Ordenação Externa

- Envolve arquivos onde o número de registros é maior que a memória interna do computador
- Estruturas têm que levar em consideração que os dados estão armazenados em unidades externas
- Fatores
  - » Custo - relacionado com a transferência de dados
  - » Restrições de acesso aos dados - forma de acesso (sequencial, direto etc)
  - » Dependente do estado atual da tecnologia - pode tornar os métodos dependente de vários parâmetros
- Métodos em geral quebram o arquivo em blocos, intercalando-os:
  - » os algoritmos devem reduzir o número de passadas sobre o arquivo (10 passadas)

Ordenação por Intercalação

- Intercalar: combinar dois ou mais blocos **ordenados** em um único bloco
- A intercalação é utilizada como uma operação auxiliar de ordenação

- Estratégia geral:
  - » Realiza-se uma primeira passada no arquivo, quebrando-o em blocos de acordo com o tamanho da memória principal disponível
  - » Cada bloco é então ordenado na memória principal
  - » Os blocos ordenados são intercalados (com várias passadas pelo arquivo). São criados blocos cada vez maiores até que todo o arquivo esteja ordenado.

Intercalação Direta

- Distribuir os registros iniciais em dois arquivos **A** e **B,** de tal forma que o número de registros em cada arquivo seja igual ou varie em no máximo um registro.

Exemplo:

Arquivo Original: **2 12 17 16 14 30 17 2 50 65 20 32 48 58 16 20 15 10 30 45 16**

Distribuição Inicial:

A: 2 17 14 17 50 20 48 16 15 30 16

B: 12 16 30 2 65 32 58 20 10 45

Primeira Intercalação:

2 12 | 16 17 | 14 30 | 2 17 | 50 65 | 20 32 | 48 58 | 16 20 | 10 15 | 30 45 | 16

Segunda Distribuição:

A: 2 12 | 14 30 | 50 65 | 48 58 | 10 15 | 16

B: 16 17 | 2 17 | 20 32 | 16 20 | 30 45

Segunda Intercalação:

2 12 16 17 | 2 14 17 30 | 20 32 50 65 | 16 20 48 58 | 10 15 30 45 | 16

Terceira Distribuição:

A: 2 12 16 17 | 20 32 50 65 | 10 15 30 45

B: 2 14 17 30 | 16 20 48 58 | 16

Terceira Intercalação:

2 2 12 14 16 17 17 30 | 16 20 20 32 48 50 58 65 | 10 15 16 30 45

Quarta Distribuição:

A: 2 2 12 14 16 17 17 30 | 10 15 16 30 45

B: 16 20 20 32 48 50 58 65

Quarta Intercalação:

2 2 12 14 16 16 17 17 20 20 30 32 48 50 58 65 | 10 15 16 30 45

Finalmente ...

Quinta Distribuição:

A: 2 2 12 14 16 16 17 17 20 20 30 32 48 50 58 65

B: 10 15 16 30 45

Quinta Intercalação:

2 2 10 12 14 15 16 16 16 17 17 20 20 30 30 32 45 48 50 58 65

Arquivo Ordenado por Intercalação Direta

Implementação

```
struct aluno {
     long  cod, nom;
};


long   tamanho()
{
  long   num_reg;
  FILE  *original;

  if  ((original = fopen( "c:\\temp\\dado", "rb" ))  ==  NULL )
    printf( "Erro na abertura do arquivo de dados\n" );

  fseek(original, 0L, SEEK_END);
  num_reg  =   ftell (original)  /  sizeof(aluno);
  _fcloseall( );
  return num_reg;
}
```

```
void  distribui (long int   fator)
{
  aluno   al;    long   lidos;    FILE *original, *a, *b;

  if ((original = fopen( "c:\\temp\\dado", "rb" ))  ==  NULL )
    printf( "Erro na abertura do arquivo dado\n" );

  if( (a = fopen( "c:\\temp\\a", "wb" )) == NULL )
    printf( "Erro na abertura do arquivo  a \n" );

  if( (b = fopen( "c:\\temp\\b", "wb" )) == NULL )
    printf( "Erro na abertura do arquivo  b \n" );

  while  (! feof(original))  {
   lidos = 0;
   while ( (lidos++ < fator)    &&
             (fread(&al, sizeof(aluno), 1, original)) )
        fwrite(&al, sizeof(aluno), 1, a);

   lidos = 0;
   while ( (lidos++ < fator)    &&
             (fread(&al, sizeof(aluno), 1, original)))
       fwrite(&al, sizeof(aluno), 1, b);
  }

  _fcloseall( );
}
```

```
void  intercala (long int   fator)  {

  FILE *original, *a, *b;     aluno  al1,  al2;
  long int   i,  reg_lido_a = fator,   reg_lido_b = fator;

  //  Abrindo arquivos

 fread(&al1, sizeof(aluno), 1, a);
 fread(&al2, sizeof(aluno), 1, b);

  while  ( !feof(a)   &&   !feof(b) ) {
    while (reg_lido_a && reg_lido_b) {
      if  (al1.cod < al2.cod) {
        reg_lido_a--;
        fwrite(&al1, sizeof(aluno), 1, original);
        fread(&al1,  sizeof(aluno), 1, a);
        if  (feof(a)) break;
      } else {
       reg_lido_b--;
       fwrite(&al2, sizeof(aluno), 1, original);
       fread(&al2,  sizeof(aluno), 1, b);
       if (feof(b)) break;
      }
    }
    for (i = 0; (i  <  reg_lido_a)  &&  !feof(a); i++) {
         fwrite(&al1, sizeof(aluno), 1, original);
         fread(&al1,  sizeof(aluno), 1, a);
    }
    for (i = 0; (i  <  reg_lido_b)  &&   !feof(b);  i++) {
           fwrite(&al2, sizeof(aluno), 1, original);
           fread(&al2,  sizeof(aluno), 1, b);
    }
    reg_lido_a = reg_lido_b = fator;
  }  // continua
```

```
 if  ( feof(a)  &&  !feof(b) )
    do {
          fwrite(&al2, sizeof(aluno), 1, original);
      } while (fread(&al2,  sizeof(aluno), 1, b));

  if  ( !feof(a)  &&  feof(b))
    do {
        fwrite(&al1, sizeof(aluno), 1, original);
    }  while (fread(&al1,  sizeof(aluno), 1, a));

  _fcloseall( );
}     //   fim  Intercala
```

Intercalação Natural

- Distribuir os registros em dois arquivos **A** e **B,** de tal forma que se preserve as correntes já existentes

Exemplo:

Arquivo Original: **2 12 17 16 14 30 17 2 50 65 20 32 48 58 16 20 15 10 30 45 16**

Distribuição Inicial:
A: 2 12 17 | 14 30 | 2 50 65 | 16 20 | 10 30 45
B: 16 17 | 20 32 48 58 | 15 16

Primeira Intercalação:
2 12 16 17 17 | 14 20 30 32 48 58 | 2 15 16 50 65 | 16 20 | 10 30 45

Segunda Distribuição:

A: 2 12 16 17 17 | 2 15 16 50 65 | 10 30 45

B: 14 20 30 32 48 58 | 16 20 |

Segunda Intercalação:

2 12 14 16 17 17 20 30 32 48 58 | 2 15 16 16 20 50 65 | 10 30 45

Terceira Distribuição:

A: 2 12 14 16 17 17 20 30 32 48 58 | 10 30 45

B: 2 15 16 16 20 50 65

Terceira Intercalação:

2 2 12 14 15 16 16 16 17 17 20 20 30 32 48 50 58 65 | 10 30 45

Quarta Distribuição:

A: 2 2 12 14 15 16 16 16 17 17 20 20 30 32 48 50 58 65

B: 10 30 45

Quarta Intercalação:

2 2 10 12 14 15 16 16 16 17 17 20 20 30 30 32 45 48 50 58 65

Implementação

```
void distribui()
{    //  declara variaveis abre arquivos
     fread(&al, sizeof(aluno), 1, original);
     atual = al.cod;
     while (!feof(original)) {
       do {
            fwrite(&al, sizeof(aluno), 1, a);
            fread(&al, sizeof(aluno), 1, original);
            anterior = atual;
            atual = al.cod;
       } while ( !feof (original) && (atual >= anterior) );
       if (feof(original)) {
            if (atual < anterior)                fwrite(&al,
     sizeof(aluno), 1, b);
       } else {
            do {
                 fwrite(&al, sizeof(aluno), 1, b);
                 fread(&al, sizeof(aluno), 1, original);
                 anterior = atual;
                      atual = al.cod;
            } while ( (atual >= anterior) && !feof(original) );
            if (feof(original) && (atual < anterior) )
                      fwrite(&al, sizeof(aluno), 1, a);
       }
     }
     fcloseall( );
}
```

```
long intercala()
{

    // declara variaveis abre arquivos

     fread(&al1, sizeof(aluno), 1, a);
     fread(&al2, sizeof(aluno), 1, b);
     anterior_A = al1.cod;
     anterior_B = al2.cod;

     while (!feof(a) && !feof(b)) {
      while ((anterior_A <= al1.cod) && (anterior_B <= al2.cod)) {
          if (al1.cod < al2.cod) {
              fwrite(&al1, sizeof(aluno), 1, original);
              anterior_A = al1.cod;
              fread(&al1,  sizeof(aluno), 1, a);
              if (feof(a)) break;
          }  else {
               fwrite(&al2, sizeof(aluno), 1, original);
               anterior_B = al2.cod;
               fread(&al2,  sizeof(aluno), 1, b);
               if (feof(b)) break;
            }
      }
```

```
        while ((anterior_A <= al1.cod) && !feof(a)) {
                fwrite(&al1, sizeof(aluno), 1, original);
                anterior_A = al1.cod;
                fread(&al1,  sizeof(aluno), 1, a);
                if (feof(a)) break;
        }
        while ((anterior_B <= al2.cod) && !feof(b)) {
                fwrite(&al2, sizeof(aluno), 1, original);
                anterior_B = al2.cod;
                fread(&al2,  sizeof(aluno), 1, b);
                if (feof(b)) break;
        }
        anterior_A = al1.cod;
        anterior_B = al2.cod;
    }
    if (feof(a) && !feof(b))  do {
                fwrite(&al2, sizeof(aluno), 1, original);
    } while (fread(&al2,  sizeof(aluno), 1, b));

    if (!feof(a) && feof(b)) do {
                fwrite(&al1, sizeof(aluno), 1, original);
    } while (fread(&al1,  sizeof(aluno), 1, a));

    vazio = ftell(a) && ftell(b);
    fcloseall( );
    return vazio;
}
```