# 问题概述

生成树问题是指在一个连通图中选择若干条边，是这些选择的边和所有图中的点组成一棵树的问题。而如果这个连通图是带权连通图，那么最小生成树指的就是所有生成树中边权之和最小的生成树。

# Prim算法

Prim算法可以理解为“加点法”。算法的流程是，随意选择一个起始点$s$把图$G=\langle E,V \rangle$的点集$V$分为两个部分$S,T$即$S \bigcup T = V$且$S \cap T = \emptyset$初始情况下$S={s}$$T$即是剩下的点的集合。每次操作，我们选择连接两个集合的边中最短的那个边，将其加入我们选择的生成树，并把这条边所连接的那个不在$S$中的点从$T$中移动到$S$中。当我们选择了$n-1$条边后就得到了最小生成树。

我们会发现Prim算法的流程于Dijkstra算法的流程十分相似，事实上，朴素Prim算法的时间复杂度也是$O(V^2)$并且它也可以用二叉堆来优化，优化后的时间复杂度为$O(ElogV)$此外Prim算法还可以用斐波那契堆来优化，优化后的时间复杂度为$O(E+VlogV)$

# Kruskal算法

Kruskal算法可以理解为“加边法”。先将图中所有的边从小到大排序，然后从小往大尝试添加，如果一条边的两端不在同一个集合就将其加入，这一过程需要使用到并查集，当加入$n-1$条边后就得到了原图的最小生成树。

显然Kruskal算法的时间复杂度为$O(ElogE)$

# 分析及证明

两种最小生成树算法都运用了贪心的策略，但是贪心策略不同，时间复杂度也有区别。朴素Prim算法更适合点相对于边来说较少的图，例如边数接近完全图的连通图。优化后的Prim算法和Kruskal算法都更适合稀疏图，即边没有比点多很多的图。

两种算法的严格证明都略为复杂，可以在**算法导论**的第23章找到，简略的证明可以看这篇博客

# 例题

## 洛谷P3366——最小生成树模板题

### 题目大意

就是个模板题

**解题思路**

两种算法均可，但是就这道题的数据来说Kruskal更优

**代码实现**

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;

struct edge{
    int f, t, w;
}E[200010];
bool vis[5010];
int father[5010], level[5010];
int n, m, ans = 0;

inline int read()
{
    char ch;
    int a = 0;
    bool flag = 0;
    while(!((ch = getchar()) >= '0' && ch <= '9') || (ch == '-')));
    if(ch == '-')
        flag = 1;
    else
    {
        a = a * 10;
        a += ch - '0';
    }
    while(((ch = getchar()) >= '0' && ch <= '9'))
    {
        a = a * 10;
        a += ch - '0';
    }
    if(flag == 1)
        a = -a;
    return a;
}

inline bool cmp(edge a, edge b)
{
    return a.w < b.w;
}

inline int find(int x)
{
```

```
    if(father[x] != x)
        father[x] = find(father[x]);
    return father[x];
}

inline bool Union(int x, int y)
{
    int fx = find(x), fy = find(y);
    if(fx == fy) return 0;
    if(level[fx] > level[fy])
        father[fy] = fx;
    else
    {
        father[fx] = fy;
        if(level[fx] == level[fy])
            level[fy]++;
    }
    return 1;
}

inline void init()
{
    for(int i = 1;i <= n;++i)
        father[i] = i;
}

int main()
{
    n = read();m = read();
    for(int i = 0;i < m;++i)
    {
        E[i].f = read();
        E[i].t = read();
        E[i].w = read();
    }
    init();
    sort(E, E + m, cmp);
    int flag = 0;
    for(int i = 0;i < m;++i)
    {
        if(Union(E[i].f, E[i].t) == 1)
        {
            ans += E[i].w;
            flag++;
        }
        if(flag == n - 1)
            break;
    }
    printf("%d\n", ans);
    return 0;
```

```
}
```

## Codeforces597Div2-D

### 题目大意

给出$n$个二维平面上的城市，现在需要给所有城市通电，在城市$i$建立发电站需要花费代价$c_i$在城市$i,j$之间连接电缆需要花费代价$(k_i + k_j) \times d$其中$d$为两个城市间的曼哈顿距离，最小化花费。

### 解题思路

先连边，然后建立超级源点和每个城市连权值为$c_i$的边，求最小生成树。

### 代码实现

```
#include <map>
#include <set>
#include <ctime>
#include <cmath>
#include <stack>
#include <queue>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

const int maxn = 2100;

int n, tot = 0;
int fa[maxn];
long long x[maxn], y[maxn];
long long c[maxn], k[maxn], out = 0, num = 0;
struct Edge{
    long long s, t, w;
}e[maxn * maxn];
vector<int> tc;
vector<Edge> tk;

inline long long dis(int i, int j)
{
    return abs(x[i] - x[j]) + abs(y[i] - y[j]);
}
```

```
inline Edge form(int a, int b, long long c)
{
    Edge ret;
    ret.s = a;
    ret.t = b;
    ret.w = c;
    return ret;
}

inline void init()
{
    for(int i = 1;i <= n;++i)
        fa[i] = i;
}

inline int find(int a)
{
    if(fa[a] == a)
        return a;
    return fa[a] = find(fa[a]);
}

inline bool cmp(Edge a, Edge b)
{
    return a.w < b.w;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1;i <= n;++i)
        scanf("%lld%lld", &x[i], &y[i]);
    for(int i = 1;i <= n;++i)
        scanf("%lld", &c[i]);
    for(int i = 1;i <= n;++i)
        scanf("%lld", &k[i]);

    for(int i = 1;i <= n;++i)
        for(int j = 1;j <= i - 1;++j)
            e[++tot] = form(i, j, dis(i, j) * (k[i] + k[j]));

    for(int i = 1;i <= n;++i)
        e[++tot] = form(n + 1, i, c[i]);

    sort(e + 1, e + tot + 1, cmp);

    init();
    for(int i = 1;i <= tot;++i)
    {
        int fs = find(e[i].s), ft = find(e[i].t);
```

```
        if(fs == ft)
            continue;
        fa[ft] = fs;
        if(e[i].s == n + 1)
            tc.push_back(e[i].t);
        else
            tk.push_back(e[i]);
        out += e[i].w;
        ++num;
        if(num == n)
            break;
    }

    int C = tc.size(), K = tk.size();
    printf("%lld\n", out);
    printf("%d\n", C);
    for(int i = 0;i < C;++i)
        printf("%d ", tc[i]);
    printf("\n");
    printf("%d\n", K);
    for(int i = 0;i < K;++i)
        printf("%lld %lld\n", tk[i].s, tk[i].t);

    return 0;
}
```

**by MisakaTao 2020.5.2**