

SME0827 - Estruturas de Dados

Árvores Binárias de Busca Aula 15

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Moisés Rocha dos Santos
Monitora: Marília Costa Rosendo Silva



Xadrez

- Em 1997, campeão mundial da época, Gary Kasparov, jogou 6 jogos contra deep blue
 - Programa escrito por pesquisadores da IBM
 - Deep Blue venceu 3, perdeu 2, empatou 1
 - Deep Blue avaliava 126.000.000 tabuleiros por segundo
 - Avaliava 30 bilhões de posições por movimento, atingindo, várias vezes, profundidade 14



© André de Carvalho - ICMC/USP

4

Hoje

- Busca
- Busca binária
- Dicionário
- Dicionários ordenados em árvores binárias
- Operações básicas em árvores binárias
 - Busca
 - Inserção
 - Remoção

© André de Carvalho - ICMC/USP

2

Dicionários

- Tipo abstrato de dados (TAD) *Dictionary*
 - Cada item tem uma parte chave e uma parte dados
 - Conjunto dinâmico com métodos:
 - Search (D, k): método de consulta que retorna um ponteiro x para um item, onde $x.chave = k$
 - Insert(D, x): método que adiciona ao dicionário D o item apontado por x
 - Delete (D, x): método que remove do dicionário D o item apontado por x

© André de Carvalho - ICMC/USP

5

Introdução

- Algoritmos de busca são utilizados em várias aplicações
 - Jogos
 - Os melhores jogadores de dama, go e xadrez são algoritmos de busca
 - Encontrar caminho mínimo
 - Caixeiro viajante, Waze
 - Busca na internet
 - Encontrar os sites mais relevantes
 - Buscar contatos em redes sociais

© André de Carvalho - ICMC/USP

3

Dicionários ordenados

- Além das funções anteriores, também deve permitir operações de fila de prioridades
 - Min(D)
 - Max(D)
- Também seria útil incluir os métodos
 - Antecessor(D, k)
 - Sucessor(D, k)
- Essas operações precisam que as chaves sejam comparáveis

© André de Carvalho - ICMC/USP

6

Dicionários ordenados

- Estrutura de dados básica para dicionários ordenados
 - Lista encadeada ordenada
 - Array ordenado

© André de Carvalho - ICMC/USP 7

Busca binária

- Buscar por um item com uma dada chave em uma sequência de itens ordenados pela chave
 - Faz buscas sucessivas pelo item do meio de uma sequência de itens com chave
 - Pode ser implementada por um array de itens ordenados pelo valor da chave
 - A cada passo, o número de itens candidatos é reduzido pela metade
 - Complexidade: $O(\log n)$
 - Ex.: encontrar item com chave 7 na sequência a seguir

© André de Carvalho - ICMC/USP 10

Arrays ordenados

- $O(n)$ insert/delete:
- $O(\lg(n))$ search, $O(1)$ select:

Search: Busca binária para ver se 3 está no array A

Select: A[3]

© André de Carvalho - ICMC/USP 8

Busca binária

Se m tem a chave procurada (valor da chave = 7)
 Então encontrou o item com a chave procurada
 Senão Se a chave for menor que a chave de m
 Então buscar no lado esquerdo de m
 Senão buscar no lado direito de m

© André de Carvalho - ICMC/USP 11

Listas encadeadas

- $O(1)$ insert/delete
 - Supor ter um ponteiro para a localização da operação inserir/excluir:
- $O(n)$ search/select:

© André de Carvalho - ICMC/USP 9

Árvores

- Árvores binárias de busca
 - Binary Search Trees (BSTs)
- Estrutura de dados baseada em busca binária
- Por que árvores?
 - Por que muitos problemas reais de busca são representados por árvores

© André de Carvalho - ICMC/USP 12

Complexidade da busca

	Arrays ordenados	Listas encadeadas	Árvores binárias de busca (BST)
Search	$O(\lg(n))$	$O(n)$	$O(\lg(n))$
Insert / Delete	$O(n)$	$O(1)$	$O(\lg(n))$

© André de Carvalho - ICMC/USP

13

Problema de busca em árvores

- Entrada:
 - Descrição dos nós inicial e alvo
 - Procedimento que gera os sucessores para um nó
- Saída:
 - Sequência válida de nós, do nó inicial ao nó alvo
 - Exemplo: palavras cruzadas

© André de Carvalho - ICMC/USP

16

Árvores

- Estrutura hierárquica

© André de Carvalho - ICMC/USP

14

Exemplo: palavras cruzadas

André Ponce de Leon F de Carvalho

17

Árvores

- Grafos acíclicos conexos em que cada nó

© André de Carvalho - ICMC/USP


15

Exemplo: labirinto

André Ponce de Leon F de Carvalho

18

Pausa



André de Carvalho - ICMC/USP

Varredura de árvore

- Permite imprimir as chaves de uma árvore
 - Ex.: Varredura infixa (central)
 - Imprime a chave de um nó entre a impressão das chaves das sub-árvores da esquerda e da direita

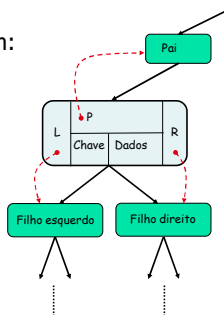
```
InorderTreeWalk(x)
01 if x ≠ NIL then
02     InorderTreeWalk(x.left())
03     print x.chave()
04     InorderTreeWalk(x.right())
```

- Algoritmo de divisão e conquista
- Imprime itens em ordem crescente
- Complexidade: $O(n)$

© André de Carvalho - ICMC/USP

Árvores binárias de busca BST

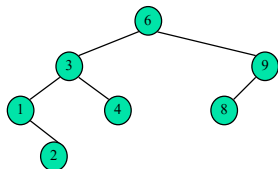
- Árvore em que cada nó tem:
 - Dados: valores armazenados em cada nó
 - Chave: identifica item, permitindo sua ordenação
 - L: ponteiro para filho da esquerda (pode ser None)
 - R: ponteiro para filho da direita (pode ser None)
 - P: ponteiro para nó pai (None para a raiz)



© André de Carvalho - ICMC/USP

Varredura de árvore

- Uma varredura infixa (ou central) de uma árvore binária de busca
 - Visita as chaves em ordem crescente de valor



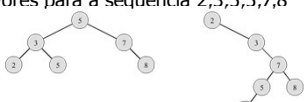
1, 2, 3, 4, 6, 8, 9

© André de Carvalho - ICMC/USP

Árvores binárias de busca BST

- Uma árvore binária em que:
 - Cada nó interno x armazena um item (k, d)
 - As chaves armazenadas em nós da subárvore esquerda de x são menores ou iguais a k
 - As chaves armazenadas em nós da subárvore direita de x são maiores ou iguais a k
 - Cada sub-árvore é uma árvore binária de busca
 - Ex.: Possíveis árvores para a sequência 2,3,5,5,7,8

BST é uma representação gráfica de uma sequência ordenada de chaves



© André de Carvalho - ICMC/USP

Outras varreduras de árvores

- Varredura em pré-ordem (ou prefixa)
 - Imprime (processa) cada nó antes de imprimir (processar) seus filhos
- Varredura em pós-ordem (ou posfixa)
 - Imprime (processa) cada nó após imprimir(processar) seus filhos
- Como fica a impressão nesses dois casos para a árvore anterior?

© André de Carvalho - ICMC/USP

Dividir para conquistar

- Abordagem natural para algoritmos em árvores
- Exemplo: Encontre a altura da árvore:
 - Se a árvore é NIL, Então altura = -1
 - Senão, altura = altura máxima dos seus filhos + 1

© André de Carvalho - ICMC/USP 25

Exemplo

- Search(7, 6)

© André de Carvalho - ICMC/USP 28

Busca de chave em BST

- Para buscar item com chave k, traçar caminho descendente começando na raiz
 - O próximo nó visitado depende da comparação de k com a chave do nó atual
 - Se chegar a uma folha, não encontrou a chave
 - Ex: encontrar item com chave 4:
 - Chamada TreeSearch(4, raiz)
 - Os algoritmos para consultas aos vizinhos mais próximos na sequência são semelhantes
 - Antes ou depois da chave

© André de Carvalho - ICMC/USP 26

Busca de chave na árvore

```
# Busca uma chave em uma BST
def search(raiz, chave):
    if raiz is None or raiz.val == chave: # Caso base da recursão
        return raiz
    if raiz.val < chave: # Chave é menor que chave da raiz?
        return search(raiz.right, chave)
    return search(raiz.left, chave) # Chave é maior que chave da raiz
```

© André de Carvalho - ICMC/USP 29

Exemplo

- Search(7, 11)

© André de Carvalho - ICMC/USP 27

Exemplo

- Insert 8

© André de Carvalho - ICMC/USP 30

Inserir item na árvore

```
# Operação de inserção em uma BST
class Node: # Classe que representa os nós na BST
    def __init__(self, chave):
        self.left = None
        self.right = None
        self.val = chave

def insert(raiz, node): # inserção de nó com uma dada chave
    if raiz is None:
        raiz = node
    else:
        if raiz.val < node.val:
            if raiz.right is None:
                raiz.right = node
            else:
                insert(raiz.right, node)
        else:
            if raiz.left is None:
                raiz.left = node
            else:
                insert(raiz.left, node)
```

```
def inorder(raiz): # varredura infixa
    if raiz:
        inorder(raiz.left)
        print(raiz.val)
        inorder(raiz.right)

r = Node(50) # constrói árvore
insert(r, Node(30))
insert(r, Node(20))
insert(r, Node(40))
insert(r, Node(70))
insert(r, Node(60))
insert(r, Node(80))

inorder(r) # Imprime árvore inorder
```

© André de Carvalho - ICMC/USP 31

Exemplo de remoção 0

■ Se x não tem filhos, basta remover x

Remover o nó 4

© André de Carvalho - ICMC/USP 34

Inserir item na árvore

```
# Operação de inserção em uma BST
class Node: # Classe que representa os nós na BST
    def __init__(self, chave):
        self.left = None
        self.right = None
        self.val = chave

def insert(raiz, node): # inserção de nó com uma dada chave
    if raiz is None:
        raiz = node
    else:
        if raiz.val < node.val:
            if raiz.right is None:
                raiz.right = node
            else:
                insert(raiz.right, node)
        else:
            if raiz.left is None:
                raiz.left = node
            else:
                insert(raiz.left, node)
```

```
def inorder(raiz): # varredura infixa
    if raiz:
        inorder(raiz.left)
        print(raiz.val)
        inorder(raiz.right)

r = Node(50) # constrói árvore
insert(r, Node(30))
insert(r, Node(20))
insert(r, Node(40))
insert(r, Node(70))
insert(r, Node(60))
insert(r, Node(80))

inorder(r) # Imprime árvore inorder
```

20

30

40

50

60

70

80

© André de Carvalho - ICMC/USP 32

Exemplo de remoção 1

■ Se x tem exatamente um filho, para excluir x, fazer com que pai de x aponte para esse filho

Remover o nó 10

© André de Carvalho - ICMC/USP 35

Remoção de item

- Excluir nó x de uma árvore T
- Podemos distinguir três casos
 - Nó não tem filhos
 - Remover x
 - Nó tem um filho
 - Fazer com que pai de x aponte para filho de x
 - Nó tem dois filhos
 - Mais complicado

© André de Carvalho - ICMC/USP 33

Exemplo de remoção 3

Remover o nó 5

■ Se x tem dois filhos, então para excluí-lo temos que:

- Encontrar seu sucessor y
 - Nó y que segue x em uma travessia inorder
- Substituir x por y
- Remover y anterior
 - Note que y tem no máximo uma criança
 - Porque?

© André de Carvalho - ICMC/USP 36

Exemplo de remoção 3

Remover o nó D

- Se x tem dois filhos, então para excluí-lo temos que:
 - Encontrar seu sucessor y
 - Nó y que segue x em uma travessia inorder
 - Substituir x por y
 - Remover y anterior
 - Note que y tem no máximo uma criança
 - Porque?
 - Por que é o item de menor chave na árvore a direita de x

© André de Carvalho - ICMC/USP 37

Conclusão

- Importância da busca
- Busca binária
- Dicionários ordenados
- Operações básicas em árvores binárias
 - Busca
 - Inserção
 - Remoção

© André de Carvalho - ICMC/USP 40

Remover item da árvore 1

```
class Node: # Nó BST
    def __init__(si, chave): # construtor cria novo nó
        si.chave = chave
        si.left = None
        si.right = None

    def inorder(raiz): # varredura infixa de BST
        if raiz is not None:
            inorder(raiz.left)
            print(raiz.chave)
            inorder(raiz.right)

    def minValueNode( node): # retorna nó com menor chave
        current = node # de BST não vazia
        while(current.left is not None): # loop até folha mais a esq
            current = current.left
        return current
```

© André de Carvalho - ICMC/USP 38

Perguntas

© André de Carvalho - ICMC/USP 41

Remover item da árvore 2

```
# Dada uma BST e uma chave, deleta chave e
# retorna nova raiz
def deleteNode(raiz, chave): # deleta nó da chave e retorna
    # nova raiz
    if raiz is None: # caso base
        return raiz
    # Se a chave < raiz, chave está na subárvore da esquerda
    if chave < raiz.chave:
        raiz.left = deleteNode(raiz.left, chave)
    # Se a chave > raiz, chave está na subárvore da direita
    elif chave > raiz.chave:
        raiz.right = deleteNode(raiz.right, chave)
    # Se a chave < raiz, chave está no nó a ser deletado
    else: # Não tem um ou nenhum filho
        if raiz.left is None:
            temp = raiz.right
            raiz = None
            return temp
        elif raiz.right is None:
            temp = raiz.left
            raiz = None
            return temp
        # Não tem dois filhos, pegar sucessor infixo
        # (menor da subárvore direita)
        temp = minValueNode(raiz.right)
        # Copia sucessor infixo neste nó
        raiz.chave = temp.chave
        # Deleta sucessor infixo
        raiz.right = deleteNode(raiz.right, temp.chave)
        return raiz

raiz = None
raiz = insert(raiz, 50)
raiz = insert(raiz, 30)
raiz = insert(raiz, 20)
raiz = insert(raiz, 40)
raiz = insert(raiz, 70)
raiz = insert(raiz, 60)
raiz = insert(raiz, 80)
print("Delete 20")
raiz = deleteNode(raiz, 20)
print("Varredura Inorder da nova BST")
```

Delete 20
Varredura Inorder da nova BST
30
40
50
60
70
80

© André de Carvalho - ICMC/USP 39