3º PERÍODO DE ENGENHARIA DE SOFTWARE ANO 2020

ESTRUTURA DE DADOS

PROFESSOR: FÁBIO GARCEZ BETTIO

ESTUDANTE: CLÍSTENES GRIZAFIS BENTO

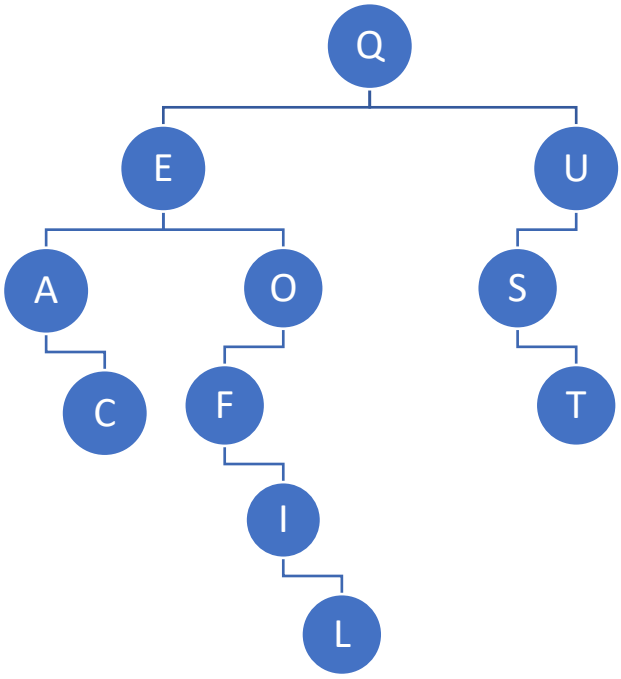
APS 3 – ÁRVORE BINÁRIA DE BUSCA

1) Sobre árvores binárias de pesquisa:

a) Qual a principal propriedade deste tipo árvore?

R: A principal propriedade e vantagem de uma arvore de pesquisa binária é o critério de ordenação de dados, que permite melhor busca entre os valores armazenados.

b) Desenhe a árvore binária de pesquisa que resulta da inserção sucessiva das chaves QUESTAOFCIL em uma árvore inicialmente vazia.



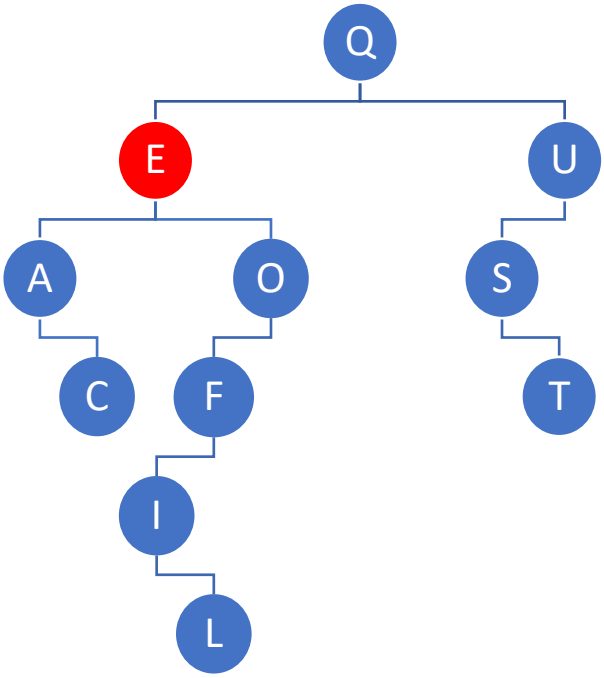
c) Represente a árvore sob a forma de matriz. (slide 22).

Índice	Info	Esq.	Dir.
0	Q	E	U
1	U	S	-1
2	E	A	O
3	S	-1	T
4	T	-1	-1
5	A	-1	C
6	O	F	-1
7	F	-1	I
8	C	-1	-1
9	I	-1	L
10	L	-1	-1

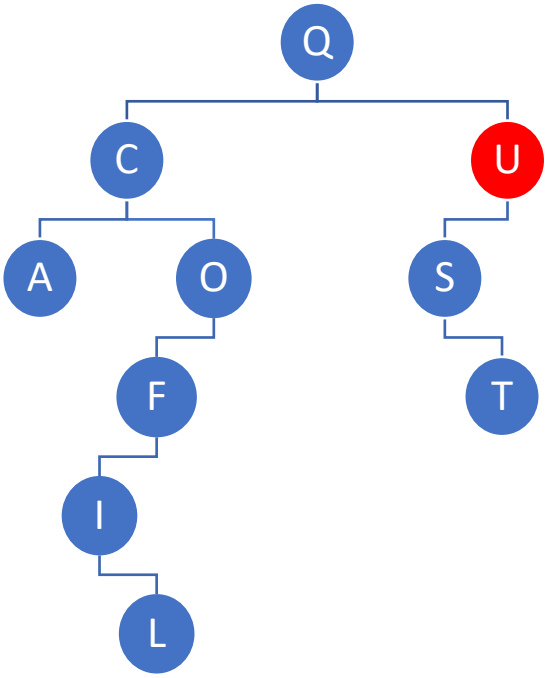
d) Anulada

e) Desenhe as árvores resultantes das retiradas dos elementos E e depois U da árvore do item anterior por cópia.

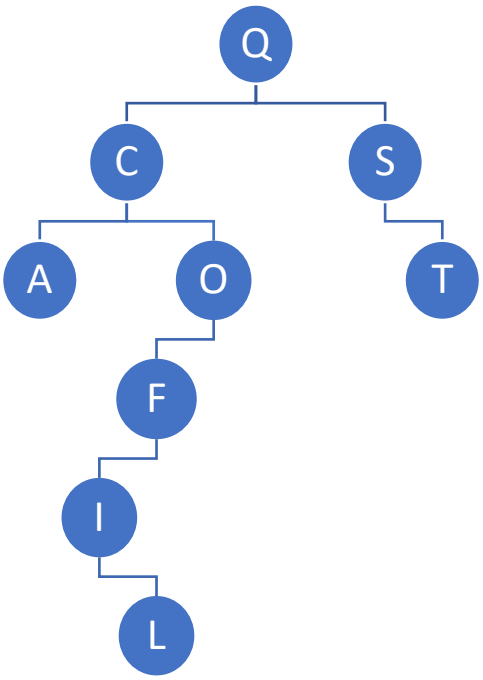
1º



2º



3º



2) Elaborar um programa que apresente o seguinte menu:

- Inserção em árvore binária
- Remoção em árvore binária
- Apresentação da árvore

Obs: A árvore deve ser apresentada da seguinte forma: (impressão por nível)

#Raíz: 25 FE: 20 FD: 30

Nó: 20 FE:10 FD: 23

Nó: 30 FE: 28 FD: 40

Nó: 10 FE: 5 FD: 15

Nó: 23 FE: -1 FD: -1

Nó: 28 FE: -1 FD: -1

Nó: 40 FE: -1 FD: -1

Nó: 5 FE: -1 FD: -1

Nó: 15 FE: -1 FD: -1

```c
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>

struct NO{

    int nivel;
    int valor;
    struct NO *ancestral;
    struct NO *FE;
    struct NO *FD;

};

struct NIVEL{
    struct NO *endereco;
    int nivel;
    struct NIVEL *proximo;
};
struct NO *raiz = NULL;
struct NIVEL *nraiz = NULL;

void insercao();
void buscaPosicao(struct NO *busca, struct NO *posicao);
void remocao();
void impressaoNivel(struct NIVEL *no);
void impressaoOrdemSimetrica(struct NO *impressao);
void buscaRemocao(int busca, struct NO *posicao);
void remocaoNo(struct NO *no);

int main(){

    setlocale(LC_ALL,"portuguese");

    while(1){
```

```
        int op;

        system("cls");
        printf("BIENVENIDO\n");
        printf("\nMENU\n");
        printf("Por gentileza escolha uma op磧:");
        printf("\n(1) Inser磧 em □ore bin□a");
        printf("\n(2) Remo磧 em □ore bin□a");
        printf("\n(3) Apresenta磧 da □ore");
        printf("\n(4) Sair");
        printf("\nEscolha: ");
        scanf("%d",&op);
        fflush(stdin);

        switch(op){
            case 1:
                insercao();
                system("pause");
                break;
            case 2:
                remocao();
                system("pause");
                break;
            case 3:
                printf("Por gentileza escolha uma op磧:");
                printf("\n(1) Largura");
                printf("\n(2) Ordem sim鵠ica");
                printf("Escolha: ");
                scanf("%d",&op);
                switch(op){
                    case 1:
                        impressaoNivel(nraiz);
                        system("pause");
                        break;
                    case 2:
                        impressaoOrdemSimetrica(raiz);
                        system("pause");
                        break;
                    default:
                        printf("Escolha inv□da\n\n");
                        system("pause");
                        break;
                }
                break;
            case 4:
                printf("tchal!!\n\n");
                free(raiz);
                free(nraiz);
                system("pause");
                return 0;
                break;
            default:
                printf("Escolha inv□da\n\n");
                system("pause");
                break;
        }

    }

}
```

```
void insercao(){
    struct NO *arvore = NULL;
    struct NO *auxiliar = NULL;

    if(raiz == NULL){

        raiz = (struct NO*)malloc(sizeof(struct NO));
        raiz->nivel = 1;
        raiz->FE = NULL;
        raiz->FD = NULL;
        raiz->ancestral = NULL;

        nraiz = (struct NIVEL*)malloc(sizeof(struct NIVEL));
        nraiz->endereco = raiz;
        nraiz->nivel = raiz->nivel;
        nraiz->proximo = NULL;


        system("cls");


        printf("Por gentileza, digite o valor do desejado: ");
        scanf("%d",&raiz->valor);
        fflush(stdin);

        printf("\nValor cadastrado com sucesso!!\n\n");
    //  printf("%x, %d, %d, %x, %x", raiz, raiz->nivel, raiz->valor,raiz-
>FE, raiz->FD);
    //  system("pause");

    }
    else{

        auxiliar = raiz;

        arvore = (struct NO*)malloc(sizeof(struct NO));
        arvore->FE = NULL;
        arvore->FD = NULL;
        system("cls");

        printf("Por gentileza, digite o valor do desejado: ");
        scanf("%d",&arvore->valor);
        fflush(stdin);

        buscaPosicao(arvore, auxiliar);

    }
}

void buscaPosicao(struct NO *busca, struct NO *posicao){

    if(busca->valor<posicao->valor){
        if(posicao->FE==NULL){
            posicao->FE = busca;
            busca->ancestral = posicao;
            busca->nivel = posicao->nivel+1;

            struct NIVEL *arvoreAuxiliar = nraiz;
            struct NIVEL *arvoreAtual;
            arvoreAtual = (struct NIVEL*)malloc(sizeof(struct NIVEL));
```

```
            arvoreAtual->nivel = busca->nivel;
            arvoreAtual->endereco = busca;

            while(arvoreAuxiliar != NULL){

                if(arvoreAtual->nivel==arvoreAuxiliar->nivel && arvoreAtual
!= arvoreAuxiliar){
                    arvoreAtual->proximo = arvoreAuxiliar->proximo;
                    arvoreAuxiliar->proximo = arvoreAtual;

                }
                if(arvoreAtual->nivel>arvoreAuxiliar->nivel &&
arvoreAuxiliar->proximo == NULL){
                    arvoreAuxiliar->proximo = arvoreAtual;
                    arvoreAtual->proximo = NULL;

                }
                else{
                    arvoreAuxiliar = arvoreAuxiliar->proximo;
                }
            }
            printf("\nValor cadastrado com sucesso!!\n\n");
        //  printf("\n%x, %d, %d, %x, %x", posicao, posicao->nivel,
posicao->valor,posicao->FE, posicao->FD);
        //  printf("\n%x, %d, %d, %x, %x\n", busca, busca->nivel, busca-
>valor,busca->FE, busca->FD);
        }
        else
            buscaPosicao(busca,posicao->FE);
    }
    else if(busca->valor>posicao->valor){
        if(posicao->FD==NULL){
            posicao->FD = busca;
            busca->ancestral =posicao;
            busca->nivel = posicao->nivel+1;

            struct NIVEL *arvoreAuxiliar = nraiz;
            struct NIVEL *arvoreAtual;
            arvoreAtual = (struct NIVEL*)malloc(sizeof(struct NIVEL));
            arvoreAtual->nivel = busca->nivel;
            arvoreAtual->endereco = busca;
            arvoreAtual->proximo = NULL;

            while(arvoreAuxiliar != NULL){

                if(arvoreAtual->nivel==arvoreAuxiliar->nivel && arvoreAtual
!= arvoreAuxiliar){
                    arvoreAtual->proximo = arvoreAuxiliar->proximo;
                    arvoreAuxiliar->proximo = arvoreAtual;

                }
                if(arvoreAtual->nivel>arvoreAuxiliar->nivel &&
arvoreAuxiliar->proximo == NULL){
                    arvoreAuxiliar->proximo = arvoreAtual;
                    arvoreAtual->proximo = NULL;

                }
                else{
                    arvoreAuxiliar = arvoreAuxiliar->proximo;
                }
            }
```

```
            printf("\nValor cadastrado com sucesso!!\n\n");
        //  printf("\n%x, %d, %d, %x, %x", posicao, posicao->nivel,
posicao->valor,posicao->FE, posicao->FD);
        //  printf("\n%x, %d, %d, %x, %x\n", busca, busca->nivel, busca-
>valor,busca->FE, busca->FD);
        }
        else{
            buscaPosicao(busca, posicao->FD);
        }
    }
    else{
        printf("\n\nValor repetido!!\n\n");
    }
}

void remocao(){
    system("cls");
    struct NO *enderecoNo = NULL;
    int valor;
    printf("Por gentileza digite o valor que deseja remover:");
    scanf("%d", &valor);
    buscaRemocao(valor, raiz);
}

void buscaRemocao(int busca, struct NO *posicao){

    if(posicao==NULL)
        printf("Valor n棒cadastrado na ore");
    else if(busca == posicao->valor)
        remocaoNo(posicao);
    else if(busca<posicao->valor)
        buscaRemocao(busca, posicao->FE);
    else if(busca>posicao->valor)
         buscaRemocao(busca, posicao->FD);
}

void remocaoNo(struct NO *no){
    struct NO *auxiliar = no->ancestral;
//  struct NO *alteraNivel = NULL;
    struct NO *proximo = NULL;
    struct NO *anterior = NULL;

    if(auxiliar == NULL){

        if(no->FE == NULL && no->FD == NULL){
            free(no);
            raiz = NULL;
        }
        else if(no->FE != NULL && no->FD == NULL){
            raiz = no->FE;
            raiz->nivel--;
            free(no);

        }
        else if(no->FE == NULL && no->FD != NULL){
            raiz = no->FD;
            raiz->nivel--;
            free(no);

        }
        else{
```

```
            proximo = no->FE;
            anterior = no;
            while(proximo->FD != NULL){
                anterior = proximo;
                proximo = proximo->FD;
            }
            if(proximo->FE==NULL && anterior != no){
                anterior->FD = NULL;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
                raiz = proximo;
            }
            else if(anterior == no){
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
                raiz = proximo;
            }
            else{
                anterior->FD = proximo->FE;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
                raiz = proximo;
            }
            free(no);
            free(proximo);

        }
    }

    else if(no->FD==NULL && no->FE==NULL){

        if(auxiliar->FD == no)
            auxiliar->FD = NULL;
        else if(auxiliar->FE == no)
            auxiliar->FE = NULL;
        free(no);
        //printf("\n%X\n",no);

    }
    else if (no->FE!=NULL && no->FD == NULL){

        if(auxiliar->FE == no){
            auxiliar->FE = no->FE;
            auxiliar = no->FE;
            auxiliar->ancestral = no->ancestral;
            auxiliar->nivel--;
```

```
            no->valor = NULL;
            free(no);
        }
        else if(auxiliar->FD == no){
            auxiliar->FD = no->FE;
            auxiliar = no->FE;
            auxiliar->ancestral = no->ancestral;
            auxiliar->nivel--;
            no->valor = NULL;
            free(no);
        }

    }
    else if(no->FE==NULL && no->FD != NULL){
        if(auxiliar->FE == no){
            auxiliar->FE = no->FD;
            auxiliar = no->FD;
            auxiliar->ancestral = no->ancestral;
            auxiliar->nivel--;
            no->valor = NULL;
            free(no);
        }
        else if(auxiliar->FD == no){
            auxiliar->FD = no->FD;
            auxiliar = no->FD;
            auxiliar->ancestral = no->ancestral;
            auxiliar->nivel--;
            no->valor = NULL;
            free(no);
        }

    }

    else{

        proximo = no->FE;
        anterior = no;
        while(proximo->FD != NULL){
            anterior = proximo;
            proximo = proximo->FD;
        }
        if(proximo->FE==NULL && anterior != no){
            if(auxiliar->FE == no){
                auxiliar->FE = proximo;
                anterior->FD = NULL;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
            else{
                auxiliar->FD = proximo;
                anterior->FD = NULL;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
```

```
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
        }
        else if(anterior == no){
            if(auxiliar->FE == no){
                auxiliar->FE = proximo;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
            else{
                auxiliar->FD = proximo;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
        }
        else{
            if(auxiliar->FE == no){
                auxiliar->FE = proximo;
                anterior->FD = proximo->FE;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
            else{
                auxiliar->FD = proximo;
                anterior->FD = proximo->FE;
                proximo->FE = no->FE;
                proximo->FD = no->FD;
                proximo->ancestral = no->ancestral;
                proximo->nivel = no->nivel;
                anterior = proximo->FE;
                anterior->ancestral=proximo;
                anterior = proximo->FD;
                anterior->ancestral = proximo;
            }
        }
        free(no);
    }

}

void impressaoNivel(struct NIVEL *impressao){

    struct NO *no;

        while(impressao != NULL){
```

```
            no = impressao->endereco;

            struct NO *fe = no->FE;
            struct NO *fd = no->FD;
            int valor = no->valor;

            if(fe == NULL){
                fe = (struct NO*)malloc(sizeof(struct NO));
                fe->valor = -1;
            }
            if(fd == NULL){
                fd = (struct NO*)malloc(sizeof(struct NO));
                fd->valor = -1;
            }
            if(valor == NULL)
                valor = -1;
            if(no->nivel==1)
                printf("Raiz: %d | FE: %d | FD: %d \n", valor, fe->valor,
fd->valor);
            else
                printf("N□d | FE: %d | FD: %d \n", valor, fe->valor, fd-
>valor);

            impressao = impressao->proximo;
    }

}

void impressaoOrdemSimetrica(struct NO *impressao){

    if(impressao != NULL){
        impressaoOrdemSimetrica(impressao->FE);

        struct NO *fe = impressao->FE;
        struct NO *fd = impressao->FD;
        int valor = impressao->valor;
        if(fe == NULL){
        fe = (struct NO*)malloc(sizeof(struct NO));
        fe->valor = -1;
        }
        if(fd == NULL){
        fd = (struct NO*)malloc(sizeof(struct NO));
        fd->valor = -1;
        }
        if(valor == NULL)
        valor = -1;
        if(impressao->nivel==1)
        printf("Raiz: %d | FE: %d | FD: %d \n", valor, fe->valor, fd-
>valor);
        else
        printf("N□d | FE: %d | FD: %d \n", valor, fe->valor, fd->valor);

        impressaoOrdemSimetrica(impressao->FD);
    }

}
```