

# Prüfer序列

## Prüfer Sequence

(注：来源 <https://oi-wiki.org/graph/prufer/> | [https://github.com/e-maxx-eng/e-maxx-eng/blob/master/src/graph/pruefer\\_code.md](https://github.com/e-maxx-eng/e-maxx-eng/blob/master/src/graph/pruefer_code.md))

## 概述

**Prüfer 序列**可以将一个带标号  $n$  个结点的树用  $[1, n]$  中的  $n - 2$  个整数表示。可以把它理解为完全图的生成树与数列之间的双射。常用于组合计数问题上。

其定义是：每次选择一个编号最小的叶结点并删掉它，然后在序列中记录下它连接到的那个结点。重复  $n - 2$  次后就只剩下两个结点，算法结束。

注：不考虑  $n = 1$  特殊情况！

## 构建 Prüfer 序列

### $O(n \lg n)$ 方法

直接用堆模拟即可。

### 线性方法

注意到叶节点数量是单调不增的（要么删掉后不新增叶节点，要么删掉一个新增一个）。

设一个指针  $\text{ptr}$ ，始终保证  $[1, \text{ptr}]$  中最多只有一个叶节点（就是要删的那个节点）。也就是说， $[1, \text{ptr}]$  中的其他节点要么已经被删掉，要么不是叶节点；而我们还没有删大于  $\text{ptr}$  的那些节点。

在第一种情况下（删掉后不新增叶节点），我们只需要从  $\text{ptr} + 1$  开始往后找下一个要删的节点；在第二种情况下（删掉后新增一个叶节点），如果新增的节点小于  $\text{ptr}$ ，那么它就是我们要找的下一个节点；否则，从  $\text{ptr} + 1$  开始往后找下一个要删的节点。

鉴于  $\text{ptr}$  始终增加，该算法是  $O(n)$  的。

## Prüfer 序列的性质

---

- 构建完序列之后，剩下的两个点，其中一个一定是  $n$ ，另一个不确定。
- 每个节点在 **Prüfer** 序列中出现次数为度数减 1。因为对于每个点来说，它的度数每减 1，就会被加入序列一次，直到它的度数变成 1 然后被删掉。

## 从 Prüfer 序列建树

---

和构建 Prüfer 序列类似，注意到建树时，叶节点数量是单调不减的。

通过 Prüfer 序列我们可以得到每个节点的度数信息，于是每次找到度为 1 的点，把它与当前遍历的这个 Prüfer 序列中的点连起来，然后二者度数减 1。

同样设一个指针 ptr，分情况讨论。 $O(n)$

可以理解为，Prüfer 序列与带标号的无根树之间构成了双射。

## Cayley 公式

---

完全图  $K_n$  有  $n^{n-2}$  棵生成树。

使用 Prüfer 序列证明：任意一个长度为  $n - 2$  的在  $[1, n]$  之间取值的整数序列都可以通过 Prüfer 序列双射关系对应一颗生成树，一共  $n^{n-2}$  种。

## 图连通方案数

---

一个  $n$  个点  $m$  条边的带标号无向图有  $k$  个连通块，欲加  $k - 1$  条边使之连通，求方案数。

由于连通块内部不能连边，把每一个连通块看成一个点，这问题就和在完全图中搜寻生成树很相似了。不过，设第  $i$  个连通块有  $s_i$  个点，它们都有可能成为往外连的点，所以我们讨论一下度数情况：

设  $d_i$  表示第  $i$  个连通块的度数，根据度数的两倍等于边数，有

$$\sum_{i=1}^k d_i = 2k - 2$$

第  $i$  个连通块在长度为  $k - 2$  的 Prüfer 序列中出现  $d_i - 1$  次，所有连通块能构成的 Prüfer 序列就是**多项式系数**：

$$\binom{k-2}{d_1-1, d_2-1, \dots, d_k-1} = \frac{(k-2)!}{(d_1-1)!(d_2-1)!\dots(d_k-1)!}$$

又第  $i$  块的每个度数都有  $s_i$  种选择，所以对于某种  $d$  序列，方案数为：

$$\binom{k-2}{d_1-1, d_2-1, \dots, d_k-1} \prod_{i=1}^k s_i^{d_i}$$

现在枚举  $d$  序列得到答案：

$$\sum_{d_i \geq 1, \sum_{i=1}^k d_i = 2k-2} \binom{k-2}{d_1-1, d_2-1, \dots, d_k-1} \prod_{i=1}^k s_i^{d_i}$$

根据**多项式定理**：

$$(x_1 + \dots + x_m)^p = \sum_{c_i \geq 0, \sum_{i=1}^m c_i = p} \binom{p}{c_1, \dots, c_m} \prod_{i=1}^m x_i^{c_i}$$

作换元  $e_i = d_i - 1$ ，那么答案的式子改写为：

$$\begin{aligned} & \sum_{e_i \geq 0, \sum_{i=1}^k e_i = k-2} \binom{k-2}{e_1, e_2, \dots, e_k} \prod_{i=1}^k s_i^{e_i+1} \\ &= (s_1 + s_2 + \dots + s_k)^{k-2} \cdot \prod_{i=1}^k s_i \\ &= n^{k-2} \cdot \prod_{i=1}^k s_i \end{aligned}$$

## Code

```
1 namespace Prufer{
2     void getFa(int x, int f){ // ATT: initially dfs from n
3         fa[x] = f;
4         for(auto &to : edge[x]){
5             if(to == f) continue;
6             getFa(to, x);
7         }
8     }
```

```

9      vector<int> code(){
10          vector<int> res(n+5);
11          int ptr = 0;
12          while(deg[ptr] != 1)    ptr++;
13          int leaf = ptr;
14          for(int i = 1; i <= n - 2; i++){
15              int next = fa[leaf];
16              res[i] = next;
17              if(--deg[next] == 1 && next < ptr)    leaf = next;
18              else{
19                  ptr++; while(deg[ptr] != 1) ptr++;
20                  leaf = ptr;
21              }
22          }
23          return res;
24      }
25      vector< pair<int, int> > decode(vector<int> &code){
26          vector< pair<int, int> > edges(n+5);
27          for(int i = 1; i <= n; i++) deg[i] = 1;
28          for(int i = 1; i <= n - 2; i++) deg[code[i]]++;
29          int ptr = 0;
30          while(deg[ptr] != 1)    ptr++;
31          int leaf = ptr;
32          for(int i = 1; i <= n - 2; i++){
33              edges.emplace_back(mp(leaf, code[i])), fa[leaf] =
code[i];
34              if(--deg[code[i]] == 1 && code[i] < ptr)    leaf =
code[i];
35              else{
36                  ptr++; while(deg[ptr] != 1) ptr++;
37                  leaf = ptr;
38              }
39          }
40          edges.emplace_back(mp(leaf, n)), fa[leaf] = n;
41          return edges;
42      }
43
44  }

```