**Nomes:** Jonas Antonio Gomes Vicente e Juan Ferreira Carlos

**Trabalho de Ordenação Externa**

**1 - Descrição dos Métodos:**

Ordenação Externa consiste em ordenar arquivos maiores do que a memória interna disponível possa suportar, onde os algoritmos devem buscar a redução do número de acessos às unidades de memória externa, relativamente muito mais lentas do que a memória principal, onde os registros são acessados sequencialmente ou em grandes blocos

**1.1 - Intercalação Balanceada de Vários Caminhos:**

Consiste em um método de ordenação em que se divide o arquivo que se quer ordenar de acordo com o limite de memória interna do computador, ou seja, se o computador possuir espaço para 4 registros e a fita magnética possuir tamanho 12**,** na primeira etapa leremos 4 registros e o armazenaremos em fitas e continuaremos o processo até que na segunda etapa iremos intercalar estes registros na memória interna até a terceira etapa onde teremos 3 fitas ordenadas, repetiremos o processo até obtermos a ordenação completa dos registros.

**1.2 - Intercalação Polifásica:**

É um método de ordenação criado para resolver os problemas da Intercalação Balanceada de Vários Caminhos, a qual necessita de um grande número de fitas, portanto um número maior de leituras e escritas entre as fitas envolvidas, além do custo de criar uma cópia adicional do arquivo. O funcionamento da Intercalação Polifásica ocorre da seguinte maneira: Os blocos ordenados são distribuídos de forma desigual entre as fitas disponíveis de modo que uma fita fique disponível. Logo, ocorre a execução da intercalação de blocos ordenados até que uma das fitas de entrada fique vazia. Tal fita torna-se a próxima fita de saída. A intercalação é realizada em várias fases, as quais não envolvem todos os blocos, porém nenhuma cópia é realizada diretamente entre as fitas.

**1.3 - Quicksort Externo:**

Este método de ordenação consiste no paradigma dividir para conquistar, onde se divide o arquivo e se armazena os registros em memória secundária de acesso randômico, utilizando-se apenas da memória interna para a ordenação, não se fazendo necessário o uso de memória externa. Consideremos que há um arquivo A={R1, … , RN} onde, (1 <= i <=N), Ri é o registro que se encontra na i-ésima posição de A, o primeiro passo é particionar A em: {R1 , … , Ri} <= Ri+1 <= Ri+2 <= … <= Rj-2 <= Rj-1 <= (Rj , … , RN) , utilizando-se de um Buffer na memória interna de tamanho K = j-i+1, com K>=3. É extremamente eficiente e necessita apenas de uma pequena pilha como memória auxiliar.

**2 - Atividades :**

**2.1 - Código Fonte do Programa em C dos exercícios 1 e 2 página 20 do slide de nº 11:**

#include <stdio.h>

#include <stdlib.h>

#define Maxn 1000

typedef int TipoChave;

typedef struct Registro {

TipoChave Chave;

} Registro;

typedef int Indice;

typedef struct Tabela {

Registro Item[Maxn + 1];

Indice n;

} Tabela;

void Inicializa(Tabela \*T) {

T->n=0;

}

Indice Pesquisa(TipoChave x, Tabela \*T) {

int i;

T->Item[0].Chave = x;

i = T->n+1;

do{

i--;

}while(T->Item[i].Chave != x);

if(i==0){

printf("Valor %d nao existe na tabela!\n", x);

}else {

printf("Valor %d existe na posição %d da tabela!\n", x, i);

}

return i;

}

Indice PesquisaMais(TipoChave x, Tabela \*T) {

int i;

T->Item[0].Chave = x;

i = T->n+1;

if(i==0){

printf("Valor %d nao pode ser encontrado!\n", x);

} else {

while(i>0){

if(T->Item[i].Chave == x){

printf("Valor %d encontrado na posicao [%d]!\n", x, i);

}

i--;

}

}

return i;

}

void Insere(Registro reg, Tabela \*T) {

if(T->n == Maxn) {

printf("Erro: tabela repleta\n");

exit(1);

}

else {

T->n++;

T->Item[T->n] = reg;

printf("Valor %d inserido na tabela!\n", reg.Chave);

}

}

void Remove(TipoChave x, Tabela \*T) {

int i = Pesquisa(x, T);

if(i) {

T->Item[i] = T->Item[T->n];

T->n -= 1;

printf("Valor %d removido da tabela!\n", x);

}

}

int main(){

Tabela Tab;

Inicializa(&Tab);

Registro reg;

int menu = 1;

do {

printf("\n\t ----MENU---- \n");

printf("\t1 - Inserir\n");

printf("\t2 - Remover\n");

printf("\t3 - Pesquisar\n");

printf("\t4 - Pesquisar (2ou+)\n");

printf("\t0 - Sair\n");

printf("\t ------------ \n\n");

scanf("%d", &menu);

system("cls || clear");

switch(menu) {

case 1:

scanf("%d", &reg.Chave);

Insere(reg, &Tab);

break;

case 2:

scanf("%d", &reg.Chave);

Remove(reg.Chave, &Tab);

break;

case 3:

scanf("%d", &reg.Chave);

Pesquisa(reg.Chave, &Tab);

break;

case 4:

scanf("%d", &reg.Chave);

PesquisaMais(reg.Chave, &Tab);

break;

case 0:

break;

default:

printf("\nDigite uma opcao valida\n");

}

} while(menu);

}

**2.2 - Código Fonte do Programa em C dos exercícios 1 e 2 página 29 do slide de nº 11:**

#include <stdio.h>

#include <stdlib.h>

#define Maxn 1000

typedef char TipoChave;

typedef struct Registro {

TipoChave Chave;

} Registro;

typedef int Indice;

typedef struct Tabela {

Registro Item[Maxn + 1];

Indice n;

} Tabela;

void Inicializa(Tabela \*T) {

T->n=0;

}

Indice Pesquisa(TipoChave x, Tabela \*T) {

int i;

T->Item[0].Chave = x;

i = T->n+1;

do{

i--;

}while(T->Item[i].Chave != x);

if(i==0){

printf("Valor %c nao existe na tabela!\n", x);

}else {

printf("Valor %c existe na posição %d da tabela!\n", x, i);

}

return i;

}

Indice Binaria(TipoChave x, Tabela \*T) {

Indice i, esq, dir;

if(T->n == 0) {

printf("Valor %c nao existe na tabela!\n", x);

return 0;

} else {

esq = 1;

dir = T->n;

do{

i = (esq+dir)/2;

if(x > T->Item[i].Chave){

esq = i + 1;

} else {

dir = i - 1;

}

}while(x != T->Item[i].Chave && esq<=dir);

if(x == T->Item[i].Chave) {

printf("Valor %c existe na posição %d da tabela!\n", x, i);

return i;

} else {

return 0;

}

}

}

void Insere(Registro reg, Tabela \*T) {

if(T->n == Maxn) {

printf("Erro: tabela repleta\n");

exit(1);

}

else {

T->n++;

T->Item[T->n] = reg;

printf("Valor %c inserido na tabela!\n", reg.Chave);

}

}

void Remove(TipoChave x, Tabela \*T) {

int i = Pesquisa(x, T);

if(i) {

T->Item[i] = T->Item[T->n];

T->n -= 1;

printf("Valor %c removido da tabela!\n", x);

}

}

int main(){

Tabela Tab;

Inicializa(&Tab);

Registro reg;

int menu = 1;

do {

printf("\n\t ----MENU---- \n");

printf("\t1 - Inserir\n");

printf("\t2 - Remover\n");

printf("\t3 - Pesquisa Sequencial\n");

printf("\t4 - Pesquisa Binaria\n");

printf("\t0 - Sair\n");

printf("\t ------------ \n\n");

scanf("%d", &menu);

system("cls || clear");

switch(menu) {

case 1:

scanf(" %c", &reg.Chave);

Insere(reg, &Tab);

break;

case 2:

scanf(" %c", &reg.Chave);

Remove(reg.Chave, &Tab);

break;

case 3:

scanf(" %c", &reg.Chave);

Pesquisa(reg.Chave, &Tab);

break;

case 4:

scanf(" %c", &reg.Chave);

Binaria(reg.Chave, &Tab);

break;

case 0:

break;

default:

printf("\nDigite uma opcao valida\n");

}

} while(menu);

}