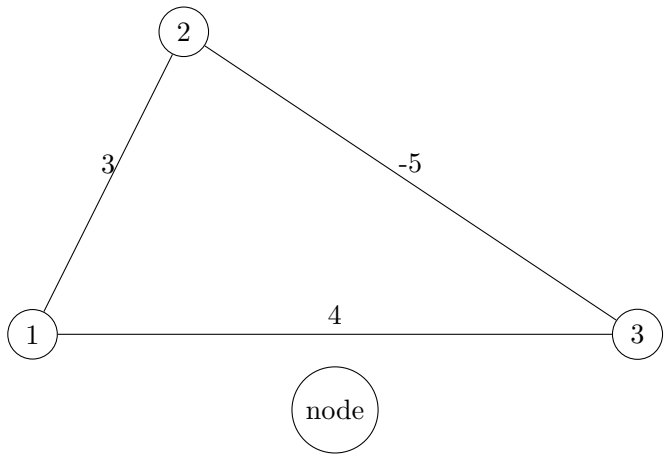
算法设计与分析课程作业作业 6

162140224 刘昊文

2023 年 5 月 2 日

1 *Dijkstra* 负边问题



Dijkstra 算法采用了贪心策略，假设源点为 1 点，此时 dist[] 数组为 0,4,3, 这时，我们会选择最近的 2 节点进行更新，此时 dist[3]>dist[2]+(-5), 所以 dist[3]=dist[2]-5 得到的 dist 为 0,3,-2, 算法得出 2 结点的最短路答案是 3，事实上，2 节点的最短路径应该为 1->3->2，为-2，但由于 2 被选择后就无法被更新，导致了算法在处理负权值时的错误, 原因就在于对于距离较大的一个点，加上负权值边后，可能距离小于已经确定的最近点，而由于最近点已被选择且无法更新，所以产生了错误。

2 钱币兑换问题

对于这道题，硬币的面值很特殊，都是 2 的 n 次幂，与兑换的值的二进制数存在关系，所以，对于需要兑换的值将其转换成二进制，如果第 t 位 (最低位算作第 1 位) 上的二进制值为 1 ，则存在一个 2^{t-1} 的硬币组成答案。

题目 1 的解答.

Algorithm 1: Coins of Money

Input: the number of money **mon**

Output: the number of conins **tot**

the coins array of money **ansCoins**

```
1 Function getCoins(mon)
2     nowCoins ← 1
      tot ← 0
      while mon ≥ 1 do
3         if mon%2 = 1 then
4             tot ← tot + 1
              ansCoins[tot] ← nowCoins
5         end
6     end
7     nowCoins ← nowCoins ∗ 2
      mon ← mon/2
      return tot, ansCoins
8 Function End
```

3 *Kruskal*

Kruskal 算法中利用了并查集避免成环，当边两个结点的并查集根节点相同时，则说明加入这条边会使图形成环，需要跳过这条边。

```
int find(int x)//并查集返回节点x的根节点并进行路径压缩
{
    if(fa[x]==x) return x;
    else return find(fa[x]);
}
struct node{//边集合
    int from,to,valve;
}ma[200005];
bool cmp(node a,node b)//排序使边权小的靠前
{
    return a.valve<b.valve;
}
int main(){
    int ans=0,tot=0,t1,t2,t3,n,m,s;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
        cin>>ma[i].from>>ma[i].to>>ma[i].valve;
    sort(ma+1,ma+1+m,cmp);
    for(int i=1;i<=n;i++)//初始化并查集
    fa[i]=i;
    for(int i=1;i<=m;i++)
    {
        int t1=find(ma[i].from),t2=find(ma[i].to);//找出当前处理边两个端点的并查集根节点
        if(t1==t2)//相等则说明如果相连则会称号，需要跳过这条边
        continue;
        fa[t1]=t2;//标记好并查集
        ans+=ma[i].valve;//更新答案
        tot++;//边数加一
        if(tot==n-1)//最小生成树边数为点数-1
        {
            cout<<ans;
            return 0;
        }
    }
    cout<<"不存在最小生成树";
    return 0;
}
```

两种算法运行结果如下:

```
10 15
1 2 4
1 3 6
1 4 8
2 3 3
2 5 7
3 4 5
3 6 9
4 6 2
4 7 10
5 6 8
5 8 11
6 7 4
6 8 6
7 9 12
8 10 13
Prim算法
最小生成树权值为:56
最小生成树各边(边结点1 边结点2 边权值)为
1 2 4
2 3 3
3 4 5
4 6 2
6 7 4
6 8 6
2 5 7
7 9 12
8 10 13

--------------------------------
```

图 1: *Prim* 算法

```
10 15
1 2 4
1 3 6
1 4 8
2 3 3
2 5 7
3 4 5
3 6 9
4 6 2
4 7 10
5 6 8
5 8 11
6 7 4
6 8 6
7 9 12
8 10 13
Kruskal算法
最小生成树权值为:56
最小生成树各边(边结点1 边结点2 边权值)为:
4 6 2
2 3 3
1 2 4
6 7 4
3 4 5
6 8 6
2 5 7
7 9 12
8 10 13

--------------------------------
```

图 2: *Kruskal* 算法

4 *Prim*

Prim 算法中利用了线性表 (链式前向星) 来存储图，利用了贪心策略，每次选择找出距离集合 S 最近的未处理的点进行处理并将其加入 S，该边进入最小生成树，循环操作直到边数为点数 -1 生成最小生成树。

```
struct node
{
    int to,val,next;
}ma[1000010];
void add(int a,int b,int c)//通过链式前向星加边
{
    ma[tot].to=b;
    ma[tot].val=c;
    ma[tot].next=head[a];
    head[a]=tot++;
}
int main()
{
    int n,m;
        cin>>n>>m;
        for(int i=1;i<=m;i++)
        {
                int t1,t2,t3;
                cin>>t1>>t2>>t3;
                add(t1,t2,t3);
                add(t2,t1,t3);
        }
    for(int i=2;i<=n;i++)//先将1节点到其他点距离标记为最大值
        dist[i]=2147483647;
    for(int i=head[1];i;i=ma[i].next)//标记好1点到其他点的距离
    {
        int y=ma[i].to,z=ma[i].val;
        dist[y]=min(dist[y],z);
    }
    now=1;
    vis[1]=true;//将1点标记好并将其加入到集合S中
    while(++num<=n-1)//最小生成树边数为点数-1
    {
        int minn=2147483647;
        now=-1;
        for(int i=1;i<=n;i++)//循环遍历找出距离集合S最近的点
```

```
        {
            if (! vis [ i]&&dist[ i]<minn)
            {
                minn=dist[ i ];
                now=i;
            }
        }
        if (now==-1) //最小生成树还未生成并且找不到最近的点了说明不存在最小生成树
            break;
        vis [now]=true;//将这个点标记好并加入到集合S中
        ans+=minn;//答案更新
        for ( int  i=head[now];i; i=ma[i].next)//更新与now相连且未被访问的点离集合S的距离，更新距离
        {
            int  y=ma[i].to,z=ma[i].val ;
            if (! vis [y]&&dist[y]>z)
                dist [y]=z;
        }
    }
    if (num==n)
    cout<<ans;
    else  cout<<"不存在最小生成树";
    return  0;
}
```

5 分发饼干

对于每个孩子，其能获得的饼干是那些尺寸大于等于他们胃口值的饼干，为了满足越多数量的孩子，所以对于每个孩子应该给他们分配最小的能满足他们胃口值的饼干，使更大的饼干能满足其他孩子的需求，这样就能使满足的孩子最多。

```
class Solution {
    public:
        int findContentChildren(vector<int>& g, vector<int>& s) {
            int n=g.size(),m=s.size();
            bool vis[30005];
            for(int i=0;i<m;i++)
            vis[i]=false;
            sort(s.begin(),s.end());//排序饼干尺寸，使每个人都尽量挑选更小的饼干
            int ans=0;
            for(int i=0;i<n;i++)
            {
                for(int j=0;j<m;j++)
                {
                    if(s[j]>=g[i]&&!vis[j])//挑选满足要求的最小的饼干
                    {
                        vis[j]=true;//标记已分配
                        ans++;
                        break;
                    }
                }
            }
            return ans;
        }
    };
```

分发饼干

提交记录

21 / 21 个通过测试用例
执行用时: 688 ms
内存消耗: 17 MB

状态: 通过
提交时间: 5 天前

6 无重叠区间

这道题要求出需要移除区间的最小数量，使剩余区间互不重叠，可以反过来思考，实际上就是使剩余区间最多，那么，现在就应该思考如何使选择的区间最多。可以利用贪心想法，对于待处理区域 L，为了使选择的区间最多，应该选择最早结束且开始时间晚于上一场结束时间的那一个会议，此时需要处理的区域为区域 L 减去会议右端点往左，此时剩下的区域，循环往复处理，最终可使得选择区间最多，需要移除区间的最小。

```
class Solution {
    public:
        int eraseOverlapIntervals(vector<vector<int>>&intervals) {
            sort(intervals.begin(),intervals.end(),[](const vector<int> &u, const vector<int> &v) {
                return u[1]<v[1]; //按照右端点升序排序
            });
            int n=intervals.size(),nowright=intervals[0][1]; //选择最早结束的会议
            int ans=1;
            for(int i=1;i<n;i++)
            {
                if(intervals[i][0]>=nowright)//选择开始时间晚于上一场结束时间且是最早结束的
                {
                    ans++;
                    nowright=intervals[i][1]; //更新结束时间
                }
            }
            return n-ans;
        }
};
```