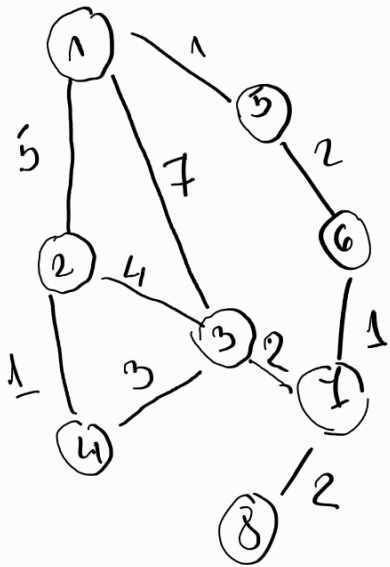


Графи - Дийкстра

задача Даден ни е граф с n върха и m претеглени ребра (ще разгл. неориентиран, но алгоритъма работи и за ориентиран). Теглата на ребрата са строго положителни. Определяме цена на път като сумата от теглата по него (① $\xrightarrow{4}$ ② $\xrightarrow{3}$ ③ цената на пътя от ① до ③ е 7). Да се намери цената на най-краткия път от ① до всеки друг връх.

Пример:



Отговор:

1 - 1 : 0
1 - 2 : 5
1 - 3 : 6
1 - 4 : 6
1 - 7 : 4
1 - 8 : 6

Идея на алгоритъма:

Нека имаме масив $d[]$, където $d[v]$ ще записваме цената на най-краткия път до v , които сме открили до момента. Първоначално $d[v] = \infty$ за всяко v , а $d[1] = 0$, защото цената на пътя от 1 до 1 е 0. Нека имаме и масив $solved[]$, където $solved[v] = 1$, ако вече сме намерили наистина най-малкия път до v . Първоначално $solved[v] = 0$ за всяко v .

Повтаряме следния алгоритъм докато има поне едно i , за които $solved[i] = 0$:


Нека V = върха, за който $solved[V] = 1$ и който е свързан с всички върхове (с $solved[V] = 0$).

За него имаме два варианта:

- или наистина $d[V]$ е най-краткият път от S до V
- или има някакво U и пътят от S до U + пътят от U до V е по-малък. Обаче, тъй като $d[V]$ е минимално, значи пътят, който сме открили от S до $U > d[V] \Rightarrow$ този вариант е невъзможен.

$\Rightarrow d[V]$ наистина е най-краткият път от S до V .

маркираме $solved[V] = 1$. Сега можем използвайки пътя от S до V да стигнем до някой нов съсед по по-кратък път.

( $\Rightarrow d[U] > d[V] + w$)

Това, което правим е да обиколим всички съседи на V и за тези, които $solved$ е 0 да проверим дали $d[V] + \text{тежест на реброто между тях} < d[\text{съседа на } V]$ ако е така, обновяваме d -то на съседа.

Идеята за реализация на алгоритъма е да използваме приоритетна очередь, която да ни дава най-квсия път до кои да е от неизчислените върхове.

```
int n,m;
vector<pair<int,int>> g[105]; // имаме ребро (v, g[v].first) с тежест g[v].second;
int d[105];
bool solved[105];
void dijkstra()
{
    priority_queue<pair<int,int>> pq; // .first ни е цената на пътя (с отрицателен знак, защото искаме да ни дава минималното),
    // .second е върха, до който е този път
    pq.push({0,1}); // първоначално вкарваме само пътя до 1

    while( !pq.empty() )
    {
        auto [dist, v] = pq.top(); // взимаме това v, с най-малка цена на пътя (dist) и опашката
        pq.pop();
        if( solved[v] ) continue; // първото условие беше solved[v] = 0

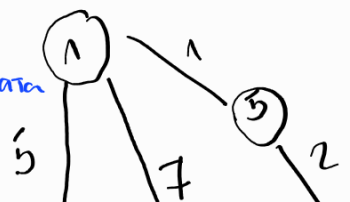
        solved[v] = 1;

        for( auto [u,w] : g[v] )
        {
            if( solved[u] ) continue; // ако вече е изчислен този връх не се занимаваме с него
            if( d[u] > d[v] + w )
            {
                d[u] = d[v] + w;
                pq.push({-d[u],u}); // с обратен знак, защото искаме минимално да ни дава
            }
        }
    }
}
```

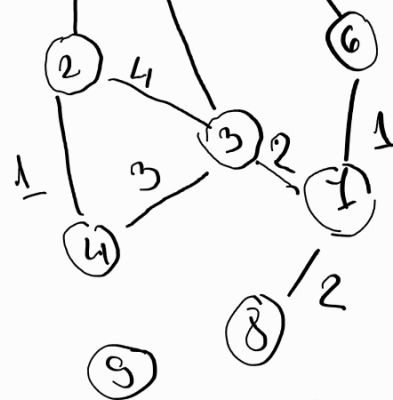
Целият код е качен на [bosna.tk/Graphs](https://www.bosna.tk/Graphs)

Задача:

Проследете на този пример как ще се изпълни програмата



Зад. Забележете, че програмата работи докато има елементи в опашката, а ако някой не е достижим от 5 (напр 9), той никога няма да влезе в опашката и съответно накрая `visited` ще му бъде 0.



Задача:

Напишете `codeforces-20C` и `tourguide` си `resho.org` → Графи-пробор.
 Помислете какво ще стане ако тежестта на някое ребро е отрицателно.