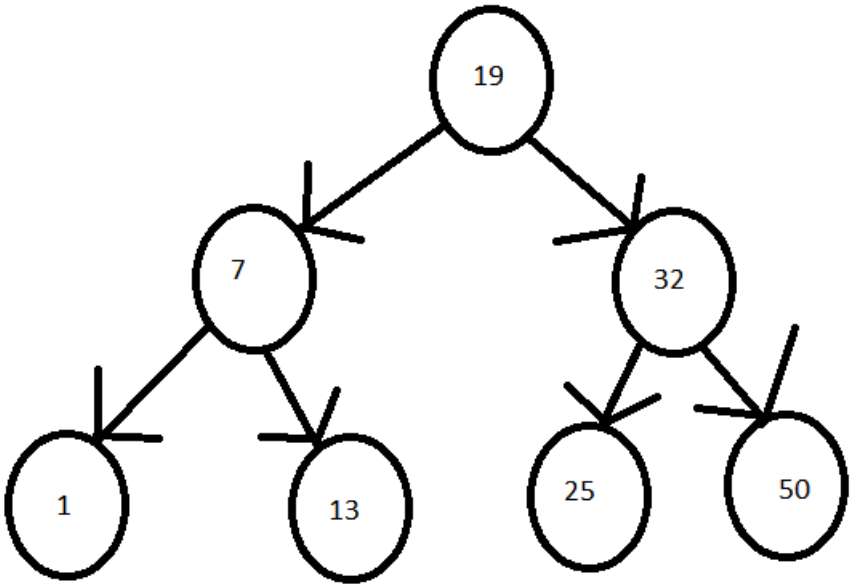
Aluno: Rafael Cruz Arantes da Silva

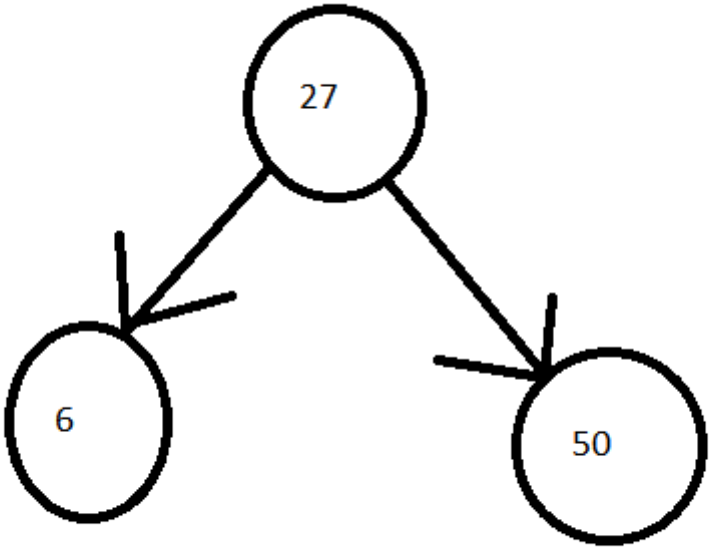
Matrícula: 20191105905

Matéria: Estrutura de Dados II

1- Desenho da Árvore de Busca Binária Balanceada, com os elementos 1,7,13,19,25,32 e 50 e fator de balanceamento de no máximo 1:



2- Desenho da Árvore AVL resultante, inicialmente vazia, sendo inseridos os elementos 6,27 e 50, em ordem:



3- Código da Árvore AVL:

Definindo a estrutura e funções para calcular altura e fator de balanceamento e funções para calcular o maior nó e calcular o maior número entre dois números:

```c
//Aluno: Rafael Cruz Arantes da Silva
#include<stdio.h>
#include<stdlib.h>

typedef struct No{
    int num;
    int altura;
    struct No *esq;
    struct No *dir;
}Arv;


int altNo(struct No* no){
    if(no == NULL){
        return -1;
    }
    else
    return no->altura;
}

int fatorBalancNo(struct No* no){
    return labs(altNo(no->esq)-altNo(no->dir));
}

Arv* maior(Arv* a){
    while(a->dir != NULL){
            a = a->dir;
            }
    return a;
}

int maiorN(int x,int y){
    if(x>y){
        return x;
    }
    else{
        return y;
    }
}
```

Funções de rotação:

```
Arv* rotacaoLL(Arv *a){
    struct No *no;
    no = a->esq;
    a->esq = no->dir;
    no->dir = a;
    a->altura = maiorN(altNo(a->esq),altNo(a->dir))+1;
    no->altura = maiorN(altNo(no->esq),a->altura)+1;
    a = no;
    return a;
}

Arv* rotacaoRR(Arv *a){
    struct No *no;
    no = a->dir;
    a->dir = no->esq;
    no->esq = a;
    a->altura = maiorN(altNo(a->esq),altNo(a->dir))+1;
    no->altura = maiorN(altNo(no->dir),(a->altura))+1;
    a = no;
    return a;
}

Arv* rotacaoLR(Arv *a){
    rotacaoRR(a->esq);
    rotacaoLL(a);
    return a;
}

Arv* rotacaoRL(Arv *a){
    rotacaoLL(a->dir);
    rotacaoRR(a);
    return a;
}
```

```c
int vazia(Arv *a){
    if(a == NULL)
        return 1;
    else
        return 0;
}

Arv* inicializar(){
    return NULL;
}

Arv* aloca(int valor){
    Arv *a;
        a = (No*) malloc(sizeof(No));
        a->num = valor;
        a->esq = NULL;
        a->dir = NULL;
        a->altura = 0;
        printf("Valor inserido com sucesso! Valor inserido: %d\n",valor);
        return a;
    }
```

Função para inserir um nó:

```c
Arv* insere(Arv* a, int valor){
        if(vazia(a)){
            a = aloca(valor);
        }
            if(valor < a->num){
            a->esq = insere(a->esq, valor);
                    if(fatorBalancNo(a) >= 2){
                        if(valor < a->esq->num){
                        a = rotacaoLL(a);
                        }
                        else{
                        a = rotacaoLR(a);
                        }
                        a->altura = maiorN(altNo(a->esq),altNo(a->dir))+1;
                    }
            }
            else if(valor > a->num){
            a->dir = insere(a->dir, valor);
                    if(fatorBalancNo(a) >= 2){
                        if(valor < a->esq->num){
                        a = rotacaoRR(a);
                        }
                        else{
                        a = rotacaoRL(a);
                        }
                        a->altura = maiorN(altNo(a->esq),altNo(a->dir))+1;
                    }
            }
    return a;

}
```

Função para buscar um nó na árvore:

```
Arv *busca(Arv *a, int valor){
    if(vazia(a)){
        printf("Valor nao encontrado!\nO valor buscado: %d\n\n",valor);
        return NULL;
    }

    else
    {
    if(a->num == valor){
        printf("Valor encontrado!\nO valor buscado: %d\n\n",valor);
        }

    else if(a->num > valor)
        busca(a->esq, valor);

    else if(a->num < valor)
        busca(a->dir, valor);
    }
    return a;
}
```

Função para remoção de um nó:

```
Arv* removevalor(Arv* raiz, int valor){
    if(vazia(raiz)) {
        printf("Valor nao encontrado.\n\n");
        return raiz;
    }
    if(valor < raiz->num){
        raiz->esq = removevalor(raiz->esq, valor);
        raiz->altura = altNo(raiz);
        if(fatorBalancNo(raiz) >= 2){
            if(altNo(raiz->dir->dir)>fatorBalancNo(raiz->dir->esq)){
            rotacaoRR(raiz);
            }
            else{
            rotacaoRL(raiz);
            }
        }
    }
    else
        if(valor > raiz->num){
            raiz->dir = removevalor(raiz->dir, valor);
            raiz->altura = altNo(raiz);
            if(fatorBalancNo(raiz)>=2){
                if(altNo(raiz->esq->esq) > altNo(raiz->esq->dir)){
                rotacaoLL(raiz);
                }
                else{
                rotacaoLR(raiz);
                }
            }
        }
        else
        {
            if(raiz->esq == NULL){
                printf("Valor removido!\n\n");
                Arv* t = raiz;
                raiz = raiz->dir;
                free(t);
            }
```

Continuação da função para remover um nó e função para imprimir os elementos em ordem:

```
            else
                if(raiz->dir == NULL){
                    printf("Valor removido!\n\n");
                    Arv* t = raiz;
                    raiz = raiz->esq;
                    free(t);
                }
            else
                if((raiz->dir != NULL)&&(raiz->esq != NULL)){
                    Arv *t = maior(raiz->esq);
                    int v = t->num;
                    removevalor(raiz,v);
                    raiz->num = v;
                    }
        }

    return raiz;
}

void imprimir(Arv* a){
    if (!vazia(a)){
        imprimir(a->esq);
        printf("%d\n", a->num);
        imprimir(a->dir);
    }
}
```

Função Main:

```
int main(){
    Arv* a;
    a = inicializar();
    int op;
    int valor = 0;
    a = insere(a, 6);
    a = insere(a, 27);
    a = insere(a, 50);
    a = insere(a, 77);
    a = insere(a, 23);
    a = insere(a, 11);


    // menu
    do{
    printf("******************\n1- INSERIR\n2- BUSCAR\n3- IMPRIMIR\n4- DELETAR VALOR\n0- SAIR\n******************\nDigite a Opcao: ");
    scanf("%d", &op);
    switch (op){
        case 1:
            printf("\nDigite o valor para inserir: ");
            scanf("%d", &valor);
            a = insere(a, valor);
            printf("\n");
            break;
        case 2:
            printf("\nDigite o valor para buscar: ");
            scanf("%d", &valor);
            a = busca(a,valor);
            break;
        case 3:
            if(vazia(a)){
            printf("\nArvore vazia.\n\n");
            }
                else{
                printf("\nResultado: \n");
                imprimir(a);
                printf("\n");
                }
```

Continuação da Main:

```
            break;
        case 4:
            printf("\nDigite o valor para remover: ");
            scanf("%d", &valor);
            a = removevalor(a, valor);
            break;
            case 0:
            break;
        default:
        printf("\nOpcao Invalida.\n\n");
        break;
        }
    }while(op != 0);
    return 0;
}
```