**Алгоритм Дейкстри**

**Алгоритм Дейкстри** — [алгоритм](http://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) на [графах](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), відкритий [Дейкстрою](http://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80). Знаходить [найкоротший шлях](http://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B9%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D1%88%D0%B8%D0%B9_%D1%88%D0%BB%D1%8F%D1%85) від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

Зберігатимемо поточну мінімальну відстань до всіх вершин **V** (від даної вершини **a**) і на кожному кроці алгоритму намагатимемося зменшити цю відстань. Спочатку встановимо відстані до всіх вершин рівними нескінченості, а до вершини **а** — нулю. Розглянемо виконання алгоритму на прикладі. Хай потрібно знайти відстані від 1-ої вершини до всіх інших. Кружечками позначені вершини, лініями — шляхи між ними («дуги»). Над дугами позначена їх «ціна» — довжина шляху. Надписом над кружечком позначена поточна найкоротша відстань до вершини.

**Крок 1**

Ініціалізація. Відстань до всіх вершин [графа](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) **V** = . Відстань до **а** = 0. Жодна вершина графа ще не опрацьована.


**Крок 2**

Знаходимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.


**Крок 3**

Перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1-у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.


**Кроки 4, 5**

Аналогічну операцію проробляєм з двома іншими сусідами 1-ї вершини — 3-ю та 6-ю.


**Крок 6**

Всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів [Дейкстра](http://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80)). Тому викреслимо її з [графа](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), щоб відмітити цей факт. 

**Крок 7**

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7. І знову намагаємося зменшити відстань до всіх сусідів 2-ої вершини, намагаючись пройти в них через 2-у. Сусідами 2-ої вершини є 1, 3, 4.

**Крок 8**

Перший (по порядку) сусід вершини № 2 — 1-а вершина. Але вона вже оброблена (або викреслена — див. крок 6). Тому з 1-ою вершиною нічого не робимо.

**Крок 8 (з іншими вхідними данними)**

Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ої + відстань між 2-ою і 4-ою вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22. 

**Крок 9**

Ще один сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ої вершини не міняємо. 

**Крок 10**

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з [графа](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29). 

**Кроки 11 — 15**

По вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» виявляється вершина № 3. Після її «обробки» отримаємо такі результати:


**Наступні кроки**

Проробляємо те саме з вершинами, що залишилися (№ по порядку: 6, 4 і 5).


Нехай u — вершина, від якої шукаються відстані, V — множина вершин [графа](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), di — відстань від вершини u до вершини i, , w(i, j) — вага «ребра» (i, j).

Алгоритм:

1. Множина вершин U, до яких відстань відома, встановлюється рівною {u}.

2. Якщо U=V, алгоритм завершено.

3. *Потенційні відстані* Di до вершин з V\U встановлюються нескінченними.

4. Для всіх ребер (i, j), де i∈U та j∈V\U, якщо Dj>di+w(i, j), то Dj присвоюється di+w(i, j).

5. Шукається i∈V\U, при якому Di мінімальне.

6. Якщо Di дорівнює нескінченності, алгоритм завершено. В іншому випадку di присвоюється значення Di, U присвоюється U∪{i} і виконується перехід до кроку 2.

|  |  |
| --- | --- |
| 23456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869 | #include <iostream>#include <set>#include <vector>using namespace std; typedef vector<int> vi;typedef pair<int,int> ii;typedef vector<ii> vii;typedef vector<vii> vvii; const int MAX = 1001;const int MAXINT = 1000000000; int n;vvii G(MAX);vi D(MAX, MAXINT); void Dijkstra(int s){    set<ii> Q;    D[s] = 0;    Q.insert(ii(0,s));     while(!Q.empty())    {        ii top = \*Q.begin();        Q.erase(Q.begin());        int v = top.second;        int d = top.first;         for (vii::const\_iterator it = G[v].begin(); it != G[v].end(); it++)        {            int v2 = it->first;            int cost = it->second;            if (D[v2] > D[v] + cost)            {                if (D[v2] != 1000000000)                {                    Q.erase(Q.find(ii(D[v2], v2)));                }                D[v2] = D[v] + cost;                Q.insert(ii(D[v2], v2));            }        }    }} int main(){    //freopen("input.txt", "r", stdin);    //freopen("output.txt", "w", stdout);     int m, s, t = 0;    scanf("%d %d %d %d", &n, &m, &s, &t);     for (int i = 0; i < m; i++)    {        int a, b, w = 0;        scanf("%d %d %d", &a, &b, &w);        G[a - 1].push\_back(ii(b - 1, w));        G[b - 1].push\_back(ii(a - 1, w));    }     Dijkstra(s - 1);     printf("%d\n", D[t - 1]);     return 0;} |