图论-Prufer序列

Prufer序列是什么？

Prufer序列，又称Prufer code。

对于一棵 $n(n \geq 2)$ 个节点的**标定树**（节点带编号），其Prufer序列是一个长度为 $n-2$ ，值域为 $[1, n]$ 的整数序列。

每棵树必定存在Prufer序列且唯一，每个Prufer序列对应的树也必定存在且唯一，即二者为双射关系。

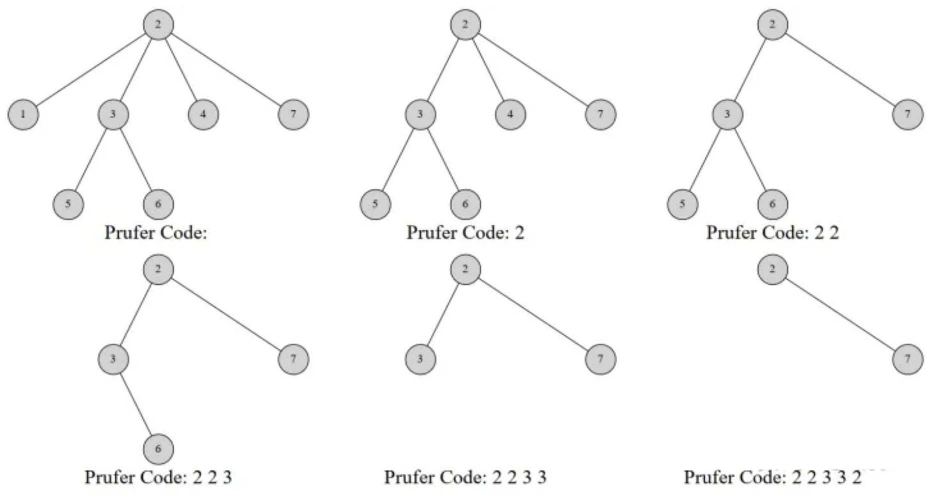
将树转化为Prufer序列

Prufer序列是这样从树转化的：

①从树上选择编号最小的叶子节点，序列的下一位为其父节点的编号。

②删去该叶子节点。

③重复①和②，直到树只剩下两个节点，此时序列的长度刚好为 $n-2$ 。



借用网上的一张图来展示一棵树化为Prufer序列的过程。

一个显而易见的做法是:

维护一个小根堆，将叶子节点依次丢入。

每次从堆顶取出一个叶子节点，将其父节点编号加入序列。

然后删去该叶子节点并减小其父节点的度，当其父节点度为1时，就变成了叶子节点，继续丢入小根堆。

重复这个过程，执行 $n-2$ 次即可。

总时间复杂度为 $O(nlogn)$ ，主要是维护小根堆的时间复杂度。

还有另外一种线性的做法：

①先找到编号最小的叶子节点，设其为 p 。

②将 p 的父节点 f 加入序列。

③若删去 p 节点后， f 节点变为叶子节点，且 $f<p$ ，则此时可以立即将 f 作为新选择的叶子节点进行操作。因为 p 已经是之前最小的叶子节点了， f 比其更小，所以删去 p 后 f 就变成了最小，可以略去这一步直接选择它。

④一直执行③直到不满足条件，此时将 p 自增，直到找到下一个还没删除的叶子节点。

p 最多自增 $n-2$ 次，操作③的执行次数最多也是 $n-2$ 次，故总的时间复杂度为 $O(n)$ 。

参考代码：

```
int degree[N],fa[N],prufer[N];
int p;
for(int i=1;i<=n;i++)
    if(degree[i]==1)
    {
        p=i;
        break;
    }//找到编号最小的叶子节点
int cnt=0;
int leaf=p;//leaf为当前选择要删去的叶子节点
while(cnt<n-2)//直到构造完整个prufer序列为止
{
    int f=fa[leaf];//当前叶子节点的父节点
    prufer[++cnt]=f;//序列的下一位
    degree[f]--;
    if(degree[f]==1&&f<p) leaf=f;
    else
    {
        p++;
        while(degree[p]!=1) p++;
        leaf=p;
    }
}
```

Prufer序列的性质

由Prufer序列构造的过程，我们可以发现其具有两个显而易见的性质。

1）构造完后剩下的两个节点里，一定有一个是编号最大的节点。

2）对于一个 n 度的节点，其必定在序列中出现 $n-1$ 次，因为每次删去其子节点它都会出现一次，最后一次则是删除其本身。一次都未出现的是原树的叶子节点。

由Prufer序列重构树

既然Prufer序列与树是双射关系，则必然也能由Prufer序列来重构一棵唯一的标定树。

这个过程与树的序列化是十分类似的——

①选择编号最小的叶子节点（即未出现在序列中的节点），其父节点就是序列的第 i（i 初始为1）个元素。

②由性质可得，其父节点的度数为其出现次数+1。将该叶子节点删去，其父节点度数-1。若度数变成1，则父节点也成为叶子节点。

③将 i 加一，然后重复①和②，直到序列的每一个元素都使用完毕。

所以我们同样可以想出一种线性的方法：

①先找到编号最小的未出现在序列中的节点，其为叶子节点，设其为 p 。

②将 p 的父节点 f 设为序列里尚未使用的第一个元素。

③若删去 p 节点后， f 节点变为叶子节点，且 $f < p$ ，则此时可以立即将 f 作为新选择的叶子节点进行操作。因为 p 已经是之前最小的叶子节点了， f 比其更小，所以删去 p 后 f 就变成了最小，可以略去这一步直接选择它。

④一直执行③直到不满足条件，此时将 p 自增，直到找到下一个还没删除的叶子节点。

p 最多自增 $n-2$ 次，操作③的执行次数最多也是 $n-2$ 次，故总的时间复杂度为 $O(n)$ 。

最后剩下两个节点，一个必然是节点 n ，将这两个节点连接即可。

参考代码：

```
int degree[N],fa[N],prufer[N],cnt[N];
for(int i=1;i<=n;i++) degree[i]=1;
for(int i=1;i<=n-2;i++) degree[prufer[i]]++;//度数为出现次数+1
int p;
for(int i=1;i<=n;i++)
    if(degree[i]==1)
    {
        p=i;
        break;
    }//找到编号最小的叶子节点
int cnt=0;
int leaf=p;//leaf为当前选择要删去的叶子节点
while(cnt<n-2)//直到构造完整个prufer序列为止
{
    int f;
    f=fa[leaf]=prufer[++cnt];//当前节点的父亲为序列未使用的第一位
    degree[f]--;
    if(degree[f]==1&&f<p) leaf=f;
    else
    {
        p++;
        while(degree[p]!=1) p++;
        leaf=p;
    }
}
fa[leaf]=n;//最后剩下两个节点，一个是n，把它们俩连接即可
```

Prufer序列的应用

Prufer序列一般是用于图论的组合计数问题。

1、无向完全图的不同生成树数：

若该完全图有 n 个节点，则任意一个长度为 $n-2$ 的Prufer序列都可以重构出其一棵生成树，且各不相同。又因为Prufer序列的值域在 $[1,n]$ ，故总数为 n^{n-2} 。

这就是有名的Cayley公式。

2、 n 个点的无根树计数：

同上问题。

3、n 个点的有根树计数：

对每棵无根树来说，每个点都可能是根，故总数为 $n^{n-2} \times n = n^{n-1}$ 。

4、 n 个点，每点度分别为 d_i 的无根树计数：

总排列为 $(n-2)!$ ，即 A_{n-2}^{n-2} 。

其中数字 i 出现了 $d_i - 1$ 次，故其重复的有 $(d_i - 1)!$ 种排列，即 $A_{d_i-1}^{d_i-1}$ 。

应当除去重复的，故总个数为 $\frac{(n-2)!}{\prod_{i=1}^{n}(d_i-1)!}$ 。

一些例题

洛谷 P6086：模板

LibreOJ 6395：城市地铁规划

CodeForces 156D：Clues

UVA 10843：Anne's game

CodeForces 1267F：Foolpruf Security