

Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br

The image features decorative blue lines in the corners. On the left, a vertical line descends from the top, then steps right and down. At the bottom left, a line starts with a small circle, goes up, then right, then down, then right again. At the top right, a line descends from the top edge, then steps left and down. At the bottom right, multiple parallel lines ascend from the bottom edge, then step right and down.

Ordenação Externa

Ordenação

Conjunto Ordenado: Elementos dispostos sob uma determinada ordem.

Ordenação

Conjunto Ordenado: Elementos dispostos sob uma determinada ordem.

- **Objetivos da Classificação/Ordenação:**
 - Facilitar a recuperação;
 - Tornar mais eficiente o acesso aos dados.

Ordenação

Conjunto Ordenado: Elementos dispostos sob uma determinada ordem.

- **Objetivos da Classificação/Ordenação:**

- Facilitar a recuperação;
- Tornar mais eficiente o acesso aos dados.

- **Tipos de Classificação:**

- **Interna:** Todos os registros estão na memória principal;
- **Externa:** Alguns registros podem estar em dispositivos auxiliares de armazenamento.

Ordenação

Ordenação interna:

- São os métodos que não necessitam de uma memória secundária para o processo, a ordenação é feita na memória principal do computador;

Ordenação

Ordenação externa:

- Quando o arquivo a ser ordenado **não cabe na memória principal** e, por isso, tem de ser armazenado em disco.

Ordenação

- A principal diferença entre os dois grupos é que no **método de ordenação interna qualquer registro pode ser acessado diretamente**, enquanto no **método externo é necessário fazer o acesso em algum dispositivo externo**.

Ordenação Externa

- **A ordenação externa consiste** em ordenar arquivos de tamanho maior que a memória interna disponível

Ordenação Externa

- **A ordenação externa consiste** em ordenar arquivos de tamanho maior que a memória interna disponível
- Os métodos de ordenação externa são diferentes dos algoritmos de ordenação interna
 - Os algoritmos devem diminuir o número de acessos às unidades de memória externa.

Ordenação Externa

- Nas memórias externas, os dados ficam em um arquivo sequência
 - Apenas um registro pode ser acesso em um dado momento.

Ordenação Externa

- Fatores que determinam as diferenças das técnicas de ordenação externa em relação à ordenação interna:

Ordenação Externa

- Fatores que determinam as diferenças das técnicas de ordenação externa em relação à ordenação interna:
 - **Maior custo de acesso à memória** para **transferência** dos dados entre a memória interna e externa
 - **Restrições de acesso a dados**, como o acesso **apenas sequencial** ou o **custo do acesso**
 - Os métodos são dependentes do estado atual da tecnologia.

Ordenação Externa

- O método mais importante de ordenação externa é o de ordenação por intercalação
 - Intercalar é combinar dois ou mais blocos ordenados em um único bloco ordenado
 - A intercalação é utilizada como operação auxiliar na ordenação

Ordenação Externa

- O método mais importante de ordenação externa é o de ordenação por intercalação
 - Intercalar é combinar dois ou mais blocos ordenados em um único bloco ordenado
 - A intercalação é utilizada como operação auxiliar na ordenação
- **Foco dos algoritmos de ordenação externa é reduzir o número de acessos ao arquivo**
 - Uma boa medida de complexidade de um algoritmo de ordenação por intercalação é o número de vezes que um item é lido ou escrito na memória interna.

Ordenação Externa

- O método mais importante de ordenação externa é o de ordenação por intercalação
 - Intercalar é combinar dois ou mais blocos ordenados em um único bloco ordenado
 - A intercalação é utilizada como operação auxiliar na ordenação
- **Foco dos algoritmos de ordenação externa é reduzir o número de acessos ao arquivo**
 - Uma boa medida de complexidade de um algoritmo de ordenação por intercalação é o número de vezes que um item é lido ou escrito na memória interna.
 - **Bons métodos de ordenação externa geralmente envolvem menos do que dez passadas sobre o arquivo.**

Ordenação Externa

- Estratégia geral dos métodos de ordenação externa:
 1. **Quebre o arquivo em blocos do tamanho da memória interna disponível**
 2. **Ordene cada bloco na memória interna**
 3. **Intercale os blocos ordenados**, fazendo várias passadas sobre o arquivo
 - Cada acesso são criados blocos ordenados cada vez maiores, até que todo o arquivo esteja ordenado



MergeSort - Externo

MergeSort Externo

- Método mais comum da ordenação externa
- Utiliza a técnica de intercalação: combina dois ou mais blocos ordenados em um único bloco, maior, ordenado.

MergeSort Externo

- **Funcionamento:**

- RAM comporta “n” registros de dados
- Carregar parte do arquivo na RAM
- Ordenar os dados na RAM com um algoritmo
- In-place: ex. QuickSort
- Salvar os dados ordenados em um arquivo separado
- Repetir os passos anteriores até terminar o arquivo original
- Ao final, temos “k” arquivos ordenados
- “Multi-way merging”

MergeSort Externo

- Funcionamento:
 - “Multi-way merging”
 - Criar “ $K+1$ ” buffers de tamanho “ $n/(k+1)$ ” (“1” de saída, “ k ” de entrada)

MergeSort Externo

- Funcionamento:
 - “Multi-way merging”
 - Criar “ $K+1$ ” buffers de tamanho “ $n/(k+1)$ ” (“1” de saída, “ k ” de entrada)
 - **Carregar partes dos arquivos ordenados nos “buffers de entrada”, intercalar no “buffer de saída”**

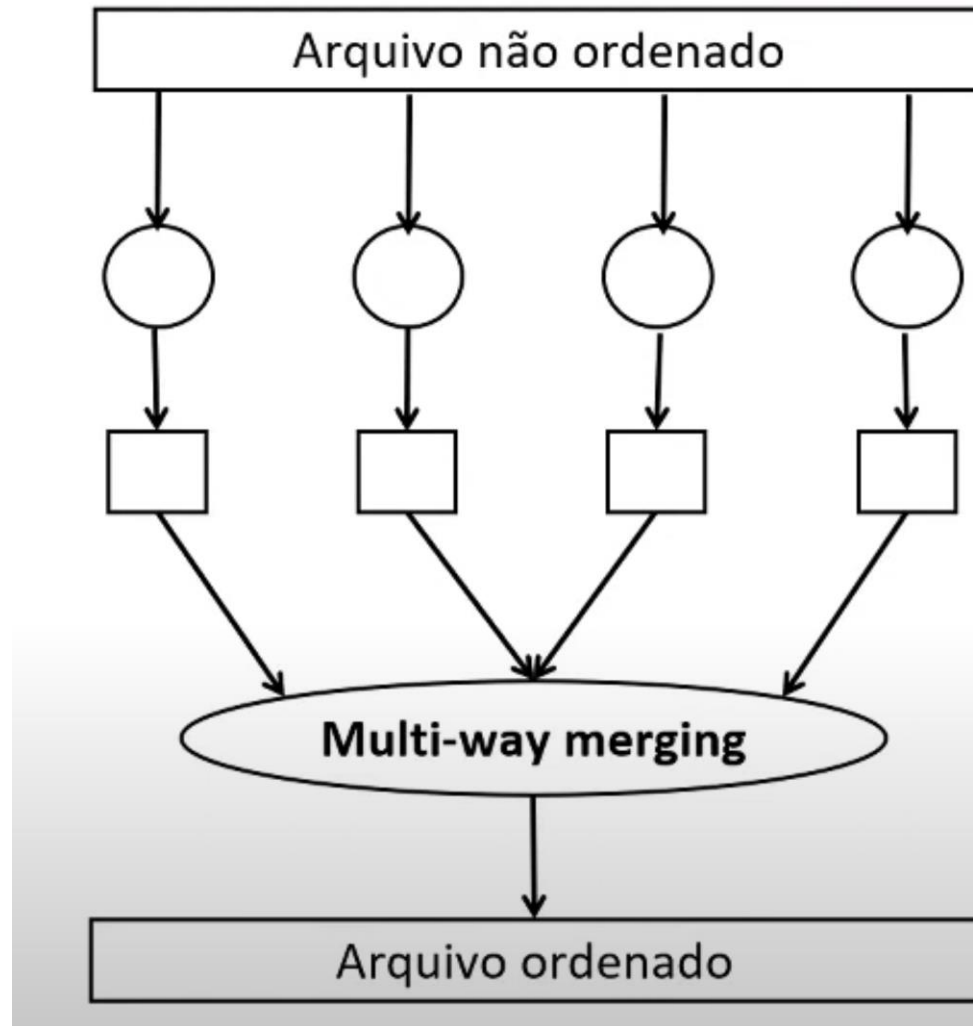
MergeSort Externo

- Funcionamento:
 - “Multi-way merging”
 - Criar “ $K+1$ ” buffers de tamanho “ $n/(k+1)$ ” (“1” de saída, “ k ” de entrada)
 - Carregar partes dos arquivos ordenados nos “buffers de entrada”, intercalar no “buffer de saída”
 - **Buffers de entrada “vazio”**: carregar mais dados

MergeSort Externo

- Funcionamento:
 - “Multi-way merging”
 - Criar “ $K+1$ ” buffers de tamanho “ $n/(k+1)$ ” (“1” de saída, “ k ” de entrada)
 - Carregar partes dos arquivos ordenados nos “buffers de entrada”, intercalar no “buffer de saída”
 - Buffers de entrada “vazio”: carregar mais dados
 - **Buffers de saída “cheio”: salvar dados**

MergeSort Externo



K ordenações na RAM

K arquivos ordenados

MergeSort Externo

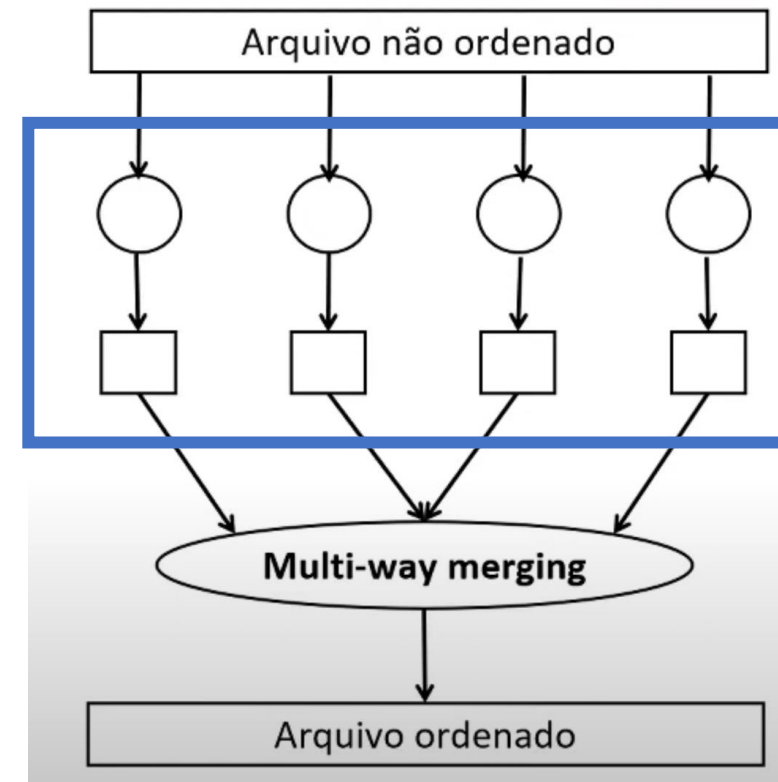
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define N 100
6  // Quantidade de valores que é possível ter na memória RAM
7  // #define K 10
8
9  void criArquivoTeste(char *nome) {
10     int i;
11     FILE *f = fopen(nome, "w");
12     srand(time(NULL));
13     for(i=1; i < 1000; i++)
14         fprintf(f, "%d\n", rand());
15     fprintf(f, "%d", rand());
16     fclose(f);
17 }
18
19 int main() {
20     criArquivoTeste("dados.txt");
21     system("pause");
22     mergeSortExterno("dados.txt");
23     verificaArquivoOrdenado("dados.txt");
24     system("pause");
25     return 0;
26 }
27
```

MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```



MergeSort Externo

Número de arquivos que serão criados

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

MergeSort Externo

Tamanho que a memória comporta

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

MergeSort Externo

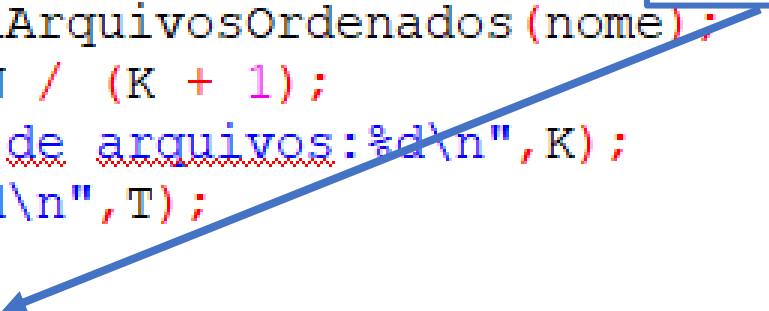
Nro de buffers

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

Remove o
arquivo original



MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

Mesmo nome
do arquivo

MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

Número de
arquivos
criados

MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

Tamanho do
Buffer



MergeSort Externo

```
- void mergeSortExterno(char *nome) {  
    char novo[20];  
    int K = criaArquivosOrdenados(nome);  
    int i, T = N / (K + 1);  
    printf("Nro de arquivos:%d\n", K);  
    printf("K:%d\n", T);  
  
    remove(nome);  
    merge(nome, K, T);  
  
    for(i=0; i<K; i++){  
        sprintf(novo, "Temp%d.txt", i+1);  
        remove(novo);  
    }  
    printf("Fim!\n");  
}
```

Depois do merge os
arquivos temporários
são removidos

MergeSort Externo

Controlar o número total de arquivos na memória RAM

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

Buffer cheio:
Salva em disco

Sobraram dados no
Buffer: Salva em disco

MergeSort Externo

Percorre o arquivo

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

MergeSort Externo

Controlar o número total de arquivos na memória RAM

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

Buffer cheio:
Salva em disco

Sobraram dados no
Buffer: Salva em disco

MergeSort Externo

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

Lê valor atual



MergeSort Externo

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

Ordena o Vetor com
qsort

MergeSort Externo

```
int criaArquivosOrdenados(char *nome) {
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)) {
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N) {
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0) {
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

Zera para o próximo
arquivo

MergeSort Externo

```
int criaArquivosOrdenados(char *nome){
    int V[N];
    char novo[20];
    int cont = 0, total = 0;
    FILE *f = fopen(nome, "r");
    while(!feof(f)){
        fscanf(f, "%d", &V[total]);
        total++;
        if(total == N){
            cont++;
            sprintf(novo, "Temp%d.txt", cont);
            qsort(V, total, sizeof(int), compara);
            salvaArquivo(novo, V, total, 0);
            total = 0;
        }
    }

    if(total > 0){
        cont++;
        sprintf(novo, "Temp%d.txt", cont);
        qsort(V, total, sizeof(int), compara);
        salvaArquivo(novo, V, total, 0);
    }
    fclose(f);
    return cont;
}
```

nro de registros, caso
não seja múltiplo

MergeSort Externo

```
void salvaArquivo(char *nome, int *V, int tam, int mudaLinhaFinal){  
    int i;  
    FILE *f = fopen(nome, "a");  
    for(i=0; i < tam-1; i++)  
        fprintf(f, "%d\n", V[i]);  
    if(mudaLinhaFinal == 0)  
        fprintf(f, "%d", V[tam-1]);  
    else  
        fprintf(f, "%d\n", V[tam-1]);  
    fclose(f);  
}
```

MergeSort Externo

```
void merge(char *nome, int K, int T){
    char novo[20];
    int i;
    int *buffer = (int*)malloc(T*sizeof(int));

    struct arquivo* arq;
    arq = (struct arquivo*)malloc(K*sizeof(struct arquivo));
    for(i=0; i < K; i++){
        sprintf(novo, "Temp%d.txt", i+1);
        arq[i].f = fopen(novo, "r");
        arq[i].MAX = 0;
        arq[i].pos = 0;
        arq[i].buffer = (int*)malloc(T*sizeof(int));
        preencheBuffer(&arq[i], T);
    }
```

```
//enquanto houver arquivos para processar
int menor, qtdBuffer = 0;
while(procuraMenor(arq, K, T, &menor) == 1){
    buffer[qtdBuffer] = menor;
    qtdBuffer++;
    if(qtdBuffer == T){
        salvaArquivo(nome, buffer, T, 1);
        qtdBuffer = 0;
    }
}

if(qtdBuffer != 0)
    salvaArquivo(nome, buffer, qtdBuffer, 1);

for(i=0; i<K; i++)
    free(arq[i].buffer);
free(arq);
free(buffer);
}
```

MergeSort Externo

```
//enquanto houver arquivos para processar
int menor, qtdBuffer = 0;
while (procuraMenor(arq, K, T, &menor) == 1) {
    buffer[qtdBuffer] = menor;
    qtdBuffer++;
    if (qtdBuffer == T) {
        salvaArquivo(nome, buffer, T, 1);
        qtdBuffer = 0;
    }
}

if (qtdBuffer != 0)
    salvaArquivo(nome, buffer, qtdBuffer, 1);

for (i=0; i<K; i++)
    free(arq[i].buffer);
free(arq);
free(buffer);
}
```

Se existe menor elemento?
Sim, coloca no buffer de saída

MergeSort Externo

```
//enquanto houver arquivos para processar
```

```
int menor, qtdBuffer = 0;
```

```
while (procuraMenor(arq, K, T, &menor) == 1) {
```

```
    buffer[qtdBuffer] = menor;
```

```
    qtdBuffer++;
```

```
    if (qtdBuffer == T) {
```

```
        salvaArquivo(nome, buffer, T, 1);
```

```
        qtdBuffer = 0;
```

```
    }
```

```
}
```

```
if (qtdBuffer != 0)
```

```
    salvaArquivo(nome, buffer, qtdBuffer, 1);
```

```
for (i=0; i<K; i++)
```

```
    free(arq[i].buffer);
```

```
free(arq);
```

```
free(buffer);
```

} qtdBuffer == T
Sim, então buffer cheio

MergeSort Externo

```
//enquanto houver arquivos para processar
int menor, qtdBuffer = 0;
while (procuraMenor(arq, K, T, &menor) == 1) {
    buffer[qtdBuffer] = menor;
    qtdBuffer++;
    if (qtdBuffer == T) {
        salvaArquivo(nome, buffer, T, 1);
        qtdBuffer = 0;
    }
}

if (qtdBuffer != 0)
    salvaArquivo(nome, buffer, qtdBuffer, 1);

for (i=0; i<K; i++)
    free(arq[i].buffer);
free(arq);
free(buffer);
}
```

} Sobraram dados, guarda no buffer

MergeSort Externo

```
int procuraMenor(struct arquivo* arq, int K, int T, int* menor) {
    int i, idx = -1;
    for(i=0; i<K; i++){
        if(arq[i].pos < arq[i].MAX){
            if(idx == -1)
                idx = i;
            else{
                if(arq[i].buffer[arq[i].pos] < arq[idx].buffer[arq[idx].pos])
                    idx = i;
            }
        }
    }
    if(idx != -1){
        *menor = arq[idx].buffer[arq[idx].pos];
        arq[idx].pos++;
        if(arq[idx].pos == arq[idx].MAX)
            preencheBuffer(&arq[idx], T);
        return 1;
    }else
        return 0;
}
```

Procura menor valor na primeira posição de cada buffer

MergeSort Externo

```
int procuraMenor(struct arquivo* arq, int K, int T, int* menor) {
    int i, idx = -1;
    for(i=0; i<K; i++){
        if(arq[i].pos < arq[i].MAX){
            if(idx == -1)
                idx = i;
            else{
                if(arq[i].buffer[arq[i].pos] < arq[idx].buffer[arq[idx].pos])
                    idx = i;
            }
        }
    }
    if(idx != -1){
        *menor = arq[idx].buffer[arq[idx].pos];
        arq[idx].pos++;
        if(arq[idx].pos == arq[idx].MAX)
            preencheBuffer(&arq[idx], T);
        return 1;
    }else
        return 0;
}
```

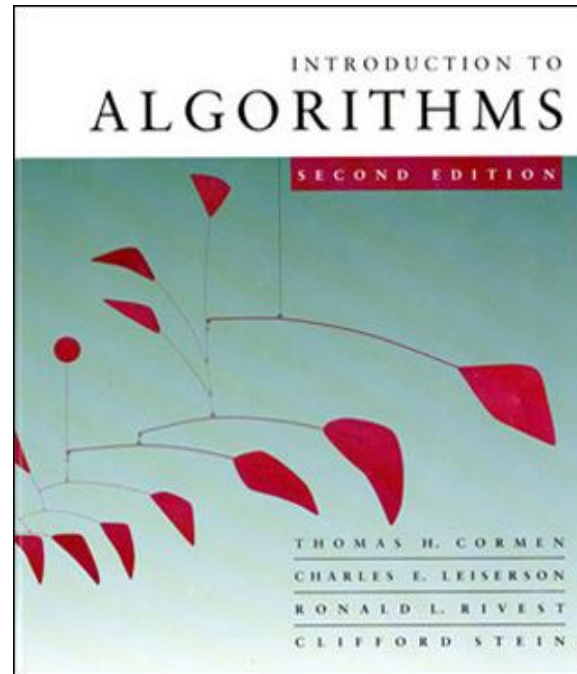
Achou menor, atualiza posição do buffer

Bibliografia

- <https://www.youtube.com/watch?v=0Whr9aOqgrI>

Leitura importante

livro “Algorithms” de Cormen et al.



Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br