位翻转置换（Bit Reversal Permutation）快速生成算法

起草：季文骢，修订：冯疏桐，版本：12

第一部分：概述

为方便表述，定义 N 位非负整数 x（其中 $N > 0$ 且为整数）的二进制表示法在本文中统一用以下记号表示：

$$x = (x_{N-1}x_{N-2}\dots x_1x_0)_2 = \sum_{i=0}^{N-1} x_i \cdot 2^i$$

对于所有满足条件 $0 \le i < 2^N$（其中 $N > 0$ 且为整数）的非负整数 i，定义 N 阶位翻转函数 $BRV_N(i) = BRV_N((i_{N-1}i_{N-2}\dots i_1i_0)_2) = (i_0i_1\dots i_{N-2}i_{N-1})_2$，那么 N 阶位翻转置换序列可定义为 $x(k) = BRV_N(k)$（$0 \le k < 2^N$）。以 3 阶位翻转置换序列为例，表 1 为该序列中的各项。

表 1：3 阶位翻转置换中的各项

k	$x(k) = BRV_3(k)$		k	$x(k) = BRV_3(k)$
0（000_2）	0（000_2）		4（100_2）	1（001_2）
1（001_2）	4（100_2）		5（101_2）	5（101_2）
2（010_2）	2（010_2）		6（110_2）	3（011_2）
3（011_2）	6（110_2）		7（111_2）	7（111_2）

本文提供了一种适用于在计算机中快速生成这样的 N 阶位翻转置换序列的算法及其正确性证明，最后给出了该算法在 FFT（快速傅立叶变换）中的应用。

第二部分：理论基础（若干定理的证明）

定理 1：对于两个 N 位非负整数 $a = (a_{N-1}a_{N-2}\dots a_1a_0)_2$、$b = (b_{N-1}b_{N-2}\dots b_1b_0)_2$（其中 $0 \le a, b < 2^N$），关系 $a > b$ 成立的充分必要条件是存在整数 $0 \le t < N$，使得 $a_t > b_t$ 且对于所有 $t < p < N$ 的整数 p 都有 $a_p = b_p$。

证明：

先证充分性，若数 a、b 的所有二进制位都对应相等，那么显然 $a = b$，排除掉这种情况。那么从最高位开始寻找，必然可以找到这么一个下标为 t 的位，使得对于所有的 $t < p < N$ 都有 $a_p = b_p$，且 $a_t \ne b_t$。

对两数作差：

$$a - b = \sum_{i=0}^{N-1}(a_i - b_i)\,2^i = \begin{cases} (a_t - b_t)2^t + \sum_{i=0}^{t-1}(a_i - b_i)\,2^i & (t \ne 0) \\ a_0 - b_0 & (t = 0) \end{cases}$$

当 $t = 0$ 时，若 $a_0 > b_0$，则必有 $a - b > 0$，即 $a > b$。

当 $t \ne 0$ 时，由等比数列求和公式，有：

$$\sum_{i=0}^{t-1}(a_i - b_i)2^i \ge \sum_{i=0}^{t-1}(-1)2^i$$

$$= -\frac{1 \cdot (1 - 2^t)}{1 - 2}$$

$$= 1 - 2^t$$

只要 $a_t > b_t$，必有 $a_t - b_t = 1$，于是：

$$\begin{aligned} a - b &= 2^t + \sum_{i=0}^{t-1} (a_i - b_i)\, 2^i \\ &\geq 2^t + 1 - 2^t \\ &= 1 > 0 \end{aligned}$$

充分性证毕。

再证必要性，显然有：

$$\begin{aligned} a - b &= \sum_{i=0}^{N-1} (a_i - b_i)\, 2^i \\ &= \sum_{i=0}^{t} (a_i - b_i)\, 2^i + \sum_{p=t+1}^{N-1} (a_p - b_p)\, 2^p \\ &= \sum_{i=0}^{t} (a_i - b_i)\, 2^i \\ &= 2^t + \sum_{i=0}^{t-1} (a_i - b_i)\, 2^i \\ &\geq 2^t + 1 - 2^t \\ &= 1 > 0 \end{aligned}$$

必要性也证毕。□

推论 1：若存在整数 $0 \leq t < N$，使得 $a_t > b_t$ 且对于所有 $t < p < N$ 的整数 p 都有 $a_p = b_p$，那么构造两个整数 $a' = (a_t a_{t-1} \dots a_1 a_0)_2$、$b' = (b_t b_{t-2} \dots b_1 b_0)_2$，必有 $a' - b' = a - b$，这意味着 a、b 之间的大小关系与 a′、b′ 之间的大小关系保持一致。

证明：

由定理 1 可以直接得出。□

在进行定理 2 的给出及证明之前，先定义 N 位二进制按位取反运算（下称“按位取反”）。对于任意 N 位非负整数 $a = (a_{N-1} a_{N-2} \dots a_1 a_0)_2$，其按位取反的结果定义为非负整数 $INV_N(a) = (\overline{a_{N-1}} \overline{a_{N-2}} \dots \overline{a_1} \overline{a_0})_2$，其中 $\overline{a_i} = 1 - a_i$（$0 \leq i < N$）。

定理 2：对于 N 位非负整数 $a = (a_{N-1} a_{N-2} \dots a_1 a_0)_2$ 有 $(2^N - 1) - a = INV_N(a)$。

证明:

$$\begin{aligned} (2^N - 1) - a &= \sum_{i=0}^{N-1} 2^i - a \\ &= \sum_{i=0}^{N-1} (1 - a_i) 2^i \end{aligned}$$

$$= \mathrm{INV}_N(a)$$

证毕。□

定理 3：对于 N 位非负整数 $i = (i_{N-1}i_{N-2}\dots i_1 i_0)_2$，若关系 $\mathrm{BRV}_N(i) < i$ 成立，那么关系 $\mathrm{INV}_N(i) < \mathrm{INV}_N(\mathrm{BRV}_N(i))$ 也成立。

证明：

根据定理 2 有：

$$\begin{aligned}\mathrm{INV}_N(\mathrm{BRV}_N(i)) &= (2^N - 1) - \mathrm{BRV}_N(i)\\ &> (2^N - 1) - i\\ &= \mathrm{INV}_N(i)\end{aligned}$$

证毕。□

第二部分：基于穷举法的位翻转置换构造算法及其改良

构造位翻转置换的典型算法是穷举法，通过对调的方式生成整个位翻转置换序列，其伪代码如下：

```
a[0...2^N − 1] := {0 ... 2^N − 1};
for (i := 0 ... 2^N − 1) {
   if (i < BRV_N(i)) {
      对调 a[i] 和 a[BRV_N(i)];
   }
}
```

直观上看，如果能够直接找到所有满足关系 $0 \le i < \mathrm{BRV}_N(i) < 2^N - 1$ 的整数 i，那么就可以在某种程度上降低算法的迭代次数，下面将给出找出所有满足这样的关系的一种算法。

为方便表示，令 $M = \left\lfloor \frac{N}{2} \right\rfloor$，当 N 为偶数时，令 $i = (b_{M-1}b_{M-2}\dots b_0 a_0 \dots a_{M-2}a_{M-1})_2$，反之则令 $i = (b_{M-1}b_{M-2}\dots b_0 m a_0 \dots a_{M-2}a_{M-1})_2$（m 等于 0 或 1）。令 $A = (a_{M-1}a_{M-2}\dots a_0)_2$，$B = (b_{M-1}b_{M-2}\dots b_0)_2$。

在这些定义的基础上，证明下面如下性质。

性质 1：i 与 $\mathrm{BRV}_N(i)$ 的大小关系与 B 和 A 之间的大小关系保持一致（也就是说这一关系成立与否与 N 的奇偶性以及 m 的选值无关）。

证明：

在 N 为偶数的情况下，有：

$$i = B \cdot 2^M + \mathrm{BRV}_M(A)$$
$$\mathrm{BRV}_N(i) = A \cdot 2^M + \mathrm{BRV}_M(B)$$

对二者作差：

$$i - \mathrm{BRV}_N(i) = (B - A)2^M + \mathrm{BRV}_M(A) - \mathrm{BRV}_M(B)$$

若 $A = B$，则 $i - \mathrm{BRV}_N(i) = 0$，即 $i = \mathrm{BRV}_N(i)$。

否则，若 $A \neq B$，不妨令 $B < A$，此时有：

$$\begin{aligned}\mathrm{BRV}_M(A) - \mathrm{BRV}_M(B) &= \sum_{k=0}^{M-1}(a_{M-k-1} - b_{M-k-1})2^k\\ &\le \sum_{k=0}^{M-1} 1 \cdot 2^k\\ &= 2^M - 1\end{aligned}$$

$$\begin{aligned}\mathrm{i}-\mathrm{BRV_N(i)} &\le (\mathrm{B}-\mathrm{A})2^{\mathrm{M}}+2^{\mathrm{M}}-1\\ &=(\mathrm{B}-\mathrm{A}+1)\cdot 2^{\mathrm{M}}-1\end{aligned}$$

由于 $\mathrm{B}<A$，因此 $\mathrm{B}-\mathrm{A}\le -1$，那么 $\mathrm{B}-\mathrm{A}+1\le 0$，因此可以断言：

$$(\mathrm{B}-\mathrm{A}+1)\cdot 2^{\mathrm{M}}-1<0$$

于是必有 $\mathrm{i}-\mathrm{BRV_N(i)}<0$，即 $\mathrm{i}<\mathrm{BRV_N(i)}$。

在 N 为奇数的情况下，有：

$$\mathrm{i}=\mathrm{B}\cdot 2^{\mathrm{M}+1}+\mathrm{m}\cdot 2^{\mathrm{M}}+\mathrm{BRV_M(A)}$$

$$\mathrm{BRV_N(i)}=\mathrm{A}\cdot 2^{\mathrm{M}+1}+\mathrm{m}\cdot 2^{\mathrm{M}}+\mathrm{BRV_M(B)}$$

若 $\mathrm{A}=\mathrm{B}$，则 $\mathrm{i}-\mathrm{BRV_N(i)}=0$，即 $\mathrm{i}<\mathrm{BRV_N(i)}$。

否则，还是不妨令 $\mathrm{B}<\mathrm{A}$，此时有：

$$\begin{aligned}\mathrm{i}-\mathrm{BRV_N(i)} &\le (\mathrm{B}-\mathrm{A})\cdot 2^{\mathrm{M}+1}+2^{\mathrm{M}}-1\\ &=(2(\mathrm{B}-\mathrm{A})+1)2^{\mathrm{M}}-1\end{aligned}$$

由于 $\mathrm{B}<A$，因此 $\mathrm{B}-\mathrm{A}\le -1$，那么 $2(\mathrm{B}-\mathrm{A})+1\le -1$，因此可以断言：

$$(2(\mathrm{B}-\mathrm{A})+1)2^{\mathrm{M}}-1<0$$

于是必有 $\mathrm{i}-\mathrm{BRV_N(i)}<0$，即 $\mathrm{i}<\mathrm{BRV_N(i)}$。

证毕。□

下面考察所有满足条件的 i（与对应的 $\mathrm{BRV_N(i)}$）的选值，根据性质 1，只需要考虑所有满足关系 $\mathrm{B}<A$ 的 A 和 B 的选值。

考虑 A 和 B 的最高位 $\mathrm{a_{M-1}}$ 和 $\mathrm{b_{M-1}}$，同时定义：

$$\mathrm{A'}=(\mathrm{a_{M-2}a_{M-3}\dots a_1a_0})_2$$

$$\mathrm{B'}=(\mathrm{b_{M-2}b_{M-3}\dots b_1b_0})_2$$

基于定理 1 及推论 1，即可将 i 满足条件的情况分成如表 2 所示的三种。

表 2：满足条件 $\mathrm{B}<\mathrm{A}$ 的三种情况

情况	满足 $\mathrm{B}<\mathrm{A}$ 所需达成的条件		
	$\mathrm{a_{M-1}}$	$\mathrm{b_{M-1}}$	$\mathrm{A'}$ 与 $\mathrm{B'}$ 的关系
1	0	0	$\mathrm{B'}<\mathrm{A'}$
2	1	1	
3	1	0	任意

这里不难发现情况 1 和 2 之间存在一定的联系。不妨取任意一个满足情况 1 的 i 值（$\mathrm{a_{M-1}}=0$，$\mathrm{b_{M-1}}=0$，$\mathrm{B'}<\mathrm{A'}$），根据定理 3，关系 $\mathrm{INV_N(i)}<\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)$ 也成立，而 $\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)=\left(\overbrace{\overline{\mathrm{a_{M-1}}}}^{=1}\,\overline{\mathrm{a_{M-2}}}\dots\overline{\mathrm{a_0}}(\overline{\mathrm{m}})\overline{\mathrm{b_0}}\dots\overline{\mathrm{b_{M-2}}}\,\overbrace{\overline{\mathrm{b_{M-1}}}}^{=1}\right)_2$ 恰好包含在情况 2 中。

同样地，取任意一个满足情况 2 的 i 值（$\mathrm{a_{M-1}}=1$，$\mathrm{b_{M-1}}=1$，$\mathrm{B'}<\mathrm{A'}$），同样根据定理 3，关系 $\mathrm{INV_N(i)}<\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)$ 也成立，对 $\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)$ 的各个位进行考察，不难发现 $\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)=\left(\overbrace{\overline{\mathrm{a_{M-1}}}}^{=0}\,\overline{\mathrm{a_{M-2}}}\dots\overline{\mathrm{a_0}}(\overline{\mathrm{m}})\overline{\mathrm{b_0}}\dots\overline{\mathrm{b_{M-2}}}\,\overbrace{\overline{\mathrm{b_{M-1}}}}^{=0}\right)_2$，这个值又恰好包含在情况 1 中。

不妨定义映射 $\varphi:\mathrm{i}\to\mathrm{INV_N}\big(\mathrm{BRV_N(i)}\big)$，显然有 $\varphi\big(\varphi(\mathrm{i})\big)=\mathrm{i}$，因此映射 φ 有双边逆映射 $\varphi^{-1}=\varphi$，这意味着映射 φ 是双射。因此可以断言，只要遍历出所有满足情况 1（或情况 2）的 i 值，在映射 φ 的作用下，所有的 $\varphi(\mathrm{i})$ 值必然完全且不重复地覆盖情况 2（或情况 1）。也就是说，对于情况 1、2，只需要遍历其中的一种即可。

对于情况 3（$a_{M-1}=1$，$b_{M-1}=0$），$\mathrm{INV_N}\left(\mathrm{BRV_N(i)}\right)=\left(\overbrace{\overline{a_{M-1}}}^{=0}\overline{a_{M-2}}\ldots\overline{a_0}(\overline{m})\overline{b_0}\ldots\overline{b_{M-2}}\overbrace{\overline{b_{M-1}}}^{=1}\right)_2$

也包含在情况 3 中，因此只需要分别遍历所有的 $0\le A'<2^{M-1}$ 以及 $0\le B'<2^{M-1}$ 即可枚举出情况 3 中的 i 值。

以 $N=6$ 的情况为例，表 3、4 列出了所有需要对调的情况。

表 3：$N=6$ 时需要对调的元素及其对应的 $B<A$ 的情况

情况	B		A		i	$\mathrm{BRV_N(i)}$	$\mathrm{INV_N}\left(\mathrm{BRV_N(i)}\right)$	$\mathrm{INV_N(i)}$	对调
	b_2	b_1b_0	a_2	a_1a_0					
1, 2	0_2	00_2	0_2	01_2	000100_2	001000_2	110111_2	111011_2	4-8 55-59
	0_2	00_2	0_2	10_2	000010_2	010000_2	101111_2	111101_2	2-16 47-61
	0_2	01_2	0_2	10_2	001010_2	010100_2	101011_2	110101_2	10-20 43-53
	0_2	00_2	0_2	11_2	000110_2	011000_2	100111_2	111001_2	6-24 39-57
	0_2	01_2	0_2	11_2	001110_2	011100_2	100011_2	110001_2	14-28 35-49
	0_2	10_2	0_2	11_2	010110_2	011010_2	100101_2	101001_2	22-26 37-41
3	0_2	00_2	1_2	00_2	000001_2	100000_2	不适用		1-32
	0_2	00_2	1_2	01_2	000101_2	101000_2			5-40
	0_2	00_2	1_2	10_2	000011_2	110000_2			3-48
	0_2	00_2	1_2	11_2	000111_2	111000_2			7-56
	0_2	01_2	1_2	00_2	001001_2	100100_2			9-36
	0_2	01_2	1_2	01_2	001101_2	101100_2			13-44
	0_2	01_2	1_2	10_2	001011_2	110100_2			11-52
	0_2	01_2	1_2	11_2	001111_2	111100_2			15-60
	0_2	10_2	1_2	00_2	010001_2	100010_2			17-34
	0_2	10_2	1_2	01_2	010101_2	101010_2			21-42
	0_2	10_2	1_2	10_2	010011_2	110010_2			19-50
	0_2	10_2	1_2	11_2	010111_2	111010_2			23-58
	0_2	11_2	1_2	00_2	011001_2	100110_2			25-38
	0_2	11_2	1_2	01_2	011101_2	101110_2			29-46
	0_2	11_2	1_2	10_2	011011_2	110110_2			27-54
	0_2	11_2	1_2	11_2	011111_2	111110_2			31-62

表 4：$N=6$ 时所有的需要考虑的 i 及 $\mathrm{BRV_N(i)}$

i	$\mathrm{BRV_N(i)}$	$i<\mathrm{BRV_N(i)}$		i	$\mathrm{BRV_N(i)}$	$i<\mathrm{BRV_N(i)}$
0（000000_2）	0（000000_2）			32（100000_2）	1（000001_2）	
1（000001_2）	32（100000_2）	是		33（100001_2）	33（100001_2）	
2（000010_2）	16（010000_2）	是		34（100010_2）	17（010001_2）	

3（000011_2）	48（110000_2）	是		35（100011_2）	49（110001_2）	是
4（000100_2）	8（001000_2）	是		36（100100_2）	9（001001_2）	
5（000101_2）	40（101000_2）	是		37（100101_2）	41（101001_2）	是
6（000110_2）	24（011000_2）	是		38（100110_2）	25（011001_2）	
7（000111_2）	56（111000_2）	是		39（100111_2）	57（111001_2）	是
8（001000_2）	4（000100_2）			40（101000_2）	5（000101_2）	
9（001001_2）	36（100100_2）	是		41（101001_2）	37（100101_2）	
10（001010_2）	20（010100_2）	是		42（101010_2）	21（010101_2）	
11（001011_2）	52（110100_2）	是		43（101011_2）	53（110101_2）	是
12（001100_2）	12（001100_2）			44（101100_2）	13（001101_2）	
13（001101_2）	44（101100_2）	是		45（101101_2）	45（101101_2）	
14（001110_2）	28（011100_2）	是		46（101110_2）	29（011101_2）	
15（001111_2）	60（111100_2）	是		47（101111_2）	61（111101_2）	是
16（010000_2）	2（000010_2）			48（110000_2）	3（000011_2）	
17（010001_2）	34（100010_2）	是		49（110001_2）	35（100011_2）	
18（010010_2）	18（010010_2）			50（110010_2）	19（010011_2）	
19（010011_2）	50（110010_2）	是		51（110011_2）	51（110011_2）	
20（010100_2）	10（001010_2）			52（110100_2）	11（001011_2）	
21（010101_2）	42（101010_2）	是		53（110101_2）	43（101011_2）	
22（010110_2）	26（011010_2）	是		54（110110_2）	27（011011_2）	
23（010111_2）	58（111010_2）	是		55（110111_2）	59（111011_2）	是
24（011000_2）	6（000110_2）			56（111000_2）	7（000111_2）	
25（011001_2）	38（100110_2）	是		57（111001_2）	39（100111_2）	
26（011010_2）	22（010110_2）			58（111010_2）	23（010111_2）	
27（011011_2）	54（110110_2）	是		59（111011_2）	55（110111_2）	
28（011100_2）	14（001110_2）			60（111100_2）	15（001111_2）	
29（011101_2）	46（101110_2）	是		61（111101_2）	47（101111_2）	
30（011110_2）	30（011110_2）			62（111110_2）	31（011111_2）	
31（011111_2）	62（111110_2）	是		63（111111_2）	63（111111_2）	

根据上面的各种性质，得到改进后的位翻转置换构造算法，伪代码如下（输入为阶数 N，输出为数组 a[] 或针对数组 a[] 的所有对调操作）：

```
a[0...2^N − 1] := {0 ... 2^N − 1};
M = floor(N / 2);
if (N 是偶数) {
    //  遍历情况 1、2
    for (A = 1...2^(M−1) − 1) {
        for (B = 0...A − 1) {
            i = B * 2^M + BRV_M(A);
            ri = A * 2^M + BRV_M(B);
            对调 a[i] 和 a[ri];
            对调 a[INV_N(ri)] 和 a[INV_N(i)];
        }
    }
    //  遍历情况 3
    for (A = 2^(M−1)...2^M − 1) {
        for (B = 0...2^(M−1) − 1) {
            i = B * 2^M + BRV_M(A);
            ri = A * 2^M + BRV_M(B);
            对调 a[i] 和 a[ri];
        }
    }
```

```
} else {
   //  遍历中间位
   for (m in {0 * 2^M, 1 * 2^M}) {
      //  遍历情况 1、2
      for (A = 1...2^(M-1) - 1) {
         for (B = 0...A - 1) {
            i = B * 2^(M+1) + m + BRV_M(A);
            ri = A * 2^(M+1) + m + BRV_M(B);
            对调 a[i] 和 a[ri];
            对调 a[INV_N(ri)] 和 a[INV_N(i)];
         }
      }
      //  遍历情况 3
      for (A = 2^(M-1)...2^M - 1) {
         for (B = 0...2^(M-1) - 1) {
            i = B * 2^(M+1) + m + BRV_M(A);
            ri = A * 2^(M+1) + m + BRV_M(B);
            对调 a[i] 和 a[ri];
         }
      }
   }
}
```

这一算法虽然能够直接找出所有待对调的元素，但需要预先构造 $\mathrm{BRV_M(k)}$（$0 \le \mathrm{k} < 2^{\mathrm{M}}$）序列，后文将给出快速构造该序列的一种方法。

第三部分：基于递推法的位翻转置换构造算法

Anne Cathrine Elster 在其论文[1]中给出了一种时间复杂度为 $\mathrm{O}(2^{\mathrm{N}})$ 的基于递推法的位翻转置换构造算法，笔者对其算法进行了一定的修改并给出了修改后的算法正确性证明及递推公式。

定理 4：对于任意的 N 位正整数 $\mathrm{i} = (\mathrm{i_{N-1}i_{N-2}} \dots \mathrm{i_1 i_0})_2 \neq 0$，有且仅有一个奇数 $\mathrm{c_i}$，使得 $\mathrm{BRV_N(i)} = \mathrm{c_i} \cdot 2^{\mathrm{k}}$，其中 k 是 i 的二进制表示法中前导零的个数。

证明：

先证存在性，由于 $\mathrm{i} \neq 0$，必然能找到这样一个整数 k，使得对于所有的 $1 \le \mathrm{p} \le \mathrm{k}$ 有 $\mathrm{i_{N-p}} = 0$ 且 $\mathrm{i_{N-k-1}} = 1$（不然就有 $\mathrm{i} = 0$，这与前提是矛盾的）。如此一来便有：

$$\begin{aligned}\mathrm{BRV_N(i)} &= (\mathrm{i_0 i_1} \dots \mathrm{i_{N-2} i_{N-1}})_2 \\ &= (\mathrm{i_0 i_1} \dots \mathrm{i_{N-k-1} i_{N-k}} \dots \mathrm{i_{N-2} i_{N-1}})_2 \\ &= \left(\mathrm{i_0 i_1} \dots \mathrm{i_{N-k-1}} \overbrace{0 \dots 0}^{\times \mathrm{k}}\right)_2 \\ &= (\mathrm{i_0 i_1} \dots \mathrm{i_{N-k-1}})_2 \cdot 2^{\mathrm{k}}\end{aligned}$$

令 $\mathrm{c_i} = (\mathrm{i_0 i_1} \dots \mathrm{i_{N-k-1}})_2$，由 $\mathrm{i_{N-k-1}} = 1$ 可知 $\mathrm{c_i}$ 是奇数，由 k 的定义可知 k 是 i 的二进制表示法中前导零的个数。

再证唯一性，假设存在 $1 \le \mathrm{k_1} < \mathrm{k_2} < N$ 使得 $\mathrm{BRV_N(i)} = \mathrm{c_{i,1}} \cdot 2^{\mathrm{k_1}} = \mathrm{c_{i,2}} \cdot 2^{\mathrm{k_2}}$（其中 $\mathrm{c_{i,1}}$、$\mathrm{c_{i,2}}$ 都是奇数），那么就有 $\mathrm{c_{i,1}} = \mathrm{c_{i,2}} \cdot 2^{\mathrm{k_2 - k_1}}$，可见 $\mathrm{c_{i,1}}$ 是偶数，与假设矛盾，唯一性得证。□

由定理 4 可推导出推论 2、3。

推论 2：对于任意的 $\mathrm{N-1}$ 位正整数 $0 < \mathrm{i} = (0\mathrm{i_{N-2}} \dots \mathrm{i_1 i_0})_2 < 2^{\mathrm{N-1}}$，有 $\mathrm{BRV_N(2i)} = \frac{\mathrm{BRV_N(i)}}{2}$。

证明：令 k 是 i 的二进制表示法中前导零的个数，那么有：

$$\begin{aligned}2\mathrm{i} &= (0\mathrm{i_{N-2}} \dots \mathrm{i_{N-k} i_{N-k-1}} \dots \mathrm{i_1 i_0})_2 \cdot 2 \\ &= (\mathrm{i_{N-2}} \dots \mathrm{i_{N-k} i_{N-k-1}} \dots \mathrm{i_1 i_0 0})_2\end{aligned}$$

显然 $k-1$ 是 $2i$ 的二进制表示法中前导零的个数，因此有：

$$\begin{aligned}BRV_N(2i)\cdot 2 &= (0i_0i_1\dots i_{N-k-1}i_{N-k}\dots i_{N-2})_2\cdot 2\\ &= (0i_0i_1\dots i_{N-k-1})_2\cdot 2^{k-1}\cdot 2\\ &= (i_0i_1\dots i_{N-k-1})_2\cdot 2^k\\ &= BRV_N(i)\end{aligned}$$

证毕。□

推论 3：对于任意的 $N-1$ 位正整数 $0<i=(0i_{N-2}\dots i_1i_0)_2<2^{N-1}$，有 $BRV_N(2i+1)=BRV_N(2i)+2^{N-1}$。

证明：令 k 是 i 的二进制表示法中前导零的个数，那么有：

$$\begin{aligned}2i+1 &= (0i_{N-2}\dots i_{N-k}i_{N-k-1}\dots i_1i_0)_2\cdot 2+1\\ &= (i_{N-2}\dots i_{N-k}i_{N-k-1}\dots i_1i_01)_2\end{aligned}$$

显然 $k-1$ 是 $2i+1$ 的二进制表示法中前导零的个数，因此有：

$$\begin{aligned}BRV_N(2i+1) &= (1i_0i_1\dots i_{N-k-1}i_{N-k}\dots i_{N-2})_2\\ &= (1i_0i_1\dots i_{N-k-1})_2\cdot 2^{k-1}\\ &= \left(1\cdot 2^{N-k}+(0i_0i_1\dots i_{N-k-1})_2\right)\cdot 2^{k-1}\\ &= 2^{N-1}+(0i_0i_1\dots i_{N-k-1})_2\cdot 2^{k-1}\\ &= 2^{N-1}+BRV_N(2i)\end{aligned}$$

证毕。□

另外易证 $BRV_N(0)=0$、$BRV_N(1)=2^{N-1}$，由此便得到生成 $BRV_N(i)$ 的递推式：

$$BRV_N(i)=\begin{cases}0 & (i=0)\\ 2^{N-1} & (i=1)\\ \frac{1}{2}BRV_N\left(\frac{i}{2}\right) & (i\geq 2\text{ 且为偶数})\\ BRV_N(i-1)+2^{N-1} & (i>2\text{ 且为奇数})\end{cases}\qquad(0\leq i<2^N)$$

算法伪代码如下（输入为阶数 N，输出为数组 a[]）：

```
a[0] = 0;
a[1] = 2^(N-1);
for (i = 2; i < 2^N; i += 2) {
   a[i] = (a[i >> 1] >> 1);
   a[i + 1] = a[i] + 2^(N-1);
}
```

第四部分：位翻转置换在 FFT 中的应用

Cooley-Tukey 算法是最常见的一种 FFT（快速傅立叶变换）算法。一般来说，对于最普遍的以 2 为底的 Cooley-Tukey 算法实现，不论是采取按时间抽取还是按频率抽取，其蝶形变换后的存放于存储器中的输出序列的序号 j 与输入序列的序号 i 之间具有如下对应关系[2]：

$$i\mapsto j=BRV_{\lceil\log_2 W\rceil}(i)\qquad(0\leq i<W)$$

其中，W 为 FFT 的运算窗口长度。

在实际的应用场景中，实际期望的输出序列与输入序列的关系是简单的 X(i)~x(i)，因此需要利用位翻转置换的生成算法来对输出序列 X(n) 进行顺序上的还原。包括 Apache Commons Math [3]在内的绝大多数库都是基于未优化的穷举法来实现的，这使得还原 X(n) 的过程需要经历 W 次的迭代。

但如果事先利用递推公式推导出 $BRV_M(i)$ 序列（$N = \lceil \log_2 W \rceil$，$M = \left\lfloor \frac{N}{2} \right\rfloor$），那么利用优化后的穷举算法，就可以直接得到所有需要交换的下标 i 和 $BRV_N(i)$（$i < BRV_N(i)$），并不需要遍历整个序列。

在绝大多数情况下，事先推导出需要的 $BRV_M(i)$ 序列是可行的。以 Java 语言为例，数组的长度和下标都是采用 int 型来表示的，该类型为 32 位有符号整数，由于数组长度不可能为负数，因此最多有 31 位被使用（符号位必然为 0），那么就有 $\max(N) = 31$，$\max(M) = \left\lfloor \frac{\max(N)}{2} \right\rfloor = 15$，因此 $BRV_{\max(M)}(i)$ 序列的长度为 $2^{15} = 32768$，这是可以接受的。

对上述总体流程进行整理和总结，可得如下算法伪代码（输入为阶数 N、序列 X(n) 数组 X[]，伪代码对数组 X[] 的次序进行对调，第一阶段产生的 BRV_M[] 数组可以被缓存下来以供后续使用）：

```
M = (N >> 1);
INV = (1 << N) - 1;

//
//  第一阶段（初始化）：初始化 BRV_M(i) 序列
//
BRV_M[0] = 0;
BRV_M[1] = (1 << (M - 1));
for (i = 2; i < (1 << M); i += 2) {
   BRV_M[i] = (BRV_M[i >> 1] >> 1);
   BRV_M[i + 1] = BRV_M[i] + (1 << (M - 1));
}

//
//  第二阶段（处理 X(i) 序列）：对序列 X(i) 中元素的次序进行调换
//
if ((N & 1) == 0) {
   //  遍历情况 1、2
   for (A = 1; A < (1 << (M - 1)); ++A) {
      for (B = 0; B < A; ++B) {
         i = (B << M) + BRV_M[A];
         ri = (A << M) + BRV_M[B];
         对调 X[i] 和 X[ri];
         对调 X[INV ^ ri] 和 X[INV ^ i];
      }
   }
   //  遍历情况 3
   for (A = (1 << (M - 1)); A < (1 << M); ++A) {
      for (B = 0; B < (1 << (M - 1)); ++B) {
         i = (B << M) + BRV_M[A];
         ri = (A << M) + BRV_M[B];
         对调 X[i] 和 X[ri];
      }
   }
} else {
   //  遍历情况 1、2
   for (A = 1; A < (1 << (M - 1)); ++A) {
      for (B = 0; B < A; ++B) {
         i = (B << (M + 1)) + BRV_M[A];
         ri = (A << (M + 1)) + BRV_M[B];
         对调 X[i] 和 X[ri];
         对调 X[INV ^ ri] 和 X[INV ^ i];
         i += (1 << M);
         ri += (1 << M);
         对调 X[i] 和 X[ri];
         对调 X[INV ^ ri] 和 X[INV ^ i];
      }
   }
   //  遍历情况 3
   for (A = (1 << (M - 1)); A < (1 << M); ++A) {
      for (B = 0; B < (1 << (M - 1)); ++B) {
         i = (B << (M + 1)) + BRV_M[A];
         ri = (A << (M + 1)) + BRV_M[B];
         对调 X[i] 和 X[ri];
         对调 X[i + (1 << M)] 和 X[ri + (1 << M)];
      }
```

```
    }
}
```

第五部分：性能测试

本文针对三种不同的算法进行了测试，第一种是未经任何优化的穷举法（算法 A），第二种是 Apache Commons Math 中实现的一种优化的穷举法（算法 B），第三种是本文所述的算法（算法 C）。

算法 A 的伪代码已在第二部分的开头部分给出，算法 C 的伪代码已在第四部分的结尾部分给出，算法 B 的伪代码如下：

```
a[0...2^N − 1] := {0 ... 2^N − 1};
j = 0;
for (i = 0... 2^N − 1) {
    if (i < j) {
        对调 a[i] 和 a[j];
    }
    k = 2^(N−1);
    while (k <= j && k > 0) {
        j -= k;
        k >>= 1;
    }
    j += k;
}
```

测试程序运行的硬件环境采用 Intel(R) Core(TM) i7-6700 CPU、DDR4 32GB 2133MT/s 内存，操作系统为 Ubuntu Linux 18.04.2 桌面版（内核版本 5.4.0），Python 运行时采用 CPython 3.6.9。

测试结果已在表 5 中给出，其中 N 为位翻转置换序列的阶数，L 为重复生成置换的次数（即测试得到的耗时为重复生成 L 次 N 阶位翻转置换序列的结果）。

表 5：三种算法在不同数据规模下的运行耗时

数据规模		算法	耗时				
N	L						
8	1000	A	704 ms	783 ms	686 ms	730 ms	764 ms
		B	123 ms	117 ms	120 ms	114 ms	111 ms
		C	22 ms	23 ms	22 ms	23 ms	22 ms
10	1000	A	3361 ms	3191 ms	3107 ms	3121 ms	3008 ms
		B	440 ms	463 ms	477 ms	472 ms	484 ms
		C	89 ms	88 ms	88 ms	89 ms	97 ms
12	1000	A	14980 ms	16229 ms	16491 ms	16743 ms	15761 ms
		B	1959 ms	1898 ms	1902 ms	1895 ms	1896 ms
		C	363 ms	361 ms	364 ms	364 ms	366 ms
6	10000	A	1303 ms	1273 ms	1316 ms	1253 ms	1218 ms
		B	268 ms	273 ms	343 ms	267 ms	277 ms
		C	55 ms	57 ms	55 ms	53 ms	54 ms
16	1	A	326 ms	312 ms	328 ms	322 ms	357 ms
16	100	A	（耗时过大，未列出）				
		B	3125 ms	3093 ms	2979 ms	3243 ms	3109 ms
		C	705 ms	633 ms	646 ms	670 ms	699 ms

最后用量化的方式说明未优化的穷举算法效率低的原因，还是令 N 为位翻转置换序列的阶

数，$M=\left\lfloor\frac{N}{2}\right\rfloor$。

当 N 为偶数时，没有中间位，因此不需要对调的数（即二进制表示法下的回文数）的个数为 2^M，需要对调的数对的个数为 $\frac{1}{2}(2^N-2^M)=2^{2M-1}-2^{M-1}$，需要对调的数对的个数与数的总数的比值为：

$$\frac{2^{2M-1}-2^{M-1}}{2^{2M}}=\frac{1}{2}-\frac{1}{2^{M+1}}$$

当 N 为奇数时，中间位有两种可能（0 或 1），因此不需要对调的数的个数为 $2^M\cdot 2=2^{M+1}$，需要对调的数对的个数为 $\frac{1}{2}(2^N-2^{M+1})=2^{2M}-2^M$，需要对调的数对的个数与数的总数的比值为：

$$\frac{2^{2M}-2^M}{2^{2M+1}}=\frac{1}{2}-\frac{1}{2^{M+1}}$$

因此不论 N 的奇偶性如何，需要对调的数对的个数与数的总数的比值（用函数 $H(N)$ 来表示）恒为：

$$\begin{aligned}H(N)&=\frac{1}{2}-\frac{1}{2^{M+1}}\\&=\frac{1}{2}-\frac{1}{2^{\left\lfloor\frac{N}{2}\right\rfloor+1}}\end{aligned}$$

考察 $H(N)$ 的若干性质：

$$\lim_{N\to+\infty}H(N)=0.5$$

$$H(N)<0.5$$

因此不难得到结论，未优化的穷举算法的遍历过程有超过一半是无用的（即不需要对调），再加上每次遍历都需要对一个 N 位非负整数进行位翻转操作，因此未优化的穷举法的效率较低。相比之下，本文所述的算法所需的遍历次数始终不大于需要对调的数对的个数，且每次遍历只需要经过简单的查表和位运算后就可以进行对调操作了，因此效率相对较高。

参考文献

编号	引用文献	DOI/URL
1	A. C. Elster, "Fast bit-reversal algorithms," International Conference on Acoustics, Speech, and Signal Processing,, 1989, pp. 1099-1102 vol.2, doi: 10.1109/ICASSP.1989.266624.	10.1109/ICASSP.1989.266624
2	Alan V. Oppenheim and Ronald W. Schafer. 2009. Discrete-Time Signal Processing (3rd. ed.). Prentice Hall Press, USA.	10.5555/77000
3	Apache Commons Math (3.6.1).	GitHub