最短路（上）

从图中的**某个顶点**出发到达**另外一个顶点的所经过的边的权重和最小的一条路径**，称为最短路径.

单源最短路径问题(Single Source Shortest Path, SSSP 问题)，给定一张有向图 $G=(V,E)$，V 是点集，E 是边集，$|V|=n$，$|E|=m$，节点以 $[1,n]$ 之间的连续整数编号，(x,y,z) 描述一条从 x 出发，到达 y，长度为 z 的有向边. 设 1 号点为起点，求长度为 n 的数组 $dist$，其中 $dist[i]$ 表示从 1 到节点 i 的最短路径的长度.

1. $Dijkstra$ 算法：

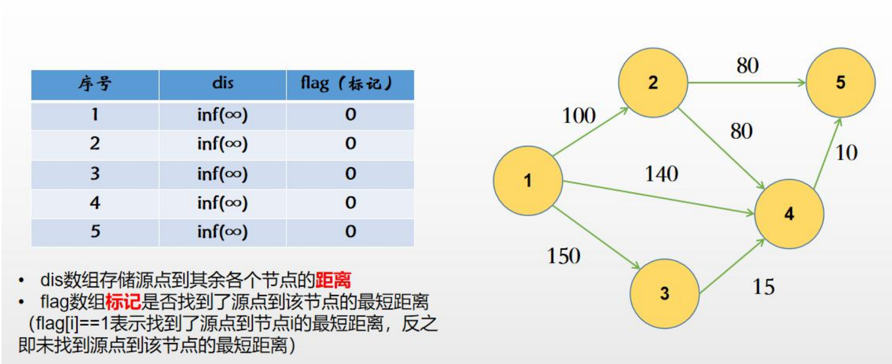
$Dijkstra$ 算法步骤如下：

1. 初始化起点 $start$ 的 $dist[start]=0$，其余结点的 $dist$ 的值为**正无穷大**；
2. 找出一个**未被标记**的、$dist[x]$ **最小的结点** x，然后标记结点 x；
3. 扫描结点 x 的所有出边 (x,y,z)，若 $dist[y]>dist[x]+z$，则使用 $dist[x]+z$ 更新 $dist[y]$；
4. 重复上述 $2\sim3$ 两个步骤，直到所有结点都被标记.

$Dijkstra$ 算法基于贪心的思想，它只适用于**所有边的长度都是非负数**的图. 当边长 z 都是非负数时，全局最小值不可能再被其他结点更新，故在第一步选出的结点 x 必然满足：$dist[x]$ 已经是起点到 x 的最短路径. 我们不断选择全局最小值进行标记和扩展，最终可得到起点 $start$ 到每一个结点的最短路径的长度.

举例：

1. 用一个 dis 数组保存源点到其余各个节点的距离, $dis[i]$ 表示源点到节点 i 的距离. 初始时 dis 数组的各个元素为无穷大; 用一个数组 $flag$ 标记是否找到了源点到该节点的最短距离, 初始时 $flag$ 数组的各个元素清为 0；

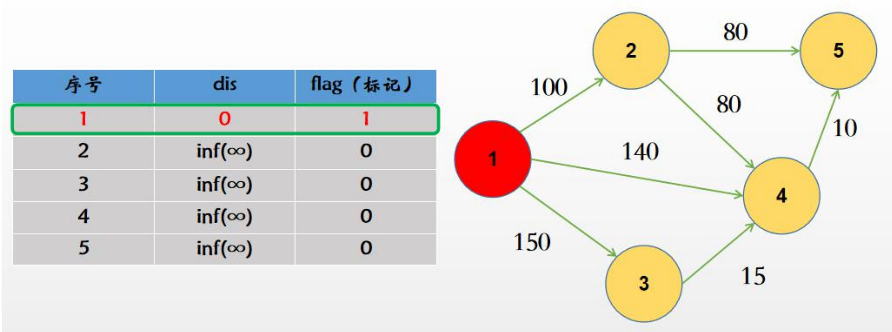


序号	dis	flag（标记）
1	inf(∞)	0
2	inf(∞)	0
3	inf(∞)	0
4	inf(∞)	0
5	inf(∞)	0

2. 假设源点为 1, 将其 $dis[1]$ 变为 0（源点到源点距离为 0）；

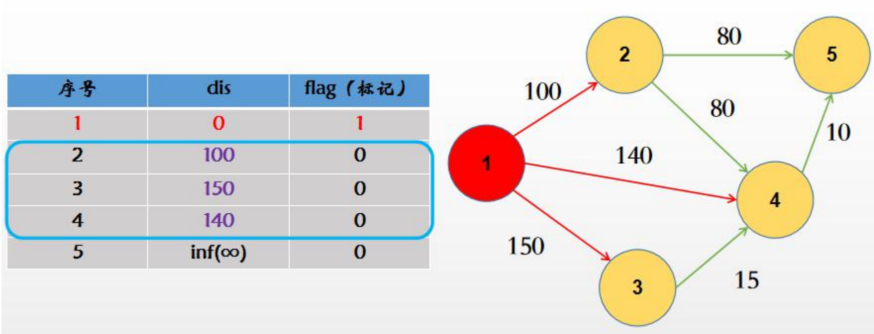
序号	dis	flag（标记）
1	0	0
2	inf(∞)	0
3	inf(∞)	0
4	inf(∞)	0
5	inf(∞)	0

3. 遍历 dis 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, $flag[i]$ 置为 1；



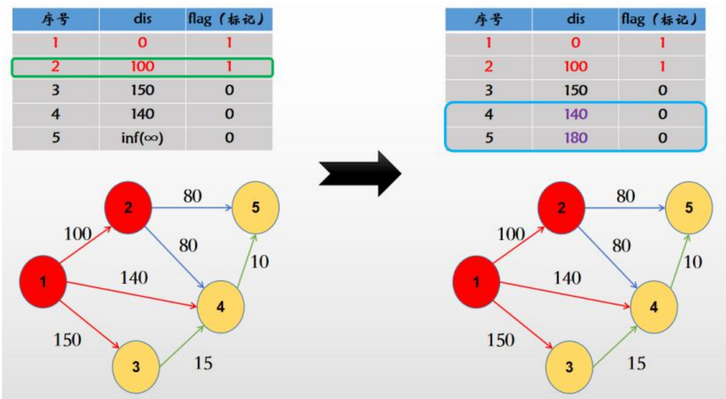
序号	dis	flag（标记）
1	0	1
2	inf(∞)	0
3	inf(∞)	0
4	inf(∞)	0
5	inf(∞)	0

4. 遍历 i 所有可以到达的节点 j，如果 $dis[j] > dis[i]+w[i][j]$（$w[i][j]$ 表示 $i \rightarrow j$ 的距离），则更新 $dis[j]$ 为 $dis[i]+w[i][j]$；



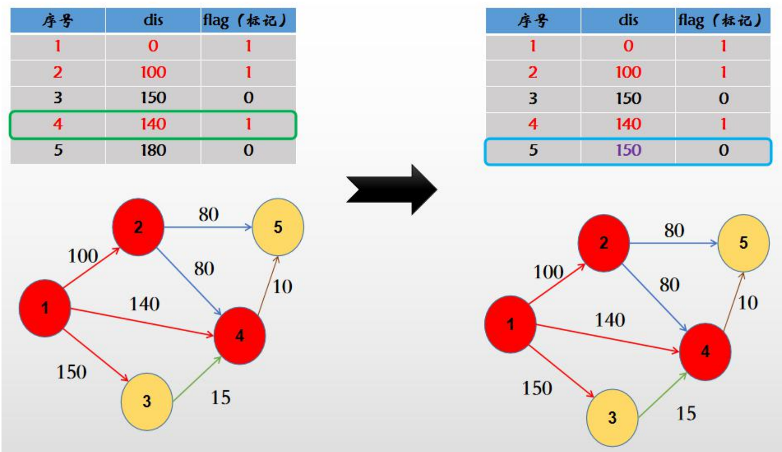
序号	dis	flag（标记）
1	0	1
2	100	0
3	150	0
4	140	0
5	inf(∞)	0

5. 重复 $3 \sim 4$ 步骤，直到所有节点的 $flag$ 都标记为 1；



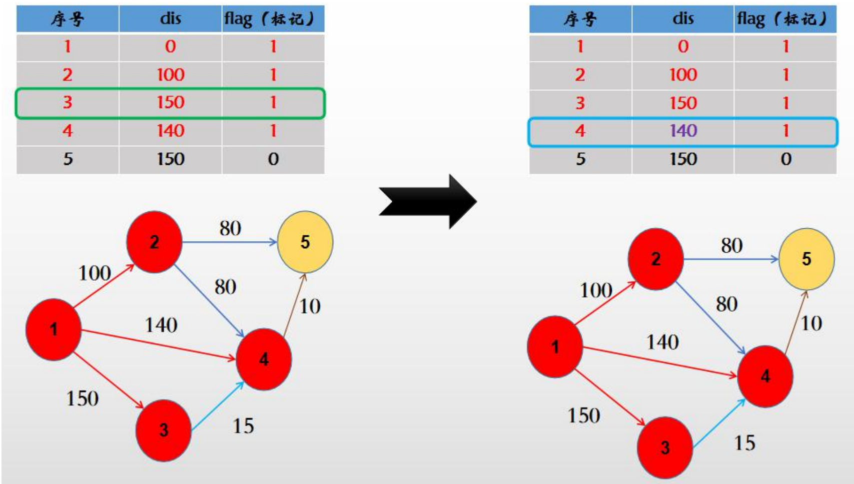
序号	dis	flag（标记）
1	0	1
2	100	1
3	150	0
4	140	0
5	inf(∞)	0

序号	dis	flag（标记）
1	0	1
2	100	1
3	150	0
4	140	0
5	180	0



序号	dis	flag（标记）
1	0	1
2	100	1
3	150	0
4	140	1
5	180	0

序号	dis	flag（标记）
1	0	1
2	100	1
3	150	0
4	140	1
5	150	0



6. 此时 *dis* 数组中，就保存了源点到其余各个节点的最短距离.

序号	dis	flag（标记）
1	0	1
2	100	1
3	150	1
4	140	1
5	150	1

代码链接

```
#include<iostream>
#include<cstring>
#include<algorithm>

#define MAXN 3010

using namespace std;

const int inf=0x3f3f3f3f;
int n,m;
int dist[MAXN],vis[MAXN];
int g[MAXN][MAXN];//用邻接矩阵存图

void dijkstra(int start){
    memset(dist,0x3f,sizeof(dist));//将dist 数组置为无穷大
    memset(vis,0,sizeof(vis));//将标记数组vis 清零
    dist[start]=0;
    for(int i=1;i<=n-1;i++){//重复进行n-1 次
        int pos=-1;//找到未标记结点中dist 最小的
```

```
        for(int j=1;j<=n;j++)
            if(vis[j]==0&&(pos==-1||dist[j]<dist[pos]))
                pos=j;
        vis[pos]=1;
        //用全局最小值点pos 更新其他结点
        for(int j=1;j<=n;j++)
            dist[j]=min(dist[j],dist[pos]+g[pos][j]);
    }
}

int main(){
    scanf("%d %d",&n,&m);
    memset(g,0x3f,sizeof(g));//将邻接矩阵置为无穷大
    for(int i=1;i<=n;i++)
        g[i][i]=0;
    for(int i=1;i<=m;i++){
        int x,y,z;
        scanf("%d %d %d",&x,&y,&z);
        g[x][y]=min(g[x][y],z);//由于存在重边,需更新为该边的最小值
    }
    dijkstra(1);//求单源最短路径
    for(int i=1;i<=n;i++){
        if(dist[i]==inf)//该点无法从起点start 到达
            printf("-1\n");
        else printf("%d\n",dist[i]);
    }
    return 0;
}
```

上面程序的时间复杂度为 $O(n^2)$，主要瓶颈在于第一步寻找全局最小值的过程，可用二叉堆（优先队列 $priority\_queue$）对 $dist$ 数组进行维护，用 $O(\log n)$ 的时间获取最小值并从堆中删除，用 $O(\log n)$ 的时间执行一条边的扩展和更新，最终可在 $O(m\log n)$ 的时间内实现 $Dijkstra$ 算法．这部分可自行了解．　[代码链接]

2. spfa 算法

$spfa$ 算法步骤如下：

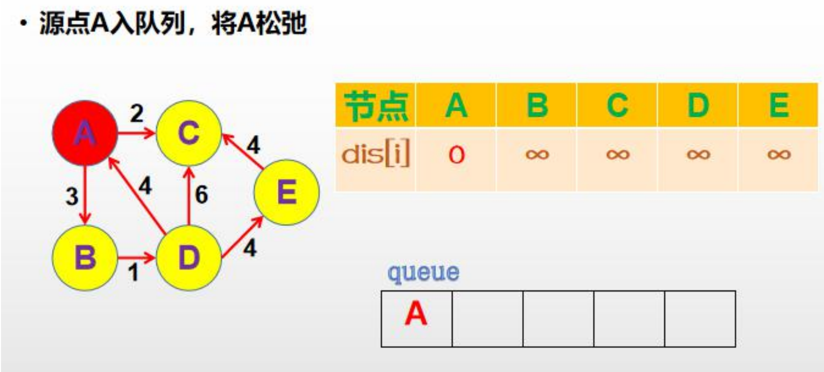
1. 建立一个队列，最初队列中只含有起点 $start$.

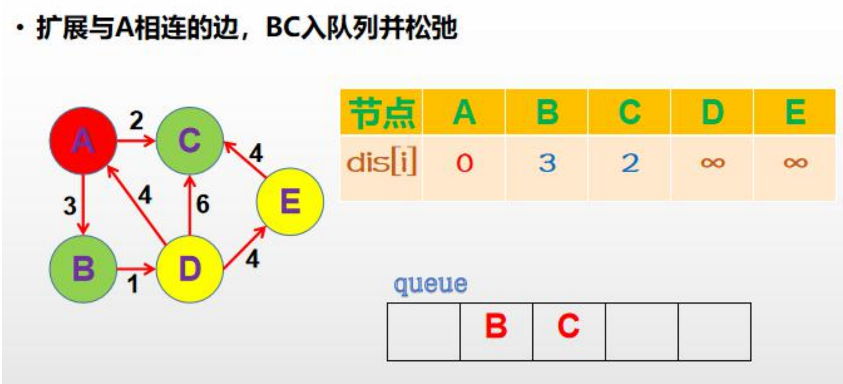
2. 取出队头节点 x ，扫描它的所有出边 (x,y,z) ，若 $dist[y] > dist[x] + z$ ，则使用 $dist[x] + z$ 更新 $dist[y]$ ，同时，若 y 不在队列，则把 y 入队.

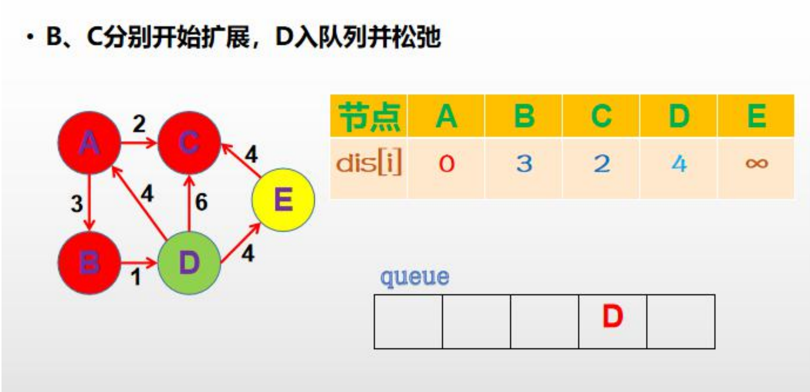
3. 重复上述操作，直到队列为空.

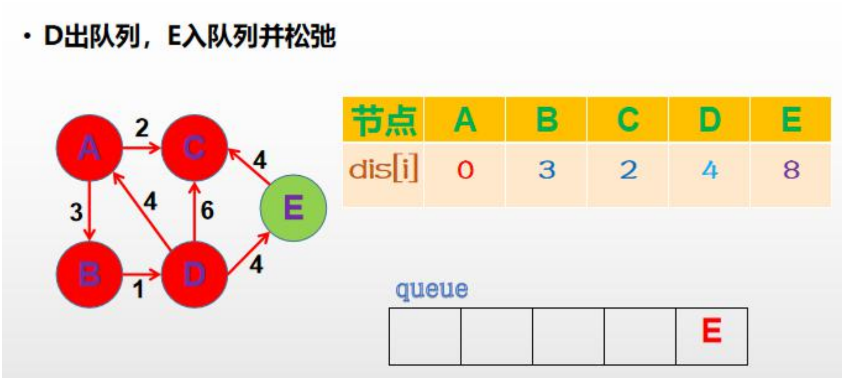
在任意时刻，该算法的队列都保存了待扩展的结点，每次入队相当于完成一次 $dist$ 数组的更新操作，使其中满足三角形不等式，一个结点可能会入队、出队多次. 最终，图中结点收敛到全部满足三角形不等式的状态. 在随机图上运行效率为 $O(km)$ 级别，其中 k 是一个较小的常数. 在特殊构造的图上，该算法很可能退化为 $O(nm)$.

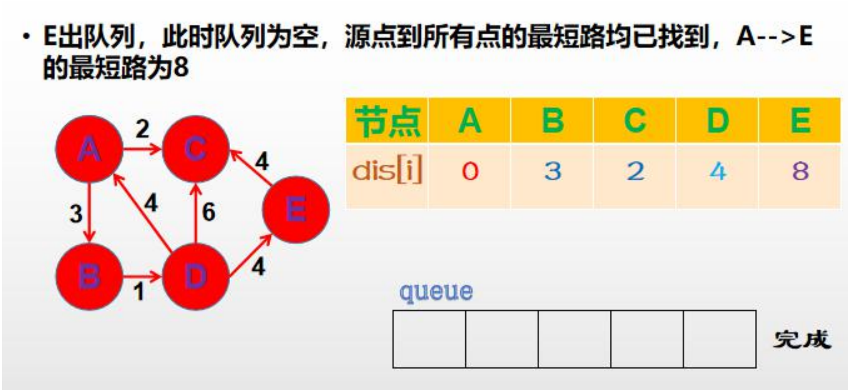
举例：











代码链接

```
#include<iostream>
#include<cstring>
#include<algorithm>
#include<queue>

#define MAXN 100010

using namespace std;

```

```
const int inf=0x3f3f3f3f;
int n,m;
int head[MAXN],ed[MAXN],val[MAXN],nex[MAXN],idx;
int dist[MAXN],vis[MAXN];

void add(int x,int y,int z){
    ed[idx]=y;
    val[idx]=z;
    nex[idx]=head[x];
    head[x]=idx++;
}

void spfa(int start){
    memset(dist,0x3f,sizeof(dist));
    memset(vis,0,sizeof(vis));//是否在队列中
    dist[start]=0;
    vis[start]=1;
    queue<int> q;
    q.push(start);
    while(q.size()>0){
        int temp=q.front();
        q.pop();
        vis[temp]=0;
        for(int i=head[temp];i!=-1;i=nex[i]){//扫描所有出边
            if(dist[ed[i]]>dist[temp]+val[i]){
                dist[ed[i]]=dist[temp]+val[i];
                if(vis[ed[i]]==0){
                    q.push(ed[i]);
                    vis[ed[i]]=1;
                }
            }
        }
    }
}

int main(){
    memset(head,-1,sizeof(head));
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y,z;
        scanf("%d %d %d",&x,&y,&z);
        add(x,y,z);
    }
    spfa(1);
```

```
    for(int i=1;i<=n;i++){
        if(dist[i]==inf)
            printf("-1\n");
        else printf("%d\n",dist[i]);
    }
    return 0;
}
```