Ordenação Externa

Joaquim Quinteiro Uchôa
Juliana Galvani Greghi

Comentários Iniciais - i

Métodos de ordenação tradicionais trabalham considerando que os dados estão todos em memória primária.

Cálculo de complexidade leva em conta apenas número de operações.

Não é incomum precisar ordenar dados com tamanho maior que a capacidade de memória.

Ex.: um registro de um aluno no SIG ocupa facilmente 1KB: nome (100B), endereço completo e telefone (300B), etc. Se inclui fotos, pula rapidamente para 100KB. 10000 alunos e já se ocuparam 1GB.

Comentários Iniciais - ii

Ordenação externa leva em conta o fato que os dados não podem ser ordenados **internamente** na memória primária ou de trabalho, precisando fazer uso da memória secundária ou de armazenamento (HDD, SSD, pendrives, etc.).

Problema: os métodos dependem fortemente do **estado da tecnologia**. Métodos em fita devem levar em conta a ausência de acesso aleatório, por exemplo.

Comentários Iniciais - iii

Duas medidas estão relacionadas ao desempenho da memória secundária:

- **Latência** ou **tempo de acesso**: tempo necessário para acessar o primeiro trecho em que está o dado solicitado;
- **Taxa de transferência**: quantidade de bits ou bytes transferidos por segundo, pode ser diferente para leitura e escrita.

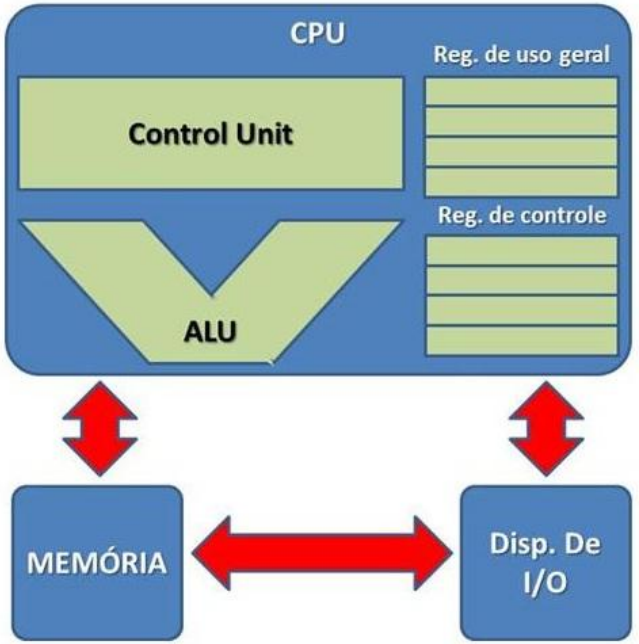
Obs: B/s (byte/s) ≠ b/s (bit/s). Em geral é necessário dividir velocidade em b/s por 8 para se obter a velocidade em B/s.

Arquitetura de Von Neumman

Base dos sistemas computacionais atuais, por questões tecnológicas, inclusive.

Dados são transferidos da unidade de armazenamento (HD, etc.) para memória de trabalho e processados na CPU, usando os registradores.

Implica em uma série de questões e em hierarquia de memória.



Hierarquia de Memória - Registradores

Processador XXX com Cache 8MB 4.2GHz - Aprox. R$ 1.500.00 em fev/2017.

Este processador possui 8 registradores em modo de 32 bits e 16 registradores em modo de 64 bits.

Registradores são unidades básicas de memória de trabalho e operam na velocidade do processador, sem qualquer atraso. A tecnologia utilizada permite alta velocidade, mas com altíssimo custo (por isso a quantidade minúscula).

Para minimizar esse fato, não é incomum processadores possuírem cache em até 3 níveis (L1, L2 e L3), com tamanhos e desempenhos diferentes.

Hierarquia de Memória - Memória RAM

Memória YYY 8GB 2400Mhz DDR4 CL15 - Aprox. 400,00 em fev/2017.

Esta memória, classe PC4-19200, possui latência de 12ns e taxa de transferência de 19.200 MB/s, ou 19,2 GB/s.

Não armazena informações entre desligamentos.

Hierarquia de Memória - Disco Rígido

HD ZZZ 1TB 7200RPM 64MB Cache SATA 6Gb/s - Aprox. R$ 250,00 em fev/2017.

Disco com taxa média de dados para leitura/gravação de 156MB/s. Taxa máxima para leitura/gravação: 210MB/s. Desempenho varia por conta do cache de 64MB.

Fabricante não informa tempo de acesso, mas testes apontam que velocidade média para acesso aleatório é de 12MB/s.

Hierarquia de Memória - Alternativa ao HD, o SSD

SSD WWW SATA III 6Gb/s 240GB - Aprox. 450,00 em fev/2017.

Latência é mínima comparada às velocidades de leitura e gravação:

- Leituras: 530MB/s
- Gravações: 440MB/s

É muito melhor que HDD, mas ainda assim é muito inferior à memória principal.

Hierarquia de Memória: quanto mais veloz, mais cara e em menor quantidade disponível.

Outras Limitações: Barramentos

Além do limite do dispositivo, a transferência ocorre por meio de barramentos que limitam a velocidade. Ex.:

- USB 3.0 (5 Gbit/s - 625 MB/s)
- SATA 3.0 (6 Gbit/s - 600 MB/s)
- PCIe v. 4.0:
  - 1.969 GB/s (×1)
  - 31.51 GB/s (×16)

Obs: velocidades obtidas na Wikipedia.

O Disco Rígido - i

O Disco Rígido - ii

O Disco Rígido - iii

O Disco Rígido - iv

Ordenação Externa - Visão Geral

O acesso a disco deve ser minimizado pelo método de ordenação, para melhoria de performance. Para efeitos de simplificação, consideraremos que, em geral, os dados serão lidos do meio de armazenamento para a memória em blocos.

A base para ordenação externa em geral é o uso de ordenação por intercalação, que implica em combinar dois ou mais blocos ordenados em um único bloco ordenado.

Métodos devem minimizar número de passadas sobre o arquivo, que é base para a medida de complexidade. Isso torna complicado o uso de abordagens gerais e paralelização.

Estratégia Geral

1. Dividir o arquivo em blocos do tamanho da memória interna disponível.

2. Ordenar cada bloco na memória interna.

3. Intercalar os blocos ordenados, fazendo várias passadas sobre o arquivo. A cada passada são criados blocos ordenados cada vez maiores, até que todo o arquivo esteja ordenado.

Merge Sort Externo

Merge Sort em Memória - Visão Geral

O merge sort é um algoritmo de ordenação eficiente bastante popular, sendo a implementação preferida em listas encadeadas e como base para ordenação externa.

É um algoritmo recursivo, baseado na ideia de dividir para conquistar, cujas bases estão no problema da intercalação (ou separação):

dados dois vetores ordenados a e b, de tamanhos m e n, respectivamente, intercalá-los de forma a gerar o vetor c, de tamanho m+n, contendo os elementos de a e b ordenados.

Problema da Intercalação - Exemplo

Sejam a = [2, 4, 8, 50, 61, 72]
e b = [0, 1, 3, 9, 60, 64, 65, 65, 90, 100].

Então o resultado da intercalação será dado por
c = [0, 1, 2, 3, 4, 8, 9, 50, 60, 61, 64, 65, 72, 90, 100]

Intercalação de Trechos de um Vetor

Não é interessante, para o processo de ordenação, ficar copiando de um vetor para outro em várias etapas desse processo. Assim, para o merge sort, é melhor intercalar trechos de um mesmo vetor:

Dados trechos ordenados a[p .. q-1] e a[q .. r], construir a[p .. r] também ordenado. Nesse caso p é o início, q é o meio e r é o fim do trecho a ser intercalado.

Intercalação de Trechos - i

```
void intercala(int a[], int inicio, int meio, int fim) {
    int i = inicio, j = meio+1;
    int tamanho = fim - inicio + 1;
    int aux[tamanho]; // vetor auxiliar
    for (int k=0; k < tamanho; k++) {
        if ((i <= meio) and (j <= fim)){
            if (a[i] <= a[j]){
                aux[k] = a[i]; // copia trecho1 em aux[]
                i++;           // avança em trecho1
            } else { //
                aux[k] = a[j]; // copia trecho2 em aux[]
                j++;           // avanca em trecho2
            }
```

Intercalação de Trechos - ii

```
} else if (i > meio) { // terminou o trecho1
            aux[k] = a[j];
            j++;
        } else {                // terminou o trecho2
            aux[k] = a[i];
            i++;
        }
    }
    // terminando: copiar de aux[] em a[inicio:fim]
    for (int k=0; k < tamanho; k++){
        a[inicio + k] = aux[k];
    }
}
```

Merge Sort em Memória

O merge sort consiste em aplicar, recursivamente, o processo de divisão do vetor original em metades, até que cada metade tenha um único elemento, intercalando as metades ordenadas após isso:

```
void mergesort(int a[], int inicio, int fim){
    int meio;
    if (inicio < fim) {
        meio = (inicio + fim)/2;
        mergesort(a, inicio, meio);
        mergesort(a, meio+1, fim);
        intercala(a,inicio,meio,fim);
    }
}
```

Reescrevendo a Intercala()

Existem várias implementações para a função de intercalação, em geral com a mesma eficiência. Deixamos a cargo dos interessados a pesquisa por alternativas.

Entretanto, uma versão reescrita é bastante popular, sendo aqui apresentada. Ela é muito semelhante à versão apresentada, com diferenças pontuais.

Reescrita da Intercala() - i

```
// intercala v[p..q-1] e v[q..r] em v[p..r]
void intercala(int v[], int p, int q, int r){
    int i = p, j = q;
    int tamanho = r - p + 1;
    int w[tamanho]; // vetor auxiliar
    int k = 0;
    while ((i < q) and (j <= r)) {
       if (v[i] <= v[j])  {
           w[k++] = v[i++]; /* w[k] = v[i]; k++; i++; */
       } else  {
           w[k++] = v[j++]; /* w[k] = v[j]; k++; j++; */
       }
    }
```

Reescrita da Intercala() - ii

```
    // terminou um dos vetores, agora copia o outro
    while (i < q) {
       w[k++] = v[i++];
    }

    while (j <= r) {
       w[k++] = v[j++];
    }

    // agora copiamos do vetor auxiliar aux[] em v[p:r]
    for (int m = 0; m < tamanho; m++){
       v[p + m] = w[m];
    }
}
```

Merge Sort Top-Down

A versão tradicional do merge sort utiliza uma abordagem inicialmente *top-down* (de cima para baixo) ao sair dividindo os vetores e depois volta intercalando, em passagens *bottom-up* (de baixo para cima). Por conta disso, o método é chamado de implementação ***top-down***.

É possível pular a etapa *top-down*, indo direto às etapas *bottom-up*, em uma versão iterativa do merge sort.

Merge Sort Bottom-Up Iterativo

O raciocínio por trás de uma versão ***bottom-up*** iterativa do merge sort é considerar que o vetor já está inicialmente partido em pedaços de tamanho um e ir aumentando o tamanho dos pedaços.

Em cada etapa, o pedaço dobra de tamanho, até que os pedaços sejam de tamanho igual ou maior ao próprio vetor.

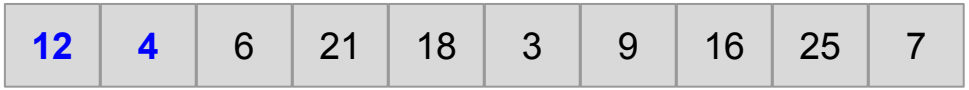
Essa versão do algoritmo é base para o algoritmos de ordenação externa (em disco).

Merge Sort Iterativo - Algoritmo

```
void mergeiterativo (int v[], int tam) {
    int p, r, b = 1;
    while (b < tam) {
        p = 0;
        while (p + b < tam) {
            r = p + 2*b - 1;
            if (r >= tam) r = tam - 1;
            intercala(v, p, p+b, r);
            p = p + 2*b;
        }
        b = 2*b; // dobra o tamanho do bloco
    }
}
```

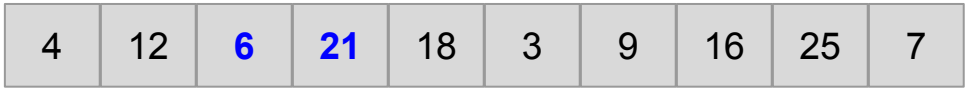
Merge Sort Iterativo - Exemplo - i

tam = 10 p = 0 r = 1 b = 1



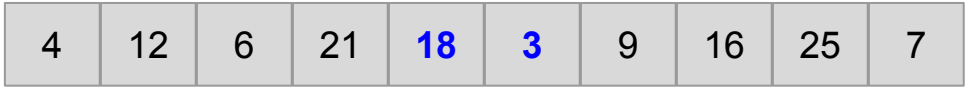
intercala (v, 0, 1, 1)

tam = 10 p = 2 r = 3 b = 1



intercala (v, 2, 3, 3)

tam = 10 p = 4 r = 5 b = 1



Merge Sort Iterativo - Exemplo - ii

intercala (v, 4, 5, 5)

tam = 10 p = 6 r = 7 b = 1

4	12	6	21	3	18	**9**	**16**	25	7

intercala (v, 6, 7, 7)

tam = 10 p = 8 r = 9 b = 1

4	12	6	21	3	18	9	16	**25**	**7**

intercala (v, 8, 9, 9)

tam = 10 p = 10 r = 9 b = 1

4	12	6	21	3	18	9	16	7	25

Merge Sort Iterativo - Exemplo - iii

tam = 10 p = 0 r = 3 b = 2

4	**12**	**6**	**21**	3	18	9	16	7	25

intercala (v, 0, 2, 3)

tam = 10 p = 4 r = 7 b = 2

4	6	12	21	**3**	**18**	**9**	**16**	7	25

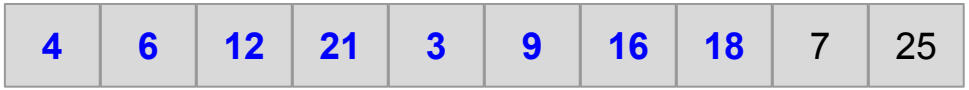
intercala (v, 4, 6, 7)

tam = 10 p = 8 r = 9 b = 2

4	6	12	21	3	9	16	18	7	25

Merge Sort Iterativo - Exemplo - iv

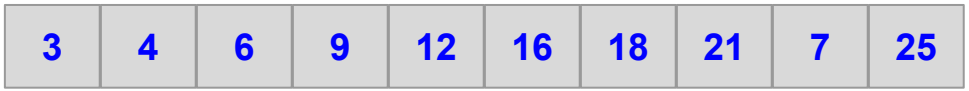
tam = 10 p = 0 r = 7 b = 4



intercala (v, 0, 4, 7)

tam = 10 p = 8 r = 7 b = 4

tam = 10 p = 0 r = 9 b = 8



Merge Sort Iterativo - Exemplo - v

intercala (v, 0, 8, 9)

3	4	6	7	9	12	16	18	21	25

Merge Sort Externo

A versão mais simples do algoritmo funciona com 4 arquivos:

- Dois arquivos que contém dados de entrada;
- Dois arquivos que contém dados de saída parcialmente ordenados a cada iteração.

Os arquivos usados como entrada e saída se alternam durante a execução. A cada iteração, tem-se um subconjunto ordenado de registros.

Merge Sort Externo - Algoritmo - i

- Dividir o arquivo original em dois arquivos de origem, referenciados como *f1* e *f2*.
- Considerar que *f1* e *f2* estão organizados e ordenados, elementos a elemento (bloco de tamanho 1)
- Inicia-se o processo de intercalação alternando o armazenamento dos blocos analisados nos arquivos de saída *s1* e *s2* (primeira rodada).
- *s1* e *s2* estão organizados e ordenados em blocos com o dobro do tamanho dos arquivos *f1* e *f2*.

Merge Sort Externo - Algoritmo - ii

- Repete-se o processo de intercalação, agora considerando os arquivos *f1* e *f2* como arquivos de saída e *s1* e *s2* como arquivos de entrada, ordenados em blocos de tamanho 2 (segunda rodada).
- Repete-se o processo até que todos os elementos estejam ordenados.

Características do método

- O número de rodadas de *f1* e *f2* difere em no máximo 1.
- No máximo um dentre os arquivos de entrada possui uma cauda - um número incompleto de registros em uma rodada.
- O tamanho das rodadas dobra a cada iteração.

Exemplo (1/6)

Arquivo de origem

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

F1 23 78 12 9 35 27 26 4 25 12

F2 45 90 64 11 5 10 8 6 49

Não há uma regra sobre como realizar a divisão dos elementos nos dois arquivos, apenas que o número de elementos em cada um deve ser igual (ou com no máximo um elemento de diferença).

Exemplo (2/6)

F1 23 78 12 9 35 27 26 4 25 12

F2 45 90 64 11 5 10 8 6 49

S1 23 45 | 12 64 | 5 35 | 8 26 | 25 49

S2 78 90 | 9 11 | 10 27 | 4 6 | 12

Exemplo (3/6)

F1 23 45 78 90 | 5 10 27 35 | 12 25 49

F2 9 11 12 64 | 4 6 8 26

S1 23 45 | 12 64 | 5 35 | 8 26 | 25 49

S2 78 90 | 9 11 | 10 27 | 4 6 | 12

Exemplo (4/6)

F1 23 45 78 90 | 5 10 27 35 | 12 25 49

F2 9 11 12 64 | 4 6 8 26

S1 9 11 12 23 45 64 78 90 | 12 25 49

S2 4 5 6 8 10 26 27 35

Exemplo (5/6)

F1 4 5 6 8 9 10 11 12 23 26 27 35 45 64 78 90

F2 12 25 49

S1 9 11 12 23 45 64 78 90 | 12 25 49

S2 4 5 6 8 10 26 27 35

Exemplo

F1 4 5 6 8 9 10 11 12 23 26 27 35 45 64 78 90

F2 12 25 49

S1 4 5 6 8 9 10 11 12 12 23 25 26 27 35 45 49 64 78 90

S2

Melhorias ao Merge Sort Externo

É possível melhorar os resultados do Merge Sort, ordenando parte dos dados em memória principal, de acordo com a capacidade disponível.

Mesmo que não seja possível ordenar todo o arquivo, acelera-se o processo, utilizando-se um método de ordenação em memória primária (quick sort ou o próprio merge sort).

Uma dificuldade de implementação é que o algoritmo precisa antes descobrir e não superestimar a quantidade de memória disponível.

Outra possibilidade de melhoria é utilizar vários arquivos para entrada e saída.

Intercalação Multi-caminhos

Uma outra abordagem para a ordenação de arquivos grandes é a divisão em *n* arquivos menores, que podem ser parcialmente ordenados em RAM, seguida da intercalação desses arquivos.

Nesse caso, são utilizados mais que 2 arquivos de entrada e 2 arquivos de saída.

A essa abordagem dá-se o nome de Intercalação Multi-caminhos (*Multi-way merging*).

Limitação

Apesar da possibilidade de aumento de eficiência, é importante levar em conta que a ordenação multi-caminhos só é vantajosa quando ***vários dispositivos*** de entrada e saída estão disponíveis ou quando o dispositivo possui facilidade de leitura e entrada em ***vários locais***.

Em caso contrário, o tempo de busca irá tornar o método inviável, pior que o merge sort tradicional.

Exemplo (1/3)

Arquivo origem

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

f1 23 90 9 5 26 6 12
f2 45 12 11 27 8 25
f3 78 64 35 10 4 49

s1 23 45 78 | 5 10 27 | 12
s2 12 64 90 | 4 8 26 |
s3 9 11 35 | 6 25 49 |

Exemplo (2/3)

f1 9 11 12 23 35 45 64 78 90 |
f2 4 5 6 8 10 25 26 27 49 |
f3 12

s1 23 45 78 | 5 10 27 | 12
s2 12 64 90 | 4 8 26 |
s3 9 11 35 | 6 25 49 |

Exemplo

f1 9 11 12 23 35 45 64 78 90 |
f2 4 5 6 8 10 25 26 27 49 |
f3 12

s1 4 5 6 8 9 10 11 12 12 23 25 26 27 35 45 49 64 78 90
s2
s3

Seleção por Substituição

Seleção por Substituição

- A quebra do arquivo pode ser realizada de maneira mais “eficiente”.
- A estrutura ideal para isso é o heap
  - Retira-se o menor item da fila de prioridades
  - Coloca-se um novo item no lugar
  - Reconstrói-sei o heap

Funcionamento

- São inseridos **m** itens são inseridos na fila de prioridades inicialmente vazia.
- O menor item da fila deve ser retirado e substituído pelo próximo item de entrada.
- Se o próximo item a ser inserido é menor que o que está sendo retirado, então ele deve ser marcado como membro do próximo bloco e ser considerado como o maior do que todos os itens do bloco corrente.
- Quando um item marcado vai para o topo da fila, o bloco corrente é encerrado e um novo bloco ordenado é iniciado.

Exemplo (1/4)

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

Entrada 23 45 78

90 45 78

90 **12** 78

90 **12** **64**

9 **12** **64**

Bloco1 23 45 78 90

Exemplo (2/4)

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

Entrada 9 12 64

11 12 64

35 12 64

35 **5** 64

27 **5** 64

27 **5** **10**

Bloco2 9 11 12 35 64

Exemplo (3/4)

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

Entrada 27 5 10

27 26 10

27 26 **8** Bloco3 5 10 26 27

27 **4** **8**

6 **4** **8**

Seleção por Substituição

23 45 78 90 12 64 9 11 35 5 27 10 26 8 4 6 25 49 12

Entrada

6 4 8

6 25 8

49 25 8

49 25 12

49 25

49

Bloco4 4 6 8 12 25 49

Blocos Ordenados Gerados

Bloco1 23 45 78 90

Bloco2 9 11 12 35 64

Bloco3 5 10 26 27

Bloco4 4 6 8 12 25 49

Com os blocos ordenados, procede-se a intercalação normalmente.

Intercalação Polifásica

Intercalação Polifásica

- Os blocos ordenados pela seleção por substituição são distribuídos de forma desigual entre as fitas disponíveis.
- Uma fita deve permanecer livre.
- Em seguida, a intercalação dos blocos deve ocorrer até que uma das fitas de entrada fique vazia.
- A fita vazia torna-se a próxima fita de saída.

Intercalação Polifásica

F1 **23** 45 78 90 9 11 12 35 64 5 10 26 27

F2 **4** 6 8 12 25 49

F3 4

Intercalação Polifásica

F1 **23** 45 78 90 9 11 12 35 64 5 10 26 27

F2 **6** 8 12 25 49

F3 4 6

Intercalação Polifásica

F1 **23** 45 78 90 9 11 12 35 64 5 10 26 27

F2 **8** 12 25 49

F3 4 6 8

Intercalação Polifásica

F1 **23** 45 78 90 9 11 12 35 64 5 10 26 27

F2 **12** 25 49

F3 4 6 8 12

Intercalação Polifásica

F1 **23** 45 78 90 9 11 12 35 64 5 10 26 27

F2 **25** 49

F3 4 6 8 12 23

Intercalação Polifásica

F1 **45** 78 90 9 11 12 35 64 5 10 26 27

F2 **25** 49

F3 4 6 8 12 23 25

Intercalação Polifásica

F1 **45** 78 90 9 11 12 35 64 5 10 26 27

F2 **49**

F3 4 6 8 12 23 25 45

Intercalação Polifásica

F1 **78** 90 9 11 12 35 64 5 10 26 27

F2 **49**

F3 4 6 8 12 23 25 45 49

Intercalação Polifásica

F1 **78** **90** 9 11 12 35 64 5 10 26 27

F2

F3 4 6 8 12 23 25 45 49 78 90

Intercalação Polifásica

F1 **9** 11 12 35 64 5 10 26 27

F2 4

F3 **4** 6 8 12 23 25 45 49 78 90

Intercalação Polifásica

F1 **9** 11 12 35 64 5 10 26 27

F2 4 6

F3 **6** 8 12 23 25 45 49 78 90

Intercalação Polifásica

F1 **9** 11 12 35 64 5 10 26 27

F2 4 6 8

F3 **8** 12 23 25 45 49 78 90

Intercalação Polifásica

F1 **9** 11 12 35 64 5 10 26 27

F2 4 6 8 9

F3 **12** 23 25 45 49 78 90

Intercalação Polifásica

F1 **11** 12 35 64 5 10 26 27

F2 4 6 8 9 11

F3 **12** 23 25 45 49 78 90

Intercalação Polifásica

F1 **12** 35 64 5 10 26 27

F2 4 6 8 9 11 12

F3 **12** 23 25 45 49 78 90

Intercalação Polifásica

F1 **35** 64 5 10 26 27

F2 4 6 8 9 11 12 12

F3 **12** 23 25 45 49 78 90

Intercalação Polifásica

F1 **35** 64 5 10 26 27

F2 4 6 8 9 11 12 12 23

F3 **23** 25 45 49 78 90

Intercalação Polifásica

F1				**35**	64		5	10	26	27
F2	4	6	8	9	11	12	12	23	25	
F3						**25**	45	49	78	90

Intercalação Polifásica

F1				**35**	64		5	10	26	27
F2	4	6	8	9	11	12	12	23	25	35
F3							**45**	49	78	90

Intercalação Polifásica

F1 **64** 5 10 26 27

F2 4 6 8 9 11 12 12 23 25 35 45

F3 **45** 49 78 90

Intercalação Polifásica

F1 **64** 5 10 26 27

F2 4 6 8 9 11 12 12 23 25 35 45 49

F3 **49** 78 90

Intercalação Polifásica

F1 **64** 5 10 26 27

F2 4 6 8 9 11 12 12 23 25 35 45 49 64

F3 **78** 90

Intercalação Polifásica

F1 5 10 26 27

F2 4 6 8 9 11 12 12 23 25 35 45 49 64 78 90

F3 **78** **90**

Intercalação Polifásica

F1 **5** 10 26 27

F2 **4** 6 8 9 11 12 12 23 25 35 45 49 64 78 90

F3

Intercalação Polifásica

F1 **5** 10 26 27

F2 **6** 8 9 11 12 12 23 25 35 45 49 64 78 90

F3 **4**

Intercalação Polifásica

F1 **10** 26 27

F2 **6** 8 9 11 12 12 23 25 35 45 49 64 78 90

F3 **4** **5**

Intercalação Polifásica

F1 **10** 26 27

F2 **8** 9 11 12 12 23 25 35 45 49 64 78 90

F3 **4** **5** **6**

Intercalação Polifásica

F1 **10** 26 27

F2 **9** 11 12 12 23 25 35 45 49 64 78 90

F3 **4** **5** **6** **8**

Intercalação Polifásica

F1 **10** 26 27

F2 **11** 12 12 23 25 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9**

Intercalação Polifásica

F1 **26** 27

F2 **11** 12 12 23 25 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10**

Intercalação Polifásica

F1 **26** 27

F2 **12** 12 23 25 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11**

Intercalação Polifásica

F1 **26** 27

F2 **12** 23 25 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11** **12**

Intercalação Polifásica

F1 **26** 27

F2 **23** 25 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11** **12** **12**

Intercalação Polifásica

F1 **26** 27

F2 **25** 35 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11** **12** **12** **23**

Intercalação Polifásica

F1 **26** 27

F2 **35** 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11** **12** **12** **23** **25**

Intercalação Polifásica

F1 **27**

F2 **35** 45 49 64 78 90

F3 **4** **5** **6** **8** **9** **10** **11** **12** **12** **23** **25** **26**

Intercalação Polifásica

F1

F2 **35 45 49 64 78 90**

F3 **4 5 6 8 9 10 11 12 12** 23 25 26 27

Intercalação Polifásica

F1

F2

F3 **4 5 6 8 9 10 11 12 12 23 25 26 27 35 45 49 64 78 90**