Merger Sort

Trương Hải Bằng
VNUHCM

Nội dung

1. Phương pháp trộn Run
2. Phương pháp trộn tự nhiên
3. Phương pháp trộn đa lối cân bằng(balanced multiway merging)
4. Phương pháp trộn đa pha(Polyphase Merge)

1. Trộn run

Có 2 Cách (khác nhau về hàm phân bố)

1. Cách 1: phân bố luân phiên dãy ban đầu vào 2 dãy con
2. Cách 2: dãy ban đầu được chia đệ quy thành hai dãy con cho đến khi n=1

1.1 Trộn run

```
void Distribute(int a[], int N, int &nb, int &nc, int k)
{
   int       i, pa, pb, pc;
   pa = pb = pc = 0;
   while (pa < N)
   {
            for (i=0; (pa<N) && (i<k); i++, pa++, pb++)
                     b[pb] = a[pa];
            for (i=0; (pa<N) && (i<k); i++, pa++, pc++)
                     c[pc] = a[pa];
   }
   nb = pb;          nc = pc;
}
```

1.1 Trộn run

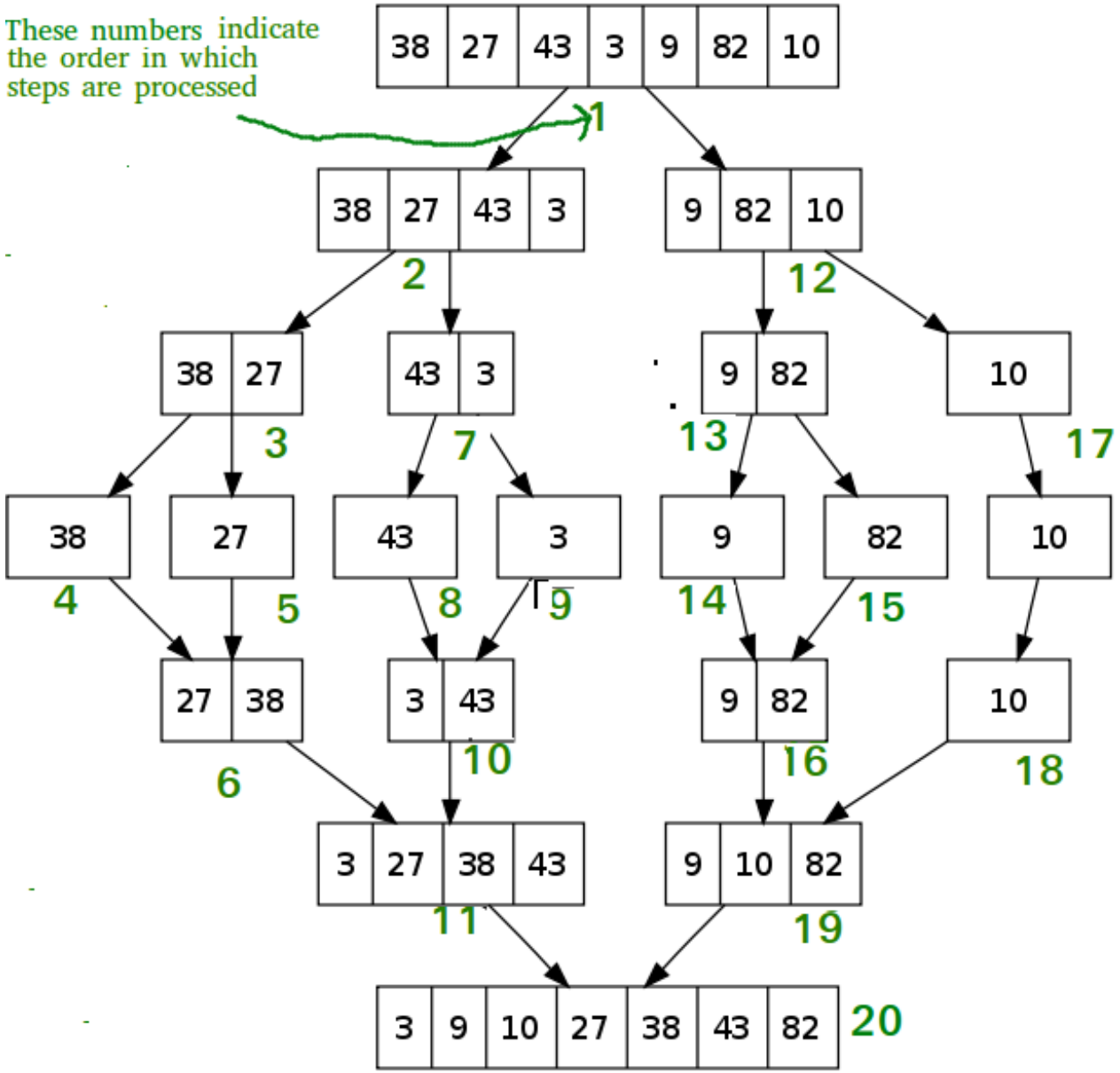
```
int b[MAX], c[MAX], nb, nc;

void MergeSort(int a[], int N)
{
   int      k;
   for (k = 1; k < N; k *= 2)
   {
            Distribute(a, N, nb, nc, k);
            Merge(a, nb, nc, k);
   }
}
```

1.2. Trộn run: Cách 1

Cách 2: dãy ban đầu được chia đệ quy thành hai dãy con cho đến khi n=1

{38, 27, 43, 3, 9, 82, 10}.

- Dãy được chia đệ quy thành hai dãy con cho đến khi n=1.
- Với n=1, các quá trình hợp nhất sẽ hoạt động và bắt đầu hợp nhất các dãy trở lại cho đến khi dãy hoàn chỉnh đã hợp nhất.



```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;
// Create temp arrays
    int L[n1], R[n2];
 // Copy data to temp arrays L[] and R[]
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
     // Merge the temp arrays back into
arr[l..r]
     // Initial index of first subarray
    int i = 0;
// Initial index of second subarray
    int j = 0;
// Initial index of merged subarray
    int k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    // Copy the remaining elements of
    // L[], if there are any
    while (i < n1) {arr[k] = L[i];
        i++;
        k++;
    }
    // Copy the remaining elements of
    // R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Hàm main()

MergeSort (arr [], l, r)

Nếu r> l

1. **Tìm điểm giữa để chia mảng thành hai nửa:**

   Ở giữa m = (l + r) / 2

2. **Hợp nhất cuộc gọi Sắp xếp cho nửa đầu:**

   Gọi mergeSort(arr, l, m)

3. **Hợp nhất cuộc gọi Sắp xếp cho nửa sau:**

   Gọi mergeSort(arr, m + 1, r)

4. **Hợp nhất hai nửa được sắp xếp ở bước 2 và 3:**

   Gọi merge(arr, l, m, r)

```
// l is for left index and r is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int arr[],int l,int r){
    if(l>=r){
        return;//returns recursively
    }
    int m =l+ (r-l)/2;
    mergeSort(arr,l,m);
    mergeSort(arr,m+1,r);
    merge(arr,l,m,r);
}
```

2. Phương pháp trộn tự nhiên

Giải thuật:

- Trong phương pháp trộn đã trình bày ở trên, giải thuật không tận dụng được chiều dài cực đại của các run trước khi phân bổ; do vậy, việc tối ưu thuật toán chưa được tận dụng.
- Đặc điểm cơ bản của phương pháp trộn tự nhiên là tận dụng độ dài "tự nhiên" của các run ban đầu; nghĩa là, thực hiện việc trộn các run có độ dài cực đại vơi nhau cho đến khi dãy chỉ bao gồm một run: dãy đã được sắp thứ tự.

2. Phương pháp trộn tự nhiên *(tt)*

Input: f0 là tập tin cần sắp thứ tự.

Output: f0 là tập tin đã được sắp thứ tự.

Lặp Cho đến khi dãy cần sắp chỉ gồm duy nhất một run.

Phân bố:

Phân bố F0 vào F1 và F2 theo các run tự nhiên

Trộn:

Trôn các run cua F1 và F2 vào F0

Quá trình này sẽ tiếp tục cho đến khi số run của F0 lá 1 thì dừng

2. Phương pháp trộn tự nhiên *(tt)*

Bước1

A	1	2	9	8	7	6	5
B	1 2 9			7	5		
C	8	6					
A'	1 2 8 9				6 7		5

Bước2

A	1 2 8 9				6 7		5
B	1 2 8 9				5		
C	6 7						
A'	1 2 6 7 8 9						5

2. Phương pháp trộn tự nhiên *(tt)*

Bước 3

A	**1 2 6 7 8 9**	**5**
B	**1 2 6 7 8 9**	
C	**5**	
A'	**1 2 5 6 7 8 9**	

Tại bước 3 ta thấy rằng file A đã được sắp xếp và kết thúc.

2. Phương pháp trộn tự nhiên *(tt)*

Nếu mô tả thuật toán như sau:

while (Số run trong tệp A >1)

{ - Lần lượt phân bổ các run tự nhiên từ file A sang 2 file B và C theo cách: một run sang B thì run tiếp theo sang C, run sau đó sang B,... cho đến khi hết run trên A. Như vậy tệp B có thể chứa nhiều hơn file C một run.

- Tiếp theo trộn từng cặp run tự nhiên trên B và C thành một run và ghi vào tệp A.

- Nếu A đã sắp xếp hay chỉ có một run thì kết thúc, nếu không quay lại vòng lặp.

}

2. Phương pháp trộn tự nhiên *(tt)*

Cài đặt: *Các hàm sử dụng trong chương trình*

```
void CreatFile(FILE *Ft,int);
void ListFile(FILE *);
void Distribute();
void Copy(FILE *,FILE *);
void CopyRun(FILE *,FILE *);
void MergeRun();
void Merge();
typedef int DataType;
FILE *F0,*F1,*F2;
int M,N,Eor;
// Bien eor dung de kiem tra ket thuc Run hoac File
DataType X1,X2,X,Y;
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void main(void)
{
CreatFile(F0,N);
  ListFile(F0);
  do
  {
            Distribute();
    M=0;
    Merge();
  }while (M != 1);
  ListFile(F0);
}
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void Copy(FILE *Fi,FILE *Fj)
{
   //Doc phan tu X tu Tap tin Fi, ghi X vao Fj
   //Eor==1, Neu het Run(tren Fi) hoac het File Fi
   fscanf(Fi,"%3d",&X);
   fprintf(Fj,"%3d",X);
   if( !feof(Fi) )
   {
            fscanf(Fi,"%3d",&Y);
            long curpos = ftell(Fi)-2;
            fseek(Fi, curpos, SEEK_SET);
   }
   if ( feof(Fi) ) Eor = 1;
   else Eor = (X > Y) ? 1 : 0 ;
}
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void Distribute()
/*      Phan bo luan phien cac Run tu nhien tu F0 vao F1 va F2   */
{
  do
  {
        CopyRun(F0,F1);
        if( !feof(F0) ) CopyRun(F0,F2);
  }while( !feof(F0) );
}
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void CopyRun(FILE *Fi,FILE *Fj)
/*      Chep 1 Run tu Fi vao Fj    */
{
  do
        Copy(Fi,Fj);
  while ( !Eor);
}
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void MergeRun()
/*      Tron 1 Run cua F1 va F2 vao F0      */
{
  do
  {
        fscanf(F1,"%3d",&X1);
        long curpos = ftell(F1)-2;
        fseek(F1, curpos, SEEK_SET);
        fscanf(F2,"%3d",&X2);
        curpos = ftell(F2)-2;
        fseek(F2, curpos, SEEK_SET);
        }
```

2. Phương pháp trộn tự nhiên *(tt)*

```
if( X1 <= X2 )
        {
                Copy(F1,F0);
                if (Eor) CopyRun(F2,F0);
        }
        else
        {
                Copy(F2,F0);
                if ( Eor ) CopyRun(F1,F0);
        }
  } while ( !Eor );
```

2. Phương pháp trộn tự nhiên *(tt)*

```
void Merge()
/* Tron cac run tu F1 va F2 vao F0                */
{
    while( (!feof(F1)) && (!feof(F2)) )
    {
            MergeRun();
            M++;
    }
    while( !feof(F1) )
    {
            CopyRun(F1,F0);
            M++;
    }
    while( !feof(F2) )
    {
            CopyRun(F2,F0);
            M++;
    }
}
```

3. Trộn n-đường cân bằng

Thuật toán sắp xếp ngoài thực hiện qua 2 giai đoạn:*Phân phối và trộn*

Giai đoạn nào góp phần làm thay đổi thứ tự ?

Chi phí cho giai đoạn phân phối ?

Rút ra kết luận

Thay vì thực hiện 2 giai đoạn, ta chỉ thực hiện 1 giai đoạn "Trộn"

Tiết kiệm được khoảng ½ chi phí copy

Cần số lượng file trung gian gấp đôi

3. Trộn n-đường cân bằng *(tt)*

Chi phí sắp xếp ngoài tỉ lệ với số bước thực hiện

Nếu mỗi bước cần N thao tác copy, và dùng 2 file trung gian: cần log2N bước . cần N*log2N thao tác copy

Để giảm số bước : phân bố các run lên nhiều hơn 2 file trung gian

Như vậy:

Dùng nhiều file trung gian để giảm số bước

Tiết kiệm chi phí copy bằng cách thực hiện 1 giai đoạn

Sử dụng 2*n file trung gian

3. Trộn n-đường cân bằng *(tt)*

Thuật toán tổng quát (tinh chế 0)

[BƯỚC 1] Gọi tập nguồn là S = {f1, f2, … , fn}

Gọi tập đích là D = {g1, g2, … , gn}

Chia xoay vòng dữ liệu của file fInput cho các file thuộc tập nguồn S, mỗi lần 1 run cho đến khi file fInput hết

[BƯỚC 2] Trộn từng bộ run của các file thuộc tập nguồn S, tạo thành run mới, lần lượt ghi lên các file thuộc tập đích D

[BƯỚC 3] Nếu (số run trên các file thuộc D > 1) thì

- Hoán vị vai trò của tập nguồn (S) và tập đích (D)

- Quay lại [B2]

Ngược lại Kết thúc thuật toán

3. Trộn n-đường cân bằng *(tt)*

VD. fInput: U Q N M K I H F D C B, *N=3*

// Phân phối (lần 1)

f1: U M H C

f2: Q K F B

f3: N I D

// Trộn (lần 1)

g1: N Q U B C

g2: I K M

g3: D F H

3. Trộn n-đường cân bằng *(tt)*

- // Trộn (lần 2)
- f1: D F H I K M N Q U
- f2: B C
- f3: NULL
- // Trộn (lần 3)
- g1: B C D F H I K M N Q U
- g2: NULL
- g3: NULL

3. Trộn n-đường cân bằng *(tt)*

Các ký hiệu:

fInput: file dữ liệu gốc cần sắp xếp

N: số phần tử trên file fInput

n: số file trung gian trên mỗi tập nguồn/đích

S: tập các file nguồn

D: tập các file đích

Sdd: tập các file nguồn đang còn run dở dang

Str: tập các file nguồn chưa hết (!EOF), còn có thể tham gia vào quá trình trộn

3. Trộn n-đường cân bằng *(tt)*

"Lượt": là 1 quá trình trộn run từ nguồn . đích, một "lượt" kết thúc khi mỗi file

đích (trong tập D) nhận được 1 run

Drun: tập các file đích đã nhận được run trong "lượt" hiện hành

Suy diễn:

S – Str: tập các file nguồn đã hết (EOF)

Str – Sdd: tập các file nguồn chưa hết (!EOF), nhưng đã kết thúc run hiện tại

D – Drun: tập các file đích chưa nhận được run trong "lượt" hiện hành

3. Trộn n-đường cân bằng *(tt)*

[BƯỚC 1]

S = {f1, f2, … , fn}

D = {g1, g2, … , gn}

// Chia xoay vòng dữ liệu của file fInput cho các file thuộc tập nguồn S

i = 1;

while (!feof(fInput)) {

Copy_1_Run(fInput, fi);

i = (i % n) + 1;

}

Str = S;

Drun = {};

nDemRun = 0;

3. Trộn n-đường cân bằng *(tt)*

[BƯỚC 2]

a. Sdd = Str

b. Gọi dhh Ỵ D – Drun là file đích hiện hành (sẽ được nhận run)

c. Đọc các phần tử xfi, fi Ỵ Sdd

d. Gọi xf0 = MIN { xfi, fi Ỵ Sdd}

e. Copy xf0 lên dhh

3. Trộn n-đường cân bằng *(tt)*

f. Nếu (file f0 hết) thì {
Str = Str – {f0}
Sdd = Sdd – {f0}
Nếu (Str == {}) thì { // Xong quá trình trộn N . D
nDemRun++;
Goto [B3]
}
ngược lại Nếu (Sdd <> {}) thì Goto [B2.d]
ngược lại { // Sdd=={}: hết bộ run hiện hành
nDemRun++;
Drun = Drun + {dhh};
Nếu (Drun==D) thì Drun= {}; // Xong 1 “lượt”
Goto [B2.a]

3. Trộn n-đường cân bằng *(tt)*

ngược lại { // File f0 chưa hết
Nếu (!EOR(f0)) thì {
Đọc phần tử kế xf0 từ file f0;
Goto [B2.d]
}
ngược lại { // Hết run hiện hành trên f0
Sdd = Sdd – {f0}
Nếu (Sdd <> {}) thì Goto [B2.d]
ngược lại { // Sdd=={}: hết bộ run hiện hành
nDemRun++;
Drun = Drun + {dhh};
Nếu (Drun==D) thì Drun= {}; // Xong 1 “lượt”
Goto [B2.a]
}

3. Trộn n-đường cân bằng *(tt)*

} // end of file f0 chưa hết

Thuật toán chi tiết…(tt):

[BƯỚC 3] Nếu (nDemRun == 1) thì Kết thúc thuật toán

ngược lại {

Nếu (nDemRun < n) thì Str = Drun; // Không đủ n run

ngược lại Str = D;

Drun = {};

nDemRun = 0;

“Hoán vị tập S, D”

Goto [B2]

}

4. Phương pháp trôn đa pha

Phương pháp trộn đa lối cân bằng đã loại bỏ các phép sao chép thuần túy thông qua việc gộp quá trình phân phối và qúa trình trộn trong cùng một giai đoạn. Tuy nhiên các tập tin các tập tin chưa được sử dụng một cách có hiệu quả bởi vì trong cùng một lần duyệt thì phân nửa số tập tin luôn luôn giữ vai trò trộn (nguồn) và phân nửa số tập tin luôn luôn giử vai trò phân phối (đích). Ta có thể cải tiến phương pháp trên bằng cách giải quyết thay đổi vai trò của các tập tin trong cùng một lần duyệt phương pháp này gọi là phương pháp trộn đa pha.

4. Phương pháp trôn đa pha *(tt)*

Phương pháp sắp xếp kiểu này do R.L. Gilstar nêu ra năm 1960 và được gọi là trộn đa pha. Có thể thấy ngay là để cho quá trình trộn đa pha thực hiện được thì dữ liệu ban đầu phải được phân bổ một cách thích hợp trên các băng nguồn. Ví dụ nếu ta có 2 băng nguồn là T1, T2 và một băng đích là T3. Nếu số run trong T1 và trong T2 là các số Fibonacci liên tiếp thì ta có thể áp dụng phương pháp trộn đa pha như sau:

4. Phương pháp trôn đa pha *(tt)*

T_1	T_2	T_3
F_n $(= F_{n-1}+F_{n-2})$	F_{n-1}	
F_{n-2}		F_{n-1} $(= F_{n-2}+F_{n-3})$
	F_{n-2} $(= F_{n-3}+F_{n-4})$	F_{n-3}
F_{n-3} $(= F_{n-4}+F_{n-5})$	F_{n-4}	
...	...	...

Ví dụ với n = 7 ta có F7 = 13, F6 = 8.

Quá trình trộn được mô tả trong bảng sau:

4. Phương pháp trôn đa pha *(tt)*

T_1	T_2	T_3	**Giải thích**
13	8		Lúc đầu có 13 run trong T_1 và 8 run trong T_2
5		8	Trộn 8 run trong mỗi run vào T_3, T_1 còn 5 run
	5	3	Trộn 5 run vào T_2, trong T_3 còn 3 run...
3	2		
1		2	
	1	1	Trộn 1 run trong T_1 với 1 run trong T_3 ở bước trước vào T_2, T_3 còn lại 1 run.
1			Trộn 1 run trong T_2 với 1 run trong T_3 ở bước trước vào T_1, chỉ còn lại một run, quá trình kết thúc.

4. Phương pháp trôn đa pha *(tt)*

Bước 1: Phân phối luân phiên các run ban đầu của f1 vào f2 và f3

Bước 2: Trộn các run của f1, f2 vào f3 . Giải thuật kết thúc nếu f3 chỉ có một run

Bước 3: Chép nửa run của f3 vào f1

Bước 4: Trộn các run của f1 và f3 vào f2. Giải thuật kết thúc nếu f2 chỉ có một run.

Bước 5: Chép nửa số run của f2 vào f1. Lặp lại *bước 2.*

4. Phương pháp trôn đa pha *(tt)*

- *Ví dụ 1: Trường hợp n=7, tổng số run ban đầu là 13+8=21 run*

Phase	F 1	F2	F3	
0	1, 1, 1, 1, 1, 1, 1, 1	1, 1, 1, 1, 1		Sort
1	1, 1, 1,		2, 2, 2, 2, 2	Merge 1
2		3, 3, 3	2, 2	Merge 2
3	5, 5	3		Merge 3
4	5		8	Merge4
5		13		Merge 5
6	21			Merge 6

4. Phương pháp trôn đa pha *(tt)*

- *Phase 0:* Phân phối các run ban đầu
- *Phase 1:* Trộn 8 run của f1 và f2 vào **f3**
- *Phase 2:* Trộn 5 run của f1 và f3 vào **f2**
- *Phase 3:* Trộn 3 run của f2 và f3 vào **f1**
- *Phase 4:* Trộn 2 run của f1 và f2 vào **f3**
- *Phase 5:* Trộn 1 run của f1 và f3 vào **f2**
- *Phase 6:* Trộn 1 run của f2 và f3 vào **f1**

4. Phương pháp trôn đa pha *(tt)*

Phase	T6	T5	T4	T3	T2	T1	Tổng số runs được xữ lý
Phase 0	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	-	129
Phase 1	1^{15}	1^{14}	1^{12}	1^{8}	-	5^{16}	80
Phase 2	1^{7}	1^{6}	1^{4}	-	9^{8}	5^{8}	72
Phase 3	1^{3}	1^{2}	-	17^{4}	9^{4}	5^{4}	68
Phase 4	1^{1}	-	33^{2}	17^{2}	9^{2}	5^{2}	66
Phase 5	-	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	65
Phase 6	129^{1}	-	-	-	-	-	129

4. Phương pháp trôn đa pha *(tt)*

- *Phase 0:* Phân phối các run ban đầu
- *Phase 1:* Trộn 16 run từ T2 đến T6 vào **T1**
- *Phase 2:* Trộn 8 run của T1, T3, T4, T5, T6 vào **T2**
- *Phase 3:* Trộn 4 run của T1, T2, T4, T5, T6 vào **T3**
- *Phase 4:* Trộn 2 run của T6, T5, T3, T1, T6 vào **T4**
- *Phase 5:* Trộn 1 run của T1, T2, T3, T4, T6 vào **T5**
- *Phase 6:* Trộn 1 run từ T1 đến T5 vào **T6.**

4. Phương pháp trôn đa pha *(tt)*

- Trong ví dụ 1, số run phan phối ban đầu cho các tập tin là 2 số Fibonacci kế tiếp nhau của dãy Fibonacci bậc 1:

  0, 1, 1, 2, 3, 5, 8 . . .

- Trong ví dụ 2 số run ban đầu phân bố cho các tập tin theo dãy Fibonacci bậc 4:

  0, 0, 0, 0, 1, 1, 2, 4, 8, 16 . . .

- Dãy Fibonacci bậc P tổng quát được định nghĩa như sau:

  $\mathbf{F^{(p)}_n = F^{(p)}_{n-1} + \ldots + F^{(p)}_{n-2} + \ldots + F^{(p)}_{n-p}}$

với $\boldsymbol{n >= p}$, $F^{(p)}_n = 0$, *với* $0 <= n <= p-2$; $F^{(p)}_{p-1} = 1$