**Trabalho de Estrutura de Dados**

Contemplando os 3 pontos que irão compor a nota da AV da disciplina de estrutura de dados, os alunos poderão escolher 1 entre 3 temas descritos a seguir para entregar como trabalho até a data definida no tópico deste trabalho.

***A – Ordenação***

Escolher um método de ordenação diferente dos 3 métodos já apresentados em aula (insert/select/bubble) e:

1 – Acrescentar o novo método no esqueleto de ordenação fornecido pelo professor no ANEXO 1

2 – Fazer o algoritmo contar a quantidade de passos realizados para a tarefa

3 – Produzir 2 ordens iniciais da coleção, uma para que cada algoritmo ganhe, e explicar o motivo das vitórias.

***B – Hashing***

Utilizando o esqueleto fornecido no ANEXO 2 com a função de Hash básica, fazer:

1 – Modificar a função original para obter uma melhor taxa de colisão

2 – Mostrar os dados utilizados nas 2 funções (original e alterada)

3 – Mostrar o cálculo de colisão para a função criada sobre os dados utilizados.

***C – Árvores Binárias***

Utilizando o material do conteúdo digital e o esqueleto fornecido no ANEXO 3, produzir:

1 – Um programa que leia uma árvore binária de busca de até 5 níveis

2 e 3 – O programa deve fornecer uma função para visualizar a árvore de maneira gráfica e proporcional na tela do terminal

O trabalho deve ser entregue até 4/6, dia da AV no calendário acadêmico seguindo a estrutura definida no ANEXO 4, em formato PDF.

**ANEXO 1**

#include <iostream>

#include <cstdlib>

using namespace std;

void inserir(int v[], int &n, int e){

if (40 == n){

cout << "ERRO: Lista cheia.";

}else{

v[n] = e;

n++;

}

}

void percorrer(int v[], int n){

for (int i = 0; i < n; i++)

cout << v[i] << " ";

}

bool troca (int v[], int a, int b){

int aux;

aux = v[a];

v[a] = v[b];

v[b] = aux;

return true;

}

int ordenaBolha(int v[], int n){

bool trocou = true;

int i, fim = n-1, aux = 6;

while (trocou){

trocou = false;

aux = aux + 2;

for (i = 0; i < fim; i++){

aux = aux + 3;

if (v[i] > v[i+1]){

trocou = troca(v,i,(i+1));

aux = aux + 5;

}

}

fim--;

aux = aux + 1;

} // fim while

return aux;

}// fim da função

int ordenaSelecao(int v[], int n){

int i, j, menor, aux = 4;

for (j = 0; j < n-1; j++){

aux = aux + 2;

menor = j;

aux = aux + 1;

for (i = j+1; i < n; i++){

aux = aux + 3;

if (v[i] < v[menor]){

menor = i;

aux = aux + 1;

}

}

troca(v,j,menor);

aux = aux + 5;

}

return aux;

}

int ordenaInsercao(int v[], int n){

int i, j, aux = 3;

for (j = 1; j < n; j++){

aux = aux + 2;

for (i=j; i > 0 && v[i-1]> v[i]; i--){

aux = aux + 4;

troca(v,i,(i-1));

aux = aux + 5;

}

}

return aux;

}

// Prog. Principal

int main(){

int v[40], tam = 0, i, temp, temp2=0;

int v2[40], tam2 = 0;

int v3[40], tam3 = 0;

cout << "\nRotina de leitura da colecao:\n\n";

for (i = 0; i < 10; i++){

cout << "Digite um numero: ";

cin >> temp;

inserir(v, tam, temp);

inserir(v2, tam2, temp);

inserir(v3, tam3, temp);

}

cout << "\n\n I N S E R T S O R T";

cout << "\n Imprimindo antes de ordenar C1: ";

percorrer(v,tam);

temp = ordenaInsercao(v,tam);

cout << "\n Imprimindo depois de ordenar C1: ";

percorrer(v,tam);

cout << "\n Precisei de " << temp << " operacoes para oredenar...\n\n";

cout << "\n\n S E L E C T S O R T";

cout << "\n Imprimindo antes de ordenar C2: ";

percorrer(v2,tam2);

temp = ordenaSelecao(v2,tam2);

cout << "\n Imprimindo depois de ordenar C2: ";

percorrer(v2,tam2);

cout << "\n Precisei de " << temp << " operacoes para oredenar...\n\n";

cout << "\n\n B U B L E S O R T";

cout << "\n Imprimindo antes de ordenar C3: ";

percorrer(v3,tam3);

temp2 = ordenaBolha(v3,tam3);

cout << "\n Imprimindo depois de ordenar C3: ";

percorrer(v3,tam3);

cout << "\n Precisei de " << temp2 << " operacoes para oredenar...\n\n";

system("pause");

return 0;

}

**ANEXO 2**

#include <iostream>

#include <cstdlib>

#include <cstring>

#define TABLESPACE 2000

using namespace std;

struct agenda{

bool livre;

char nome[50];

char enredeco[50];

char tel[10];

};

void iniciaAgenda (agenda a[]){

int i;

for (i=0; i<TABLESPACE; i++){

a[i].livre = true;

}

}

int minhaHash(char \*chave){

return (chave[i]-65);

}

int main(int argc, char\* argv[]){

int i;

agenda a[TABLESPACE];

char nome[50];

iniciaAgenda(a);

int op=1, pos;

while (op != 0){

cout << "Digite um nome: ";

cin >> nome;

pos = minhaHash(nome);

if (a[pos].livre){

cout << "Esse nome foi armazenado na posicao ";

cout << pos << ".\n";

strcpy(nome,a[pos].nome);

a[pos].livre=false;

cout << "Continuar? [0 - Nao / 1 - Sim] ";

cin >> op;

} else {

cout << "Houve colisao na posicao ";

cout << pos << ".\n";

cout << "Nome alocado: ";

cout << a[pos].nome;

op = 0;

}

}

return 0;

}

**ANEXO 3**

#include <iostream>

using namespace std;

struct no{

int valor;

struct no \*esq;

struct no \*dir;

};

struct no \*raiz;

void iniciaArvore ( ){

cout << "\nArvore Binaria Iniciada!\n";

raiz = NULL;

}

bool testaArvoreVazia (){

return (raiz == NULL);

}

void insereArvore(int num){

struct no \*pt, \*b;

pt = new struct no;

pt -> valor = num;

pt -> esq = NULL;

pt -> dir = NULL;

if(testaArvoreVazia()){

cout << "Inserido na raiz !\n";

raiz = pt;

}else {

b = raiz;

while (b != NULL){

cout << "\nAnalisando no " << b->valor << "...\n";

if(num < b->valor){

if ( b->esq == NULL ){

cout << "Achei o local de inclusão!";

b->esq = pt;

b = NULL;

} else {

cout << "Buscando local a esquerda ...\n";

b = b->esq;

}

}else{

if ( b->dir == NULL ){

cout << "Achei o local de inclusão!";

b->dir = pt;

b = NULL;

} else {

cout << "Buscando local a esquerda ...\n";

b = b->dir;

}

}

}

}

cout << "Elemento " << num << " inserido... \n";

}

void erd (struct no \*r) {

if (r != NULL) {

erd (r->esq);

cout << "\n" << r->valor;

erd (r->dir);

}

}

void red (struct no \*r) {

if (r != NULL) {

cout << "\n" << r->valor;

red (r->esq);

red (r->dir);

}

}

void edr (struct no \*r) {

if (r != NULL) {

red (r->esq);

red (r->dir);

cout << "\n" << r->valor;

}

}

int main ( ){

int i,temp;

iniciaArvore();

for (i=0; i<7; i++){

cout << "\n Informe um numero: ";

cin >> temp;

insereArvore(temp);

}

cout << "\n\n percurso Em Ordem: \n";

erd(raiz);

cout << "\n\n percurso Pre Ordem: \n";

red(raiz);

cout << "\n\n percurso pos Ordem: \n";

edr(raiz);

system("PAUSE");

return 0;

}

**ANEXO 4**

ALUNO:

MATRÍCULA:

TEMA ESCOLHIDO:

CÓDIGO GERADO:

EXPLICAÇÃO DO ITEM 1 DO TRABALHO:

EXPLICAÇÃO DO ITEM 2 DO TRABALHO:

EXPLICAÇÃO DO ITEM 3 DO TRABALHO:

EVIDÊNCIAS EXTRAS SE NECESSÁRIO: