

Stern–Brocot 树与 Farey 序列

引入

本文介绍存储最简分数的数据结构以及其它相关概念。它们与 [连分数](#) 紧密相关，在算法竞赛中可以用于解决一系列数论问题，并可能作为某些题目的隐含背景出现。

Stern–Brocot 树

Stern–Brocot 树是一种维护分数的优雅的结构，包含所有不同的正有理数。这个结构分别由 Moritz Stern 在 1858 年和 Achille Brocot 在 1861 年独立发现。

构造

逐层构造

Stern–Brocot 树可以在迭代构造第 k 阶 Stern–Brocot 序列（Stern–Brocot sequence of order k ）的过程中得到。第 0 阶 Stern–Brocot 序列由两个简单的分数组成：

$$\frac{0}{1}, \frac{1}{0}.$$

此处的 $\frac{1}{0}$ 严格意义上并不算是有理分数，可以理解为表示 ∞ 的最简分数。

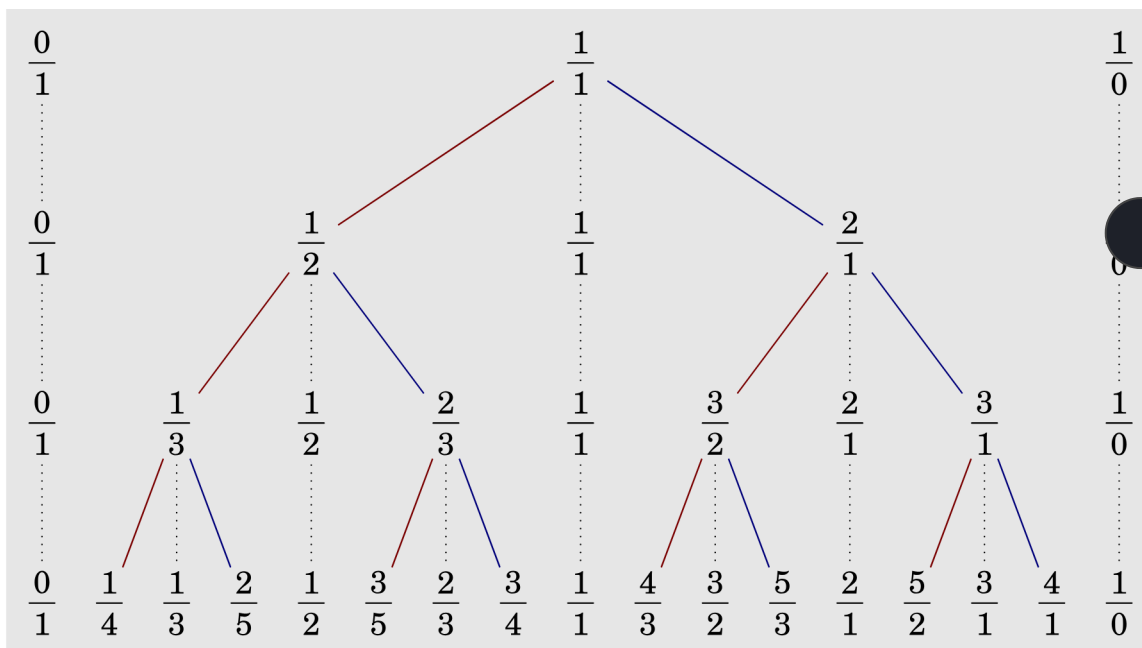
在第 k 阶 Stern–Brocot 序列相邻的两个分数 $\frac{a}{b}$ 和 $\frac{c}{d}$ 中间插入它们的中位分数（mediant）¹ $\frac{a+c}{b+d}$ ，就得到第 $k+1$ 阶 Stern–Brocot 序列。尽管中位分数的定义本身允许分数的约分，但是在 Stern–Brocot 树的构造中，只需要直接将分子和分母分别相加即可，而不必担心约分的问题。由此，可以迭代地构造出所有阶的 Stern–Brocot 序列。前几次迭代的结果如下：

$$\frac{0}{1}, \frac{1}{1}, \frac{1}{0}$$

$$\frac{0}{1}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{1}{0}$$

$$\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}, \frac{3}{2}, \frac{2}{1}, \frac{3}{1}, \frac{1}{0}$$

将每次迭代中新添加的分数连结成树状结构，就得到了 Stern–Brocot 树。如下图所示：



第 k 阶 Stern-Brocot 序列，不计左右端点，就是深度为 $k - 1$ 的 Stern-Brocot 树的中序遍历。

三元组构造

另一种等价的构造方式是以三元组

$$\left(\frac{0}{1}, \frac{1}{1}, \frac{1}{0} \right)$$

作为根节点，且在每个节点

$$\left(\frac{a}{b}, \frac{p}{q}, \frac{c}{d} \right)$$

后都分别添加

$$\left(\frac{a}{b}, \frac{a+p}{b+q}, \frac{p}{q} \right), \left(\frac{p}{q}, \frac{p+c}{q+d}, \frac{c}{d} \right)$$

为左右子节点，就可以构造出整个 Stern-Brocot 树。Stern-Brocot 树的每个节点记录的三元组中，实际存储的分数是位于三元组中间的分数 $\frac{p}{q}$ ，而左右两个分数 $\frac{a}{b}$ 和 $\frac{c}{d}$ 是更早就出现过的分数。而且，考虑前一种构造就会发现，分数 $\frac{p}{q}$ 正是通过插入 $\frac{a}{b}$ 和 $\frac{c}{d}$ 的中位分数得到的。

矩阵表示与 Stern-Brocot 数系

三元组构造意味着每个 Stern-Brocot 树上的节点都对应着一个矩阵

$$S = \begin{pmatrix} b & d \\ a & c \end{pmatrix}.$$

Stern-Brocot 树的根节点是单位矩阵 I ，而向左子节点移动和向右子节点移动则分别对应将当前节点矩阵右乘以矩阵

$$L = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, R = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

每个节点实际对应的分数就是 $f(S) = \frac{a+c}{b+d}$ 。每个节点的矩阵 S 都可以写成一系列矩阵 L 和 R 的乘积，这也可以理解为由 L 和 R 构成的字符串，表示了从根节点到达它的路径。将所有正有理数唯一地表示为这样的字符串，这可以看作得到了正有理数的一种表示，故而也称作 **Stern–Brocot 数系** (Stern–Brocot number system)。

建树实现

建树算法只需要模拟上述过程即可。下面是对前 n 层的 Stern–Brocot 树做中序遍历的代码。

建树

C++

```
1 // In-Order Transversal of Stern-Brocot Tree till Layer N.
2 void build(int n, int a = 0, int b = 1, int c = 1, int d = 0, int
3 level = 1) {
4     if (level > n) return; // Only first n layers.
5     int x = a + c, y = b + d;
6     build(n, a, b, x, y, level + 1);
7     std::cout << x << '/' << y << ' '; // Output the current
8     fraction.
9     build(n, x, y, c, d, level + 1);
10 }
```

Python

```
1 # In-Order Transversal of Stern-Brocot Tree till Layer N.
2 def build(n, a=0, b=1, c=1, d=0, level=1):
3     if level > n:
4         return
5     x, y = a + c, b + d
6     build(n, a, b, x, y, level + 1)
7     print(f"{x}/{y}", end=" ")
8     build(n, x, y, c, d, level + 1)
```

建树算法的复杂度是 $O(n^2)$ 的。

性质

接下来讨论 Stern–Brocot 树的性质。简而言之，Stern–Brocot 树是包含所有正的最简有理分数的 **二叉搜索树**，它也是分子和分母的 **堆**，也是分母和分子构成的二元组的 **笛卡尔树**。如果考虑前文的三元组构造中的左右端点形成的区间，则 Stern–Brocot 树也可以看作是 $[0, \infty]$ 上的 **线段树**。这些陈述可以通过如下的三条基本性质导出：

单调性

在上面的构造中，每一层的分数都是单调递增的。为此只需要归纳证明。因为如果 $\frac{a}{b} < \frac{c}{d}$ ，那么必然有

$$\frac{a}{b} < \frac{a+c}{b+d} < \frac{c}{d}.$$

这一点可以通过消去不等式的分母得出。归纳起点是 $\frac{0}{1} < \frac{1}{0}$ ，单调性也显然成立。

最简性

在上面的构造中，每个分数都是最简分数。为此同样需要归纳证明上面的构造中，每一层相邻的分数 $\frac{a}{b}$ 和 $\frac{c}{d}$ 都满足等式

$$bc - ad = \det \begin{pmatrix} b & d \\ a & c \end{pmatrix} = 1.$$

根节点处是单位矩阵，这显然成立。向下移动时，乘以的矩阵 L 和 R 的行列式都是 1，根据 [行列式](#) 的性质可知，在下一层同样成立

$$\det \begin{pmatrix} b & b+d \\ a & a+c \end{pmatrix} = \det \begin{pmatrix} b+d & d \\ a+c & c \end{pmatrix} = 1.$$

对于归纳起点 $\frac{0}{1}$ 和 $\frac{1}{0}$ 这也是显然的。由此，根据 [裴蜀定理](#)，必然可知每个分数的分子和分母都是互素的，即所有分数都是最简分数。

完全性

最后，需要说明 Stern-Brocot 树包括了所有正的最简分数。因为前两条性质已经说明 Stern-Brocot 树是二叉搜索树，而任何正的最简分数 $\frac{p}{q}$ 必然位于 $\frac{0}{1}$ 和 $\frac{1}{0}$ 之间，根据二叉搜索树上做搜索的方法，二叉搜索树上没有 $\frac{p}{q}$ 的唯一可能性就是搜索过程无限长。这是不可能的。

假设现在已经知道

$$\frac{a}{b} < \frac{p}{q} < \frac{c}{d},$$

那么必然有

$$bp - aq \geq 1, cq - dp \geq 1.$$

将两个不等式分别乘以 $(c+d)$ 和 $(a+b)$ ，就能够得到

$$(c+d)(bp - aq) + (a+b)(cq - dp) \geq a + b + c + d.$$

利用前面已经说明的等式 $bc - ad = 1$ 可知

$$p + q \geq a + b + c + d.$$

每次搜索更深入一层的时候，等式右侧都严格地增加，而左侧保持不变，因此搜索必然在有限步内停止。

查找分数

在实际应用 Stern-Brocot 树的过程中，经常需要查询给定分数在 Stern-Brocot 树上的位置。

朴素算法

因为 Stern-Brocot 是二叉搜索树，只需要通过比较当前分数和要寻找的分数的大小关系，就可以确定从根节点到给定分数的路径。将路径上向左子节点移动和向右子节点移动分别记作 L 和 R ，则每个路径就对应一个由 L 和 R 构成的字符串，这就是上文提到的有理数在 Stern-Brocot 数系中的表示。求得到达某个有理数的路径的过程就相当于求得该有理数在 Stern-Brocot 数系中的表示。

朴素的分数查找算法的实现如下：

朴素分数查找

C++

```
1 // Locate a given fraction in the Stern-Brocot tree.
2 std::string find(int x, int y, int a = 0, int b = 1, int c = 1,
3 int d = 0) {
4     int m = a + c, n = b + d;
5     if (x == m && y == n) return "";
6     if (x * n < y * m) {
7         return 'L' + find(x, y, a, b, m, n);
8     } else {
9         return 'R' + find(x, y, m, n, c, d);
10    }
}
```

Python

```
1 # Locate a given fraction in the Stern-Brocot tree.
2 def find(x, y, a=0, b=1, c=1, d=0):
3     m = a + c
4     n = b + d
5     if x == m and y == n:
6         return ""
7     if x * n < y * m:
8         return "L" + find(x, y, a, b, m, n)
9     else:
10        return "R" + find(x, y, m, n, c, d)
```

算法的复杂度是 $O(p + q)$ 的，因此在算法竞赛中并不实用。

在 Stern-Brocot 数系中，每个正的无理数都对应着唯一的无限长的字符串。可以使用同样的算法构造出这个字符串。这个无限长的字符串的每个前缀都对应着一个有理的最简分数。将这些最简分数排成一列，数列中的分数的分母是严格递增的，而数列的极限就是该无理数。因此，

Stern-Brocot 树可以用于找到某个无理数的任意精度的有理逼近。但是，应当注意的是，这个有理数数列和无理数之间的差距并非严格递减的。对于有理逼近的严格理论，应当参考连分数页面的 [丢番图逼近](#) 一节。在实际应用 Stern-Brocot 树寻找某个实数在分母不超过某范围的最佳逼近时，最后应当注意比较左右区间端点到该实数的距离。

快速算法

查找分数的朴素算法效率并不高，但是经过简单的优化就可以得到 $O(\log(p + q))$ 的快速查找算法。优化的关键是将连续的 L 和 R 合并处理。

如果要查找的分数 $\frac{p}{q}$ 落入 $\frac{a}{b}$ 和 $\frac{c}{d}$ 之间，那么连续 t 次向右移动时，右侧边界保持不动，而左侧节点移动到 $\frac{a+tc}{b+td}$ 处；反过来，连续 t 次向左移动时，左侧边界保持不动，而右侧节点移动到 $\frac{ta+c}{tb+d}$ 处。因此，可以直接通过 $\frac{a+tc}{b+td} < \frac{p}{q}$ 或 $\frac{p}{q} < \frac{ta+c}{tb+d}$ 来确定向右和向左移动的次数。此处取严格不等号，是因为算法移动的是左右端点，而要查找的分数是作为最后得到的端点的中位分数出现的。

快速分数查找

C++

```
1 // Locate a given fraction in the Stern-Brocot tree.
2 auto find(int x, int y) {
3     std::vector<std::pair<int, char>> res;
4     int a = 0, b = 1, c = 1, d = 0;
5     bool right = true;
6     while (x != a + c || y != b + d) {
7         if (right) {
8             auto t = (b * x - a * y - 1) / (y * c - x * d);
9             res.emplace_back(t, 'R');
10            a += t * c;
11            b += t * d;
12        } else {
13            auto t = (y * c - x * d - 1) / (b * x - a * y);
14            res.emplace_back(t, 'L');
15            c += t * a;
16            d += t * b;
17        }
18        right ^= 1;
19    }
20    return res;
21 }
```

Python

```
1 # Locate a given fraction in the Stern-Brocot tree.
2 def find(x, y):
3     res = []
4     a, b, c, d = 0, 1, 1, 0
5     right = True
6     while x != a + c or y != b + d:
7         if right:
8             t = (b * x - a * y - 1) // (y * c - x * d)
9             res.append((t, "R"))
10            a += t * c
11            b += t * d
12        else:
13            t = (y * c - x * d - 1) // (b * x - a * y)
14            res.append((t, "L"))
15            c += t * a
16            d += t * b
17        right ^= 1
18    return res
```

当前查找算法需要在分数 $\frac{p}{q}$ 已知。如果目标分数未知，往往需要在每次向右或向左移动时，对移动次数进行倍增查找或者二分查找。此时，分数查找算法的复杂度是 $O(\log^2(p + q))$ 。

基于连分数的算法

对于分数已知的情形，可以利用连分数给出更为简单的算法。不妨假设首组移动是向右的；如果不然，则设首组向右移动的次數为零。向右、向左交替移动端点，将每组移动后的端点位置排列如下：

$$\frac{p_0}{q_0}, \frac{p_1}{q_1}, \frac{p_2}{q_2}, \dots, \frac{p_{n-2}}{q_{n-2}}, \frac{p_{n-1}}{q_{n-1}}, \frac{p_n}{q_n}.$$

其中，偶数组移动是向右的，故而记录的是左端点的位置；奇数组移动是向左的，故而记录的是右端点的位置。在这一列端点前面再添加两个端点

$$\frac{p_{-2}}{q_{-2}} = \frac{0}{1}, \frac{p_{-1}}{q_{-1}} = \frac{1}{0}.$$

设第 k 组移动的次數为 t_k ，那么根据上面得到的移动次數与端点位置之间的关系可知

$$\frac{p_k}{q_k} = \frac{t_k p_{k-1} + p_{k-2}}{t_k q_{k-1} + q_{k-2}}.$$

根据连分数的 [递推关系](#) 就可以知道，端点

$$\frac{p_k}{q_k} = [t_0, t_1, \dots, t_k].$$

最后得到的连分数是

$$\frac{p}{q} = \frac{p_k + p_{k-1}}{q_k + q_{k-1}} = [t_0, t_1, \dots, t_{n-1}, t_n, 1].$$

因此，在目标分数的末尾为一的 [连分数表示](#) 中，不计最后的一，前面的项就编码了 Stern-Brocot 树上自根节点到当前节点的路径。其中，偶数项（下标自 0 开始）是向右子节点的边，奇数项是向左子节点的边。

有理数的连分数表示可以通过辗转相除法求得，因此基于连分数表示的分数查找算法的复杂度是 $O(\log \min\{p, q\})$ 的。

基于连分数的分数查找

C++

```
1 // Locate a given fraction in the Stern-Brocot tree.
2 auto find(int x, int y) {
3     std::vector<std::pair<int, char>> res;
4     bool right = true;
5     while (y) {
6         res.emplace_back(x / y, right ? 'R' : 'L');
7         x %= y;
8         std::swap(x, y);
9         right ^= 1;
10    }
11    --res.back().first;
12    return res;
13 }
```

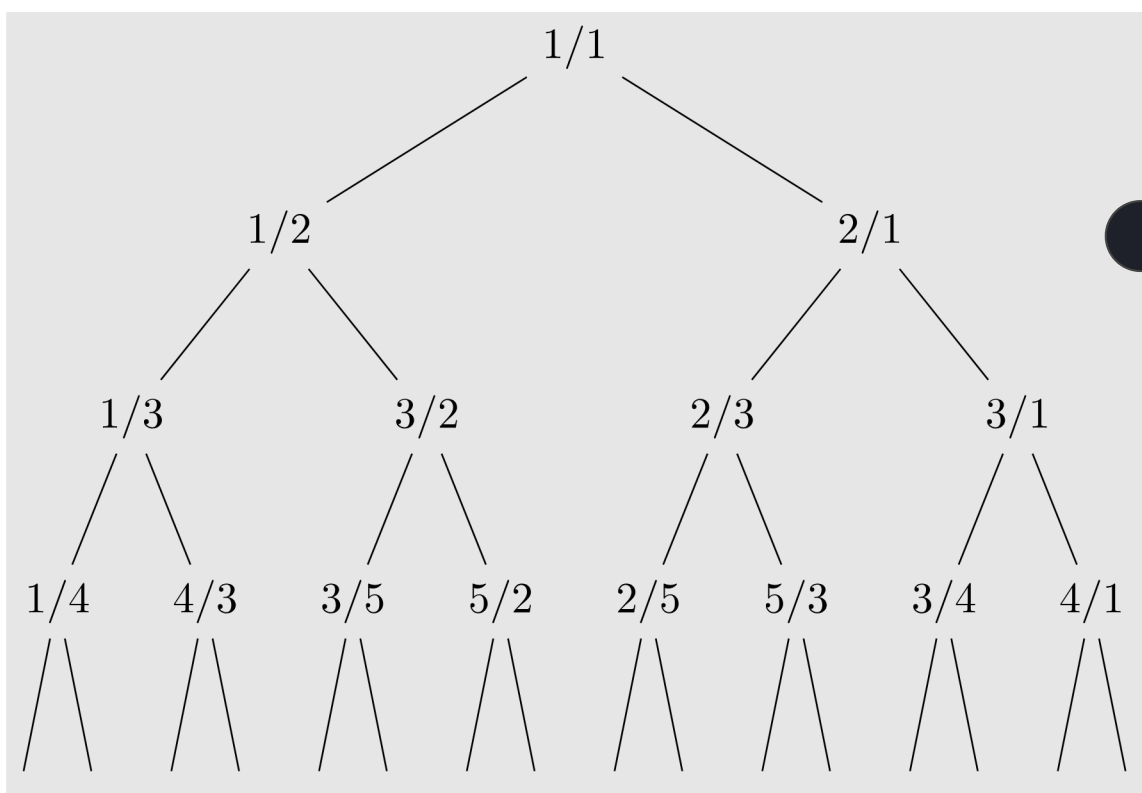
Python

```
1 # Locate a given fraction in the Stern-Brocot tree.
2 def find(x, y):
3     res = []
4     right = True
5     while y:
6         res.append([x // y, ("R" if right else "L")])
7         x, y = y, x % y
8         right ^= 1
9     res[-1][0] -= 1
10    return res
```

利用连分数表示，可以简单地表达出某个节点的父节点和子节点。对于节点 $[t_0, t_1, \dots, t_n, 1]$ ，它的父节点就是沿最后的移动方向少移动一步的节点：在 $t_k > 1$ 时，父节点是 $[t_0, t_1, \dots, t_{n-1}, 1]$ ；否则，父节点是 $[t_0, t_1, \dots, t_{n-1}, 1]$ 。它的两个子节点则分别是 $[t_0, t_1, \dots, t_n + 1, 1]$ 和 $[t_0, t_1, \dots, t_n, 1, 1]$ ；两个节点哪个是左子节点，哪个是右子节点，需要根据 n 的奇偶性判断。

Calkin-Wilf 树

另外一种更为简单的存储正有理分数的结构是 Calkin-Wilf 树。它通常如下所示：



树的根节点为 $\frac{1}{1}$ 。然后，对于分数 $\frac{p}{q}$ 所在的节点，其左右子节点分别为 $\frac{p}{p+q}$ 和 $\frac{p+q}{q}$ 。与 Stern-Brocot 树类似，它的每个分数都是最简分数，且包括全体正的最简分数各恰好一次。

与连分数的关系

与 Stern-Brocot 树不同，Calkin-Wilf 树不是二叉搜索树，因此不能用于二分查找有理数。

在 Calkin-Wilf 树中，当 $p > q$ 时，分数 $\frac{p}{q}$ 的父节点为 $\frac{p-q}{q}$ ；当 $p < q$ 时，为 $\frac{p}{q-p}$ 。对于第一种情形，自 $\frac{p}{q}$ 出发，它是父节点的右子节点，可以一直通过父节点的右边向上移动，直到分子不再大于分母为止，此时节点存储的分数是 $\frac{p \bmod q}{q}$ ，本组移动的次数是 $\left\lfloor \frac{p}{q} \right\rfloor$ ；对于第二种情形，它是父节点的左子节点，可以一直通过父节点的左边向上移动，直到分母不再大于分子为止，此时节点存储的分数是 $\frac{p}{q \bmod p}$ ，本组移动的次数是 $\left\lfloor \frac{q}{p} \right\rfloor$ 。

利用连分数的语言，设当前的节点存储的是某个连分数的余项 s_k ，则沿父节点的右边向上移动 $\lfloor s_k \rfloor$ 次，到达分数 $\frac{1}{s_{k+1}}$ ，随后沿父节点的左边移动 $\lfloor s_{k+1} \rfloor$ 次，到达分数 s_{k+2} 。因此，从节点 $s_0 = \frac{p}{q}$ 开始向上到达根节点 $\frac{1}{1}$ 的路径由连分数 $[t_0, t_1, \dots, t_n, 1]$ 编码：除去最后的 1，偶数项（下标自 0 开始）表示沿父节点的右边移动的次數，奇数项表示沿父节点的左边移动的次數。

对于分数 $\frac{p}{q} = [t_0, t_1, \dots, t_n, 1]$ 所在的节点，它的父节点有如下表示：

1. 当 $t_0 > 0$ 时，它的父节点为 $\frac{p-q}{q} = [t_0 - 1, t_1, \dots, t_n, 1]$ ；

2. 当 $t_0 = 0$ 且 $t_1 > 1$ 时, 它的父节点为 $\frac{p}{q-p} = [0, t_1 - 1, t_2, \dots, t_n, 1]$;

3. 当 $t_0 = 0$ 且 $t_1 = 1$ 时, 它的父节点为 $\frac{p}{q-p} = [t_2, t_3, \dots, t_n, 1]$ 。

反过来, 它的子节点则分别是 $\frac{p+q}{q} = [t_0 + 1, t_1, \dots, t_n, 1]$ 和 $\frac{p}{p+q} = [0, 1, t_0, t_1, \dots, t_n, 1]$ 。对于第二个子节点的连分数表示, 在 $t_0 = 0$ 时, 应当理解为 $[0, 1 + t_1, \dots, t_n, 1]$ 。

与 Stern-Brocot 树的关系

同样和连分数建立起联系, Stern-Brocot 树中路径上节点呈现的是渐近分数的递归关系, 而 Calkin-Wilf 树中路径上节点呈现的是余项的递归关系。同一个分数的连分数表示是一定的, 因此它在 Calkin-Wilf 树上到达根节点的路径的编码和 Stern-Brocot 树的根节点到它的路径的编码是完全一样的。但是, 由于路径的方向相反, 所以虽然 Stern-Brocot 树和 Calkin-Wilf 树同一层存储的分数是一样的, 但是位置并不相同。

如果对两个树分别做 [广度优先搜索](#) 并依次对节点编号, 根节点编号为 1, 那么编号为 v 的节点的左右子节点分别是 $2v$ 和 $2v + 1$ 。从编号的二进制表示来看, 除去起始的 1, 从高位至低位每个 1 均表示向右子节点移动, 每个 0 均表示向左子节点移动。Calkin-Wilf 树上, 连分数 $[t_0, t_1, \dots, t_n, 1]$ 表示的有理数所在节点的编号是

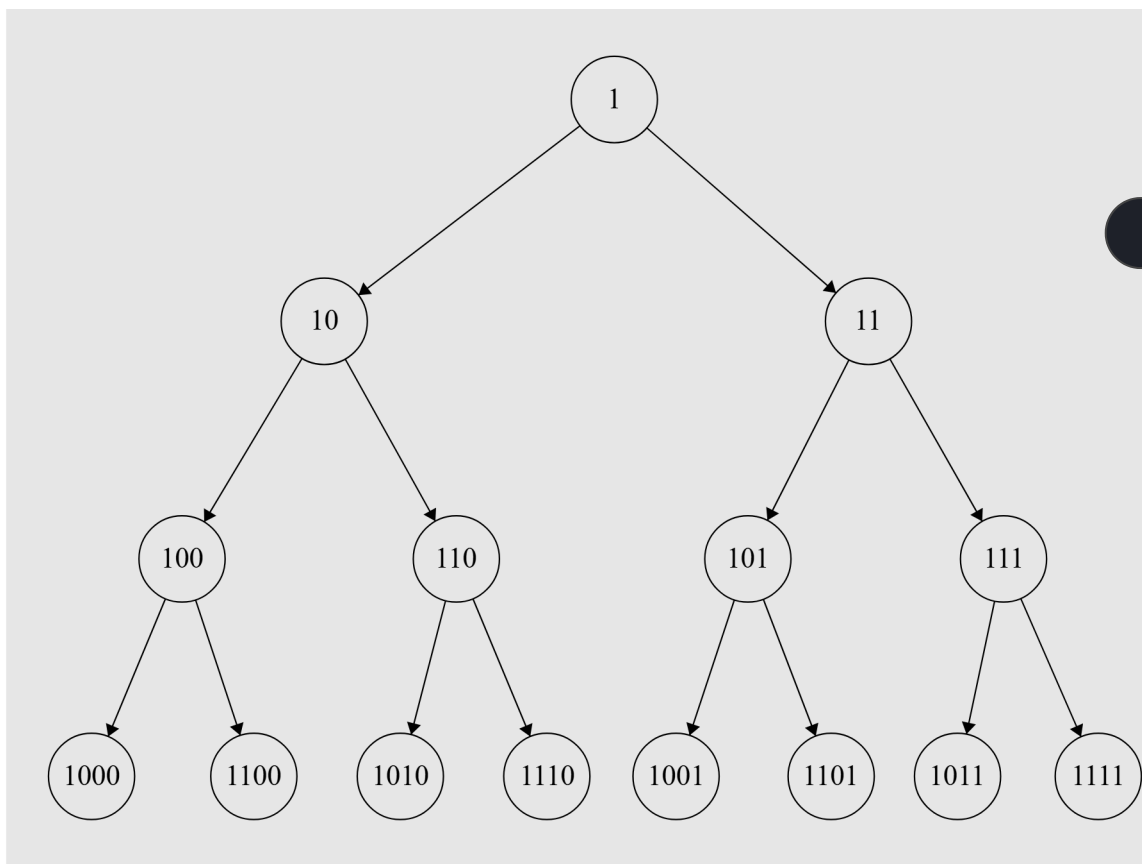
$$1 \dots \underbrace{0 \dots 01}_{t_3} \dots \underbrace{10}_{t_2} \dots \underbrace{01}_{t_1} \dots \underbrace{1}_{t_0}.$$

相应地, Stern-Brocot 树上, 连分数 $[t_0, t_1, \dots, t_n, 1]$ 表示的有理数所在节点的编号是

$$1 \underbrace{1 \dots 10}_{t_0} \dots \underbrace{01}_{t_1} \dots \underbrace{10}_{t_2} \dots \underbrace{0}_{t_3} \dots$$

删去初始的 1, 余下的二进制位组成的编号, 恰为同一层的顶点自左向右的编号 (自 0 开始)。此处的推导说明, Stern-Brocot 树和 Calkin-Wilf 树中同一层的分数的排列互为位逆序置换 (bit-reversal permutation), 即将下标的二进制位 (补齐起始的 0) 反转得到的 $0 \sim (2^k - 1)$ 上的置换。

正因如此, Stern-Brocot 树上的节点有时会按照对应的 Calkin-Wilf 树上的节点编号, 由此得到的编号如下图所示:



这个编号可以递归地构造：根节点编号为 1，每次移动到左子节点时，就将编号首位的 1 替换成 10，而移动到右子节点时，就将编号首位的 1 替换成 11。对该编号自右向左解读就可以得到自根节点到该节点的路径。

Stern 双原子序列

将 Calkin-Wilf 树中的所有分数按照广度优先搜索的编号排列，或将 Stern-Brocot 树中的所有分数按照上图所示的编号排列，就得到如下序列：

$$\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{3}{1}, \frac{1}{4}, \frac{4}{3}, \frac{3}{5}, \frac{5}{2}, \dots$$

利用 Calkin-Wilf 树的构造过程，可以证明，该序列中相邻的两个分数，前一个的分母必然等于后一个的分子。将分子单独列出，就得到 Stern 双原子序列（Stern diatomic sequence, [OEIS A002487](#)），也称为 Stern-Brocot 序列（Stern-Brocot sequence）。上述序列的编号从 1 开始，并补充规定第 0 个数是 0。

设 a_n 是 Stern 双原子序列中第 n 个数，那么，它满足如下递归关系：

$$\begin{aligned} a_{2n} &= a_n, \\ a_{2n+1} &= a_n + a_{n+1}. \end{aligned}$$

递归起点是 $a_0 = 0$ 和 $a_1 = 1$ 。要求得 Stern 双原子序列中 a_n 的值，直接利用递归关系复杂度为 $O(\log^2 n)$ ，并不优秀。更好的做法是，将它视为 Calkin-Wilf 树上编号为 n 的分数的分子，利用上文描述的基于连分数的递归关系求解，复杂度为 $O(\log n)$ 。

Farey 序列

Farey 序列与 Stern–Brocot 树有着极其相似的特征。记第 n 阶 **Farey 序列** (Farey sequence of order n) 为 F_n 。它表示把分母小于等于 n 的所有位于 $[0, 1]$ 之间的最简分数按大小顺序排列形成的序列：

$$F_1 = \left\{ \frac{0}{1}, \frac{1}{1} \right\}$$

$$F_2 = \left\{ \frac{0}{1}, \frac{1}{2}, \frac{1}{1} \right\}$$

$$F_3 = \left\{ \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1} \right\}$$

$$F_4 = \left\{ \frac{0}{1}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1} \right\}$$

$$F_5 = \left\{ \frac{0}{1}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1} \right\}$$

根据 Farey 序列的定义，它自然满足单调性、最简性和完全性。如上图所示， F_k 相较于 F_{k-1} 新添加的分数总是 F_{k-1} 中相邻分数的中位分数。

上文中构建 Stern–Brocot 树的算法同样适用于构建 Farey 序列。因为 Stern–Brocot 树中包括所有最简分数，因此只要将边界条件修改为对分母的限制就可以得到构造 Farey 序列的代码。可以将第 n 阶 Farey 序列 F_n 看作是第 $n - 1$ 阶 Stern–Brocot 序列的子序列。

构建 Farey 序列

C++

```
1 // Farey Sequence of Order N.
2 void build(int n, int a = 0, int b = 1, int c = 1, int d = 1,
3           bool init = true) {
4     int x = a + c, y = b + d;
5     if (y > n) return; // Only first n layers.
6     if (init) std::cout << "0/1 "; // Output 0/1;
7     build(n, a, b, x, y, false);
8     std::cout << x << '/' << y << ' '; // Output the current
9     fraction.
10    build(n, x, y, c, d, false);
11    if (init) std::cout << "1/1 "; // Output 1/1;
12 }
```

Python

```
1 # Farey Sequence of Order N.
2 def build(n, a=0, b=1, c=1, d=1, init=True):
3     x, y = a + c, b + d
4     if y > n:
5         return
6     if init:
7         print("0/1", end=" ")
8     build(n, a, b, x, y, False)
9     print(f"{x}/{y}", end=" ")
10    build(n, x, y, c, d, False)
11    if init:
12        print("1/1", end=" ")
```

直接构建 Farey 序列的复杂度是 $O(|F_n|) = O(n^2)$ 的。

序列长度与分数查找

Farey 序列的长度可以递归求出。相较于 F_{n-1} ，序列 F_n 多出来的分数的分母都是 n ，而分子不超过 n 且与 n 互素，因此有：

$$|F_n| = |F_{n-1}| + \varphi(n) = 1 + \sum_{k=1}^n \varphi(k).$$

此处的 $\varphi(n)$ 是 [欧拉函数](#)。该式利用 [线性筛](#) 可以 $O(n)$ 求出，而利用 [杜教筛](#) 可以将复杂度降到 $O(n^{2/3})$ 。

相较于直接求出序列的长度，更为常见的情景是需要求出序列 F_k 中某一分数 $r = \frac{p}{q}$ 的编号。这相当于求值

$$1 + \sum_{k=1}^n \sum_{i=1}^{\lfloor rk \rfloor} [i \perp k] = 1 + \sum_{d=1}^n \mu(d) \sum_{j=1}^{\lfloor n/d \rfloor} [rj].$$

要得到右式，应用了 [莫比乌斯反演](#)。线性筛和枚举因子结合，可以做到 $O(n)$ 预处理和 $O(n \log n)$ 询问；杜教筛和 [类欧几里得算法](#) 结合，可以做到 $O(n^{2/3})$ 预处理和 $O(\sqrt{n} \log n)$ 询问。

反过来，已知编号求解分数，需要对 $[0, 1]$ 之间的实数进行二分，或者在 Stern–Brocot 树上进行 [二分](#)。前者可能受到浮点数精度限制，对分数编号的询问次数为 $O(\log V)$ ，其中 V 是精度范围；后者不受浮点数精度限制，但是对分数编号的询问次数为 $O(\log^2 n)$ 。

Farey 邻项

如果分数 $\frac{a}{b}$ 和 $\frac{c}{d}$ 在某个 Farey 序列中相邻，就称它们为 **Farey 邻项** (Farey neighbors)，也称为 Farey 对 (Farey pair)。

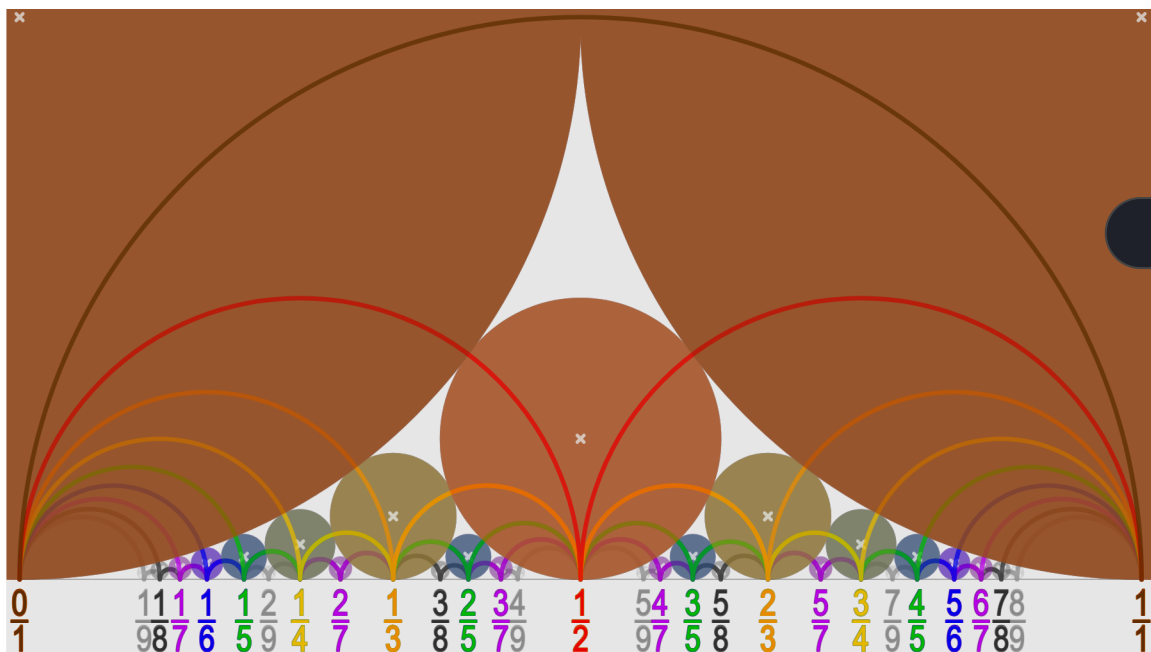
设 $\frac{a}{b} < \frac{c}{d}$ 。从 Farey 序列的构造过程可知，两个相邻分数中后加入的分数必然是另一个分数与它先前的邻项的中位分数，因此 Farey 邻项在某一阶 Stern–Brocot 序列中也相邻，根据 [最简性](#) 中证明的结论可知，必然也有

$$bc - ad = 1.$$

反过来，这也是两个最简真分数成为 Farey 邻项的充分条件。现在说明这一点。不妨设 $\frac{a}{b}$ 是两个分数中分母较大的，那么两个分数都会出现在 F_b 中。设 $\frac{e}{f}$ 是序列 F_b 中排在 $\frac{a}{b}$ 右侧的一项，按照已经说明的必要性可知， $be - af = 1$ 。但是，[线性同余方程](#) $bx - ay = 1$ 只有一组 $y \leq b$ 的正整数解，故而必然有 $(e, f) = (c, d)$ 。

其实，因为下一个会出现在两者之间的最简分数必然是 $\frac{a+c}{b+d}$ ，所以 $\frac{a}{b}$ 和 $\frac{c}{d}$ 在第 $\max\{b, d\}$ 到第 $(b+d-1)$ 阶 Farey 序列中都是相邻的。

分母不超过 9 的 Farey 邻项关系如下图所示：



图中的圆称为 **Ford 圆**：对于每个 $[0, 1]$ 内的最简分数 $\frac{p}{q}$ ，都以 $\left(\frac{p}{q}, \frac{1}{2q^2}\right)$ 为圆心、以 $\frac{1}{2q^2}$ 为半径绘制一个圆。图示说明，两个分数对应的 Ford 圆只能相切或相离，且两圆相切，当且仅当两分数是 Farey 邻项。而且，对于图中相切的任何两个圆，总存在唯一的第三个圆和两圆都相切。这第三个圆对应的分数就是两个圆对应的分数的中位分数。

要验证相切的圆总是对应着 Farey 邻项，可以直接计算两圆心的距离：

$$\left(\frac{a}{b} - \frac{c}{d}\right)^2 + \left(\frac{1}{2b^2} - \frac{1}{2d^2}\right)^2 = \left(\frac{1}{2b^2} + \frac{1}{2d^2}\right)^2 + \frac{(bc - ad)^2 - 1}{b^2d^2}.$$

因为两个最简分数并不相同，所以 $|bc - ad| \geq 1$ ，故而两圆只能相切或相离。而且，两圆相切，当且仅当 $|bc - ad| = 1$ ，这就等价于两分数是 Ford 邻项。

最后，要计算 Farey 邻项的数目。除了 $\left(\frac{0}{1}, \frac{1}{1}\right)$ ，其余 Farey 邻项的分母都不相同。不妨设 $\frac{p}{q}$ 是其中分母较大的那个，则另一个分数必然可以从二元一次不定方程

$$qx - py = \pm 1$$

中解得。两个方程各恰有一组满足 $y < q$ 的正整数解，分别对应位于 $\frac{p}{q}$ 左右两侧的邻项。因而，每个位于 $(0, 1)$ 内的真分数都有两个分母小于它的分母的 Farey 邻项，再加上 $\frac{0}{1}$ 和 $\frac{1}{1}$ ，这说明序列 F_n 中共计 $(2|F_n| - 3)$ 对 Farey 邻项。

当然，在这个过程中找到的分数 $\frac{a}{b} < \frac{c}{d}$ 就是在序列中插入 $\frac{p}{q}$ 时它的左右邻项。因此，它们本就是 Farey 邻项且 $\frac{p}{q}$ 是它们的中位分数。设 $\frac{p}{q} = [t_0, t_1, \dots, t_n, 1]$ ，则这两个分母更小的 Farey 邻项就分别是 $[t_0, t_1, \dots, t_n]$ 和 $[t_0, t_1, \dots, t_{n-1}]$ 。

要计算当前分数 $\frac{p}{q}$ 的其它 Farey 邻项，只需要利用 **扩展欧几里得算法** 求出所有符合条件的解即可。

递推关系

Farey 序列有一个简洁的递推关系，可以自左向右地顺序求出第 n 阶 Farey 序列的全部分数。

首先，前面的分析指出，在 F_n 中，新添加的项 $\frac{p}{n}$ 总是左右相邻两个分数的中位分数。其实，这个关系对于 F_n 中（除了两 endpoints）所有分数都成立。设 $\frac{a}{b} < \frac{p}{q} < \frac{c}{d}$ ，根据 Farey 邻项的充条件，总是有

$$bp - aq = 1 = cq - dp \iff \frac{p}{q} = \frac{a+c}{b+d}.$$

不过，对于一般的情形，分数 $\frac{a+c}{b+d}$ 可能需要约分。

利用这个观察就可以构建如下的递推关系。设 $\frac{a}{b}$ 和 $\frac{p}{q}$ 是已知的，要求出第三个分数 $\frac{c}{d}$ 的值。此时，存在 k 使得

$$a + c = kp, \quad b + d = kq$$

成立。因为差值

$$\frac{kp - a}{kq - b} - \frac{p}{q} = \frac{bp - aq}{q(kq - b)} = \frac{1}{q(kq - b)}$$

随着 k 增加而减小，且紧邻着 $\frac{p}{q}$ 的应该是所有满足 $kq - d \leq n$ 的分数中该差值最小的那个，因而，必然有

$$k = \left\lfloor \frac{n+b}{q} \right\rfloor.$$

可以验证这样得到的分数必然在 F_n 中。因此， F_n 中的分数的分子和分母满足递推关系：

$$\begin{aligned} p_k &= \left\lfloor \frac{n + q_{k-2}}{q_{k-1}} \right\rfloor p_{k-1} - p_{k-2}, \\ q_k &= \left\lfloor \frac{n + q_{k-2}}{q_{k-1}} \right\rfloor q_{k-1} - q_{k-2}. \end{aligned}$$

递推起点是 $(p_0, q_0) = (0, 1)$ 和 $(p_1, q_1) = (1, n)$ 。

习题

以本文材料为背景的题目：

- [LOJ 6685. 迷宫](#)
- [UVa 10077. The Stern-Brocot Number System](#)
- [Luogu P8058. \[BalkanOI2003\] Farey 序列](#)
- [UVa 12995. Farey Sequence](#)
- [UVa 10408. Farey Sequences](#)

- [UVa 12438. Farey Polygon](#)
- [AtCoder ARC123F. Insert Addition](#)

需要在 Stern–Brocot 树上作二分的题目：

- [AtCoder ABC333G. Nearest Fraction](#)
- [SPOJ DIVCNT1 - Counting Divisors](#)
- [SPOJ AFS3 - Amazing Factor Sequence \(hard\)](#)

参考资料与注释


- [Stern–Brocot tree - Wikipedia](#)
- [Calkin–Wilf tree - Wikipedia](#)
- [Farey sequence - Wikipedia](#)

本页面部分内容译自博文 [Дерево Штерна-Броко. Ряд Фарея](#) 与其英文翻译版 [The Stern–Brocot Tree and Farey Sequences](#)。其中俄文版版权协议为 **Public Domain + Leave a Link**；英文版版权协议为 **CC-BY-SA 4.0**。本页面另有部分内容译自博文 [Continued fractions](#)，版权协议为 **CC-BY-SA 4.0**。内容均有改动。

1. 译名来自张明尧、张凡翻译的《具体数学》第 4.5 节。 [←](#)

 本页面最近更新：2025/8/11 22:43:39，[更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者：[StudyingFather](#), [Enter-tainer](#), [sshwy](#), [Tiphereth-A](#), [c-forrest](#), [Great-designer](#), [H-J-Granger](#), [countercurrent-time](#), [NachtgeistW](#), [CCXXI](#), [diauweb](#), [iamtwz](#), [lr1d](#), [Xeonacid](#), [AngelKitty](#), [cjsoft](#), [Early0v0](#), [ezoixx130](#), [GekkaSaori](#), [Konano](#), [LovelyBuggies](#), [Makkiy](#), [Menci](#), [mgt](#), [minghu6](#), [P-Y-Y](#), [PotassiumWings](#), [SamZhangQingChuan](#), [Suyun514](#), [weiyong1024](#), [1804040636](#), [c8ef](#), [caijianhong](#), [Chrogeek](#), [CookiePieWw](#), [GavinZhengOI](#), [Gesrua](#), [HeRaNO](#), [jiang1997](#), [ksyx](#), [kxccc](#), [LLLMMKK](#), [lyccrius](#), [lychees](#), [Peanut-Tang](#), [shawlleyw](#), [shuzhouliu](#), [SukkaW](#), [TianKong-y](#), [untitledunrevised](#), [yanboishere](#), [yusancky](#)

© 本页面的全部内容 [在 CC BY-SA 4.0 和 SATA 协议之条款下](#) 提供，附加条款亦可能应用