Lista 3 - Estrutura de Dados 1 - 2020.1

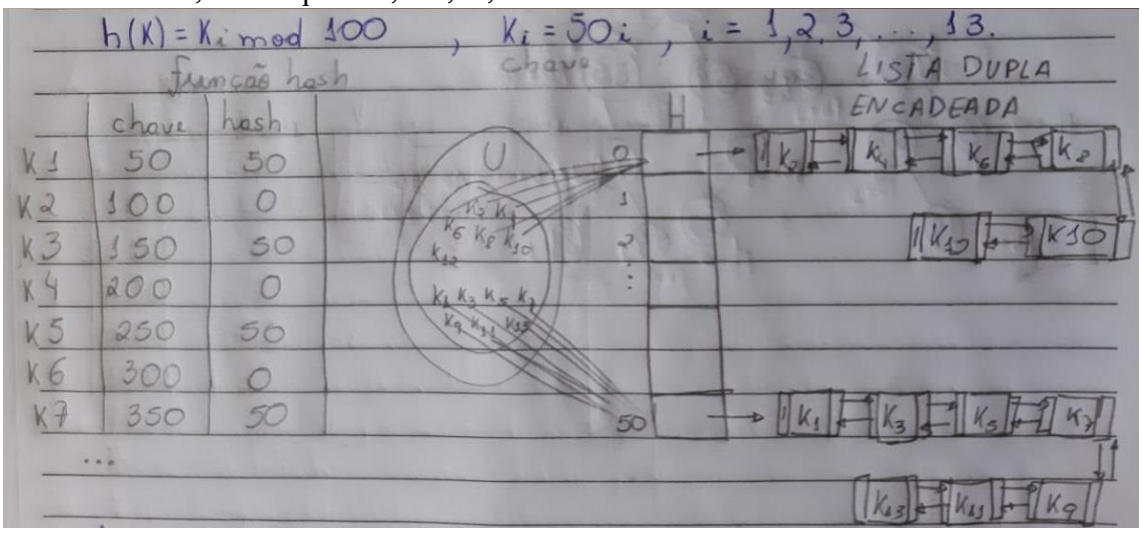
Prof. Ana Luiza Bessa de Paula Barros

Ciência da computação – UECE

Aluna: Natália Sales Aragão

1.1 Questões sobre Hash

1. função hash: h(k) = k mod 100
   chave: ki = 50i, sendo que i=1, 2 3, ..., 13.



   Como as colisões são resolvidas utilizando listas encadeadas, uma parte dos elementos vai para a posição 0 da tabela e a outra parte vai para a posição 50. Então k par vai para posição 0 e k ímpar, pro 50.

   Eu tentei implementar 2 vezes esse código usando Lista Encadeada, mas não consigo fazer o que o desenho esta fazendo, ele só bota em outra posição da tabela pra evitar colisão.

2. As chaves 12,18,13,2,3,23,5 e 15 são inseridas em uma tabela com 10 posições utilizando a função hash h(k) = k mod 10 e sondagem linear. Qual é a tabela hash resultante?

   Resposta: letra d, porque a chave 12 tem posição 2 pois 12 mod 10 = 2, e assim em diante com o resto das chaves, há uma lista encadeada.

(d)

0	
1	
2	12,2
3	13,3,23
4	
5	5,15
6	
7	
8	18
9	

2.2. Questões sobre Árvore Binária de Busca (BST)

1. a)

```
public void adicionar(T valor) {
    No<T> novoElem = new No<T>(valor);

    if(raiz == null) {
        this.raiz = novoElem;
    }else {
        No<T> atual = this.raiz;
        while(true) {
            if(novoElem.getValor().compareTo(atual.getValor()) == -1) { //menor
                if(atual.getEsq() != null) { //vai para esquerda
                    atual = atual.getEsq();
                }else {
                    atual.setEsq(novoElem);
                    break;
                }
            }else { // se for maior ou igual
                if(atual.getDir() != null) { //vai para direita
                    atual = atual.getDir();
                }else {
                    atual.setDir(novoElem);
                    break;
                }
            }
        }
    }
}
```

A árvore em ordem ficaria assim:

```
Árvore em ordem:
1
3
5
8
10
15
16
17
18
23
25
26
27
28
30
47
52
```

b)

```java
public boolean remover(T valor) {
    //buscar o elemento na arvore
    No<T> atual = this.raiz;
    No<T> paiAtual = null;
    while(atual != null) {
        if(atual.getValor().equals(valor)) {
            break;
        }else if(valor.compareTo(atual.getValor()) == -1) { //valor é menor q o atual
            paiAtual = atual;
            atual = atual.getEsq();
        }else {
            paiAtual = atual;
            atual = atual.getDir();
        }
    }
    //verifica se existe o elemento
    if(atual != null) {

        //elem tem 2 filhos ou elem tem somente filho à direita
        if(atual.getDir() != null) { //tem filhos só a direita
            No<T> substituto = atual.getDir();
            No<T> paiSubstituto = atual;
            while(substituto.getEsq() != null) {
                paiSubstituto = substituto;
                substituto = substituto.getEsq();
            }
```

```
        substituto.setEsq(atual.getEsq());

        if(paiAtual != null) {
            if(atual.getValor().compareTo(paiAtual.getValor()) == -1) { //atual < paiAtual
                paiAtual.setEsq(substituto);
            }else {
                paiAtual.setDir(substituto);
            }
        }else { //se não tem paiAtual. então é a raiz
            this.raiz = substituto;
        }
        //removeu o elemento da árvore
        if(substituto.getValor().compareTo(paiSubstituto.getValor()) == -1) { //substituto < paiSubstituto
            paiSubstituto.setEsq(null);
        }else {
            paiSubstituto.setDir(null);
        }


    }else if(atual.getEsq() != null) { //tem filho só a esquerda
        No<T> substituto = atual.getEsq();
        No<T> paiSubstituto = atual;
        while(substituto.getDir() != null) {
            paiSubstituto = substituto;
            substituto = substituto.getDir();
        }


      if(paiAtual != null) {
          if(atual.getValor().compareTo(paiAtual.getValor()) == -1) { //atual < paiAtual
              paiAtual.setEsq(substituto);
          }else {
              paiAtual.setDir(substituto);
          }
      }else { //se for a raiz
          this.raiz = substituto;
      }

      //removeu o elemento da árvore
      if(substituto.getValor().compareTo(paiSubstituto.getValor()) == -1) { //substituto < paiSubstituto
          paiSubstituto.setEsq(null);
      }else {
          paiSubstituto.setDir(null);
      }


  }else{ //não tem filho
      if(paiAtual != null) {
          if(atual.getValor().compareTo(paiAtual.getValor()) == -1) { //atual < paiAtual
              paiAtual.setEsq(null);
          }else {
              paiAtual.setDir(null);
          }
              }else { //é a raiz
                  this.raiz = null;
              }
          }

          return true;

      }else {
          return false;
      }
  }
```

Removendo cada um dos elementos [26 15 52 8 5] fica:

```
Depois da remoção do elemento [ 26 ]
1
3
5
8
10
15
16
17
18
23
25
27
28
30
47
52
```

```
Depois da remoção dos elementos [ 26 15 ]
1
3
5
8
10
16
17
25
27
28
30
47
52
```

```
Depois da remoção dos elementos [ 26 15 52 ]
1
3
5
8
10
16
17
25
27
28
30
47
```

Eu fui bem removendo esses 3 priemiros números, porém, tive bastante de remover o 8, não aparecerem ai :

```
Depois da remoção dos elementos [ 26 15 52 8 ]
5
10
16
17
25
27
28
30
47
Depois da remoção dos elementos [ 26 15 52 8 5 ]
10
16
17
25
27
28
30
47
```

c) Não pode ser considerada uma AVL pois essa árvore está desbalanceada, tem algumas subarvores que possuem o fator de balanaceamente com módulo 2 indicando necessidade de rotação da subarvore.

d)

```
Pós Ordem
1
5
3
8
17
16
23
18
15
10
27
26
28
52
47
30
25
```

3.3. Questões sobre Árvores Balanceadas (AVL)

1. AVL

a) Inserção dos elementos: [60 45 42 83 69]

```
public void inserir(T k) {
    No<T> n = new No<T>(k);
    inserirAVL(this.raiz, n);
}

public void inserirAVL(No<T> aComparar, No<T> aInserir) {
    if(aComparar == null) {
        this.raiz = aInserir;
    } else {
        if(aInserir.getChave().compareTo(aComparar.getChave()) == -1) { // aInserir for menor

            if(aComparar.getEsquerda() == null) {
                aComparar.setEsquerda(aInserir);
                aInserir.setPai(aComparar);
                verificarBalanceamento(aComparar);
            }else {
                inserirAVL(aComparar.getEsquerda(), aInserir);
            }
        }else if (aInserir.getChave().compareTo(aComparar.getChave()) == 1) { //maior

            if (aComparar.getDireita() == null) {
                aComparar.setDireita(aInserir);
                aInserir.setPai(aComparar);
                verificarBalanceamento(aComparar);
            } else {
                inserirAVL(aComparar.getDireita(), aInserir);
            }

        } else {
            System.out.println("O nó já existe");
        }
```

O método de inserção é quase idêntico à Árvore Binária, só difere no fato de verificar o balanceamento do nó adicionado.

A ávore em Ordem:

```
Inserção dos elem
Árvore em ordem:
42
45
60
69
83
```

b)Remoção dos elementos [45 83]

```java
public void remover(T k) {
    No<T> n = new No<T>(k);
    removerAVL(this.raiz, n);
}

public void removerAVL(No<T> atual, No<T> aRemover) {
    if (atual == null) {
        return;

    } else {

        if (atual.getChave().compareTo(aRemover.getChave()) == 1) { //se item for maior do q a raiz, está na esquerda
            removerAVL(atual.getEsquerda(), aRemover);

        } else if (atual.getChave().compareTo(aRemover.getChave()) == -1) { //se for menor q a raiz, na direita
            removerAVL(atual.getDireita(), aRemover);

        } else {
            removerNoEncontrado(atual);
        }
    }
}
```

```
Remoção do elemento: [ 45 ]
60
42
69
83
Remoção dos elementos: [ 45 83 ]
60
42
69
```

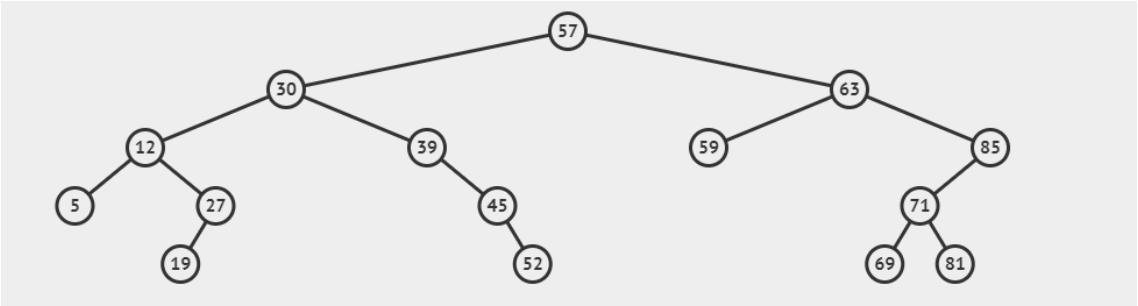
c) Inserção dos elementos [77 85 44 83 17 20]

```
Inserção dos elementos: [ 77 85 44 83 17 20 ]
60
42
17
20
44
77
69
85
83
```

d) Árvore em ordem:

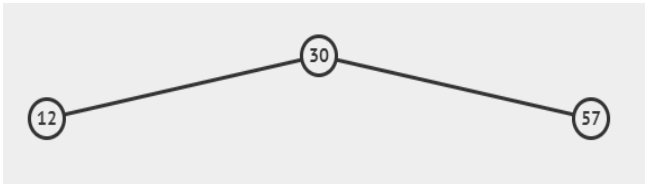
```
Árvore em Ordem
17
20
42
44
60
69
77
83
85
```

2.

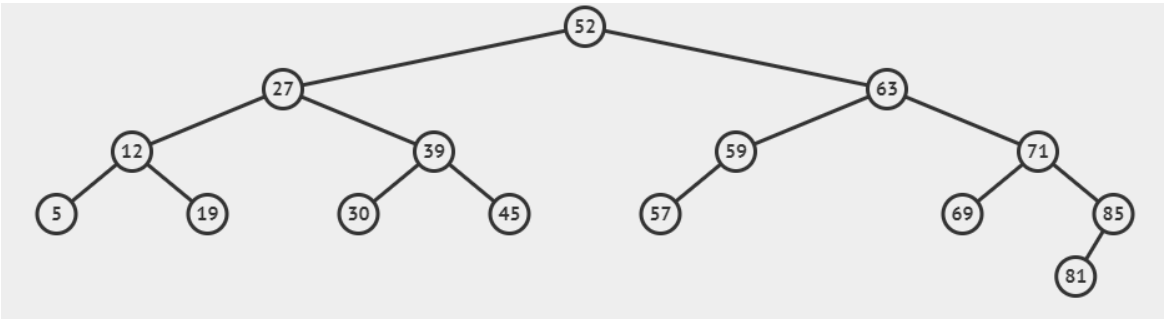


a) A árvore cima seria a árvore desbalanceada dada na questão, quando balanceamos ela podemos contar com 9 a quantidade de vezes que ela rotaciona para a esquerda ou para a direita.

b) A árvore abaixo foi inserido só os dois primeiros elementos, quando inserimos o terceiro elemento, a árvore já pesa para a esquerda, o que aciona a rotação para a direita tornando o segundo elemento a raiz da arvore agora balanceada.



A seguir, o desenho da árvore completa balanceada



A seguir, o código da verificação do balanceamento e altura:

```java
private void verificarBalanceamento(No<T> atual) {
    definaBalanceamento(atual);
    int balanceamento = atual.getBalanceamento();

    if (balanceamento == -2) {
        //Rotacionar para a direita
        if (altura(atual.getEsquerda().getEsquerda()) >= altura(atual.getEsquerda().getDireita())) {
            atual = rotacaoDireita(atual);

        } else {
            atual = duplaRotacaoEsquerdaDireita(atual);
        }
        setQuantRotacoes(1);

    } else if (balanceamento == 2) {
        //Rotacionar para a esquerda
        if (altura(atual.getDireita().getDireita()) >= altura(atual.getDireita().getEsquerda())) {
            atual = rotacaoEsquerda(atual);

        } else {
            atual = duplaRotacaoDireitaEsquerda(atual);
        }
        setQuantRotacoes(1);
    }

private void definaBalanceamento(No<T> no) {
    no.setBalanceamento(altura(no.getDireita()) - altura(no.getEsquerda()));
}

private int altura(No<T> atual) {
    if (atual == null) {
        return -1;
    }

    if (atual.getEsquerda() == null && atual.getDireita() == null) {
        return 0;

    } else if (atual.getEsquerda() == null) {
        return 1 + altura(atual.getDireita());

    } else if (atual.getDireita() == null) {
        return 1 + altura(atual.getEsquerda());

    } else { //retorna a maior das duas alturas
        return 1 + Math.max(altura(atual.getEsquerda()), altura(atual.getDireita()));
    }
}
```

A seguir, o código da rotação para a direita e esquerda:

```java
public No<T> rotacaoDireita(No<T> inicial) {

    No<T> esquerda = inicial.getEsquerda();
    esquerda.setPai(inicial.getPai());

    inicial.setEsquerda(esquerda.getDireita());

    if (inicial.getEsquerda() != null) {
        inicial.getEsquerda().setPai(inicial);
    }

    esquerda.setDireita(inicial);
    inicial.setPai(esquerda);

    if (esquerda.getPai() != null) {

        if (esquerda.getPai().getDireita() == inicial) {
            esquerda.getPai().setDireita(esquerda);

        } else if (esquerda.getPai().getEsquerda() == inicial) {
            esquerda.getPai().setEsquerda(esquerda);
        }
    }

    definaBalanceamento(inicial);
    definaBalanceamento(esquerda);

    return esquerda;
}


public No<T> duplaRotacaoEsquerdaDireita(No<T> inicial) {
    inicial.setEsquerda(rotacaoEsquerda(inicial.getEsquerda()));
    return rotacaoDireita(inicial);
}

public No<T> duplaRotacaoDireitaEsquerda(No<T> inicial) {
    inicial.setDireita(rotacaoDireita(inicial.getDireita()));
    return rotacaoEsquerda(inicial);
}
```

```java
public No<T> rotacaoEsquerda(No<T> inicial) {

    No<T> direita = inicial.getDireita();
    direita.setPai(inicial.getPai());

    inicial.setDireita(direita.getEsquerda());

    if (inicial.getDireita() != null) {
        inicial.getDireita().setPai(inicial);
    }

    direita.setEsquerda(inicial);
    inicial.setPai(direita);

    if (direita.getPai() != null) {

        if (direita.getPai().getDireita() == inicial) {
            direita.getPai().setDireita(direita);

        } else if (direita.getPai().getEsquerda() == inicial) {
            direita.getPai().setEsquerda(direita);
        }
    }

    definaBalanceamento(inicial);
    definaBalanceamento(direita);

    return direita;
}
```

Saída da árvore balanceada pré ordenada:

```
Inserção dos elementos:
52
27
12
5
19
39
30
45
63
59
57
71
69
85
81
Quantidade de Rotações
9
```