**Aluno:** Pedro Henrique Mendes Pereira

**Matrícula:** 120060006

**Universidade:** UFF – Rio das Ostras

**Curso:** Ciência da Computação

**Disciplina:** Estruturas de Dados

**Período:** 2020.2

**Professor:** Dalessandro Soares Vianna

**Data de Entrega:** 30/03/2021

**Prova I**

**1)**

// A função recebe duas arvores. A função altura é a mesma utilizada nas atividades.

bool VerificarSimilaridade(struct *arvore* \**arvore1*, struct *arvore* \**arvore2*) {

if(*arvore1* != NULL && *arvore2* != NULL) {

bool igualdade = false;

int alturaEsq1 = AlturaArvore(*arvore1*->esq);

int alturaDir1 = AlturaArvore(*arvore1*->dir);

int alturaEsq2 = AlturaArvore(*arvore2*->esq);

int alturaDir2 = AlturaArvore(*arvore2*->dir);

if(alturaDir1 == alturaDir2 && alturaEsq1 == alturaEsq2) {

igualdade = VerificarSimilaridade(*arvore1*->esq, *arvore2*->esq);

if(igualdade == false) {

return false;

}

igualdade = VerificarSimilaridade(*arvore1*->dir, *arvore2*->dir);

if(igualdade == false) {

return false;

}

return true;

}

else {

return false;

}

}

else if(*arvore1* != NULL) {

return false;

}

else if(*arvore2* != NULL) {

return false;

}

else {

return true;

}

}

**2)**

// A função recebe um ponteiro contador, iniciado com zero, que vai armazenar a quantidade de nos entre X e Y.

void NosEntreXeY(struct *arvore* \**arvore*, int *x*, int *y*, int \**contador*) {

if(*arvore* != NULL) {

if(*arvore*->info < *x*) {

NosEntreXeY(*arvore*->dir, *x*, *y*, *contador*);

}

else if(*arvore*->info > *y*) {

NosEntreXeY(*arvore*->esq, *x*, *y*, *contador*);

}

else {

*contador*[0]++;

NosEntreXeY(*arvore*->dir, *x*, *y*, *contador*);

NosEntreXeY(*arvore*->esq, *x*, *y*, *contador*);

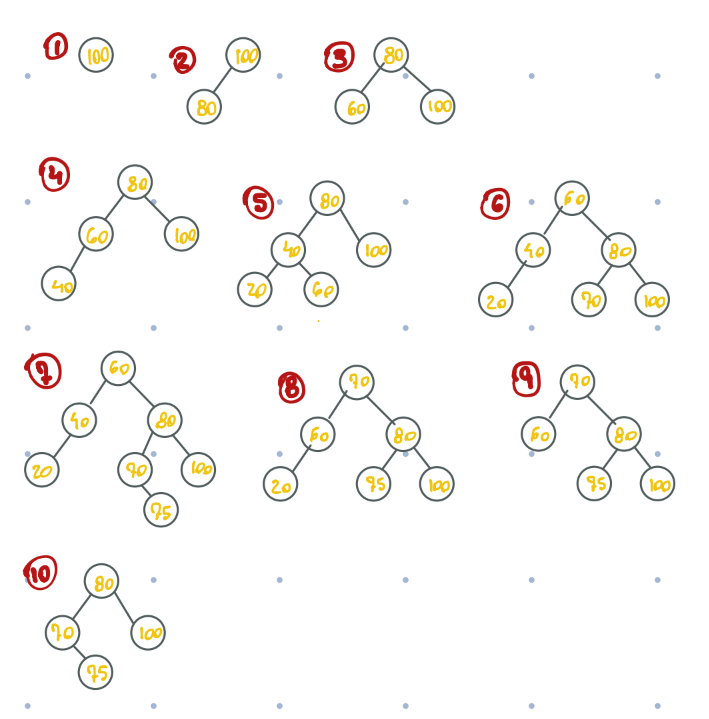
}

}

}

**3)**

**.**



.

**4)**

// A função recebe um vetor com os elementos de 1 até n, um vetor booleano setado com false do índice 0 até o índice n – 1, o k começa com 0 e o n é o tamanho do vetor de elementos.

void SubconjutosSomaImpar(int \**vetor*, int \**vetorBool*, int *k*, int *n*) {

if(*k* == *n*) {

if(SomarElementos(*vetor*, *vetorBool*, *n*) % 2 != 0) {

printf("{");

int i;

for(i = 0; i < *n*; i++) {

if(*vetorBool*[i] == true) {

printf(" %d ", *vetor*[i]);

}

}

printf("}\n");

}

}

else {

*vetorBool*[*k*] = true;

SubconjutosSomaImpar(*vetor*, *vetorBool*, *k* + 1, *n*);

*vetorBool*[*k*] = false;

SubconjutosSomaImpar(*vetor*, *vetorBool*, *k* + 1, *n*);

}

}

int SomarElementos(int \**vetor*, int \**vetorBool*, int *tamanho*) {

int i;

int soma = 0;

for(i = 0; i < *tamanho*; i++) {

if(*vetorBool*[i] == true) {

soma += *vetor*[i];

}

}

return soma;

}

**5)**

// A variável “quantidade” é definida fora na função valendo “0” na posição zero. Ela armazena a quantidade de caminhos com custo menor do que o valor passado para a função.

struct *caminho* {

int destino;

int custo;

};

void CaminhosAteCusto(struct *lista* \*\**grafo*, int *destino*, struct *caminho* \**vetor*, int *posicao*, int *valor*, int \**quantidade*) {

if(*vetor*[*posicao* - 1].destino == *destino*) {

int custoFinal = 0;

int i;

for(i = 1; i < *posicao*; i++) {

custoFinal += *vetor*[i].custo;

}

if(custoFinal < *valor*) {

*quantidade*[0]++;

}

}

else {

struct *lista* \*ponteiro = *grafo*[*vetor*[*posicao* - 1].destino];

while(ponteiro != NULL) {

if(!Existe(*vetor*, ponteiro->destino, *posicao*)) {

*vetor*[*posicao*].destino = ponteiro->destino;

*vetor*[*posicao*].custo = ponteiro->custo;

CaminhosAteCusto(*grafo*, *destino*, *vetor*, *posicao* + 1, *valor*, *quantidade*);

}

ponteiro = ponteiro->proximo;

}

}

}

bool Existe(struct *caminho* \**vetor*, int *valor*, int *n*) {

int i;

for(i = 0; i < *n*; i++) {

if(*vetor*[i].destino == *valor*) {

return true;

}

}

return false;

}