# Estruturas de Dados Básicas II - Avaliação 02

## 1. Implementação Computacional (Heap)

### 1.0 Ambiente Computacional

- **Software**:
  - Durante os testes e a implementação feitos por Vitor, foi utilizado como sistema operacional o Linux Mint 22 (Wilma), a IDE utilizada foi a CLion da JetBrains e como linguagem de programação, CPP.
- **Hardware**:
  - A máquina utilizada para os testes foi um notebook Lenovo Ideapad 1 com as seguintes configurações:
    - CPU: 12th Gen Intel© Core™ i5-1235U;
    - GPU: Intel® Iris® Xe Graphics eligible;
    - RAM: 8 GB DDR4-3.200MHz (Soldado); e
    - Armazenamento: 512 GB SSD M.2 2242 PCIe Gen4 TLC

### 1.1 Funções de Heap

A implementação de todas as funções de Heap e o HeapSort com comentários em português estão no repositório do GitHub do trabalho, no repositório. Mas segue a implmentação das funções em texto:

- Alteração de Prioridade.

```cpp
void changePriority(vector<int>& heap, const int index, const int newValue) {
    if (index < 1 || index >= heap.size()) {
        throw out_of_range("Out of range!");
    }

    const int oldValue = heap[index];
    heap[index] = newValue;

    if (compare(oldValue, newValue)) {
        heapifyUp(heap, index);
    }
    else {
        heapifyDown(heap, index);
    }
}
```

Como é possível visualizar, foram utilizados os seguintes métodos auxiliares: `compare()`, `heapfiyUp()` e `heapifyDown()`. Segue a implementação desses métodos:

```
bool compare(const int parent, const int child) const {
    return isMaxHeap ? parent < child : parent > child;
}

void heapifyUp(vector<int>& heap, const int index) {
    if (index > 1) {
        if (const int parent = index / 2;
            compare(heap[parent], heap[index])) {
            swap(heap[parent], heap[index]);
            heapifyUp(heap, parent);
        }
    }
}

void heapifyDown(vector<int>& heap, const int index, const int size = -1) {
    const int effectiveSize = (size == -1) ? heap.size() - 1 : size;
    const int left = 2 * index;
    const int right = left + 1;
    int largest = index;

    if (left <= effectiveSize &&
        compare(heap[largest], heap[left])) {
        largest = left;
    }

    if (right <= effectiveSize &&
        compare(heap[largest], heap[right])) {
        largest = right;
    }

    if (largest != index) {
        swap(heap[index], heap[largest]);
        heapifyDown(heap, largest, effectiveSize);
    }
}
```

- Inserção.

```
void insert(vector<int>& heap, const int value) {
    heap.push_back(value);
    heapifyUp(heap, heap.size() - 1);
}
```

- Remoção (da raiz).

```
int remove(vector<int>& heap) {
    if (heap.size() <= 1) {
        throw runtime_error("Heap is empty!");
    }
```

```
5
6       const int root = heap[1];
7       heap[1] = heap.back();
8       heap.pop_back();
9       heapifyDown(heap, 1);
10      return root;
11  }
```

- Construção das Heaps.

```
1   void makeHeap(vector<int>& heap, const vector<int>& arr) {
2       heap = {0};
3       heap.insert(heap.end(), arr.begin(), arr.end());
4
5       for (int i = (heap.size() - 1) / 2; i >= 1; --i) {
6           heapifyDown(heap, i);
7       }
8   }
```

Exemplos utilizados em sala de aula foram testados e os resultados apresentados no Apêndice.

## 1.2 Algoritmo Heapsort

### Pseudocódigo:

```
1   HeapSort(arr) {
2       ConstruirHeap(arr)
3
4       para i de tamanho(arr) - 1 até 1:
5           Trocar(arr[0], arr[i])
6           TamanhoDoHeap = TamanhoDoHeap - 1
7           Heapify(arr, 0, TamanhoDoHeap)
8   }
```

### Implementação:

Segue aqui também a implementação do heapSort:

```
1   void heapSort(vector<int>& arr) {
2       vector<int> heap;
3       makeHeap(heap, arr);
4
5       for (int i = arr.size() - 1; i >= 0; --i) {
6           arr[i] = heap[1];
7           heap[1] = heap[heap.size() - 1];
8           heap.pop_back();
9           heapifyDown(heap, 1, heap.size() - 1);
10      }
11  }
```

## 1.3 Listas Aleatórias

Listas geradas com os tamanhos 10.000, 100.000 e 1.000.000. As listas estão salvas em arquivos TXT disponíveis no repositório. Para criar as listas usamos um algoritmo simples cujo código implementado em C++, está no repositório.

## 1.4 Comparação de Algoritmos de Ordenação

Inicialmente, executamos o teste de ordenação das listas de dez mil, cem mil e um milhão de elementos. Para isso, rodamos o teste `./testHeap` e executamos o comando `--a`. Com isso, tivemos o seguinte resultado:

- Ordenação de 10k de elementos: 4.27783ms;
- Ordenação de 100k de elementos: 59.3884ms; e
- Ordenação de 1M de elementos: 675.067ms.

Agora, temos os seguintes resultados comparativos entre Heapsort, BubbleSort, MergeSort e QuickSort que foram implementados no trabalho anterior:

| **Método** | **1000** | **10000** | **100000** | **1000000** |
|---|---|---|---|---|
| BubbleSort Recursivo | 12.6331 ms | 645.952 ms | 37383.7 ms | - |
| MergeSort Recursivo | 0.8238 ms | 2.39819 ms | 15.0491 ms | - |
| QuickSort Recursivo | 0.483668 ms | 1.36376 ms | 9.43817 ms | - |
| BubbleSort Iterativo | 13.9092 ms | 322.769 ms | 34717.8 ms | - |
| MergeSort Iterativo | 0.553774 ms | 1.27057 ms | 15.0724 ms | - |
| QuickSort Iterativo | 1.45077 ms | 2.4314 ms | 29.2926 ms | - |
| HeapSort | - | 4.27783 ms | 59.3884 ms | 675.067 ms |

Table 1: Comparação de desempenho entre métodos de ordenação

Como esperado, o `HeapSort` é um algorítmo mais eficiente que o `BubbleSort` em qualquer caso, isso ocorre devido à complexidade quadrática do `BubbleSort`. Todavia, notamos que o `HeapSort` perde, em relação a eficiência, em todos os casos para os outros algorítmos de ordenação.

Por outro lado, o `HeapSort` foi o único algorítmo capaz de resolver, em tempo útil, os casos com um milhão de elementos.

Importante ressaltar que foi utilizado as listas de dados da avaliação anterior, assim para o heapsort foi usado os mesmos dados.

# 2. Implementação Computacional (Árvores Binárias)

## 2.0 Ambiente Computacional

- **Software**: João fez, usando o nvim, na linguagem *Haskell*. Todo o código foi escrito em programação literária, em *bird-style*. Desse modo, basta conferir o código para ver mais detalhes sobre a implementação.

## 2.1 Criação de Árvore Binária

```haskell
data Tree a where
 Nil  :: Tree a
 Tree :: { getVal   :: a
         , getLeft  :: Tree a
         , getRight :: Tree a
         }          -> Tree a
  deriving (Eq)
```

## 2.2 Percurso em Árvore Binária

- Pré-Ordem.

```haskell
inPreOrder :: Tree a -> [a]
inPreOrder Nil            = []
inPreOrder (Tree val l r) =
    [val] ++ inPreOrder l ++ inPreOrder r
```

- Ordem-Simétrica.

```haskell
inSimOrder :: Tree a -> [a]
inSimOrder Nil            = []
inSimOrder (Tree val l r) =
    inSimOrder l ++ [val] ++ inSimOrder r
```

- Pós-Ordem.

```haskell
inPosOrder :: Tree a -> [a]
inPosOrder Nil            = []
inPosOrder (Tree val l r) =
    inPosOrder l ++ inPosOrder r ++ [val]
```

- Em-Nível.

```haskell
-- usamos a função auxiliar:
projectFloor :: Integral i => i -> Tree a -> [a]
projectFloor _ Nil              = []
projectFloor num (Tree val tl tr)
    | num  < 0  = error "negative Floor"
    | num == 0  = [val]
    | num  > 0  = projectFloor (num - 1) tl
                ++ projectFloor (num - 1) tr

-- p/ finalmente definir (onde tbm definimos a height):
```

```haskell
inLevel :: Tree a -> [a]
inLevel tree = concat [projectFloor n tree | n <- [0 .. height tree - 1]]
```

## 2.3 Funções Implementadas

- Busca.

```haskell
search :: Eq a => a -> Tree a -> Bool
search x Nil = False
search x (Tree val l r) =
  x == val || search x l || search x r

```

Este outro codigo tambem permite achar todos os caminhos que terminam no valor buscado:

```haskell
find :: Eq a => a -> Tree a -> [Path]
find x Nil = []
  find x tree@(Tree val l r)
    | x == val  = case (pl, pr) of
                    ([], []) -> [[X]]
                    (_, _) -> map (X:) (pl ++ pr)

    | otherwise = case (pl, pr) of
                    ([], []) -> []
                    (xs, []) -> map (L:) xs
                    ([], ys) -> map (R:) ys
                    (xs, ys) -> map (L:) xs
                                ++ map (R:) ys

    where pl = find x l
          pr = find x r

```

- Inserção.

```haskell
insert :: Ord a => a -> Tree a -> Tree a
insert x Nil = leaf x
insert x tree@(Tree val l r)
  | x == val = tree
  | x  < val = Tree val (insert x l) r
  | x  > val = Tree val l (insert x r)
```

- Remoção.

```haskell
removeNode :: Path -> Tree a -> Tree a
removeNode _ Nil   = Nil
removeNode [] tree = tree
removeNode ds tree@(Tree x l r)
  | isLeaf      node = deleteNode ds tree
  | fatherOfOne node = killNode ds tree
  | fatherOfTwo node = deleteNode safeLeafPath $
                       tradeNodes ds safeLeafPath tree
  | otherwise        = Nil
  where node         = getNode ds tree
        safeLeafPath = concatPaths ds
                        (rightestFromLeft node)

-- Caso o leitor esteja em busca de um "removeVal", a nível de corretude,
-- basta definí-lo assim:

removeVal :: a -> Tree a -> Tree a
removeVal x tree =
    case (pathsTo x tree) of
        Nil     -> tree
        (p : _) -> removeNode p (removeVal x tree)
```

Testes realizados com exemplos em sala de aula.

# 3. Implementação Computacional (Árvores AVL)

## 3.0 Ambiente Computacional

- **Software**: Descrição do software utilizado.
  - **Sistema Operacional**: Ubuntu 22.04.5 LTS
  - **Linguagem de Programação**: C++
- **Hardware**: Descrição do hardware utilizado.
  - **Modelo do Notebook**: Lenovo IdeaPad S145
  - **Processador (CPU)**: Intel® Core™ i3-8130U CPU @ 2.20GHz
  - **Memória RAM**: 12,0 GB
  - **Armazenamento**: 1,5 TB (sendo 1TB HDD e 512GB SSD)

## 3.1 Funções Implementadas

- **Rotação:** A rotação é usada para balancear a árvore AVL, com os métodos de rotação para a direita, esquerda, e duplamente para a direita e esquerda.

```cpp
    Node* rightRotate(Node* y) {
        Node* x = y->left;
        Node* T2 = x->right;

```

```
        x->right = y;
        y->left = T2;

        updateHeight(y);
        updateHeight(x);

        return x;
    }

    Node* leftRotate(Node* x) {
        Node* y = x->right;
        Node* T2 = y->left;

        y->left = x;
        x->right = T2;

        updateHeight(x);
        updateHeight(y);

        return y;
    }

    Node* duplicateRightRotate(Node* y) {
        y->left = leftRotate(y->left);
        return rightRotate(y);
    }

    Node* duplicateLeftRotate(Node* x) {
        x->right = rightRotate(x->right);
        return leftRotate(x);
    }
```

- **Busca:**

```
    bool search(Node* root, int key) {
        if (!root) return false;
        if (root->key == key) return true;
        return key < root->key ? search(root->left, key) : search(root->right,
        ↪  key);
    }
```

- **Inserção:**

```
    Node* insert(Node* root, int key) {
        if (!root) return new Node(key);

        if (key < root->key)
            root->left = insert(root->left, key);
        else if (key > root->key)
            root->right = insert(root->right, key);
        else
            return root;

```

```
        updateHeight(root);
        return balance(root);
    }
```

- **Remoção:**

```
    Node* remove(Node* root, int key) {
        if (!root) return root;

        if (key < root->key)
            root->left = remove(root->left, key);
        else if (key > root->key)
            root->right = remove(root->right, key);
        else {
            if (!root->left || !root->right) {
                Node* temp = root->left ? root->left : root->right;
                delete root;
                return temp;
            }
            Node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = remove(root->right, temp->key);
        }

        updateHeight(root);
        return balance(root);
    }
```

O código fonte completo pode ser acessado no seguinte link: repositório

## 3.2 Teste com Valores Pré-Definidos

Segue abaixo o resultado da árvore AVL gerada a partir dos seguintes valores: **Valores:** $\{15, 18, 20, 35, 32, 38, 30, 40, 32, 45, 48, 52, 60, 50\}$

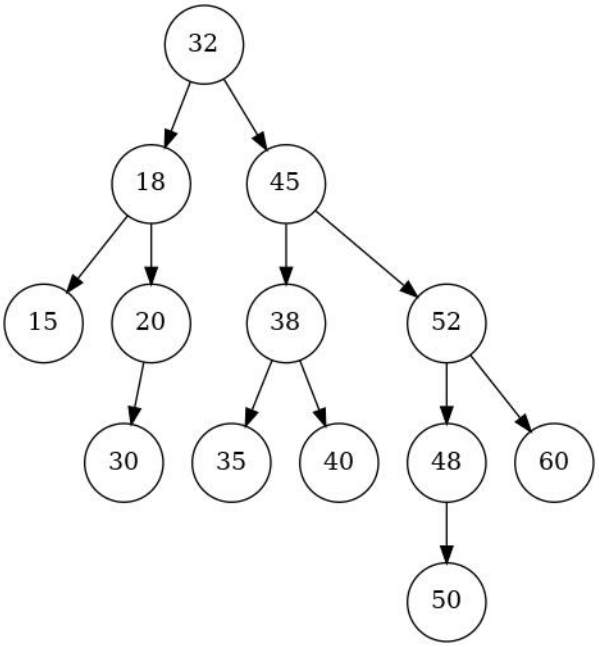


Figure 1: Representação gráfica da árvore AVL gerada

# 4. Implementação Computacional (Árvores Rubro-Negras)

## 4.0 Ambiente Computacional

- **Software**: Descrição do software utilizado.
  - **Sistema Operacional**: Windows
  - **Linguagem de Programação**: C++
- **Hardware**: Descrição do hardware utilizado.
  - **Modelo da GPU**: Nvidia GTX 1660 TI
  - **Processador (CPU)**: AMD Ryzen 5 3600
  - **Memória RAM**: 36 GB
  - **Armazenamento**: 3 TB (1TB HDD e 2TB SSD)

## 4.1 Fluxograma para a função de exclusão

Link do fluxograma

## 4.2 Funções Implementadas

- **Rotação:**

```
1       void rotacaoEsquerda(NoRB* x) {
2           NoRB* y = x->direita;
3           x->direita = y->esquerda;
```

```
4           if (y->esquerda != TNULL) {
5               y->esquerda->pai = x;
6           }
7           y->pai = x->pai;
8           if (x->pai == nullptr) {
9               root = y;
10          } else if (x == x->pai->esquerda) {
11              x->pai->esquerda = y;
12          } else {
13              x->pai->direita = y;
14          }
15          y->esquerda = x;
16          x->pai = y;
17      }
18      void rotacaoDireita(NoRB* x) {
19          NoRB* y = x->esquerda;
20          x->esquerda = y->direita;
21          if (y->direita != TNULL) {
22              y->direita->pai = x;
23          }
24          y->pai = x->pai;
25          if (x->pai == nullptr) {
26              root = y;
27          } else if (x == x->pai->direita) {
28              x->pai->direita = y;
29          } else {
30              x->pai->esquerda = y;
31          }
32          y->direita = x;
33          x->pai = y;
34      }
```

- **Busca:**

```
    NoRB* buscar(int chave) {
        NoRB* atual = root;
        while (atual != TNULL && chave != atual->item) {
            if (chave < atual->item) {
                atual = atual->esquerda;
            } else {
                atual = atual->direita;
            }
        }
        return atual;
    }

```

- **Inserção:**

```
    void inserir(int chave) {
        NoRB* novoNo = new NoRB(chave);
        novoNo->esquerda = TNULL;
        novoNo->direita = TNULL;

        NoRB* y = nullptr;
        NoRB* x = root;

        while (x != TNULL) {
            y = x;
            if (novoNo->item < x->item) {
                x = x->esquerda;
            } else {
                x = x->direita;
            }
        }

        novoNo->pai = y;
        if (y == nullptr) {
            root = novoNo;
        } else if (novoNo->item < y->item) {
            y->esquerda = novoNo;
        } else {
            y->direita = novoNo;
        }

        if (novoNo->pai == nullptr) {
            novoNo->cor = "Preto";
            return;
        }

        if (novoNo->pai->pai == nullptr) {
            return;
        }

        corrigirInsercao(novoNo);
    }

```

- **ConsertaInserção:**

```
    void corrigirInsercao(NoRB* k) {
        while (k->pai && k->pai->cor == "Vermelho") {
            if (k->pai == k->pai->pai->esquerda) {
                NoRB* tio = k->pai->pai->direita;
                if (tio && tio->cor == "Vermelho") {
                    k->pai->cor = "Preto";
                    tio->cor = "Preto";
                    k->pai->pai->cor = "Vermelho";
                    k = k->pai->pai;
                } else {
                    if (k == k->pai->direita) {
                        k = k->pai;
                        rotacaoEsquerda(k);
                    }
                    k->pai->cor = "Preto";
                    k->pai->pai->cor = "Vermelho";
                    rotacaoDireita(k->pai->pai);
                }
            } else {
                NoRB* tio = k->pai->pai->esquerda;
                if (tio && tio->cor == "Vermelho") {
                    k->pai->cor = "Preto";
                    tio->cor = "Preto";
                    k->pai->pai->cor = "Vermelho";
                    k = k->pai->pai;
                } else {
                    if (k == k->pai->esquerda) {
                        k = k->pai;
                        rotacaoDireita(k);
                    }
                    k->pai->cor = "Preto";
                    k->pai->pai->cor = "Vermelho";
                    rotacaoEsquerda(k->pai->pai);
                }
            }
        }
        root->cor = "Preto";
    }

```

- **Remoção:**

```
    void remover(int chave) {
        NoRB* z = buscar(chave);
        if (z == TNULL) return;

        NoRB* y = z;
        NoRB* x;
        string yCorOriginal = y->cor;

        if (z->esquerda == TNULL) {
            x = z->direita;
            substituirNo(z, z->direita);
        } else if (z->direita == TNULL) {
```

```
            x = z->esquerda;
            substituirNo(z, z->esquerda);
        } else {
            y = minimo(z->direita);
            yCorOriginal = y->cor;
            x = y->direita;
            if (y->pai == z) {
                x->pai = y;
            } else {
                substituirNo(y, y->direita);
                y->direita = z->direita;
                y->direita->pai = y;
            }
            substituirNo(z, y);
            y->esquerda = z->esquerda;
            y->esquerda->pai = y;
            y->cor = z->cor;
        }

        if (yCorOriginal == "Preto") {
            corrigirRemocao(x);
        }
    }
}
```

---

- **ConsertaRemoção:**

---

```
    void corrigirRemocao(NoRB* x) {
        while (x != root && x->cor == "Preto") {
            if (x == x->pai->esquerda) {
                NoRB* s = x->pai->direita;
                if (s->cor == "Vermelho") {
                    s->cor = "Preto";
                    x->pai->cor = "Vermelho";
                    rotacaoEsquerda(x->pai);
                    s = x->pai->direita;
                }
                if (s->esquerda->cor == "Preto" && s->direita->cor == "Preto")
                ↪  {
                    s->cor = "Vermelho";
                    x = x->pai;
                } else {
                    if (s->direita->cor == "Preto") {
                        s->esquerda->cor = "Preto";
                        s->cor = "Vermelho";
                        rotacaoDireita(s);
                        s = x->pai->direita;
                    }
                    s->cor = x->pai->cor;
                    x->pai->cor = "Preto";
                    s->direita->cor = "Preto";
                    rotacaoEsquerda(x->pai);
                    x = root;
                }
            } else {
                NoRB* s = x->pai->esquerda;
```

```
29                  if (s->cor == "Vermelho") {
30                      s->cor = "Preto";
31                      x->pai->cor = "Vermelho";
32                      rotacaoDireita(x->pai);
33                      s = x->pai->esquerda;
34                  }
35                  if (s->direita->cor == "Preto" && s->esquerda->cor == "Preto")
                    ↪  {
36                      s->cor = "Vermelho";
37                      x = x->pai;
38                  } else {
39                      if (s->esquerda->cor == "Preto") {
40                          s->direita->cor = "Preto";
41                          s->cor = "Vermelho";
42                          rotacaoEsquerda(s);
43                          s = x->pai->esquerda;
44                      }
45                      s->cor = x->pai->cor;
46                      x->pai->cor = "Preto";
47                      s->esquerda->cor = "Preto";
48                      rotacaoDireita(x->pai);
49                      x = root;
50                  }
51              }
52          }
53          x->cor = "Preto";
54      }
55
```

Testes realizados com exemplos em sala de aula.

## 4.3 Teste com Valores Pré-Definidos

Segue abaixo o resultado da árvore AVL gerada a partir dos seguintes valores:

**Valores:** $\{15, 18, 20, 35, 32, 38, 30, 40, 32, 45, 48, 52, 60, 50\}$

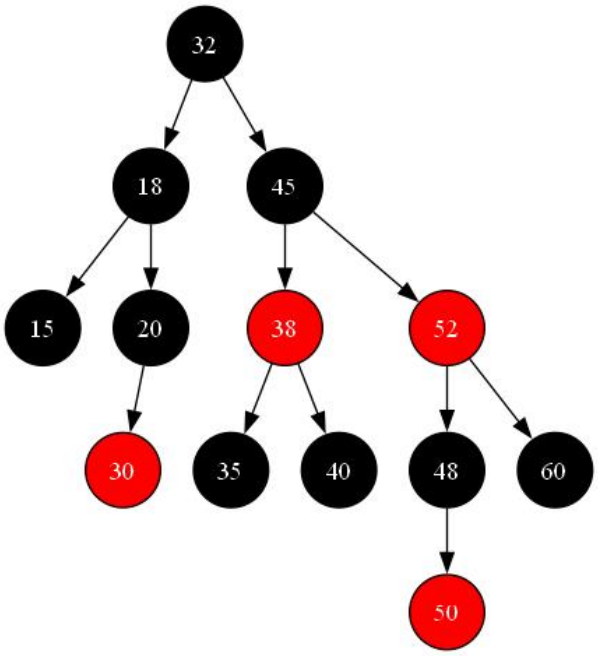


Figure 2: Representação gráfica da árvore Rubro Negra gerada

# Apêndice

## Link para o video

## Listagem das tarefas feita por cada integrante

- Felipe Augusto Lemos Barreto
  - Implementou a árvore rubro-negra em Cpp
  - Implementou as estruturas de dados em Python
  - Fez parte da organização da questão 4 no documento
- Francisca Gabrielly Lopes Freire
  - Implementou a arvoré AVL em Cpp
  - Implementou algumas árvores em JavaScript
  - Organizou a questão 3 no documento
- Gustavo Henrique Araujo de Sales Leite
  - Implementou as estruturas de dados em Java
  - Fez parte da organização da questão 4 no documento
- João Lucas de Moraes Pereira
  - Implementou a árvore binária em Cpp

  - Implementou as estruturas de dados em Haskell
  - Organizou a questão 2 no documento
  - Gravou o vídeo
- Vitor Emanuel Rodrigues de Alencar
  - Implementou a árvore binária e a heap em Cpp
  - Organizou a questão 1 no documento