DSU&Kruskal&Prim

sine_and_cosine

ZUCC ACM Group

2023.01.11

Catalogue

Catalogue

概念

- 并查集是一种用于管理元素所属集合的数据结构，实现为一个森林，其中每棵树表示一个集合，树中的节点表示对应集合中的元素。

概念

- 并查集是一种用于管理元素所属集合的数据结构，实现为一个森林，其中每棵树表示一个集合，树中的节点表示对应集合中的元素。
- 考虑初始时有n个点，然后进行q次操作（$n \le 10^5, q \le 10^5$）,操作分为2种：

  操作1-merge：将两个点所在的集合合并为一个集合
  操作2-query：查询两个点是否属于同一集合

问题解决

考虑利用树维护每个集合，则根节点代表着这个集合，若两个点所在树的根节点相同则认为两个点在一个集合内，初始时每个点为一个集合，每棵树都只有一个结点。每次合并就将后者集合的根节点接到前者集合根节点上，作为其一个新的子节点

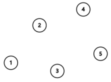


图: 初始状态

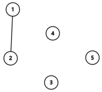


图: after merge(1,2)

图: after merge(3,5)

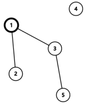


图: after merge(1,3)

```
const int N = 1e5 + 10;
int fa[N];
void init(int n) {
    for (int i = 1; i <= n; i++) fa[i] = i;
}
int find(int x) {
    while (fa[x] != x) x = fa[x];
    return x;
}
bool query(int x, int y) {
    return find(x) == find(y);
}
void merge(int x, int y) {
    int fx = find(x);
    int fy = find(y);
    if (fx != fy) {
        fa[fy] = fx;
    }
}
```

时间复杂度计算

- init复杂度为$O(n)$

时间复杂度计算

- init复杂度为$O(n)$
- query和merge的复杂度由find(x)决定，故并查集时间复杂度由find(x)决定，find(x)复杂度为$O($x的深度$)$

时间复杂度计算

- init复杂度为$O(n)$
- query和merge的复杂度由find(x)决定，故并查集时间复杂度由find(x)决定，find(x)复杂度为$O($x的深度$)$
- 最坏情况：不断将一个长链接在单个点下每次merge将导致链长度$+1$，即最深的点深度$+1$，经过$n-1$次合并最终最深深度为n

时间复杂度计算

- init复杂度为$O(n)$
- query和merge的复杂度由find(x)决定，故并查集时间复杂度由find(x)决定，find(x)复杂度为O(x的深度)
- 最坏情况：不断将一个长链接在单个点下每次merge将导致链长度$+1$，即最深的点深度$+1$，经过$n-1$次合并最终最深深度为n
- 故若每次都通过长链最低端的结点进行find操作，则第i次合并时find需要进行i次循环，最终$n-1$次合并所有结点后总时间为$1+2+3+\cdots+n-1=O(n^2)$，不能在1s内解决规模在10^5的问题

时间复杂度计算

- init复杂度为$O(n)$
- query和merge的复杂度由find(x)决定，故并查集时间复杂度由find(x)决定，find(x)复杂度为O(x的深度)
- 最坏情况：不断将一个长链接在单个点下每次merge将导致链长度+1，即最深的点深度+1，经过$n-1$次合并最终最深深度为n
- 故若每次都通过长链最低端的结点进行find操作，则第i次合并时find需要进行i次循环，最终$n-1$次合并所有结点后总时间为$1+2+3+\cdots+n-1=O(n^2)$，不能在1s内解决规模在10^5的问题
-

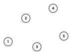


图: init

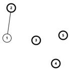


图: after merge(2,1)

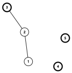


图: after merge(3,1)

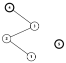


图: after merge(4,1)

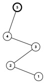


图: after merge(5,1)

优化

方法1.按秩合并

- 瓶颈在于深度的不断增加

- 瓶颈在于深度的不断增加
- 而合并时总是被挂到另一棵树上的树fy高度加一，而被挂上另一颗树的树fx高度不变

- 瓶颈在于深度的不断增加
- 而合并时总是被挂到另一棵树上的树fy高度加一，而被挂上另一颗树的树fx高度不变
- 因此可以尝试维护每棵树的大小，两颗树中总是选择较小的一棵树挂在较大树的根节点下

复杂度计算

fy高度加一，但同时由于$sz[fy] \le sz[fx]$，故合并后对于fy来说大小至少变为原来的2倍，因此每棵树最多被如此合并$\log_2 n$次，即高度最多加到$\log_2 n$，故最终单次find操作复杂度为$O(\log_2 n)$，n-1次合并复杂度为$O(n \log_2 n)$

```
int sz[N];
void merge(int x, int y) {
    int fx = find(x);
    int fy = find(y);
//  if (sz[fx] < sz[fy]) swap(fx, fy);
    if (fx != fy) {
        fa[fy] = fx;
        sz[fx] += sz[fy];
        sz[fy] = 0;
    }
}
```

- 瓶颈在于对较深结点的重复访问

- 瓶颈在于对较深结点的重复访问
- 而每次访问较深结点后真正有用的信息只有顶部根节点，被访问结点距离根节点越近则循环次数越少

优化

方法2.路径压缩

- 瓶颈在于对较深结点的重复访问
- 而每次访问较深结点后真正有用的信息只有顶部根节点，被访问结点距离根节点越近则循环次数越少
- 因此每次find操作时可以对遍历到的点都改变父节点为最后查找到的根节点，即直接将其挂在根节点上，以后访问这些结点后find的循环次数将大大减少

复杂度计算

通过构造二项树，m次询问可以把复杂度卡到$\Omega(m\log_{1+\frac{m}{n}} n)$

二项树构造：`https://www.cnblogs.com/meowww/p/6475952.html`

```
int find(int x) {
    int rt = x;
    while (fa[rt] != rt) rt = fa[rt];
    while (x != rt) {
        int tmp = fa[x];
        fa[x] = rt;
        x = tmp;
    }
    return rt;
}
```

使用递归压缩代码后：

```
int find(int x) {
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}
```

- 以上两种优化方法不冲突，可以一起使用

时间复杂度

按秩合并+路径压缩后m次操作的复杂度为$O(m\alpha(n))$
α是阿克曼函数的反函数，可以当成不超过5的常数。

```cpp
int fa[N], sz[N];
void init(int n) {
    for (int i = 1; i <= n; i++) {
        fa[i] = i, sz[i] = 1;
    }
}
int find(int x) {
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}
bool query(int x, int y) {
    return find(x) == find(y);
}
void merge(int x, int y) {
    int fx = find(x);
    int fy = find(y);
    if (sz[fx] < sz[fy]) swap(fx, fy);
    if (fx != fy) {
        fa[fy] = fx;
        sz[fx] += sz[fy];
        sz[fy] = 0;
    }
}
```

Catalogue

最小生成树（MST）概念

- 生成树：一个n个点m条边的图中,我们一定可以从中挑出$n-1$条边使n个点连通，即构成n个点的树。

最小生成树（MST）概念

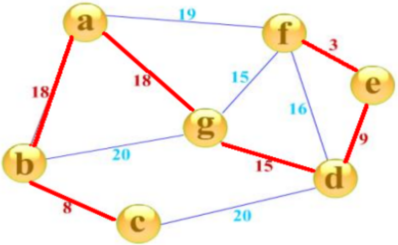
- 生成树：一个n个点m条边的图中,我们一定可以从中挑出$n-1$条边使n个点连通，即构成n个点的树。
- 当图中每条边都存在权重时,这时候我们从图中生成一棵树($n-1$ 条边)时,生成这棵树的总代价就是每条边的权重相加之和

最小生成树（MST）概念

- 生成树：一个n个点m条边的图中,我们一定可以从中挑出$n-1$条边使n个点连通，即构成n个点的树。
- 当图中每条边都存在权重时,这时候我们从图中生成一棵树($n-1$ 条边)时,生成这棵树的总代价就是每条边的权重相加之和
- 最小生成树：就是选择$n-1$条出来，连接所有的n个点。这$n-1$条边的边权之和是所有方案中最小的。

最小生成树（MST）概念

- 生成树：一个n个点m条边的图中,我们一定可以从中挑出$n-1$条边使n个点连通，即构成n个点的树。
- 当图中每条边都存在权重时,这时候我们从图中生成一棵树($n-1$ 条边)时,生成这棵树的总代价就是每条边的权重相加之和
- 最小生成树：就是选择$n-1$条出来，连接所有的n个点。这$n-1$条边的边权之和是所有方案中最小的。
- 显然一张图可能有很多生成树，也可能有很多最小生成树。



利用了贪心的思想。
连通块：无向图中相互连通的一些点称为处于同一个连通块中

- 将图的点边信息保存下来，并初始认为每一个点都是孤立的，分属于n个独立的集合。

Kruskal

利用了贪心的思想。
连通块：无向图中相互连通的一些点称为处于同一个连通块中

- 将图的点边信息保存下来，并初始认为每一个点都是孤立的，分属于n个独立的集合。
- 对所有边进行排序，按照权重升序排序

Kruskal

利用了贪心的思想。
连通块：无向图中相互连通的一些点称为处于同一个连通块中

- 将图的点边信息保存下来，并初始认为每一个点都是孤立的，分属于n个独立的集合。
- 对所有边进行排序，按照权重升序排序
- 依次遍历所有边，若当前边的所连接的两个点不在一个连通块内则将当前边加入最小生成树中，将两个连通块相连；如果这条边连接的两个点属于同一连通块，就跳过。

Kruskal

利用了贪心的思想。
连通块：无向图中相互连通的一些点称为处于同一个连通块中

- 将图的点边信息保存下来，并初始认为每一个点都是孤立的，分属于n个独立的集合。
- 对所有边进行排序，按照权重升序排序
- 依次遍历所有边，若当前边的所连接的两个点不在一个连通块内则将当前边加入最小生成树中，将两个连通块相连；如果这条边连接的两个点属于同一连通块，就跳过。
- 直到选取了n-1条边为止。

连通块的维护

- 是否可以把一个连通块内的点看做是一个集合，两个连通块相连是否等于集合的合并？
- 连通块可以用并查集维护

Code:

```cpp
int V,E;
struct edge{
    int u,v,cost;
    bool operator < (const edge e) const {
        return cost < e.cost;
    }
};
vector<edge>es;
int kruskal(){
    int ret=0;
    sort(es.begin(),es.end());
    init(V);
    for(auto e : es){
        if(!query(e.u,e.v)){
            merge(e.u,e.v);
            ret+=e.cost;
        }
    }
    return ret;
}
```

Kruskal证明

- 证明分为三步，证明了3个定理则证明了kruskal的正确性

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

定理1

对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

定理2

带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

定理3

kruskal产生的生成树T是一棵最小生成树

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明:
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2, e_1 \notin T_2$，$e_2 \notin T_1$，

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2$,$e_1 \notin T_2$，$e_2 \notin T_1$，
- 反证：若$w(e_1) \neq w(e_2)$,不妨假设有$w(e_1) < w(e_2)$

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2, e_1 \notin T_2$，$e_2 \notin T_1$，
- 反证：若$w(e_1) \neq w(e_2)$,不妨假设有$w(e_1) < w(e_2)$
- 则将e_1加入T_2后T_2产生一条回路

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2$,$e_1 \notin T_2$，$e_2 \notin T_1$，
- 反证：若$w(e_1) \neq w(e_2)$,不妨假设有$w(e_1) < w(e_2)$
- 则将e_1加入T_2后T_2产生一条回路
- 由T_2具有性质1得该回路中任意一条边应当有权值小于等于e_1

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2$,$e_1 \notin T_2$，$e_2 \notin T_1$，
- 反证：若$w(e_1) \neq w(e_2)$,不妨假设有$w(e_1) < w(e_2)$
- 则将e_1加入T_2后T_2产生一条回路
- 由T_2具有性质1得该回路中任意一条边应当有权值小于等于e_1
- 而e_2应当在该回路中，若不在，则该回路中的所有边都将属于T_1，则T_1中存在一条回路，与T_1是生成树相矛盾

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 基础；当$k = 1$时，
- 设$e_1 \in T_1$，$e_2 \in T_2$,$e_1 \notin T_2$，$e_2 \notin T_1$，
- 反证：若$w(e_1) \neq w(e_2)$,不妨假设有$w(e_1) < w(e_2)$
- 则将e_1加入T_2后T_2产生一条回路
- 由T_2具有性质1得该回路中任意一条边应当有权值小于等于e_1
- 而e_2应当在该回路中，若不在，则该回路中的所有边都将属于T_1，则T_1中存在一条回路，与T_1是生成树相矛盾
- 而若e_2在回路中，则该回路中应当有$w(e_1) \geq w(e_2)$，与假设相矛盾，故假设不成立，e_1和e_2权值应当相等

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边
- 则将e_2加入T_1后，由于T_1具有性质1，则T_1中产生一条回路，且这条回路中所有边边权应当$\leq w(e_2)$，且应当有一条边e_1使得$e_1 \in T_1$q且$e_1 \notin T_2$

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边
- 则将e_2加入T_1后，由于T_1具有性质1，则T_1中产生一条回路，且这条回路中所有边边权应当$\leq w(e_2)$，且应当有一条边e_1使得$e_1 \in T_1$q且$e_1 \notin T_2$
- 而将e_1加入T_2中得到$w(e_1) \geq w(e_2)$

Kruskal证明

定理1:对于一个图的两棵生成树$T1,T2$，若他们同时具有性质1，则$T1,T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1,T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边
- 则将e_2加入T_1后，由于T_1具有性质1，则T_1中产生一条回路，且这条回路中所有边边权应当$\leq w(e_2)$，且应当有一条边e_1使得$e_1 \in T_1$q且$e_1 \notin T_2$
- 而将e_1加入T_2中得到$w(e_1) \geq w(e_2)$
- 因此可得$w(e_1) = w(e_2)$

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明:
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边
- 则将e_2加入T_1后，由于T_1具有性质1，则T_1中产生一条回路，且这条回路中所有边边权应当$\leq w(e_2)$，且应当有一条边e_1使得$e_1 \in T_1$q且$e_1 \notin T_2$
- 而将e_1加入T_2中得到$w(e_1) \geq w(e_2)$
- 因此可得$w(e_1) = w(e_2)$
- 将T_1中的e_1删去，加入e_2得到新树T_3，则T_3与T_2只有$j-1$条边不同

Kruskal证明

定理1:对于一个图的两棵生成树$T1, T2$，若他们同时具有性质1，则$T1, T2$的边权和相等

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 数学归纳法证明：
- 令k为两个生成树不相同的边数
- 假设$1 < k < j$时，$T1, T2$权值和相同
- 则当$k = j$时，令e_2是属于T_2但不属于T_1的一条边
- 则将e_2加入T_1后，由于T_1具有性质1，则T_1中产生一条回路，且这条回路中所有边边权应当$\leq w(e_2)$，且应当有一条边e_1使得$e_1 \in T_1$q且$e_1 \notin T_2$
- 而将e_1加入T_2中得到$w(e_1) \geq w(e_2)$
- 因此可得$w(e_1) = w(e_2)$
- 将T_1中的e_1删去，加入e_2得到新树T_3，则T_3与T_2只有$j-1$条边不同
- 由假设可知，T_3与T_2权值和相等，故T_1与T_2权值和相等得证。

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$
- (2).再证明T具有性质1是T是最小生成树的充分条件

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$
- (2).再证明T具有性质1是T是最小生成树的充分条件
- 若T_1图的一棵最小生成树,则由(1)得T_1具有性质1

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$
- (2).再证明T具有性质1是T是最小生成树的充分条件
- 若T_1图的一棵最小生成树,则由(1)得T_1具有性质1
- 则由定理1 得T和T_1权值和相等

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$
- (2).再证明T具有性质1是T是最小生成树的充分条件
- 若T_1图的一棵最小生成树,则由(1)得T_1具有性质1
- 则由定理1 得T和T_1权值和相等
- 故T也是最小生成树

Kruskal证明

定理2:带权连通图中的某棵生成树T为图的最小生成树当且仅当它具有性质1

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- (1).首先证明T为最小生成树是T具有性质1的充分条件。
- 假设有一条边$e_g \in G$但$e_g \notin T$,则将e_g加入T后会产生回路
- 反证：假设回路中存在有一条边e_t使得$w(e_t) > w(e_g)$
- 则删除e_t加入e_g后T仍然是一课最小生成树，且其权值和比之前要小，与T是最小生成树矛盾
- 所以回路中的任意一条边权值应当都$\leq e_g$
- (2).再证明T具有性质1是T是最小生成树的充分条件
- 若T_1图的一棵最小生成树,则由(1)得T_1具有性质1
- 则由定理1 得T和T_1权值和相等
- 故T也是最小生成树
- 因此生成树T具有性质1是T为图的最小生成树的充要条件。

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2
- 令e_g连接的两点为a,b，则有a,b被P_2中除e_g的边相连

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2
- 令e_g连接的两点为a,b，则有a,b被P_2中除e_g的边相连
- 由于最终状态下有a,b被P_1中除e_g的边相连

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2
- 令e_g连接的两点为a,b，则有a,b被P_2中除e_g的边相连
- 由于最终状态下有a,b被P_1中除e_g的边相连
- 因此最终状态下有a,b被两条路径相连，与T是生成树相矛盾

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2
- 令e_g连接的两点为a, b，则有a, b被P_2中除e_g的边相连
- 由于最终状态下有a, b被P_1中除e_g的边相连
- 因此最终状态下有a, b被两条路径相连，与T是生成树相矛盾
- 故kruskal产生的生成树必然有加入任意一条未加入的边，其在其产生的回路中权值最大，即符合性质1

Kruskal证明

定理3:kruskal产生的生成树T是一棵最小生成树

性质1

对于一个图G的一课生成树T，若对于任意存在于图G中但不存在与生成树T中的一条边e，加入生成树的边集后形成一个回路，且该边e为该回路中权值最大的边，则认为该生成树T具有性质1

- 反证：假设kruskal产生生成树T后，存在一条边$e_g \in G$，但$e_g \notin T$，则将其加入T中将产生一个回路P_1，回路中存在一条边e_t使得$w(e_t) > w(e_g)$
- 则由于kruskal对边按边权排序，有对加入e_g的判断比e_t早
- 又由于e_g未加入T，则说明此时将e_g加入T中将产生一条回路P_2
- 令e_g连接的两点为a, b，则有a, b被P_2中除e_g的边相连
- 由于最终状态下有a, b被P_1中除e_g的边相连
- 因此最终状态下有a, b被两条路径相连，与T是生成树相矛盾
- 故kruskal产生的生成树必然有加入任意一条未加入的边，其在其产生的回路中权值最大，即符合性质1
- 由定理2得T为最小生成树，故kruskal产生的生成树T为最小生成树

Catalogue

Prim

prim算法

Prim 算法是另一种常见并且好写的最小生成树算法。该算法的基本思想是从一个结点开始，不断加点（而不是 Kruskal 算法的加边）。

- 从任意一个结点开始，将结点分成两类：已加入的，未加入的。

Prim

prim算法

Prim 算法是另一种常见并且好写的最小生成树算法。该算法的基本思想是从一个结点开始，不断加点（而不是 Kruskal 算法的加边）。

- 从任意一个结点开始，将结点分成两类：已加入的，未加入的。
- 每次从未加入的结点中，找一个与已加入的结点之间边权最小值最小的结点。

Prim

prim算法

Prim 算法是另一种常见并且好写的最小生成树算法。该算法的基本思想是从一个结点开始，不断加点（而不是 Kruskal 算法的加边）。

- 从任意一个结点开始，将结点分成两类：已加入的，未加入的。
- 每次从未加入的结点中，找一个与已加入的结点之间边权最小值最小的结点。
- 然后将这个结点加入，并连上那条边权最小的边。

prim算法

Prim 算法是另一种常见并且好写的最小生成树算法。该算法的基本思想是从一个结点开始，不断加点（而不是 Kruskal 算法的加边）。

- 从任意一个结点开始，将结点分成两类：已加入的，未加入的。
- 每次从未加入的结点中，找一个与已加入的结点之间边权最小值最小的结点。
- 然后将这个结点加入，并连上那条边权最小的边。
- 重$n-1$次即可。

Prim

prim算法

Prim 算法是另一种常见并且好写的最小生成树算法。该算法的基本思想是从一个结点开始，不断加点（而不是 Kruskal 算法的加边）。

- 从任意一个结点开始，将结点分成两类：已加入的，未加入的。
- 每次从未加入的结点中，找一个与已加入的结点之间边权最小值最小的结点。
- 然后将这个结点加入，并连上那条边权最小的边。
- 重$n-1$次即可。
- 其实跟 Dijkstra 算法一样，每次找到距离最小的一个点，可以暴力找也可以用堆维护。

Code:

```
#define INF 0x3f3f3f3f
#define pii pair<int,int>

struct v_e {
    int to;
    int cost;
};

vector<v_e> g[5050];
int mincost[5050];
int vis[5050];

int prim() {
    int ret = 0;
    memset(mincost, INF, sizeof mincost);
    memset(vis, 0, sizeof vis);
    priority_queue<pii, vector<pii>, greater<pii> > pq;
    pq.push(pii(0, 1));
    while (!pq.empty()) {
        pii p = pq.top();
        pq.pop();
        if (vis[p.second]) continue;
        vis[p.second] = 1;
        ret += p.first;
        for (auto x: g[p.second]) {
            if (mincost[x.to] > x.cost) {
                mincost[x.to] = x.cost;
                pq.push(pii(mincost[x.to], x.to));
            }
        }
    }
    return ret;
}
```

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。
- 否则考虑$T+e$的环上另一条可以加入当前边集的边f。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。
- 否则考虑$T+e$的环上另一条可以加入当前边集的边f。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。
- 否则考虑$T+e$的环上另一条可以加入当前边集的边f。
- 首先,f的权值一定不小于e的权值，否则就会选择f而不是e了。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。
- 否则考虑$T+e$的环上另一条可以加入当前边集的边f。
- 首先,f的权值一定不小于e的权值，否则就会选择f而不是e了。
- 然后，f的权值一定不大于e的权值，否则$T+e-f$就是一棵更小的生成树了。

prim证明

- 说明在每一步，都存在一棵最小生成树包含已选边集。
- 基础：只有一个结点的时候，显然成立。
- 归纳：如果某一步成立，当前边集为F，属于T这棵MST，接下来要加入边e。
- 如果e属于T，那么成立。
- 否则考虑$T+e$的环上另一条可以加入当前边集的边f。
- 首先,f的权值一定不小于e的权值，否则就会选择f而不是e了。
- 然后，f的权值一定不大于e的权值，否则$T+e-f$就是一棵更小的生成树了。
- 因此，e和f的权值相等，$T+e-f$也是一棵最小生成树，且包含了F。

例题

- DSU
  P3367 【模板】并查集
  P1197 [JSOI2008] 星球大战
  codeforces C.String Reconstruction
  codeforces G. MinOr Tree
- MST
  P3366 【模板】最小生成树
  codeforces J. Road Reform
  codeforces F. Make It Connected
  codeforces A. Gift