 4, 0, 2, 0),
 7 (0, 0, 0, 9, 0, 0),
 8 (4, 0, 0, 7, 0, 0),
 9 (0, 9, 7, 0, 0, 2),
10 (0, 0, 0, 0, 0, 8),
11 (0, 0, 0, 0, 0, 0));
12 {алгоритм Дейкстры}
13 procedure Dijkstra(GR: array[1..V, 1..V] of integer; st: integer);
14 var count, index, i, u, m, min: integer;
15 distance: vektor;
```

```
16 visited: array[1..V] of boolean;
17 begin
18 m:=st;
19 for i:=1 to V do
20 begin
21 distance[i]:=inf; visited[i]:=false;
22 end;
23 distance[st]:=0;
24 for count:=1 to V-1 do
25 begin
26 min:=inf;
27 for i:=1 to V do
28 if (not visited[i]) and (distance[i]<=min) then
29 begin
30 min:=distance[i]; index:=i;
31 end;
32 u:=index;
33 visited[u]:=true;
34 for i:=1 to V do
35 if (not visited[i]) and (GR[u, i]<>0) and (distance[u]<>inf) and
36 (distance[u]+GR[u, i]<distance[i]) then
37 distance[i]:=distance[u]+GR[u, i];
38 end;
39 write('Стоимость пути из начальной вершины до остальных:'); writeln;
40 for i:=1 to V do
41 if distance[i]<>inf then
42 writeln(m,' > ', i,' = ', distance[i])
43 else writeln(m,' > ', i,' = ', 'маршрут недоступен');
44 end;
45 {основной блок программы}
46 begin
47 clrscr;
48 write('Начальная вершина >> '); read(start);
49 Dijkstra(GR, start);
50 end.
51
```