Dijkstra

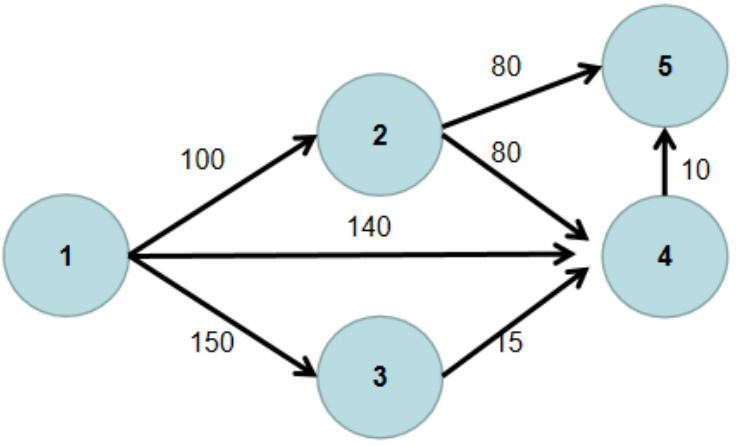
朴素Dijkstra算法

AcWing 849. Dijkstra求最短路 I

给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环，所有边权均为正值。
请你求出 1 号点到 n 号点的最短距离，如果无法从 1 号点走到 n 号点，则输出 −1。
输入：第一行包含整数 n 和 m。$1 \leq n \leq 500$, $1 \leq m \leq 10^5$
接下来 m 行每行包含三个整数 x , y , z，表示存在一条从点 x 到点 y 的有向边，边长为 z。
注：图中涉及的边长均不超过10000。
输出：一个整数，表示 1 号点到 n 号点的最短距离。
如果路径不存在，则输出 −1。
求源点到其余各点的最短距离步骤如下：

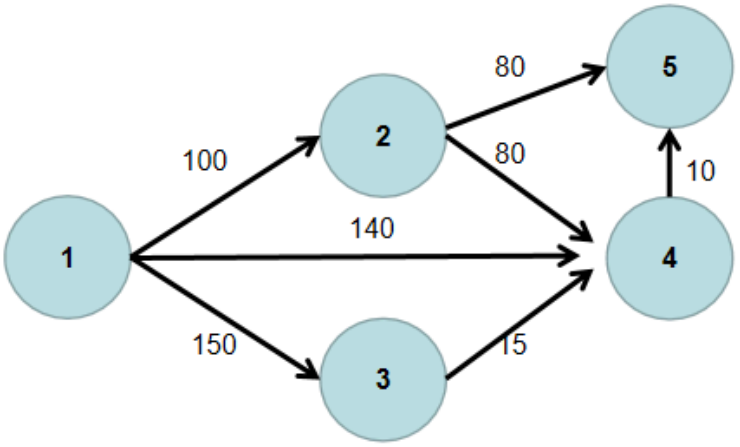
1. 用一个 dist 数组保存源点到其余各个节点的距离，dist[i] 表示源点到节点 i 的距离。初始时，dist 数组的各个元素为无穷大。用一个状态数组 state 记录是否找到了源点到该节点的最短距离，state[i] 如果为真，则表示找到了源点到节点 i 的最短距离，state[i] 如果为假，则表示源点到节点 i 的最短距离还没有找到。初始时，state 各个元素为假。

序号	dist	state
1	∞	0
2	∞	0
3	∞	0
4	∞	0
5	∞	0



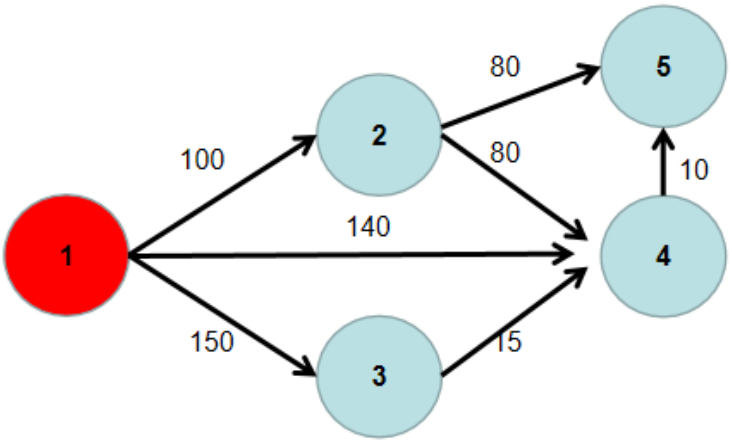
2. 源点到源点的距离为 0。即dist[1] = 0。

序号	dist	state
1	0	0
2	∞	0
3	∞	0
4	∞	0
5	∞	0



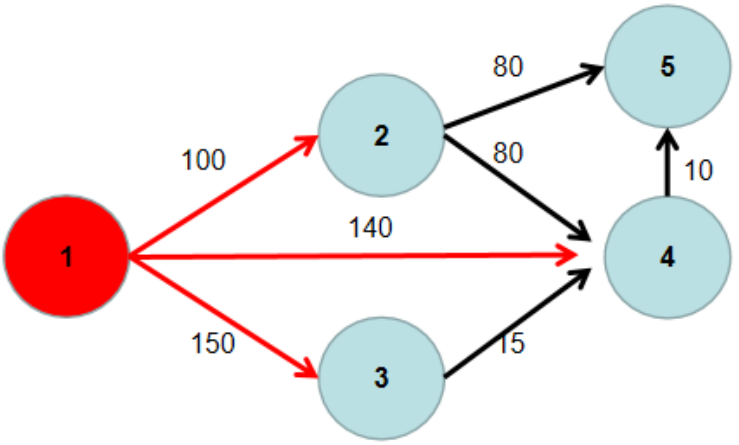
3. 遍历 dist 数组，找到一个节点，这个节点是：没有确定最短路径的节点中距离源点最近的点。假设该节点编号为 i。此时就找到了源点到该节点的最短距离，state[i] 置为 1。

序号	dist	state
1	0	1
2	∞	0
3	∞	0
4	∞	0
5	∞	0



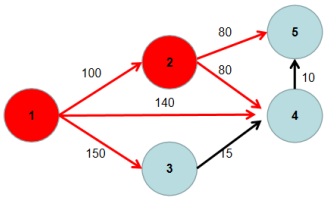
4. 遍历 i 所有可以到达的节点 j，如果 dist[j] 大于 dist[i] 加上 i -> j 的距离，即 dist[j] > dist[i] + g[i][j]（g[i][j] 为 i -> j 的距离），则更新 dist[j] = dist[i] + g[i][j]。

序号	dist	state
1	0	1
2	100	0
3	150	0
4	140	0
5	∞	0

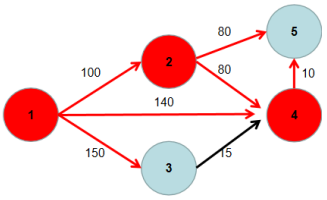


5. 重复 3 4 步骤，直到所有节点的状态都被置为 1。

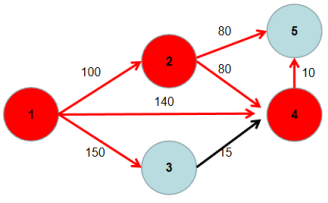
序号	dist	state
1	0	1
2	100	1
3	150	0
4	140	0
5	∞	0



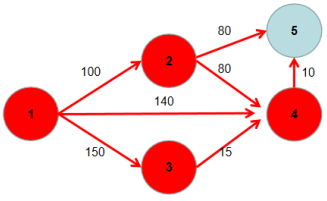
序号	dist	state
1	0	1
2	100	1
3	150	0
4	140	1
5	180	0



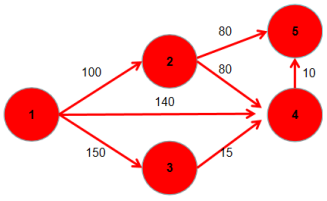
序号	dist	state
1	0	1
2	100	1
3	150	0
4	140	1
5	150	0



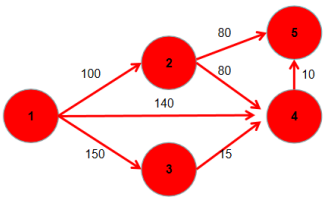
序号	dist	state
1	0	1
2	100	1
3	150	1
4	140	1
5	150	0



序号	dist	state
1	0	1
2	100	1
3	150	1
4	140	1
5	150	1

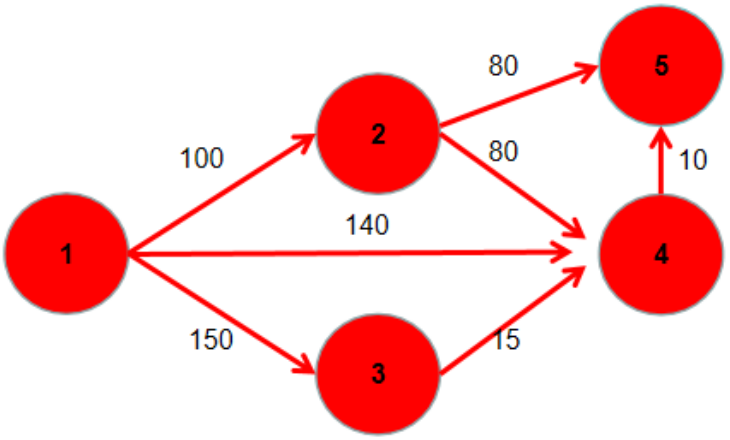


序号	dist	state
1	0	1
2	100	1
3	150	1
4	140	1
5	150	1



6. 此时 dist 数组中，就保存了源点到其余各个节点的最短距离。

序号	dist	state
1	0	1
2	100	1
3	150	1
4	140	1
5	150	1




```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 510;
int n, m, g[N][N], dist[N];
bool st[N];

int dijkstra(){
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    // 其实循环 n-1 次就可以了
    for(int i = 1; i < n; i++){
        int t = -1;
        // 寻找还未确定最短路的点中路径最短的点
        for(int j = 1; j <= n; j++)
            if(!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;

        for(int j = 1; j <= n; j++)
            dist[j] = min(dist[j], dist[t] + g[t][j]);

        st[t] = true;
    }
    // 如果起点到达不了n号节点，则返回 -1
    if(dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

int main(){
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> n >> m;
    memset(g, 0x3f, sizeof g);
    while(m--){
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c);
```

```
    }
    cout << dijkstra() << '\n';
    return 0;
}
```

堆优化Dijkstra算法

AcWing 850. Dijkstra求最短路 II

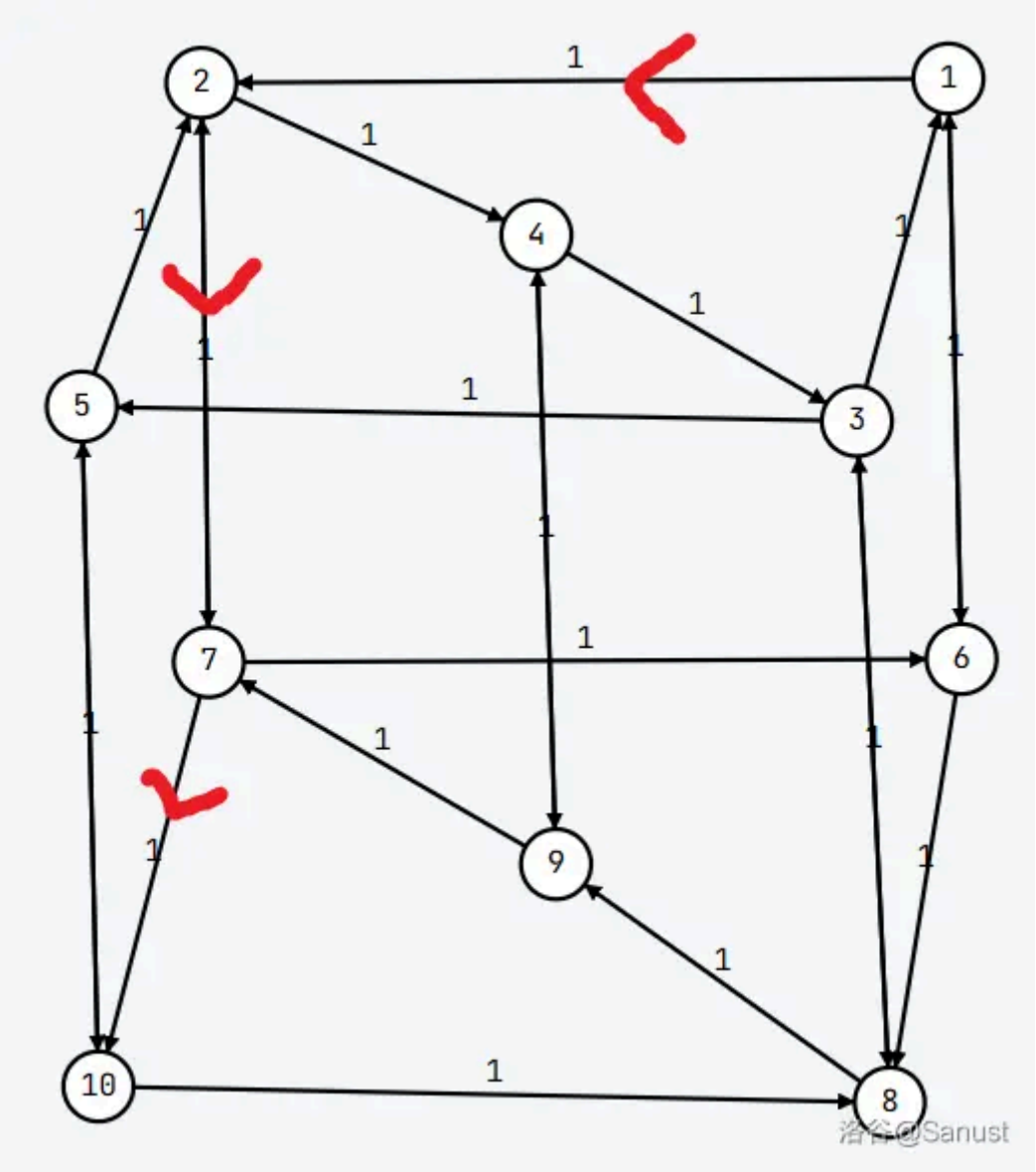
给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环，所有边权均为正值。
请你求出 1 号点到 n 号点的最短距离，如果无法从 1 号点走到 n 号点，则输出 -1。
输入：第一行包含整数 n 和 m。$1 \leq n, m \leq 1.5 \times 10^5$
接下来 m 行每行包含三个整数 x，y，z，表示存在一条从点 x 到点 y 的有向边，边长为 z。
注：图中涉及的边长均不超过10000。
如果最短路存在，则最短路的长度不超过 10^9
输出：一个整数，表示 1 号点到 n 号点的最短距离。
如果路径不存在，则输出 -1。

```
#include <bits/stdc++.h>
using namespace std;
#define pii pair<int, int>
const int N = 1.5e5 + 10, M = 1.5e5 + 10;
int n, m, h[N], w[M], e[M], ne[M], idx, dist[N];
bool st[N];

void add(int a, int b, int c){
    w[idx] = c;            // 边权重
    e[idx] = b;            // 终点
    ne[idx] = h[a];        // 下一条边
    h[a] = idx ++;         // 当前点的第一条边索引
    return;
}

int dijkstra(){
    memset(dist, 0x3f, sizeof dist);  // 初始化距离为无穷大
    dist[1] = 0;                      // 起点到自己的距离为 0
    priority_queue<pii, vector<pii>, greater<pii>> heap;  // 小根堆
    heap.push({0, 1});  // 起点入堆
    while(heap.size()){
        pii cur = heap.top();
        heap.pop();
        int distance = cur.first, ver = cur.second;
        if(st[ver]) continue; // 如果当前点已经被访问，跳过
        st[ver] = true;
        for(int i = h[ver]; i != -1; i = ne[i]){  // 遍历 ver 的所有邻接边
            int j = e[i];  // 获取邻接点
            if(dist[j] > distance + w[i]){  // 如果通过 ver 能更新 j 的距离
```

```
                dist[j] = distance + w[i];
                heap.push({dist[j], j});  // 更新后的点重新入堆
            }
        }
    }
    if(dist[n] == 0x3f3f3f3f) return -1; // 如果无法到达目标点，返回 -1
    return dist[n];
}

int main(){
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while(m--){
        int a, b, c; cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dijkstra() << '\n';
    return 0;
}
```

AT_abc395_e. Flip Edge(双层图+Dijkstra)

考虑建一个双层图，层与层之间建一条代价为 X 的**无向**边，上面一层图是原图，下面一层图是反向边后的图。这样建图后的样例二看起来就像这样：




```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 10;
#define int long long
#define pii pair<int, int>
int n, m, k, h[N], w[N], e[N], ne[N], dist[N], idx;
bool st[N];

void add(int a, int b, int c){
    w[idx] = c; // 权值
    e[idx] = b; // 终点
    ne[idx] = h[a];
    h[a] = idx ++;
    return;
}

void dijkstra(){
    memset(dist, 0x3f, sizeof dist);
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    dist[1] = 0;
```

```
    pq.push({0, 1});
    while(!pq.empty()){
        auto [distance, ver] = pq.top(); pq.pop();
        if(st[ver]) continue;
        st[ver] = true;
        for(int i = h[ver]; i != -1; i = ne[i]){
            int j = e[i];
            if(dist[j] > distance + w[i]){
                dist[j] = distance + w[i];
                pq.push({dist[j], j});
            }
        }
    }
    cout << min(dist[n], dist[n << 1]) << '\n';
}

signed main(){
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> n >> m >> k;
    memset(h, -1, sizeof h);
    for(int i = 1; i <= n; i++) // 上下两层图建边权为x的无向边
        add(i, i + n, k), add(i + n, i, k);
    while(m--){
        int a, b;
        cin >> a >> b;
        add(a, b, 1); // 原图正向边
        add(b + n, a + n, 1); // 复制图反向边
    }
    dijkstra();
    return 0;
}
```

航线(多层图+Dijkstra)

[zjhuoj(仅样例数据)]
[hduoj]

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define arr4 array<int, 4> // 时间, x, y, z
#define vi vector<int>
#define v2i vector<vi>
#define v3i vector<v2i>
int q, n, m;
int dx[4] = {-1, 1, 0, 0}, dy[4] = {0, 0, -1, 1}, dz[4] = {0, 1, 2, 3};
// 0上 1下 2左 3右
```

```
void djk(const v2i &d, const v2i &t, v3i &dist, v3i &vis){
    priority_queue<arr4, vector<arr4>, greater<arr4>> pq;
    pq.push({0, 1, 1, 3});
    while(!pq.empty()){
        auto [time, x, y, z] = pq.top(); pq.pop();
        if(vis[x][y][z]) continue;
        vis[x][y][z] = 1;
        if(x == n && y == m){
            if(z == 1) cout << time + t[x][y] << '\n';
            else{pq.push({time + d[x][y], x, y, 1}); continue;}
            return ;
        }
        for(int i = 0; i < 4; i++){
            int nx = x + dx[i], ny = y + dy[i];
            if(nx < 1 || nx > n || ny < 1 || ny > m || vis[nx][ny][i])
continue;
            if(z == i) pq.push({time + t[x][y], nx, ny, i});
            else pq.push({time + t[x][y] + d[x][y], nx, ny, i});
        }
    }
}

signed main(){
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> q;
    while(q--){
        cin  >> n >> m;
        v2i d(n + 1, vi(m + 1, 0)), t(n + 1, vi(m + 1, 0));
        v3i dist(n + 1, v2i(m + 1, vi(4, 0))), vis(n + 1, v2i(m + 1,
vi(4, 0)));
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++) cin>> t[i][j];
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++) cin>> d[i][j];
        djk(d, t, dist, vis);
    }
    return 0;
}
```