

Universidade Federal do Piauí – UFPI

Campus Senador Helvídio Nunes de Barros – CSHNB

Curso de Sistemas de Informação                      Bloco: IV

Disciplina: Estruturas de Dados II

Professora: Juliana Oliveira de Carvalho

Acadêmico:.....Matrícula:.....

## ATIVIDADE DE FIXAÇÃO 6

### Informações importantes:

1. Os exercícios que não pedirem a implementação de código favor descrever a resposta e anexar como comentário no código fonte.
2. Em todos os exercícios, caso entenda necessário ou que fique melhor, vocês podem fazer mais de um predicado para resolver o problema.
3. Para todos os exercícios vocês devem fazer um comentário no código explicando como executar o código citando exemplos.
4. Para todas as funções recursivas devem ser descritas o seu funcionamento.

- 1) Descreva passo a passo usando texto e desenho e o anexo I para inserir elementos em uma árvore Vermelha-Preta para as seguintes sequências:

Sequência: 200, 100, 300, 50, 150, 120

Sequência: 100, 200, 300, 150, 120

Sequência: 500, 100, 200, 800, 300, 400,350

- 2) Descreva o funcionamento da remoção de uma árvore vermelha-preta através de exemplo.

## ANEXO I - FUNÇÕES RELACIONADAS A ÁRVORE VERMELHO-PRETA

### Rotação à esquerda

```
01 struct NO* rotacionaEsquerda(struct NO* A){
02     struct NO* B = A->dir;
03     A->dir = B->esq;
04     B->esq = A;
05     B->cor = A->cor;
06     A->cor = RED;
07     return B;
08 }
```

### Rotação à direita

```
01 struct NO* rotacionaDireita(struct NO* A){
02     struct NO* B = A->esq;
03     A->esq = B->dir;
04     B->dir = A;
05     B->cor = A->cor;
06     A->cor = RED;
07     return B;
08 }
```

### Movendo um nó vermelho para a esquerda

```
01 struct NO* move2EsqRED(struct NO* H){
02     trocaCor(H);
03     if(cor(H->dir->esq) == RED){
04         H->dir = rotacionaDireita(H->dir);
05         H = rotacionaEsquerda(H);
06         trocaCor(H);
07     }
08     return H;
09 }
```

### Arrumando o balanceamento da rubro-negra

```
01 struct NO* balancear(struct NO* H){
02     //nó vermelho é sempre filho à esquerda
03     if(cor(H->dir) == RED)
04         H = rotacionaEsquerda(H);
05
06     //Filho da direita e neto da esquerda são vermelhos
07     if(H->esq != NULL && cor(H->dir) == RED &&
08         cor(H->esq->esq) == RED)
09         H = rotacionaDireita(H);
10
11     //2 filhos vermelhos: troca cor!
12     if(cor(H->esq) == RED && cor(H->dir) == RED)
13         trocaCor(H);
14
15     return H;
16 }
```

## Removendo um elemento da árvore rubro-negra

```

01 struct NO* remove_NO(struct NO* H, int valor){
02     if(valor < H->info){
03         if(cor(H->esq) == BLACK &&
04            cor(H->esq->esq) == BLACK)
05             H = move2EsqRED(H);
06         H->esq = remove_NO(H->esq, valor);
07     }else{
08         if(cor(H->esq) == RED)
09             H = rotacionaDireita(H);
10
11         if(valor == H->info && (H->dir == NULL)){
12             free(H);
13             return NULL;
14         }
15
16         if(cor(H->dir) == BLACK &&
17            cor(H->dir->esq) == BLACK)
18             H = move2DirRED(H);
19
20         if(valor == H->info){
21             struct NO* x = procuraMenor(H->dir);
22             H->info = x->info;
23             H->dir = removerMenor(H->dir);
24         }else
25             H->dir = remove_NO(H->dir, valor);
26     }
27     return balancear(H);
28 }
29 int remove_ArvLLRB(ArvLLRB *raiz, int valor){
30     if(consulta_ArvLLRB(raiz, valor)){
31         struct NO* h = *raiz;
32         *raiz = remove_NO(h, valor);
33         if(*raiz != NULL)
34             (*raiz)->cor = BLACK;
35         return 1;
36     }else
37         return 0;
38 }

```

### Procurando e removendo o menor elemento da rubro-negra

```
01 struct NO* removerMenor(struct NO* H) {
02     if(H->esq == NULL) {
03         free(H);
04         return NULL;
05     }
06     if(cor(H->esq) == BLACK && cor(H->esq->esq) == BLACK)
07         H = move2EsqRED(H);
08
09     H->esq = removerMenor(H->esq);
10     return balancear(H);
11 }
12 struct NO* procuraMenor(struct NO* atual) {
13     struct NO *no1 = atual;
14     struct NO *no2 = atual->esq;
15     while(no2 != NULL) {
16         no1 = no2;
17         no2 = no2->esq;
18     }
19     return no1;
20 }
```

### Mudando a cor de um nó e de seus filhos

```
01 void trocaCor(struct NO* H) {
02     H->cor = !H->cor;
03     if(H->esq != NULL)
04         H->esq->cor = !H->esq->cor;
05     if(H->dir != NULL)
06         H->dir->cor = !H->dir->cor;
07 }
```

### Inserindo um elemento na árvore rubro-negra

```

01 struct NO* insereNO(struct NO* H, int valor, int *resp){
02     if(H == NULL){
03         struct NO *novo
04         novo = (struct NO*)malloc(sizeof(struct NO));
05         if(novo == NULL){
06             *resp = 0;
07             return NULL;
08         }
09         novo->info = valor;
10         novo->cor = RED;
11         novo->dir = NULL;
12         novo->esq = NULL;
13         *resp = 1;
14         return novo;
15     }
16
17     if(valor == H->info)
18         *resp = 0; // Valor duplicado
19     else{
20         if(valor < H->info)
21             H->esq = insereNO(H->esq, valor, resp);
22         else
23             H->dir = insereNO(H->dir, valor, resp);
24     }
25
26     if(cor(H->dir) == RED && cor(H->esq) == BLACK)
27         H = rotacionaEsquerda(H);
28
29     if(cor(H->esq) == RED && cor(H->esq->esq) == RED)
30         H = rotacionaDireita(H);
31
32     if(cor(H->esq) == RED && cor(H->dir) == RED)
33         trocaCor(H);
34
35     return H;
36 }
37 int insere_ArvLLRB(ArvLLRB* raiz, int valor){
38     int resp;
39     *raiz = insereNO(*raiz, valor, &resp);
40     if((*raiz) != NULL)
41         (*raiz)->cor = BLACK;
42
43     return resp;
44 }

```