**Curso**: Superior em Sistemas Para Internet **Disciplina:** Estrutura de Dados II

Nome:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Sem.:\_\_\_\_\_\_\_\_\_\_\_\_\_Nota:\_\_\_\_\_\_

|  |
| --- |
| ***Competências e/ou habilidades:***  Recursividade. Árvores. Grafos. Tabelas de Dispersão. |
| ***Critérios de avaliação:***  A solução proposta, na forma de escolha objetiva, deve seguir as regras ditadas pelo enunciado do problema e fornecer, corretamente, os resultados solicitados. |
| **Avaliação:** |

**Atividade 01 - Recursividade (2,0 Pontos)**

Considerando o treco de programa abaixo, escrito em linguagem C, qual será a saída gerada das funções dos algoritmos abaixo, considerando o valor inicial de n=4:

a) func (int n)

{

if (n == 0)

printf(“fim”);

else

{

printf(n);

func(n-1);

}

}

Saída: 4 3 2 1 fim

b) func (int n)

{

if (n == 0)

printf(“fim”);

else

{

func(n-1);

printf(n);

}

}

Saída: fim 1 2 3 4

c) func (int n)

{

if (n == 0)

printf(“fim”);

else

{

printf(n);

func(n-1);

printf(n);

}

}

Saída: 4 3 2 1 fim 1 2 3 4

d) func (int n)

{

if (n == 0)

printf(“fim”);

else

{

func(n-1);

printf(n);

func(n-1);

}

}

Saída:

**Atividade 02 - Recursividade (1,0 Pontos)**

Considerando o treco de programa abaixo, escrito em linguagem C, qual será a saída do programa abaixo:

void move(int n, char a, char b, char c){

if(n > 0)

{

move(n-1,a,c,b);

printf("mover de %c para %c\n",a,b);

move(n-1,c,b,a);

}

}

main()

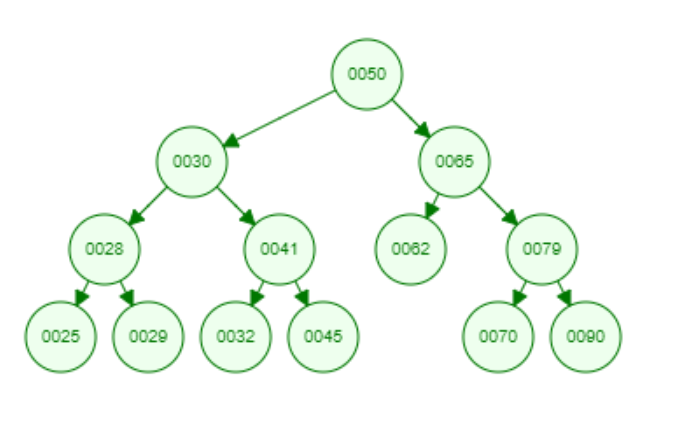
{

move(3, 'X','Y','Z');

}

**Atividade 03 - Árvores (2,0 Pontos)**

Considerando a Árvore Binária abaixo, Qual será a saída do utilizando os encaminhamentos:



void Caminhamento\_Pre\_Ordem(TArvore \*a){

if (!Vazia(a)) {

printf("%c ", a->info);

Caminhamento\_Pre\_Ordem(a->esq);

Caminhamento\_Pre\_Ordem(a->dir); sub\_dir }

}

void Caminhamento\_In\_Fixado(TArvore \*a){

if (!Vazia(a)) {

Caminhamento\_In\_Fixado(a->esq);

printf("%c ", a->info); Caminhamento\_In\_Fixado(a->dir); }

}

void Caminhamento\_Pos\_Fixado(TArvore \*a){

if (!Vazia(a)) {

Caminhamento\_Pos\_Fixado(a->esq);

Caminhamento\_Pos\_Fixado(a->dir);

printf("%c ", a->info); }

}

**Atividade 04 - Árvores (2,0 Pontos)**

Considerando a o trecho do programa abaixo referente a criação de Árvores Binárias, Represente graficamente a estrutura gerada pelo programa.

typedef struct Nodo {

struct Nodo \*esq;

char info;

struct Nodo \*dir;

} TNodo;

typedef TNodo TArvore;

TArvore \*Cria(TArvore \*esq, char info, TArvore\* dir){

TArvore \*p;

p = (TArvore\*) malloc(sizeof(TArvore));

if (p == NULL)

{

printf("ERRO FATAL: Falta de Memória\n");

getchar();

exit(0);

}

else{

p->info = info;

p->esq = esq;

p->dir = dir;

}

return p;

}

int main(void){

TArvore \*a0,\*a1,\*a2,\*a3,\*a4,\*a5,\*a6, \*a7;

a1 = Cria(NULL,'H',NULL);

a2 = Cria(NULL,'B',NULL);

a3 = Cria(a1,'C',a2);

a4 = Cria(NULL,'E',NULL);

a5 = Cria(a3,'F',a4);

a6 = Cria(NULL,'G',NULL);

a0 = Cria(a5,'A',a6);

return(0);

}

**Atividade 05 - Grafos (2,0 Pontos)**

Considerando o programa abaixo, represente graficamente o Grafo Gerado:

#define MAX\_VERTICES 7

#define LIMITE 32767

#define TRUE !0

#define FALSE 0

char visitadoOuNao[2][15] = { "Nao Visitado", "Visitado" };

const int infinito = 999999;

char cidades[MAX\_VERTICES][30] = {

"[0] - (PoA) - Porto Alegre",

"[1] - (Spa) - Sao Paulo",

"[2] - (Rio) - Rio de janeiro",

"[3] - (Vit) - Vitoria",

"[4] - (Rec) - Recife",

"[5] - (Sal) - Salvador",

"[6] - (Nat) - Natal"};

**typedef struct**{

char cidade\_adj;

int cidade\_dist;

} Tcidade;

char rotulos[MAX\_VERTICES][6] = { "PoA", "SPa", "Rio", "Vit", "Rec", "Sal", "Nat" },opcao= 'S';

**void inicializaGrafo (Tcidade grafo[][MAX\_VERTICES**]) {

int l, c;

for (l = 0; l < MAX\_VERTICES; l++)

for (c = 0; c < MAX\_VERTICES; c++)

{

grafo[l][c].cidade\_dist = -1;

grafo[l][c].cidade\_adj = 'N';

}

}

**void criaAresta (Tcidade grafo[][MAX\_VERTICES], int cid\_origem, int cid\_destino, int distancia)** {

grafo[cid\_origem][cid\_destino].cidade\_adj = 'S';

grafo[cid\_origem][cid\_destino].cidade\_dist = distancia;

}

**int buscarVerticeMinimo (int distancia[], bool visitados[])** {

int distanciaMinima = infinito, verticeMinimo;

for (int i = 0; i < MAX\_VERTICES; i++)

{

if (!visitados[i] && distancia[i] <= distanciaMinima)

{

distanciaMinima = distancia[i];

verticeMinimo = i;

}

}

return verticeMinimo;

}

**void imprimirCaminho (int menorCaminho[], int destino)** {

if (menorCaminho[destino] == -1)

{

return;

}

imprimirCaminho (menorCaminho, menorCaminho[destino]);

printf ("->%s(%d)", rotulos[destino], destino);

}

**void leOrigemDestino (int tam, int \*origem, int \*destino)** {

printf ("\n Cidade de Origem [0..%d]: ", tam);

do

{

scanf ("%d", origem);

}

while (\*origem < 0 || \*origem > tam);

printf (" Cidade de Destino [0..%d]: ", tam);

do

{

scanf ("%d", destino);

}

while (\*destino < 0 || \*destino > tam);

}

**void imprimirResultado (int distancia[], int menorCaminho[], int origem, int destino)** {

printf ("\n Caminho mais curto da origem %s ao destino %s:\n",

rotulos[origem], rotulos[destino]);

printf (" Distancia: %d ", distancia[destino]);

printf ("\n Caminho: %s(%d)", rotulos[origem], origem);

imprimirCaminho (menorCaminho, destino);

printf ("\n");

}

**void imprimeMenu (void)** {

int i;

printf ("-- Menu Opcoes de Cidades --\n");

for (i = 0; i < MAX\_VERTICES; i++)

printf (" %s\n", cidades[i]);

}

**void verificarDistanciaEntreCidades (Tcidade grafo[][MAX\_VERTICES], int origem, int destino)** {

int distancia[MAX\_VERTICES];

bool visitados[MAX\_VERTICES];

int menorCaminho[MAX\_VERTICES];

int contador = MAX\_VERTICES;

for (int i = 0; i < MAX\_VERTICES; i++)

{

distancia[i] = infinito;

visitados[i] = false;

menorCaminho[i] = -1;

}

distancia[origem] = 0;

while (contador != 0)

{

int verticeAtual = buscarVerticeMinimo (distancia, visitados);

visitados[verticeAtual] = true;

for (int l = 0; l < MAX\_VERTICES; l++)

{

bool naoFoiVisitado = !visitados[l];

bool ehCidadeVizinha = grafo[verticeAtual][l].cidade\_adj == 'S';

if (naoFoiVisitado && ehCidadeVizinha)

{

menorCaminho[l] = verticeAtual;

distancia[l] = distancia[verticeAtual] + grafo[verticeAtual][l].cidade\_dist;

}

}

contador--;

}

imprimirResultado (distancia, menorCaminho, origem, destino);

}

Tcidade grafo[MAX\_VERTICES][MAX\_VERTICES];

int precede[MAX\_VERTICES];

int origem, destino;

**int main (void)**

{

inicializaGrafo (grafo);

criaAresta (grafo, 0, 1, 1100);

criaAresta (grafo, 0, 2, 1700);

criaAresta (grafo, 1, 4, 2300);

criaAresta (grafo, 1, 5, 1500);

criaAresta (grafo, 2, 0, 1700);

criaAresta (grafo, 2, 6, 2700);

criaAresta (grafo, 2, 3, 300);

criaAresta (grafo, 3, 2, 300);

criaAresta (grafo, 3, 4, 800);

criaAresta (grafo, 4, 1, 2300);

criaAresta (grafo, 4, 3, 800);

criaAresta (grafo, 4, 6, 650);

criaAresta (grafo, 5, 1, 1500);

criaAresta (grafo, 5, 6, 1900);

criaAresta (grafo, 6, 5, 1900);

criaAresta (grafo, 6, 4, 650);

criaAresta (grafo, 6, 2, 2700);

imprimeMenu ();

leOrigemDestino (MAX\_VERTICES - 1, &origem, &destino);

verificarDistanciaEntreCidades (grafo, origem, destino);

return (0);

}

**Atividade 06 - Tabelas de Dispersão (2,0 Pontos)**

Considerando o trecho dos programas abaixo, faça um quadro comparativo que qual estratégia gerará uma estrutura mais distribuída,

**int hash (int mat)**

**{**

**return (mat%N);**

**}**

**int hash2 (int mat)**

**{**

**return (N - 2 - x%(N - 2));**

**}**

a) Considerando N = 100 e x=99, qual será o Índice?

hash=

hash2=

b) Considerando N = 100 e x=199, qual será o Índice?

hash=

hash2=

c) Considerando N = 100 e x=100, qual será o Índice?

hash=

hash2=

d) Considerando N = 100 e x=101, qual será o Índice?

hash=

hash2=

e) Considerando N = 100 e x=102, qual será o Índice?

hash=

hash2=