
最短路问题

单源

正权

Dijkstra/迪杰斯特拉算法

负权

Bellman-Ford/贝尔曼算法

负权（优化）

SPFA算法

多源

Floyd/弗洛伊德算法

适用场景

Dijkstra算法适用于查找正权图中某个顶点至其他所有顶点的最短路径

Dijkstra**算法**适用于查找正权图中某个顶点至其他所有顶点的最短路径

Greedy**贪心算法：**X **负边**

1

单源

single-source

2

正权

positive weight

3

有向/无向图

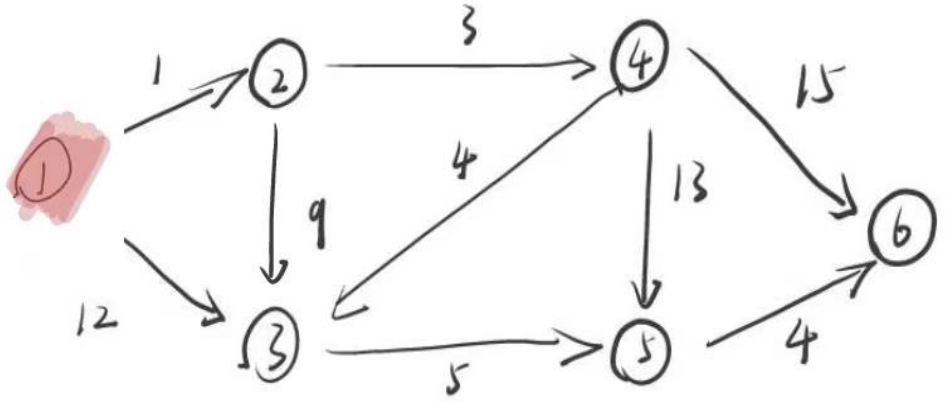
directed/undirected graph

4

对点递归

vertice

有向正权图



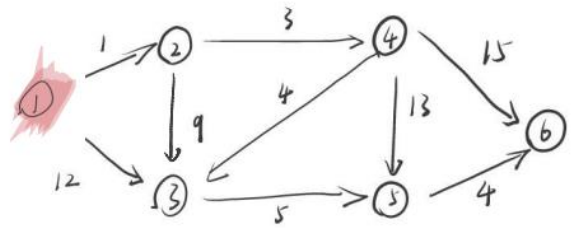
初始化

N	dis[N]	前置点	check (order)
1	0	×	
2	∞		
3	∞		
4	∞		
5	∞		
6	∞		

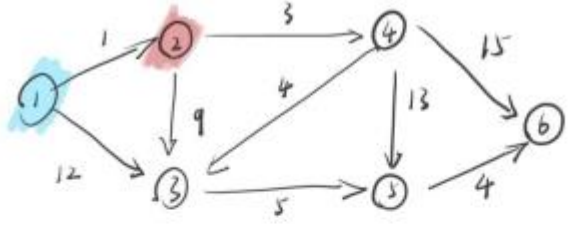
① 择：前置出发点 A → 目前

② 算：d[A ~ (未check，邻接)结点]

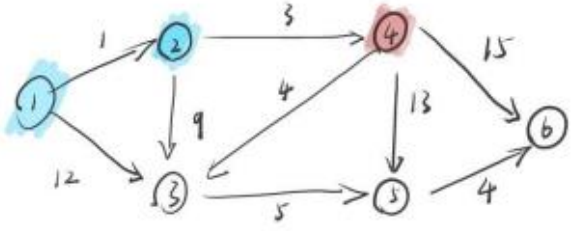
③ 比：取最优(min)，修改/记录前置点



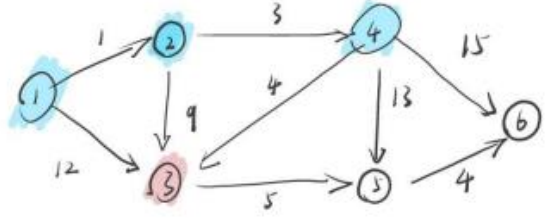
N	dis[N]	前置点	check (order)
1	0	×	1
2	∞ > 1 → 1	1	
3	∞ > 12 → 12	1	
4	∞		
5	∞		
6	∞		



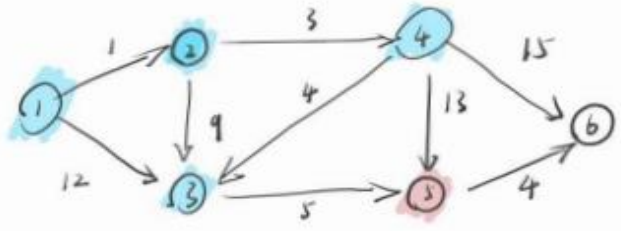
N	dis[N]	前置点	check (order)
1	0	×	1
2	1	1	2
3	12 > 10 → 10	2	
4	∞ > 4 → 4	2	
5	∞		
6	∞		



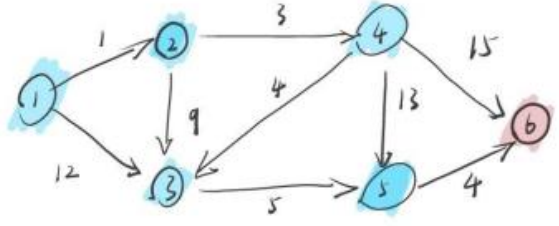
N	dis[N]	前置点	check (order)
1	0	×	1
2	1	1	2
3	10 > 8	4	
4	4	2	3
5	∞ > 17	4	
6	∞ > 19	4	



N	dis[N]	前置点	check (order)
1	0	×	1
2	1	1	2
3	8	4	4
4	4	2	3
5	17 > 13	3	
6	19	4	



N	dis[N]	前置点	check (order)
1	0	×	1
2	1	1	2
3	8	4	4
4	4	2	3
5	13	3	4
6	19 > 17	4	



N	dis[N]	前置点	check (order)
1	0	×	1
2	1	1	2
3	8	4	4
4	4	2	3
5	13	3	5
6	17	5	6

Dijkstra 板子 （洛谷：P4779）

题目描述

给定一个 n 个点，m 条有向边的带非负权图，请你计算从 s 出发，到每个点的距离。

数据保证你能从 s 出发到任意点。

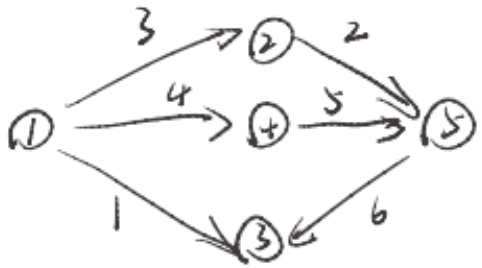
输入格式

第一行为三个正整数 n, m, s。第二行起 m 行，每行三个非负整数 u_i, v_i, w_i，表示从 u_i 到 v_i 有一条权值为 w_i 的有向边。

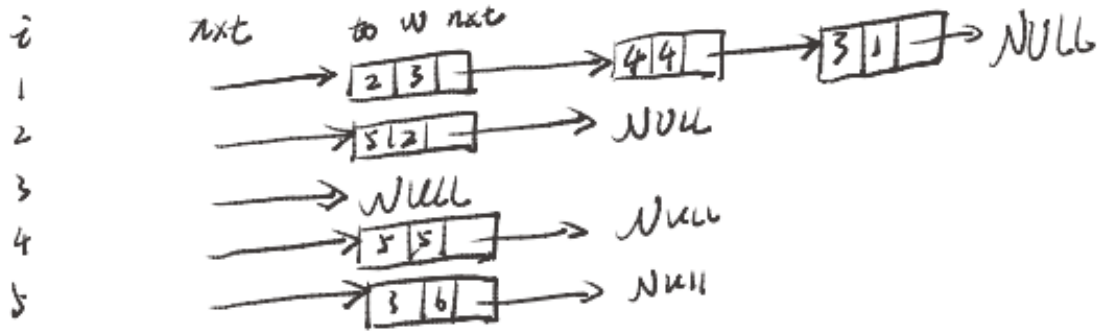
```
struct node{ //结点结构体定义
    int pos; //序号
    int dis; //d[di]:到单源的距离
    bool operator < (const node &x) const { //自定义cmp函数
        return x.dis < dis;
    }
};

priority_queue<node> Pq; //优先队列: 存点
```

```
void dijkstra() {
    //初始化数组d[i]
    memset(d,0x3f,sizeof(d)); //d[i] = inf
    d[s] = 0; //单源

    //创建队列 & 结构体node
    Pq.push(node{s, d[s]});

    //开始递归（循环）
    while (!Pq.empty()) { //队列非空 = 有意义
        node tmp = Pq.top(); Pq.pop(); //取最小dis的点from优先队列
        int x = tmp.pos;

        if (vis[x]) continue; //贪心：关闭已得到最短路的点
        vis[x] = true;

        for (int i = Head[x]; i ; i = e[i].nxt) { //链式向前星：所有邻接点
            int a = e[i].to;
            if (d[a] > d[x] + e[i].w) { //更新
                d[a] = d[x] + e[i].w;
                if (!vis[a]) {
                    Pq.push(node{a, d[a]});
                }
            }
        }
    }
}
```

链式向前星



i: 编号
nxt: 下一点的地址



链式向前星：静态邻接表

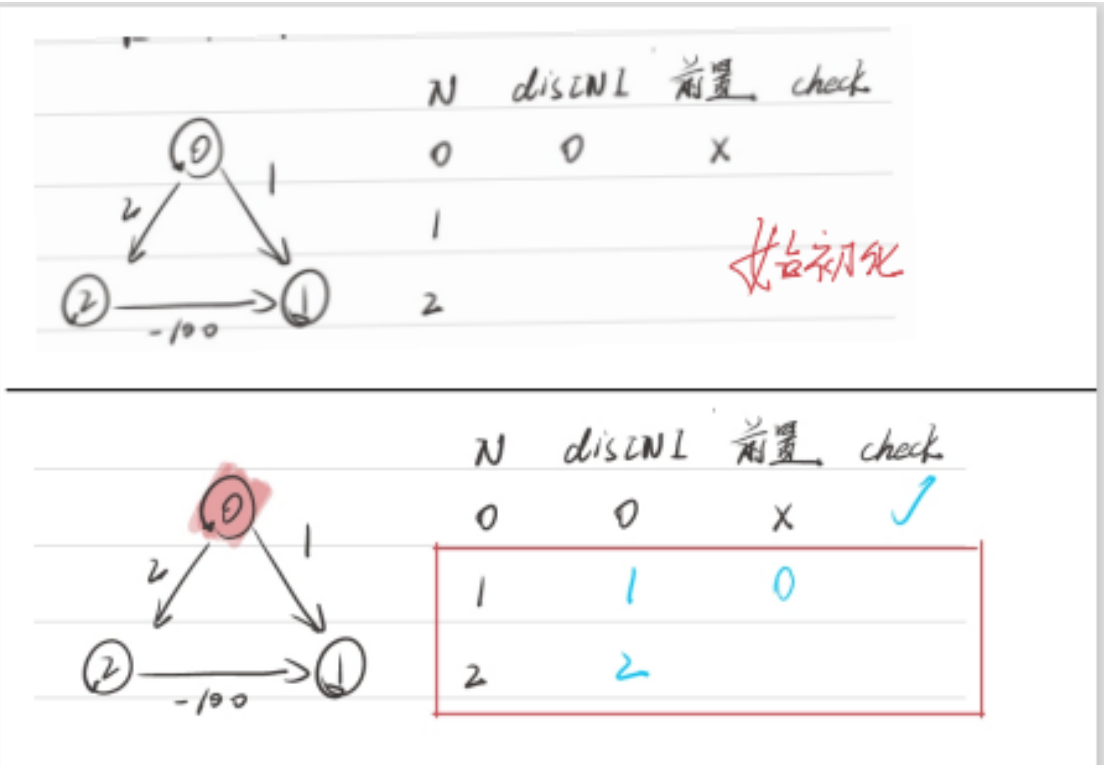
```
struct edge{
    int to;
    int w;
    int nxt;
}e[500005];
```

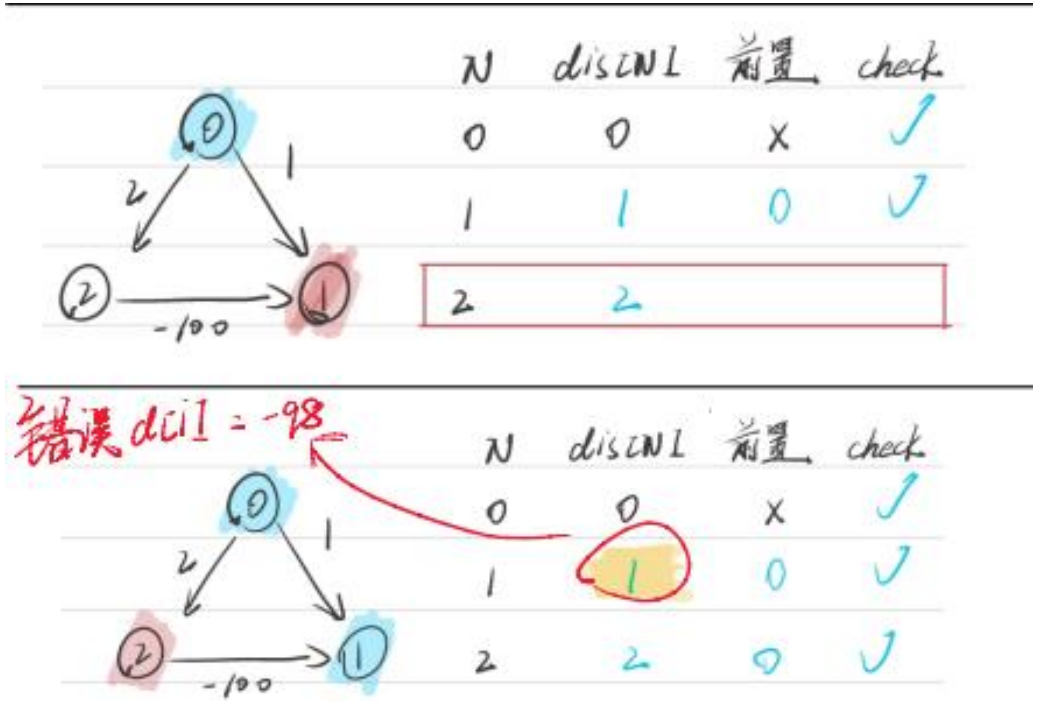
```
inline void add(int u, int v, int w) {
    e[++cnt].to = v;
    e[cnt].w = w;
    e[cnt].nxt = Head[u];
    Head[u] = cnt;
}
```

Greedy贪心算法：X 负边

当且仅当 A,B 非负 非严格单调递增
满足 A+B ≥ max(A,B)

eg. 已访问 → 关闭点 × 把重新的负边
negative cycle：永远都能找到↓dis

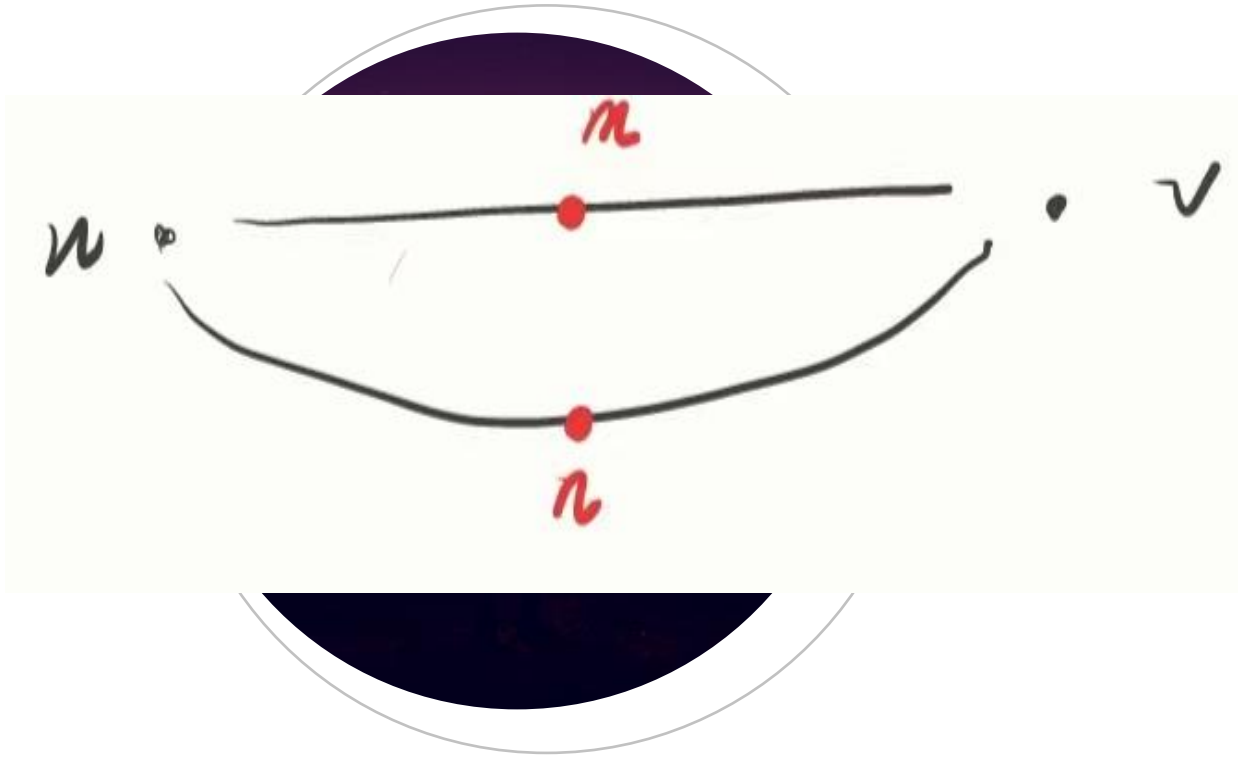




适用场景

贝尔曼福特算法适用于单源含负权的最短路问题

▲ 对边**松弛**：缩短距离→放松橡皮筋的压力

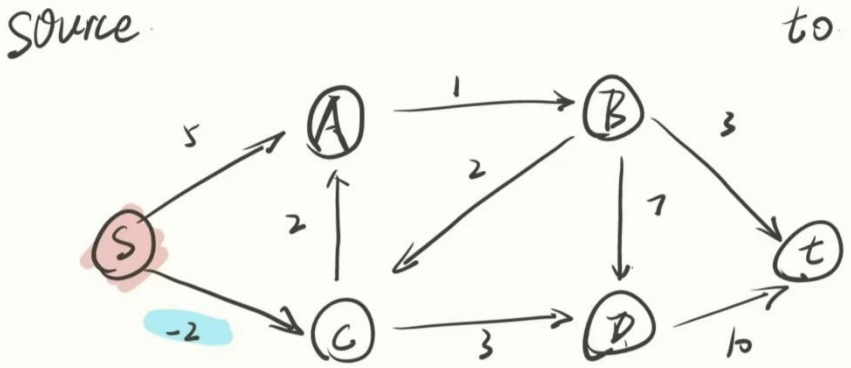


∵ d[unv] > d[umv]

∴ fn > fm

↓ d 则 f ↓

Bellman-ford



	S	A	B	C	D	t
d	0	∞	∞	∞	∞	∞
pr	X					

初始化

对边松弛 1st

	S	A	B	C	D	t
d	0	~~∞~~ 5	~~∞~~ 6	~~∞~~ -2	~~∞~~ 13 → 1	~~∞~~ 9
pr	X	~~S~~ C	A	S	~~B~~ C	B

```
void Bellman_ford() {
    memset(dis, 0x3f, sizeof(dis));//初始化
    dis[1] = 0;

    for(int i = 0; i < n - 1; i++) {//遍历所有边
        for(int j = 0; j < m; j++) {
            int a = edge[j].a, b = edge[j].b, w = edge[j].w;
            if(dis[a] != inf && dis[b] > dis[a] + w) //松弛!!!
                dis[b] = dis[a] + w;
        }
    }
}
```

V-1次松弛

Logic 原因

易知：$V=4$，共3条最路径

↳ 每条路径都受全局影响

∴ 需迭代$(V-1)$次，对全局的每条边松弛 $V-1$ 次

1次松弛 ⟶ ✓ 单源至所有点，经过边数 ≤1 的最短路

2次松弛 ⟶ ✓ 单源至所有点，经过边数 ≤2 的最短路

…

$n-1$次松弛 ⟶ ✓ 单源至所有点，经过边数 ≤$n-1$ 的最短路

适用场景

单源含负权边的最短路问题（Bellman-ford优化）

Bellman-Ford

① 松弛 n-1 次

② 不一定每轮松弛，更新多数边

× ⇓

↑ 无效判断

⇓

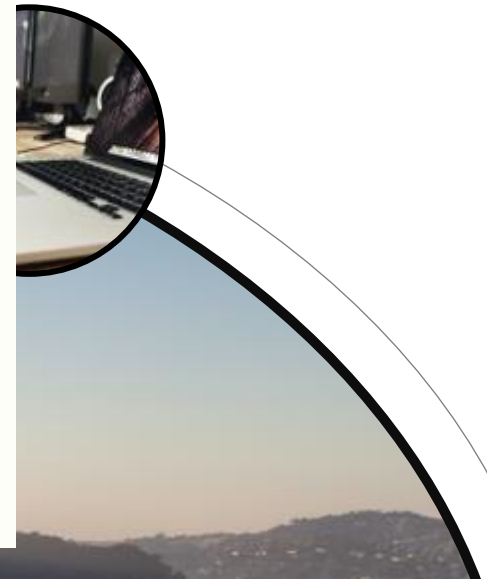
↓ 未更新点时连带更新

⇓

用队列优化：SPFA

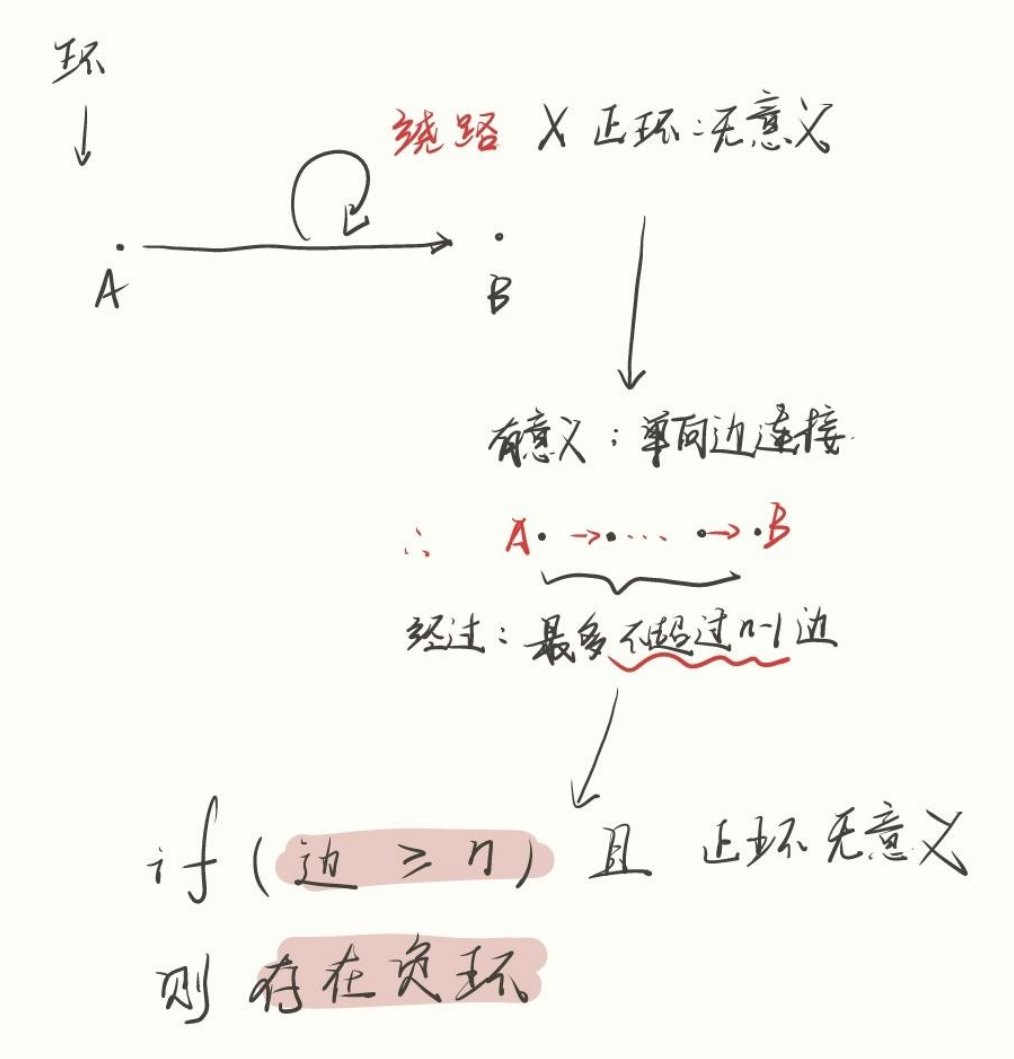
⇓

凡已更新 & 不在队列：入队



给定一个 n 个点的有向图，请求出图中是否存在**从顶点 1 出发能到达**的负环。

负环的定义是：一条边权之和为负数的回路。



```cpp
void SPFA()
{
    memset(dis,0,sizeof(dis));//初始化
    memset(cnt,0,sizeof(cnt));
    memset(vis,0,sizeof(vis));

    while(!q.empty()) q.pop();
    dis[1] = 0;
    vis[1] = 1;
    q.push(1);

    int u,v,w;
    while(!q.empty()) {
        u = q.front();
        vis[u] = 0;
        q.pop();

        for(int i = head[u]; i != -1; i=e[i].next){
            v=e[i].v; w=e[i].w;
            if(dis[u] + w <dis[v]){
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u]++;//记录最短路边数

                if(cnt[v] >= n) {//存在负环
                    printf("YES\n");
                    return;
                }
            }
            if(!vis[v]) {
                vis[v]=1;q.push(v);
            }
        }
    }
    printf("NO\n");
}
```

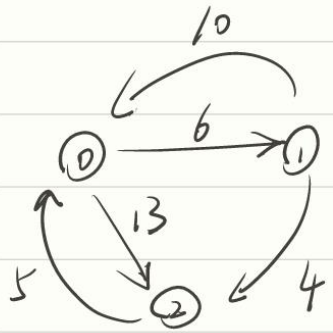
Our Team

You can choose whether or not to support certain items, If you choose to support an item, this will be identified on the item page. All supported items include a support period. Buyers can buy support extensions on these item. Some transactions may be subject to tax that may be added to the list

Floyd

下标k：以k为中继节点

不断画十字



P_{-1}

	0	1	2
0	0	6	13
1	10	0	4
2	5	∞	0

P_0

	0	1	2
0	0	6	13
1	10	0	4
2	5	∞	0

∵ $d[2-1] = \infty$

$d[2-0-1] = d[2-0]+d[0-1]$

$= 5+6 = 11$

且 $11 < \infty$

∴ $P[2-1] = 11$

	0	1	2
0	0	6	13
1	10	0	4
2	5	11	0

P_1

	0	1	2
0	0	6	13
1	10	0	4
2	5	11	0

∵ $d[0-2] = 13$

$d[0-1]+d[1-2]$

$= 6+4 = 10$

且 $10 < 13$

∴ $P[1-2] = 10$

	0	1	2
0	0	6	10
1	10	0	4
2	5	11	0

P_2

	0	1	2
0	0	6	10
1	9	0	4
2	5	11	0

题目背景

复制Markdown 展开

B 地区在地震过后，所有村庄都造成了一定的损毁，而这场地震却没对公路造成什么影响。但是在村庄重建好之前，所有与未重建完成的村庄的公路均无法通车。换句话说，只有连接着两个重建完成的村庄的公路才能通车，只能到达重建完成的村庄。

题目描述

给出 B 地区的村庄数 N，村庄编号从 0 到 $N-1$，和所有 M 条公路的长度，公路是双向的。并给出第 i 个村庄重建完成的时间 t_i，你可以认为是同时开始重建并在第 t_i 天重建完成，并且在当天即可通车。若 t_i 为 0 则说明地震未对此地区造成损坏，一开始就可以通车。之后有 Q 个询问 (x, y, t)，对于每个询问你要回答在第 t 天，从村庄 x 到村庄 y 的最短路径长度为多少。如果无法找到从 x 村庄到 y 村庄的路径，经过若干个已重建完成的村庄，或者村庄 x 或村庄 y 在第 t 天仍未重建完成，则需要返回 `-1`。

输入格式

第一行包含两个正整数N, M，表示了村庄的数目与公路的数量。

第二行包含N个非负整数$t_0, t_1, \ldots, t_{N-1}$，表示了每个村庄重建完成的时间，数据保证了$t_0 \leq t_1 \leq \ldots \leq t_{N-1}$。

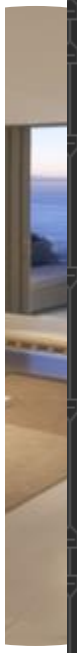
接下来M行，每行3个非负整数i, j, w，w为不超过10000的正整数，表示了有一条连接村庄i与村庄j的道路，长度为w，保证$i \neq j$，且对于任意一对村庄只会存在一条道路。

接下来一行也就是$M+3$行包含一个正整数Q，表示Q个询问。

接下来Q行，每行3个非负整数x, y, t，询问在第t天，从村庄x到村庄y的最短路径长度为多少，数据保证了t是不下降的。

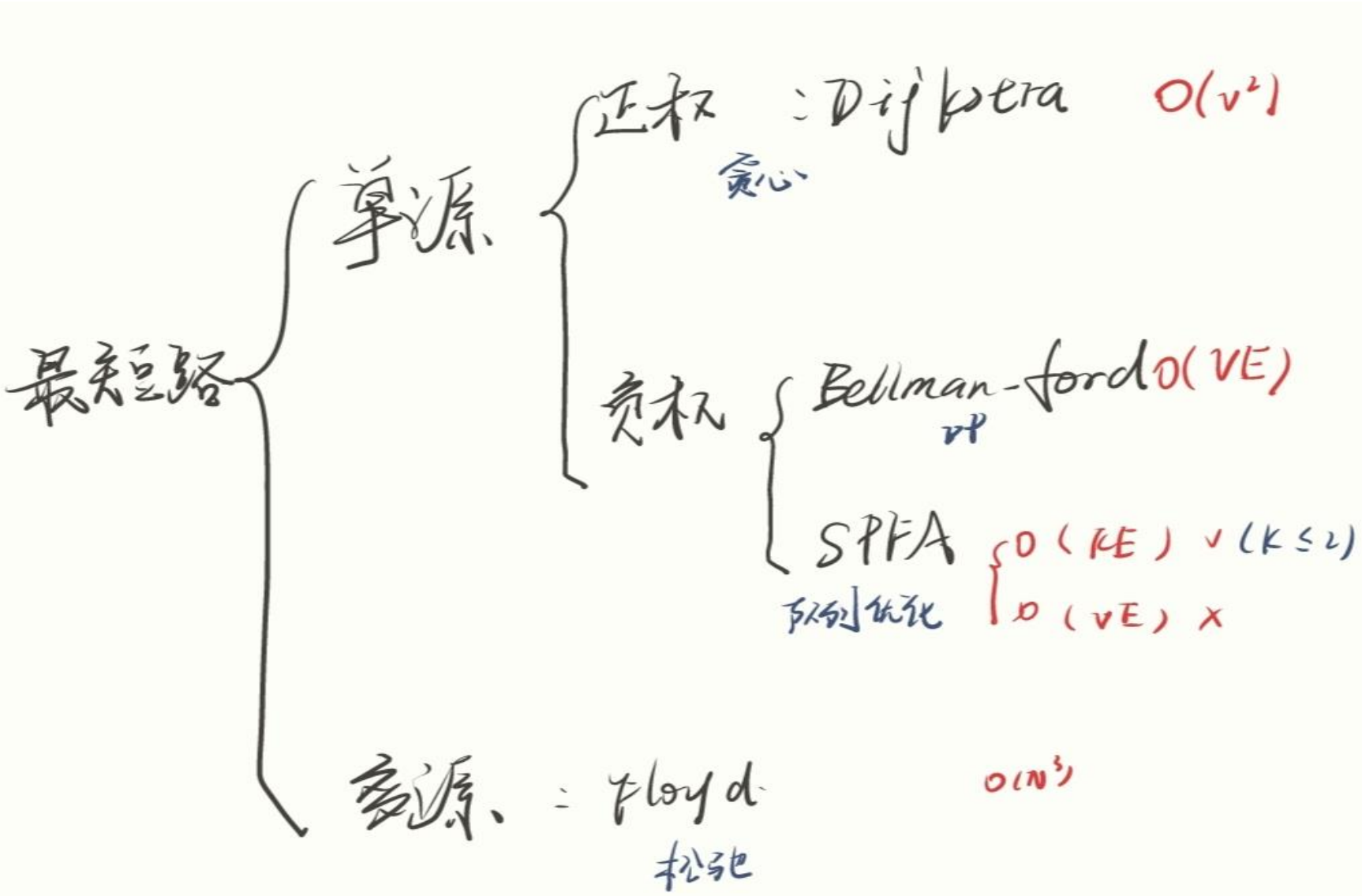
输出格式

共Q行，对每一个询问(x, y, t)输出对应的答案，即在第t天，从村庄x到村庄y的最短路径长度为多少。如果在第t天无法找到从x村庄到y村庄的路径，经过若干个已重建完成的村庄，或者村庄x或村庄y在第t天仍未修复完成，则输出-1。



```cpp
void floyd(int k) {//"三"层循环
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            edge[i][j] = min(edge[i][j], edge[i][k] + edge[k][j]);
        }
    }
}

void work() {
    int cur = 0;
    cin >> q;
    for(int i = 0; i < q; i++) {
        int u, v, tt;
        cin >> u >> v >> tt;
        while(t[cur] <= tt && cur < n) {//cur为中继点更新
            floyd(cur);
            cur++;
        }
        if(edge[u][v] == INF || t[u] > tt || t[v] > tt) {//村庄x未建成,不连通
            cout << "-1" << endl;
        }
        else {
            cout << edge[u][v] << endl;
        }
    }
}
```

总结


最短路
单源
正权 : Dijkstra O(v²)
贪心
负权
Bellman-ford O(VE)
DP
SPFA
O(KE) √ (K≤2)
队列优化
O(VE) ×
多源 : Floyd O(N³)
松弛

Thanks