# Estrutura de Dados e Algoritmos I

Trabalho Da Disciplina


Gabriel Moreira Ferreira - 220115813

Novembro, 2023

# Sumário

# 1 Introdução

O objetivo deste trabalho é elaborar um programa que seja capaz de implementar uma **Árvore Binária de Busca** juntamente com os algoritmos de: inclusão, exclusão, busca e caminhamento pré-ordem de forma iterativa. Ademais, também é implementado um algoritmo que seja capaz de exibir a estrutura da árvore binária de busca.

# 2 Desenvolvimento

O desenvolvimento deste trabalho se deu a partir da implementação de alguns algoritmos que serão exibidos a seguir.

## 2.1 Definindo classes

Primeiramente, foi necessário definir as classes que serão utilizadas pelos algoritmos do projeto. As classes definidas foram: Nó e Elemento.

A classe **Nó** é a classe que armazenará os dados (valor, elemento a esquerda, elemento a direta e elemento pai) de cada nó da árvore binária de busca. Já a classe **Elemento** é a classe que armazenará os dados (nó e próximo elemento) de cada elemento da pilha. Esta pilha será implementada de forma encadeada e terá uso no algoritmo de caminhamento em pré-ordem.

```
classe No:
  valor = nulo
  esq = nulo
  dir = nulo
  pai = nulo

classe Elemento:
  no = nulo
  prox = nulo
```

## 2.2 Algoritmos da pilha

Nesta etapa foi definido os algoritmos de empilhar e desempilhar que serão utilizados na pilha.

No algoritmo de **empilhar** é criado uma variável da classe Elemento e atribuído à sua propriedade "nó"o Nó da árvore que será recebido para ser empilhado, depois é feito com que o novo elemento possua sua propriedade "prox"para apontar para o elemento que está no topo da pilha, após isso é alterado o ponteiro do topo para o elemento recém criado e então o novo elemento do topo é retornado.

No algoritmo de **desempilhar** primeiro é verificado se a pilha não está vazia, caso não esteja, então o ponteiro de topo da pilha é atualizado para o próximo elemento. Ao fim, é retornado o elemento que foi desempilhado.

```
funcao empilhar(no, topo):
```

```
    novo = Elemento()
    novo.no = no
    novo.prox = topo
    topo = novo
    return topo

funcao desempilhar(topo):
    if (topo != nulo):
        aux = topo
        topo = topo.prox
        return aux
```

## 2.3 Algoritmo de busca

O algoritmo de **busca** é responsável por buscar um valor de chave na árvore binária de busca. Para compreender seu funcionamento, o mesmo pode ser dividido em 4 partes:

1. Primeiramente é verificado se a árvore está vazia, caso esteja, é atribuído 0 à variável f e nulo à variável pai;

2. Caso o valor de chave buscado tenha sido encontrado, é atribuído 1 à variável f e a variável pai passa a ser o pai do elemento buscado;

3. Caso o valor de chave buscado seja menor que o valor do nó que está sendo verificado no momento, é feito a verificação se o nó a esquerda é vazio, caso seja, significa que o valor de chave não foi encontrado e é ali que ele deve ser inserido caso seja executada uma chamada de inclusão, então é atribuído 2 à variável f e a variável pai passa a ser o pai do elemento buscado. Caso o nó da esquerda não seja vazio, então a função de busca é chamada recursivamente para operar na subárvore esquerda do nó atual;

4. Caso o valor de chave buscado seja maior que o valor do nó que está sendo verificado no momento, é realizado o mesmo procedimento detalhado acima, mas dessa vez com a subárvore direita do nó atual, alterando apenas o valor de retorno da variável f, que passa a ser 3 caso o valor de chave não seja encontrado.

Ao fim do algoritmo, é retornado o nó da árvore onde o valor de chave foi encontrado ou então o nó onde o elemento pode ser inserido caso seja realizada uma operação de inclusão. Além disso também é retornado o pai do nó encontrado e o valor da variável f. Caso f seja 0, indica que a árvore está vazia, caso f seja 1, indica que o valor de chave buscado existe na árvore, caso f seja 2, indica que o valor de chave pode ser inserido à esquerda do nó pai e, por fim, caso f seja 3, significa que o valor de chave pode ser inserido num nó à direita do nó pai.

```
funcao busca(pont, chave):
    if (pont == nulo):
        f = 0
        pai = nulo
    else:
```

```
    if (chave == pont.valor):
      f = 1
      pai = pont.pai
    else:
      if (chave < pont.valor):
        if (pont.esq == nulo):
          f = 2
          pai = pont.pai
        else:
          pont = pont.esq
          pont, pai, f = busca(pont, chave)
      else:
        if (pont.dir == nulo):
          f = 3
          pai = pont.pai
        else:
          pont = pont.dir
          pont, pai, f = busca(pont, chave)
  return pont, pai, f
```

## 2.4 Algoritmo de inclusão

O algoritmo de **inclusão** é responsável por adicionar um novo nó à árvore binária.

Primeiramente, é realizado um procedimento de busca usando a função busca. Se o valor de chave já existe na árvore (f == 1), é exibido uma mensagem informando que o valor já está na árvore. Se a árvore está vazia (f == 0), um novo nó é criado e atribuído como raiz. Se o valor de chave não foi encontrado (f == 2 ou f == 3), um novo nó é criado e adicionado à esquerda (f == 2) ou à direita (f == 3) do nó pai, dependendo da comparação entre valores.

```
funcao inclusao(chave, raiz):
  pont = raiz
  pont, pai, f = busca(pont, chave)
  if (f == 1):
    print("Valor ja existe na arvore binaria.")
  else:
    novo = No()
    novo.valor = chave
    if (f == 0):
      raiz = novo
    else:
      if (f == 2):
        pont.esq = novo
        novo.pai = pont
      else:
        pont.dir = novo
        novo.pai = pont
```

## 2.5 Algoritmo de exlusão

O algoritmo de **exclusão** é responsável por remover um nó específico da árvore, mantendo as propriedades da estrutura de uma árvore de busca binária.

Primeiramente, é realizado um procedimento de busca usando o algoritmo de busca. Se a árvore está vazia (f == 0), é exibido uma mensagem informando que a árvore está vazia. Se o valor de chave não foi encontrado (f == 2 ou f == 3), é exibido uma mensagem informando que o valor não existe na árvore. Se o valor de chave foi encontrado (f == 1), a exclusão é realizada.

A função busca é chamada para encontrar o nó a ser removido e obter informações sobre a árvore (valor de f). Se f for igual a 1, indica que o valor a ser removido foi encontrado na árvore e o algoritmo de exclusão é então executado:

1. Se o nó a ser removido não tem filho à esquerda, substitui o nó pelo seu filho à direita (ou nulo se não houver);

2. Se o nó a ser removido não tem filho à direita, substitui o nó pelo seu filho à esquerda;

3. Se o nó a ser removido tem ambos os filhos, encontra-se então o nó sucessor, que é o nó de menor valor na subárvore direita do nó a ser excluído. Assim que encontrado, ele passa a ocupar a posição do nó a ser removido e, em seguida, remove-se o nó de valor de chave solicitado.

```
funcao exclusao(chave, raiz):
  pont, pai, f = busca(raiz, chave)
  if (f == 0):
    print("A arvore binaria esta vazia.")
  else:
    if (f == 1):
      if (pont.esq == nulo):
        if (pont == raiz):
          raiz = raiz.dir
        else:
          if (pont == pai.esq):
            pai.esq = pont.dir
          else:
            pai.dir = pont.dir
      else:
        if (pont.dir == nulo):
          if (pont == raiz):
            raiz = raiz.esq
          else:
            if (pont == pai.esq):
              pai.esq = pont.esq
            else:
              pai.dir = pont.esq
        else:
          pont_aux = pont.dir
          pai_aux = pont
```

```
            while (pont_aux.esq != nulo):
              pai_aux = pont_aux
              pont_aux = pont_aux.esq
            if (pai_aux != pont):
              pai_aux.esq = pont_aux.dir
              pont_aux.dir = pont.dir
            pont_aux.esq = pont.esq
            if (pont == raiz):
              raiz = pont_aux
            else:
              if (pai.esq == pont):
                pai.esq = pont_aux
              else:
                pai.dir = pont_aux
        del(pont)
      else:
        print("Valor nao existe na arvore binaria.")
```

## 2.6 Algoritmo de pré-ordem

O algoritmo de **pré-ordem** é responsável por percorrer a árvore binária e visitar cada nó antes de seus filhos, seguindo a ordem: raiz, esquerda, direita.

O algoritmo inicia verificando se a árvore está vazia, se estiver, exibe uma mensagem informando que a árvore está vazia e não faz o caminhamento pré-ordem. Caso não esteja vazia, é utilizado uma pilha encadeada para armazenar os nós à medida que são visitados. Enquanto a pilha não estiver vazia ou o ponteiro pont não for nulo, continua o loop. Se o ponteiro pont não for nulo, imprime o valor do nó e o empilha. Após isso, verifica se a pilha não está vazia, e então desempilha um nó e move para sua subárvore direita.

```
funcao pre_ordem(pont, raiz, topo):
  if (raiz == nulo):
    print("A arvore binaria esta vazia.")
  else:
    while (topo != nulo or pont != nulo):
      while (pont != nulo):
        print(pont.valor)
        empilhar(pont, topo)
        pont = pont.esq
      if (topo != nulo):
        pont = desempilhar(topo).no
        pont = pont.dir
```

## 2.7 Algoritmo de mostrar árvore

O algoritmo de **mostrar árvore** é responsável por exibir a estrutura da árvore binária de busca de forma visual.

O algoritmo inicia verificando se a árvore está vazia, caso esteja, é exibido uma mensagem informando que a árvore está vazia. Caso não esteja, é utilizado uma abordagem recursiva para percorrer a árvore em ordem inversa (direita, raiz, esquerda), apresentando visualmente os nós e suas subárvores. Cada nó é exibido com espaçamento proporcional ao seu nível na árvore.

```
funcao mostrar_arvore(pont, level, raiz):
  if (raiz == nulo):
    print("A arvore binaria esta vazia.")
  if (pont != nulo):
    mostrar_arvore(pont.dir, level + 1, raiz)
    print(" " * 5 * level, "--", pont.valor)
    mostrar_arvore(pont.esq, level + 1, raiz)
```

## 2.8 Algoritmo principal

O **algoritmo principal** main() é responsável por criar um menu interativo para o usuário interagir com as operações da árvore binária de busca: inclusão, exclusão, caminhamento pré-ordem, exibição da árvore e encerramento do programa.

O algoritmo é composto por um loop que continua até o usuário escolher a opção de encerramento. Nele, é exibido um menu com opções numeradas para o usuário escolher, sendo as opções:

1. Inclusão;
2. Exclusão;
3. Caminhamento pré-ordem;
4. Mostrar a árvore;
5. Fim.

Após o usuário digitar a opção desejada, é executado o algoritmo correspondente à sua escolha (inclusao, exclusao, pre_ordem ou mostrar_arvore). Se a escolha do usuário não corresponder a nenhuma operação válida, é exibido uma mensagem indicando uma opção inválida. O loop continua até o usuário escolher a opção de encerramento.

```
funcao main():
  escolha = -1
  while (escolha != 5):
    print("Escolha uma das opcoes do menu abaixo.")
    print("""
      1 - Inclusao
      2 - Exclusao
      3 - Caminhamento pre-ordem
      4 - Mostrar a arvore
      5 - Fim
    """)
```

```
12      escolha = input("Sua escolha: ")
13
14      if (escolha == 1):
15        chave = input("Digite o valor que voce deseja
              incluir na arvore: ")
16        inclusao(chave, raiz)
17      else:
18        if (escolha == 2):
19          chave = input("Digite o valor que voce deseja
                excluir da arvore: ")
20          exclusao(chave, raiz)
21        else:
22          if (escolha == 3):
23            print("Caminhamento pre-ordem da arvore binaria
                  :")
24            pre_ordem(pont, raiz, topo)
25          else:
26            if (escolha == 4):
27              print("Exibindo a arvore binaria:")
28              mostrar_arvore(pont, level, raiz)
29            else:
30              if (escolha != 5):
31                print("Opcao invalida.")
```

# 3 Algoritmo completo

```
classe No:
  valor = nulo
  esq = nulo
  dir = nulo
  pai = nulo

classe Elemento:
  no = nulo
  prox = nulo

funcao empilhar(no, topo):
  novo = Elemento()
  novo.no = no
  novo.prox = topo
  topo = novo
  return topo

funcao desempilhar(topo):
  if (topo != nulo):
    aux = topo
    topo = topo.prox
    return aux

funcao main():
  escolha = -1
  while (escolha != 5):
    print("Escolha uma das opcoes do menu abaixo.")
    print("""
      1 - Inclusao
      2 - Exclusao
      3 - Caminhamento pre-ordem
      4 - Mostrar a arvore
      5 - Fim
    """)
    escolha = input("Sua escolha: ")

    if (escolha == 1):
      chave = input("Digite o valor que voce deseja
          incluir na arvore: ")
      inclusao(chave, raiz)
    else:
      if (escolha == 2):
        chave = input("Digite o valor que voce deseja
            excluir da arvore: ")
        exclusao(chave, raiz)
      else:
```

```
45          if (escolha == 3):
46            print("Caminhamento pre-ordem da arvore binaria
                  :")
47            pre_ordem(pont, raiz, topo)
48          else:
49            if (escolha == 4):
50              print("Exibindo a arvore binaria:")
51              mostrar_arvore(pont, level, raiz)
52            else:
53              if (escolha != 5):
54                print("Opcao invalida.")
55
56  funcao busca(pont, chave):
57    if (pont == nulo):
58      f = 0
59      pai = nulo
60    else:
61      if (chave == pont.valor):
62        f = 1
63        pai = pont.pai
64      else:
65        if (chave < pont.valor):
66          if (pont.esq == nulo):
67            f = 2
68            pai = pont.pai
69          else:
70            pont = pont.esq
71            pont, pai, f = busca(pont, chave)
72        else:
73          if (pont.dir == nulo):
74            f = 3
75            pai = pont.pai
76          else:
77            pont = pont.dir
78            pont, pai, f = busca(pont, chave)
79    return pont, pai, f
80
81  funcao inclusao(chave, raiz):
82    pont = raiz
83    pont, pai, f = busca(pont, chave)
84    if (f == 1):
85      print("Valor ja existe na arvore binaria.")
86    else:
87      novo = No()
88      novo.valor = chave
89      if (f == 0):
90        raiz = novo
91      else:
92        if (f == 2):
93          pont.esq = novo
```

```
94          novo.pai = pont
95        else:
96          pont.dir = novo
97          novo.pai = pont
98
99  funcao exclusao(chave, raiz):
100   pont, pai, f = busca(raiz, chave)
101   if (f == 0):
102     print("A arvore binaria esta vazia.")
103   else:
104     if (f == 1):
105       if (pont.esq == nulo):
106         if (pont == raiz):
107           raiz = raiz.dir
108         else:
109           if (pont == pai.esq):
110             pai.esq = pont.dir
111           else:
112             pai.dir = pont.dir
113       else:
114         if (pont.dir == nulo):
115           if (pont == raiz):
116             raiz = raiz.esq
117           else:
118             if (pont == pai.esq):
119               pai.esq = pont.esq
120             else:
121               pai.dir = pont.esq
122         else:
123           pont_aux = pont.dir
124           pai_aux = pont
125           while (pont_aux.esq != nulo):
126             pai_aux = pont_aux
127             pont_aux = pont_aux.esq
128           if (pai_aux != pont):
129             pai_aux.esq = pont_aux.dir
130             pont_aux.dir = pont.dir
131           pont_aux.esq = pont.esq
132           if (pont == raiz):
133             raiz = pont_aux
134           else:
135             if (pai.esq == pont):
136               pai.esq = pont_aux
137             else:
138               pai.dir = pont_aux
139       del(pont)
140     else:
141       print("Valor nao existe na arvore binaria.")
142
143 funcao pre_ordem(pont, raiz, topo):
```

```
144   if (raiz == nulo):
145     print("A arvore binaria esta vazia.")
146   else:
147     while (topo != nulo or pont != nulo):
148       while (pont != nulo):
149         print(pont.valor)
150         empilhar(pont, topo)
151         pont = pont.esq
152       if (topo != nulo):
153         pont = desempilhar(topo).no
154         pont = pont.dir
155 
156 funcao mostrar_arvore(pont, level, raiz):
157   if (raiz == nulo):
158     print("A arvore binaria esta vazia.")
159   if (pont != nulo):
160     mostrar_arvore(pont.dir, level + 1, raiz)
161     print(" " * 5 * level, "--", pont.valor)
162     mostrar_arvore(pont.esq, level + 1, raiz)
```

# 4 Conclusão

Ao fim deste trabalho foi possível consolidar muitos conceitos referentes à Estrutura de Dados bem como suas aplicações. Paralelamente à isto, foi enriquecedor o conhecimento obtido nas linguagens Algorítmicas e Python. Ademais, este projeto não apenas demonstra a aplicação prática de estruturas de dados, mas também ressalta a importância de uma implementação eficiente dessas estruturas para a otimização de algoritmos. Ao criar e manipular uma Árvore Binária de Busca, foi possível explorar conceitos fundamentais de estruturas de dados, contribuindo para um entendimento mais aprofundado da organização e busca em conjuntos de dados.