Bellman-Ford算法

- 执行V-1次“对每条边做松弛操作”
- 时间复杂度O(VE)，空间复杂度O(V+E)

适用情况：

- 求解单源最短路径问题
- 图中可以有负权边

特点：可以判断出图中有没有负权环。

带负权的单源最短路

描述

输入一个有向图，边的权值可正可负，求顶点到其他各点的最短路。

输入

第一行输入n，m。表示n个结点（默认顶点为1号），m条边。（n，m≤100）

接下来m行，每行三个整数，空格分隔，表示起点、终点、边权。（边权绝对值≤10000）

输出

输出一行，如果有负权回路输出“not possible”，否则输出顶点1到除自己外其他点的最短路，输出答案之间仅有一个空格。

输入样例 1

```
5 5
2 3 2
1 2 -3
1 5 5
4 5 2
3 4 3
```

输出样例 1

```
-3 -1 2 4
```

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

const int N = 110;

int n, m;
int u[N], v[N], w[N], dis[N];

int main() {

    cin >> n >> m;
    for(int i = 1;  i <= m; i++) cin >> u[i] >> v[i] >> w[i];

    memset(dis, 0x3f, sizeof dis);

    dis[1] = 0;

    for(int k = 1; k < n; k++)
        for(int i = 1; i <= m; i++)
            if(dis[v[i]] > dis[u[i]] + w[i])
```

```
                dis[v[i]] = dis[u[i]] + w[i];

    bool has_minus_circle = false;

    for(int i = 1; i <= m; i++)
        if(dis[v[i]] > dis[u[i]] + w[i]){
            has_minus_circle = true;
            break;
        }

    if(has_minus_circle) cout << "not possible" << endl;
    else{
        for(int i = 2; i <= n; i++) cout << dis[i] << " ";
    }

    return 0;
}
```

优化:

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

const int N = 110;

int n, m;
int u[N], v[N], w[N], dis[N], bak[N];

int main() {

    cin >> n >> m;
    for(int i = 1; i <= m; i++) cin >> u[i] >> v[i] >> w[i];

    memset(dis, 0x3f, sizeof dis);

    dis[1] = 0;

    for(int k = 1; k < n; k++){
        memcpy(bak, dis, sizeof dis);

        for(int i = 1; i <= m; i++)
            if(dis[v[i]] > dis[u[i]] + w[i])
                dis[v[i]] = dis[u[i]] + w[i];

        bool is_ok = true;
        for(int i = 1; i <= n; i++)
            if(bak[i] != dis[i]){
                is_ok = false;
                break;
            }

        if(is_ok) break;
    }

    bool has_minus_circle = false;
```

```
38
39      for(int i = 1; i <= m; i++){
40          if(dis[v[i]] > dis[u[i]] + w[i]){
41              has_minus_circle = true;
42              break;
43          }
44      }
45
46      if(has_minus_circle) cout << "not possible" << endl;
47      else{
48          for(int i = 2; i <= n; i++) cout << dis[i] << " ";
49      }
50
51      return 0;
52  }
```

SPFA算法

SPFA（Shortest Path Faster Algorithm）算法，即为Bellman-Ford算法的队列优化算法。

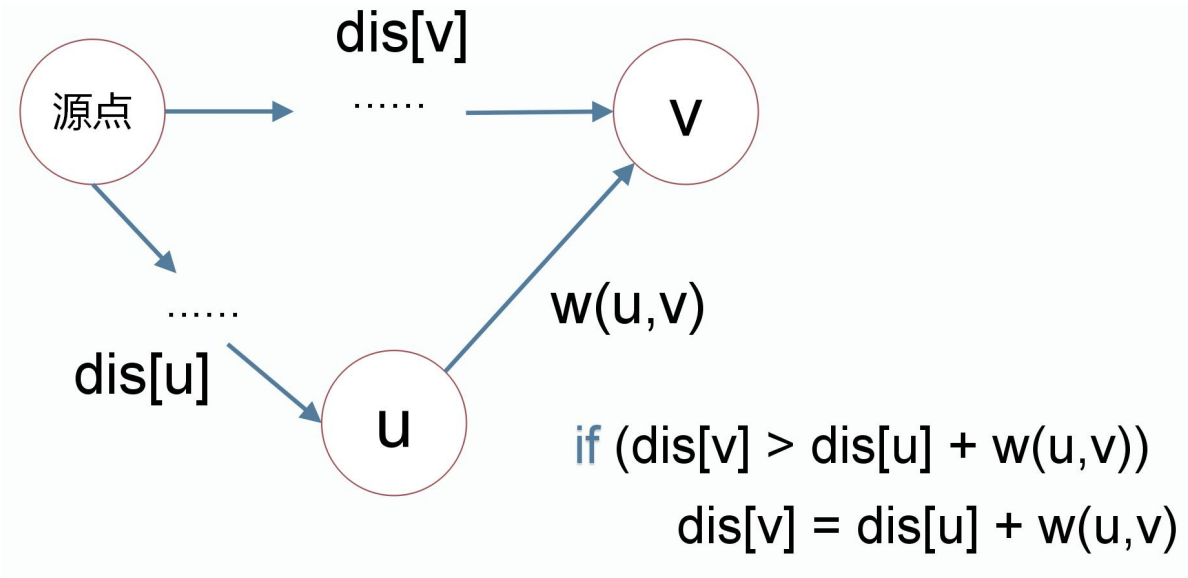
- 设dis数组，dis[i]表示已知的顶点i到源点的最短距离
- 设队列，保存距离更新后的顶点

1.将源点v0到自己的距离设为0(dis[v0] = 0)，将v0入队。

2.出队一个顶点u，对u的邻接点v做松弛操作，如果成功进行松弛操作，且v不在队列中，那么将顶点v入队。

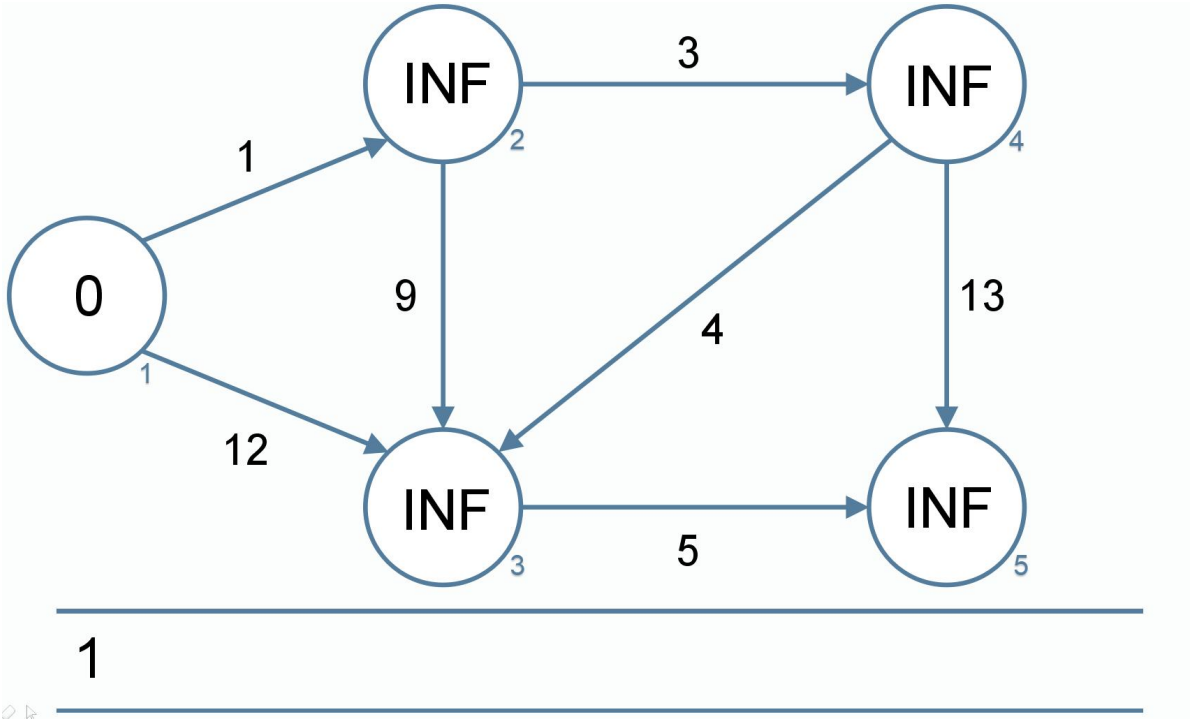
3.重复执行步骤2直到队列为空。此时dis数组中保存的就是以v0为源点的单源最短路径。

松弛操作：

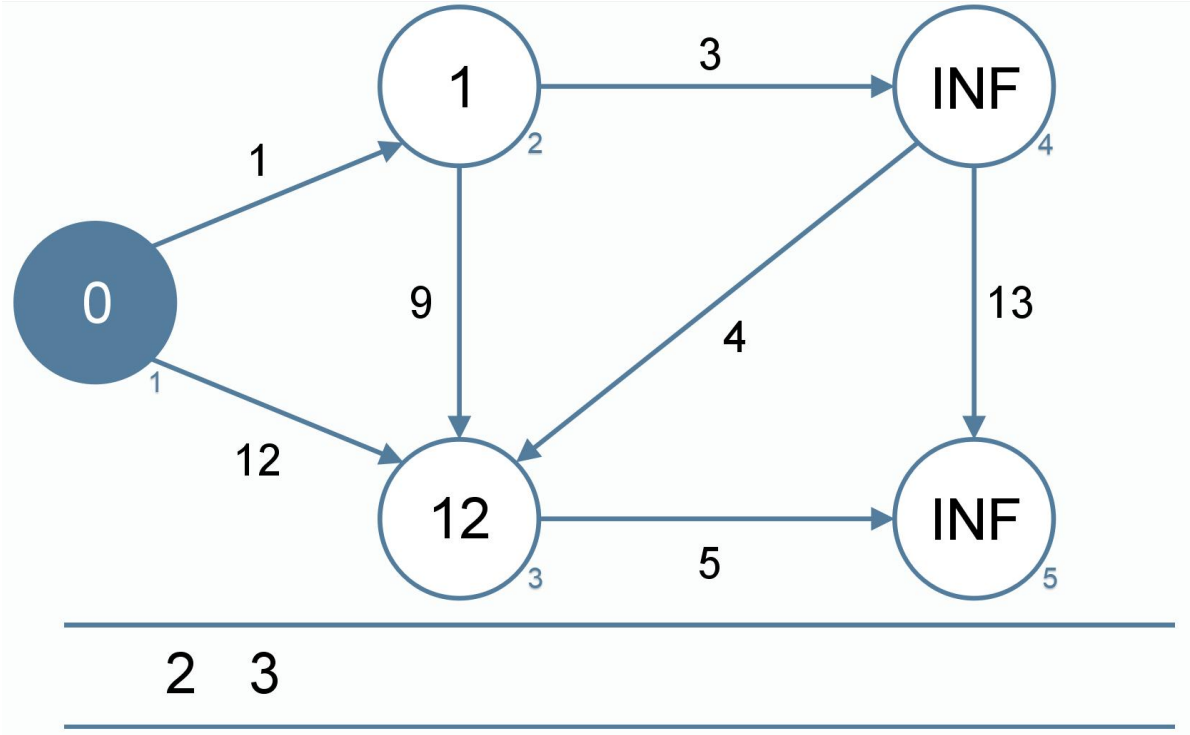


算法图解

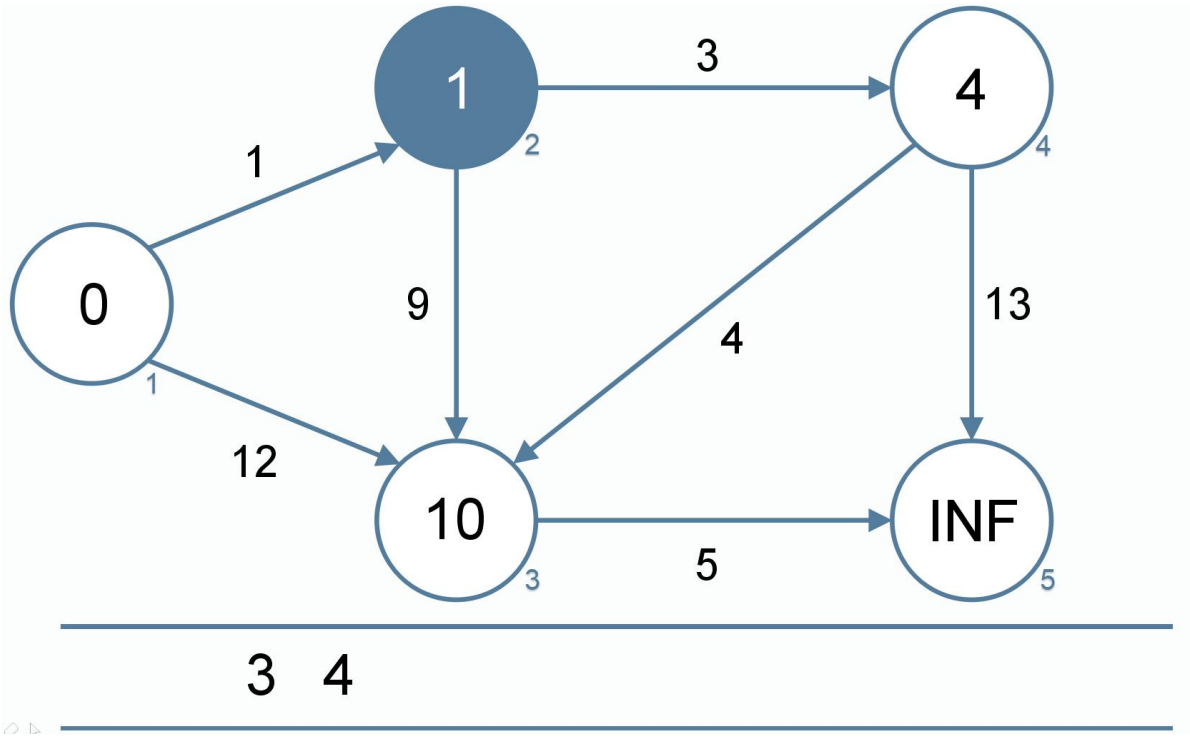
顶点1入队：

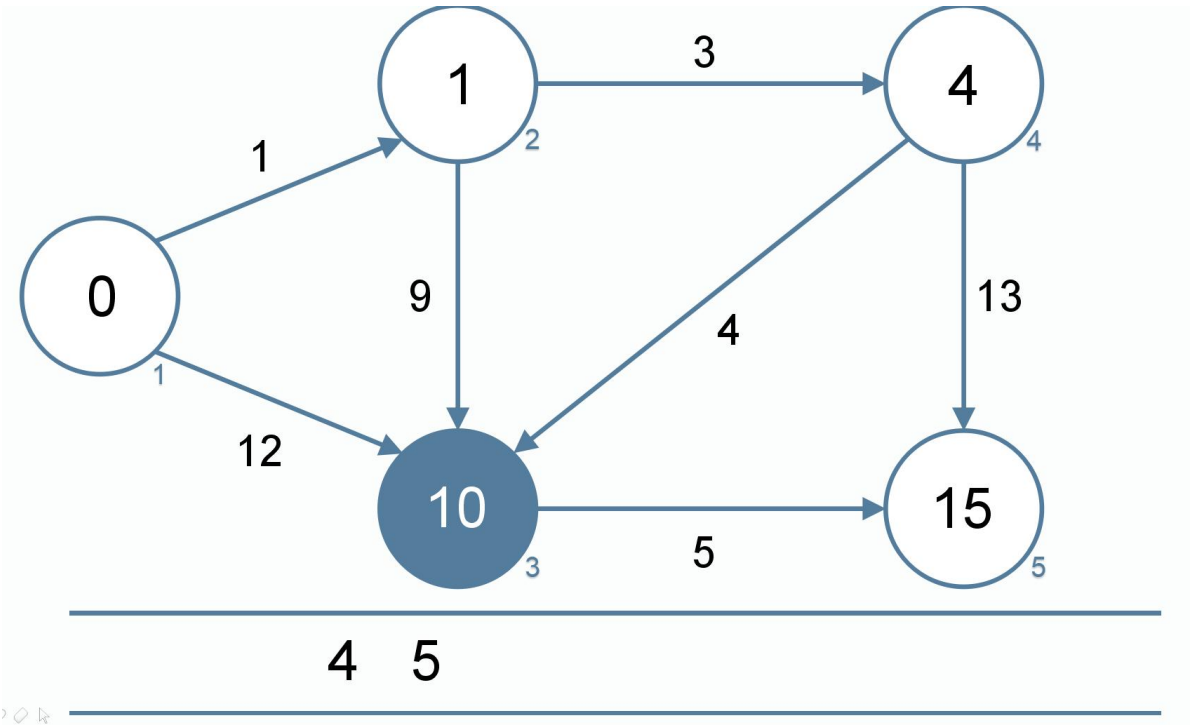


顶点1出队，顶点2、3松弛操作并入队：

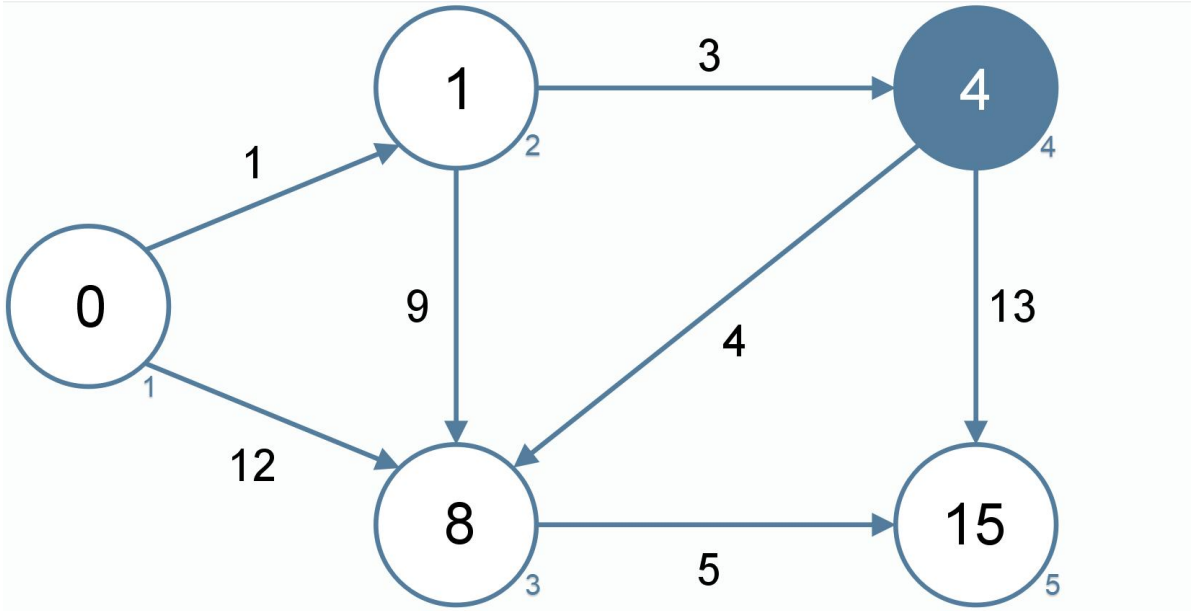


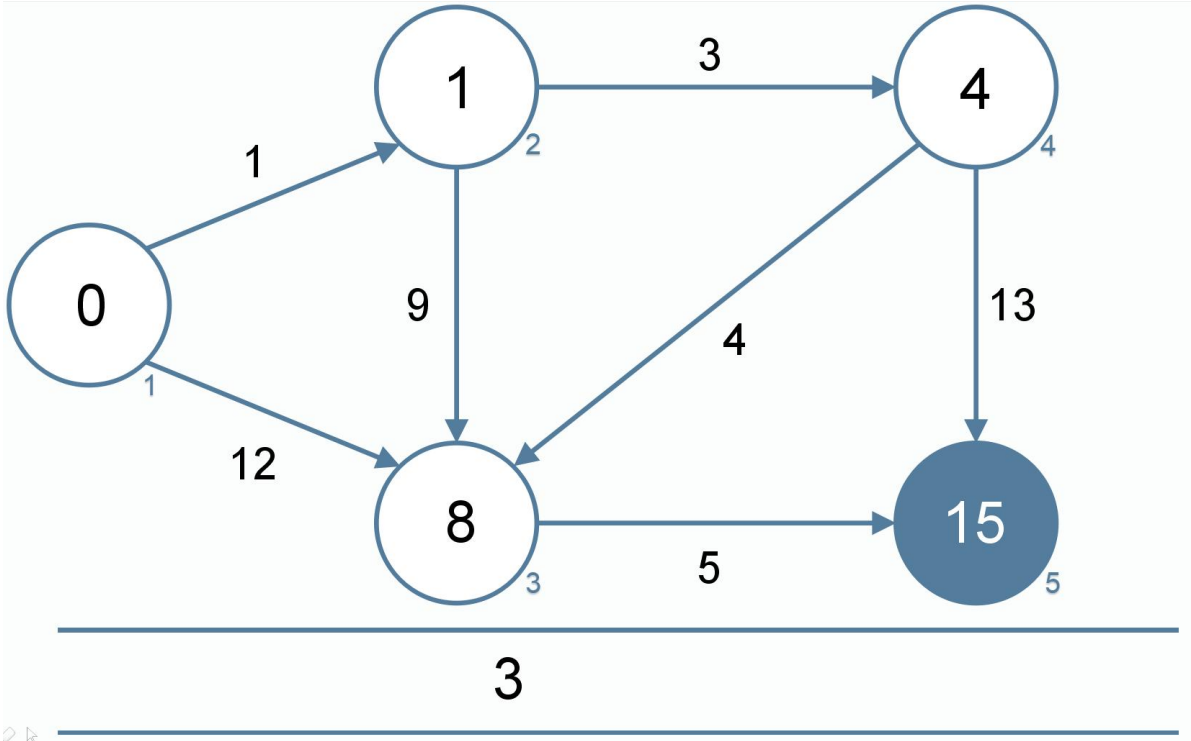
顶点2出队，顶点3、4进行松弛操作，4入队：

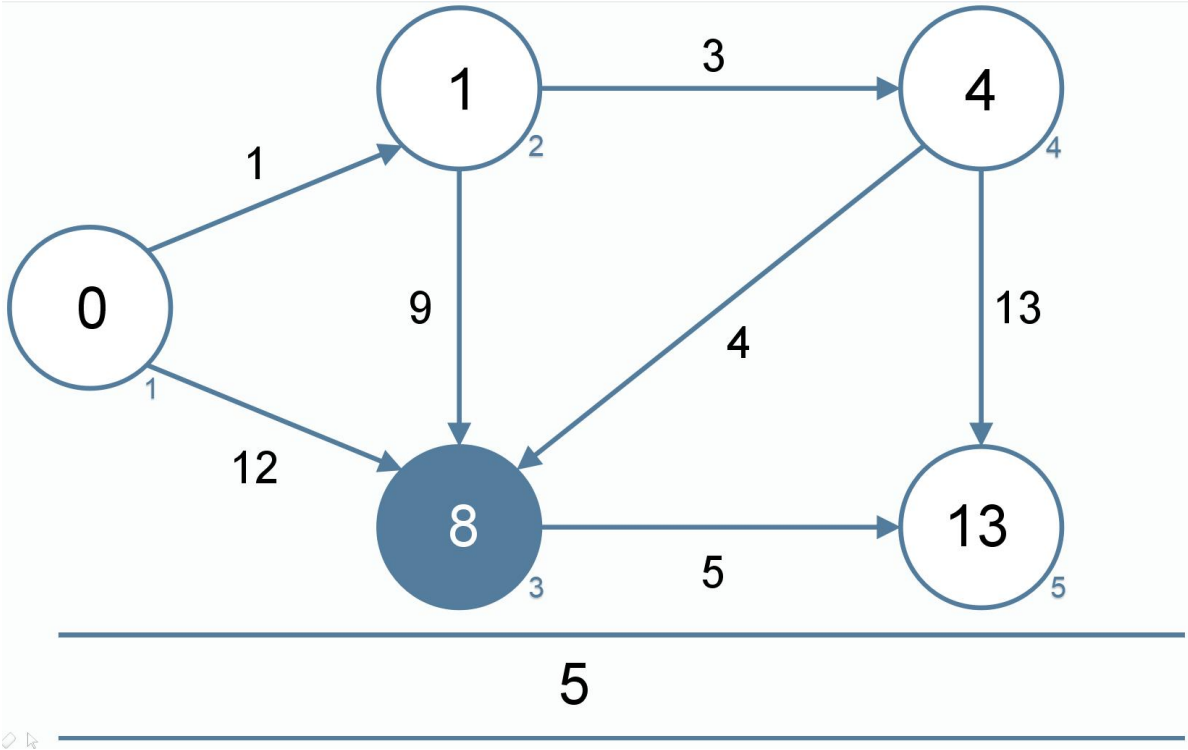




顶点4出队，3再次入队：







时间复杂度和空间复杂度

SPFA时间复杂度：O(kE)

k为各顶点平均入队次数，一般小于等于2

当图为稠密图时每个顶点入队接近V次

最坏时间复杂度：O(VE)

对比：

	Floyd算法	Dijkstra算法	SPFA算法
可解决问题	多源最短路径	单源最短路径	单源最短路径
是否可以有负权边	可以	不可以	可以
时间复杂度	$O(V^3)$	$O(V^2)$/O(ElogV)	O(KE)
空间复杂度	$O(V^2)$	O(V)	O(V)

热浪

描述

德克萨斯纯朴的民众们这个夏天正在遭受巨大的热浪！！！

他们的德克萨斯长角牛吃起来不错，可是它们并不是很擅长生产富含奶油的乳制品。

农夫John此时身先士卒地承担起向德克萨斯运送大量的营养冰凉的牛奶的重任，以减轻德克萨斯人忍受酷暑的痛苦。

John已经研究过可以把牛奶从威斯康星运送到德克萨斯州的路线。

这些路线包括起始点和终点一共有T个城镇，为了方便标号为1到T。

除了起点和终点外的每个城镇都由双向道路连向至少两个其它的城镇。

每条道路有一个通过费用（包括油费，过路费等等）。

给定一个地图，包含C条直接连接2个城镇的道路。

每条道路由道路的起点R_s，终点R_e和花费C_i组成。

求从起始的城镇T_s到终点的城镇T_e最小的总费用。

输入

第一行:4个由空格隔开的整数:T, C, T_s, T_e;

第2到第C+1行: 第i+1行描述第i条道路，包含3个由空格隔开的整数:R_s, R_e, C_i。

输出

一个单独的整数表示从T_s到T_e的最小总费用。

数据保证至少存在一条道路。

输入样例 1

```
7 11 5 4
2 4 2
1 4 3
7 2 2
3 4 3
5 7 5
7 3 3
6 1 1
6 3 4
2 4 3
5 6 3
7 2 1
```

输出样例 1

```
7
```

提示

数据范围

1≤T≤2500,
1≤C≤6200,
1≤T_s, T_e, R_s, R_e≤T,
1≤C_i≤1000。

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

const int N = 2510, M = 6200 * 2 + 10;

int n, m, S, T;
int h[N], w[M], to[M], ne[M], idx = 0;
```

```cpp
int q[N], dis[N];
bool book[N];

void add(int a, int b, int c){
    w[idx] = c, to[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void spfa(){
    memset(dis, 0x3f, sizeof dis);

    dis[S] = 0;
    int hh = 0, tt = 1;
    q[0] = S;
    book[S] = true;

    while(hh != tt){
        int t = q[hh++];
        if(hh == N) hh = 0;
        book[t] = false;

        for(int i = h[t]; ~i; i = ne[i]){
            int v = to[i];
            if(dis[v] > dis[t] + w[i]){
                dis[v] = dis[t] + w[i];
                if(!book[v]){
                    q[++tt] = v;
                    if(tt == N) tt = 0;
                    book[v] = true;
                }
            }
        }
    }
}

int main() {

    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof h);

    while(m--){
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
        add(b, a, c);
    }

    spfa();

    cout << dis[T] << endl;

    return 0;
}
```