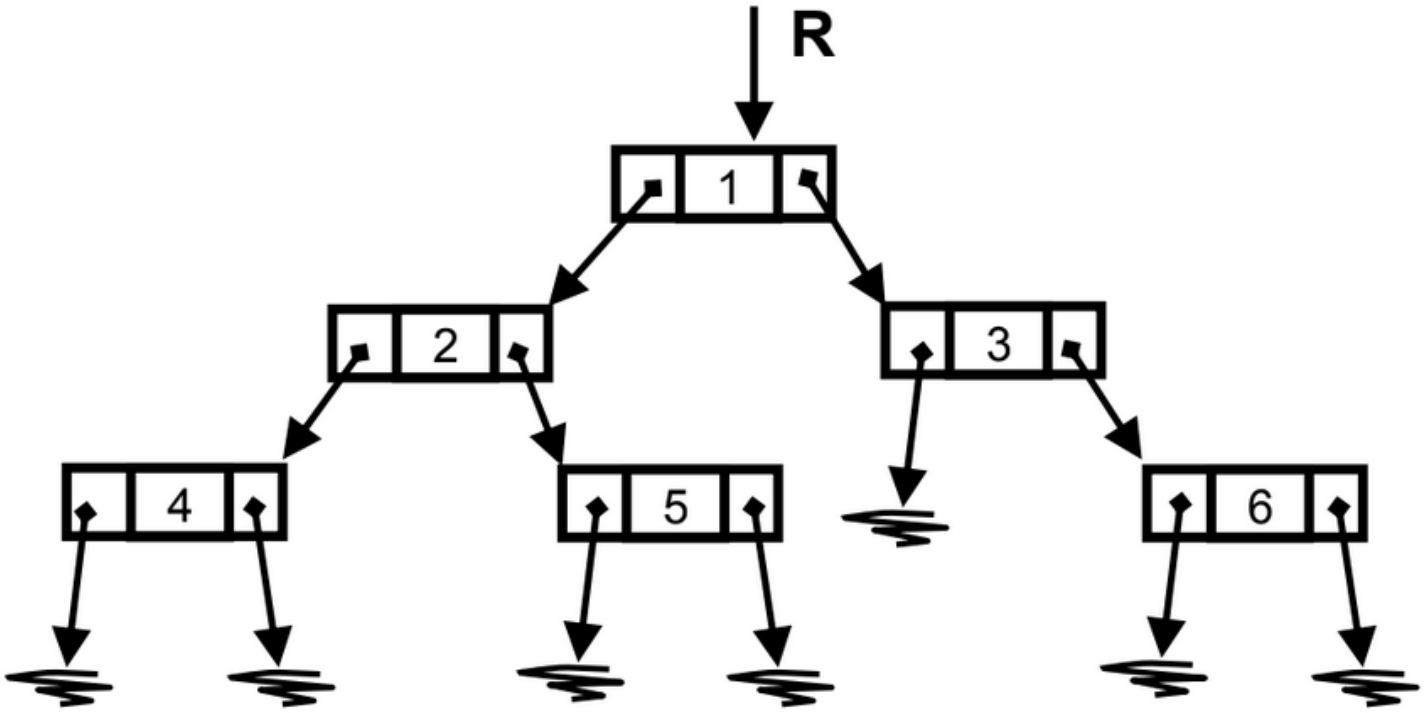
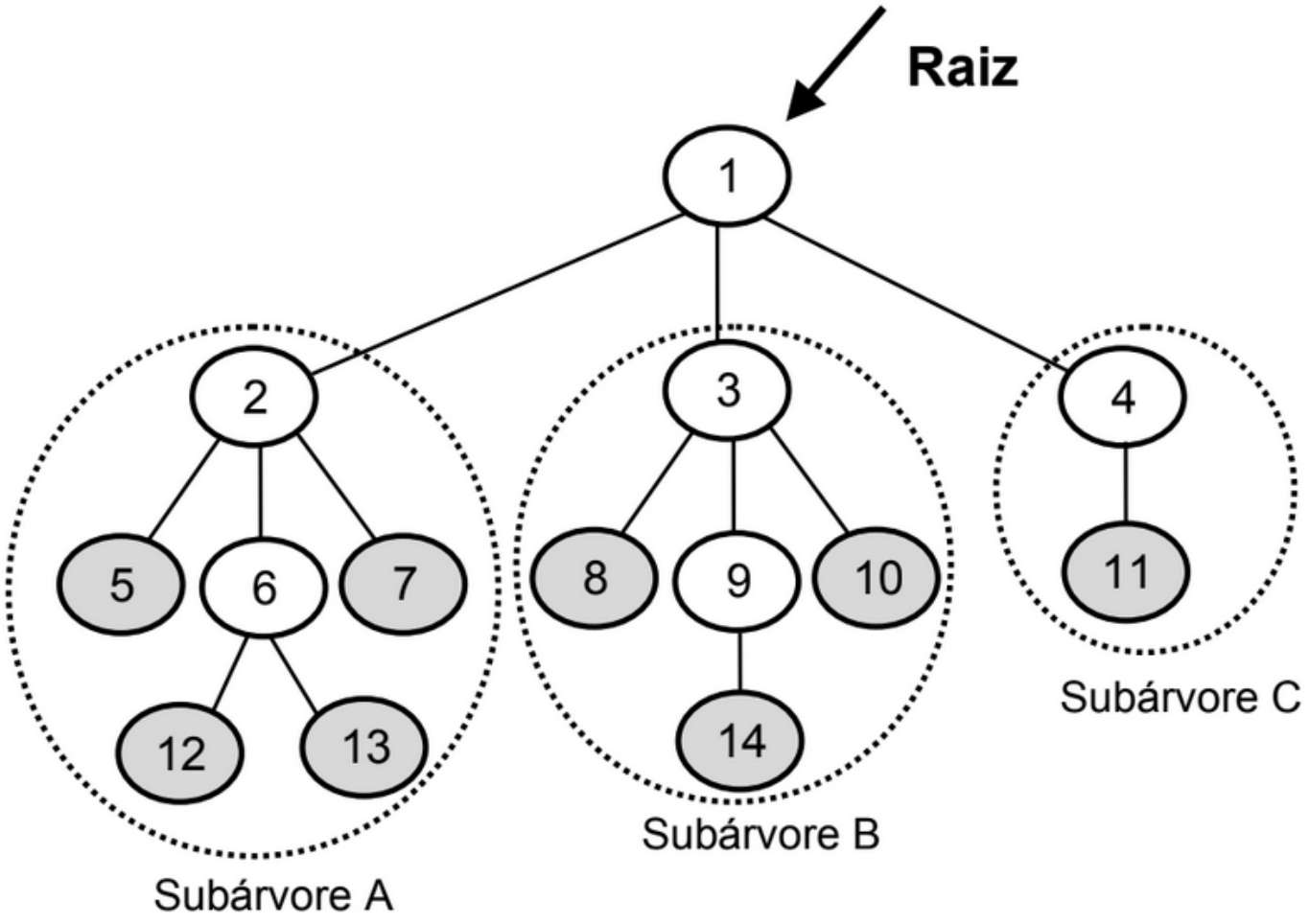
Nome: Matheus Goulart Ranzani
RA: 800278

Diagramas da Árvore Binária de Busca





Código do arquivo arvore_binaria.h

```c
#include <stdlib.h>

typedef struct Node {
    int info;
    struct Node* dir;
    struct Node* esq;
} Node;

typedef Node* NodePtr;
typedef Node* ArvoreBinaria;

void cria(ArvoreBinaria* R) {
    *R = NULL;
}

int vazia(ArvoreBinaria* R) {
    return *R == NULL;
}

void insere(ArvoreBinaria* R, int x, int* ok) {
    if (vazia(R)) {
          NodePtr p = (NodePtr) malloc(sizeof(Node));

          p->esq = NULL;
          p->dir = NULL;
          p->info = x;
          *R = p;
          p = NULL;

          *ok = 1;
    } else {
        if (x == (*R)->info) {
            *ok = 0;
            return;
        } else {
            if (x < (*R)->info) {
                ArvoreBinaria* esq = &(*R)->esq;
                insere(esq, x, ok);
            } else {
                ArvoreBinaria* dir = &(*R)->dir;
                insere(dir, x, ok);
            }
        }
    }
}
```

```c
void retira(ArvoreBinaria* R, int x, int* ok) {
    if (vazia(R)) {
        *ok = 0;
        return;
    } else {
        if (x < (*R)->info) {
            ArvoreBinaria* esq = &(*R)->esq;
            retira(esq, x, ok);
        } else {
            if (x > (*R)->info) {
                ArvoreBinaria* dir = &(*R)->dir;
                retira(dir, x, ok);
            } else {
                ArvoreBinaria aux = *R;
                *ok = 1;

                if ((*R)->esq == NULL && (*R)->dir == NULL) {
                    free(aux);
                    *R = NULL;
                } else {
                    if ((*R)->esq != NULL && (*R)->dir != NULL) {
                        aux = (*R)->esq;

                        while (aux->dir != NULL) {
                            aux = aux->dir;
                        }

                        (*R)->info = aux->info;
                        ArvoreBinaria* esq = &(*R)->esq;
                        retira(esq, (*R)->info, ok);
                    } else {
                        if ((*R)->esq == NULL) {
                            *R = (*R)->dir;
                        } else {
                            *R = (*R)->esq;
                        }

                        free(aux);
                    }
                }
            }
        }
    }
}
```

```c
void destroi(ArvoreBinaria* R) {
    if (!vazia(R)) {
        ArvoreBinaria* esq = &(*R)->esq;
        destroi(esq);
        ArvoreBinaria* dir = &(*R)->dir;
        destroi(dir);

        free(R);
    }
}

int esta_na_arvore(ArvoreBinaria* R, int x) {
    if (vazia(R)) {
        return 0;
    } else {
        if (x == (*R)->info) {
            return 1;
        } else if (x < (*R)->info) {
            ArvoreBinaria* esq = &(*R)->esq;
            return esta_na_arvore(esq, x);
        } else {
            ArvoreBinaria* dir = &(*R)->dir;
            return esta_na_arvore(dir, x);
        }
    }
}
```

Código do arquivo usa_arvore_binaria.c

```c
#include "arvore_binaria.h"

#include <stdio.h>

void imprime_todos(ArvoreBinaria R) {
    if (!vazia(&R)) {
        printf("%d ", R->info);
        imprime_todos(R->esq);
        imprime_todos(R->dir);
    }
}


int altura(ArvoreBinaria R) {
    if (vazia(&R)) {
        return 0;
    } else {
        int altura_esq = altura(R->esq);
        int altura_dir = altura(R->dir);

        if (altura_esq > altura_dir) {
            return 1 + altura_esq;
        } else {
            return 1 + altura_dir;
        }
    }
}

int main() {
    ArvoreBinaria R;
    int ok;

    cria(&R);

    printf("Adicionando os valores 6, 4, 8, 2, 5, 7, 9 (nessa ordem) na Arvore
Binaria...\n");
    insere(&R, 6, &ok);
    insere(&R, 4, &ok);
    insere(&R, 8, &ok);
    insere(&R, 2, &ok);
    insere(&R, 5, &ok);
    insere(&R, 7, &ok);
    insere(&R, 9, &ok);

    printf("\nImpressao da Arvore (Pre-ordem):\n");
```

```
    imprime_todos(R);
    printf("\nAltura da Arvore = %d\n", altura(R));

    esta_na_arvore(&R, 5) ? printf("\n5 esta na Arvore") : printf("\n5 nao esta na
Arvore");
    esta_na_arvore(&R, 9) ? printf("\n9 esta na Arvore") : printf("\n9 nao esta na
Arvore");
    esta_na_arvore(&R, 20) ? printf("\n20 esta na Arvore") : printf("\n20 nao esta na
Arvore");
    esta_na_arvore(&R, 3) ? printf("\n3 esta na Arvore") : printf("\n3 nao esta na
Arvore");

    printf("\n\nRemovendo os valores 7, 8, 5 e 2 da Arvore Binaria...");
    retira(&R, 7, &ok);
    retira(&R, 8, &ok);
    retira(&R, 5, &ok);
    retira(&R, 2, &ok);

    printf("\n\nImpressao da Arvore (Pre-ordem):\n");
    imprime_todos(R);
    printf("\nAltura da Arvore = %d", altura(R));

    printf("\n");

    destroi(&R);

    return 0;
}
```

Imagem da execução do programa

```
C:\Users\mathe\OneDrive\Área de Trabalho\UFSCar\ENPE 4\AED1\Frequências\F9
>gcc *.c -o usa_arvore_binaria

C:\Users\mathe\OneDrive\Área de Trabalho\UFSCar\ENPE 4\AED1\Frequências\F9
>gcc usa_arvore_binaria.c -o usa_arvore_binaria

C:\Users\mathe\OneDrive\Área de Trabalho\UFSCar\ENPE 4\AED1\Frequências\F9
>usa_arvore_binaria
Adicionando os valores 6, 4, 8, 2, 5, 7, 9 (nessa ordem) na Arvore Binaria
...

Impressao da Arvore (Pre-ordem):
6 4 2 5 8 7 9
Altura da Arvore = 3

5 esta na Arvore
9 esta na Arvore
20 nao esta na Arvore
3 nao esta na Arvore

Removendo os valores 7, 8, 5 e 2 da Arvore Binaria...

Impressao da Arvore (Pre-ordem):
6 4 9
Altura da Arvore = 2
```