



Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

3 utad
Pedro Melo-Pinto 2022

Um algoritmo de

PESQUISA

é um algoritmo que procura um elemento (conhecido à partida) numa lista.

O resultado deste algoritmo é, geralmente, a posição na lista em que o elemento procurado se encontra (ou -1, posição impossível, caso o elemento não tenha sido encontrado na lista em causa).

Este problema é também um dos problemas fundamentais em engenharia e em muitas das suas aplicações.

4 utad
Pedro Melo-Pinto 2022

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

um algoritmo de

PESQUISA

é um algoritmo que procura um elemento (conhecido à partida) numa lista.

exemplo: pesquisar o elemento 1 na lista {-2,5,6,1,8,7,2,7}

-2	5	6	1	8	7	2	7
0	1	2	3	4	5	6	7

resultado: 3 (posição do elemento 1 na lista)

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

5
utad
Pedro Melo-Pinto 2022

RELEMBRAR

01 PESQUISA SEQUENCIAL

força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

exemplo: **pesquisar** o elemento 1 na lista {-2,5,6,1,8,7,2,7}

-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7

0 1 2 3 4 5 6 7

resultado: 3 (posição do elemento 1 na lista)

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

6
utad
Pedro Melo-Pinto 2022

RELEMBRAR

01 PESQUISA SEQUENCIAL

força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

Algoritmo

```
function linear_search
    pass in LISTA INT L, INT x
    var: INT n, i
    n ← call: length(L)
    repeat for j = 1 to n
        if x = L[j]
            pass out: i
        end if
    end repeat
    pass out: -1
end function
```

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

RELEMBRAR
01 PESQUISA SEQUENCIAL
força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

lista {-2,5,6,1,8,7,2,7}
elemento a pesquisar: 2

-2	5	6	1	8	7	2	7
0	1	2	3	4	5	6	7

comparações: 1
comparações: 1

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

RELEMBRAR
01 PESQUISA SEQUENCIAL
força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

lista {-2,5,6,1,8,7,2,7}
elemento a pesquisar: -2

-2	5	6	1	8	7	2	7
0	1	2	3	4	5	6	7

comparações: 1

caso-melhor
quando o elemento procurado está na primeira posição, o número de comparações é: 1

O(1)

Algoritma (e Estruturas de Dados)
ALGORITMOS: PESQUISA

9
utad
Pedro Melo-Pinto 2022

RELEMBRAR

01 PESQUISA SEQUENCIAL

força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

lista {-2,5,6,1,8,7,2,7}
elemento a pesquisar: 3

-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7

comparações: 1
comparações: 1

caso-pior
quando o elemento procurado não se encontra na lista, o número de comparações é: $n + 1$

$O(n)$

Algoritma (e Estruturas de Dados)
ALGORITMOS: PESQUISA

10
utad
Pedro Melo-Pinto 2022

RELEMBRAR

01 PESQUISA SEQUENCIAL

força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

lista {-2,5,6,1,8,7,2,7}
elemento a pesquisar: 7

-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7
-2	5	6	1	8	7	2	7

comparações: 1
comparações: 1

caso-médio
em média, o número de comparações é: $\frac{1}{2}n$

$O(n)$

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

11 utad
Pedro Melo-Pinto 2022

RELEMBRAR

01 PESQUISA SEQUENCIAL

força bruta

Pesquisa numa lista de n elementos (vamos assumir, sem perda de generalidade, que são do tipo inteiro).
Procura-se sequencialmente até encontrar o elemento em causa, ou até atingir o fim da lista (caso em que a pesquisa não é bem sucedida).

O resultado deve ser a posição onde se encontra o elemento que se procura, ou -1 se esse elemento não se encontrar na lista.

Pode melhorar-se o desempenho com recurso a listas auto-organizadas.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

12 utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

The idea of sorting a list of items to allow for faster searching dates back to antiquity. The earliest known example was the Inakibit-Anu tablet from Babylon dating back to c. 200 BC mentioned by John Mauchly, 1946 (for $n = 2^k - 1$). D.H. Lehmer, 1960 (for all n)

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

13

02 PESQUISA BINÁRIA
divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- 1 Começa por se considerar como intervalo toda a lista.
- 2 Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- 3 Caso contrário considera-se a parte superior do intervalo anterior.
- 4 Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

exemplo: pesquisar o elemento 11 na lista {1,3,4,6,7,9,10,11,13,17}

1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
0	1	2	3	4	5	6	7	8	9

resultado: 7 (posição do elemento 11 na lista)

Pedro Melo-Pinto 2022

utad

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

14

02 PESQUISA BINÁRIA
divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- 1 Começa por se considerar como intervalo toda a lista.
- 2 Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- 3 Caso contrário considera-se a parte superior do intervalo anterior.
- 4 Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Algoritmo (versão iterativa)

```

function binary_search
pass in: LISTA INT L, INT x
var: INT n, low, high, mid
n ← call: length(L)
low ← 0
high ← n-1
repeat while (low<= high)
    mid ← (low+high) / 2
    if L[mid]<x
        low← mid + 1
    else if L[mid]<x
        high ← mid - 1
    else
        pass out: mid
    end if
end repeat
pass out: -1
end function

```

Pedro Melo-Pinto 2022

utad

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

15

utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- 1 Começa por se considerar como intervalo toda a lista.
- 2 Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- 3 Caso contrário considera-se a parte superior do intervalo anterior.
- 4 Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Algoritmo (versão recursiva)

```
function binary_search
    pass in: LISTA INT L, INT x, low, high
    var: INT mid

    if high >= low
        mid ← (high+low) / 2
        if L[mid] = x
            pass out: mid
        else if L[mid] > x
            pass out : binary_search(L,x,low,mid-1)
        else
            pass out : binary_search(L,x, mid+1,high)
        end if
    end if
    else pass out: -1
    end if
end function
```

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

16

utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- 1 Começa por se considerar como intervalo toda a lista.
- 2 Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- 3 Caso contrário considera-se a parte superior do intervalo anterior.
- 4 Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Complexidade

pesquisar o elemento 11

1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
0	1	2	3	4	5	6	7	8	9

O número de vezes que o intervalo de pesquisa é dividido depende das situações

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

17
utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- Começa por se considerar como intervalo toda a lista.
- Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- Caso contrário considera-se a parte superior do intervalo anterior.
- Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Complexidade

pesquisar o elemento 7

1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
0	1	2	3	4	5	6	7	8	9

caso-melhor
quando o elemento procurado está no meio da lista, o número de comparações é: ~ 1

O(1)

resultado: 4

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

18
utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- Começa por se considerar como intervalo toda a lista.
- Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- Caso contrário considera-se a parte superior do intervalo anterior.
- Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Complexidade

pesquisar o elemento 2

1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17

caso-pior
quando o elemento procurado não está na lista, o número de comparações por intervalo analisado é: ~ 1
o número de vezes que o ciclo *while* é realizado é: $\log n$

O(log n)

resultado: -1

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

19

02 PESQUISA BINÁRIA
divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- Começa por se considerar como intervalo toda a lista.
- Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- Caso contrário considera-se a parte superior do intervalo anterior.
- Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

Complexidade

pesquisar o elemento 3

1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
1	3	4	6	7	9	10	11	13	17
0	1	2	3	4	5	6	7	8	9

caso-médio
o número de comparações por intervalo analisado é : ~ 1
em média, necessitamos de metade das divisões do caso-pior : $1/2 \log n$

$O(\log n)$

resultado: 1

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

20

02 PESQUISA BINÁRIA
divisão-e-conquista

Pesquisa numa lista ordenada dividindo repetidamente o intervalo de pesquisa (a meio).

- Começa por se considerar como intervalo toda a lista.
- Se o valor a pesquisar é menor do que o valor do meio do intervalo, repete-se o processo considerando como novo intervalo a parte inferior do intervalo anterior.
- Caso contrário considera-se a parte superior do intervalo anterior.
- Repetem-se os passos 2-3 até encontrar o valor ou ficar com um intervalo vazio, caso em que a pesquisa não é bem sucedida.

**E A ORDENAÇÃO,
NÃO SE CONTABILIZA NA COMPLEXIDADE ?**

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

21 utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

The diagram shows a binary search tree and a corresponding sorted array. The tree has root 21. Level 1: left child 14, right child 28. Level 2: left child 11, right child 18 under 14; left child 25, right child 32 under 28. Level 3: left child 5, right child 12 under 11; left child 15, right child 19 under 18; left child 23, right child 27 under 25; left child 30, right child 37 under 32. A pink circle highlights node 21. Below the tree is a sorted array [5, 11, 12, 14, 15, 18, 19, 21, 23, 25, 27, 28, 30, 32, 37]. A pink arrow points from the array to node 21. Text indicates "steps: 0".

Binary search tree
steps: 0

Sorted array
steps: 0

www.mathwarehouse.com

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

22 utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

Para lá desta operação, as árvores binárias de pesquisa suportam operações como Mínimo, Máximo, Sucessor e Antecessor.

Cada uma delas é de tipo $O(h)$, sendo h a altura da árvore binária.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
23 Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.

exemplo: lista {10,16,20,6,15,5,12}, elemento a procurar: $x = 3$
a BST pode ser construída de diversas formas, correspondendo a diferentes alturas h

The figure shows four binary search trees (BSTs) with the following node configurations:

- Tree 1 (Left):** Root 20, left child 16, right child 15. 16 has left child 12, right child 10. 10 has left child 6, right child 5.
- Tree 2 (Middle):** Root 12, left child 6, right child 16. 6 has left child 5, right child 10. 16 has left child 15, right child 20.
- Tree 3 (Right):** Root 15, left child 12, right child 16. 12 has left child 5, right child 10. 16 has left child 14, right child 20.
- Tree 4 (Far Right):** Root 5, left child 6, right child 10. 6 has left child 12, right child 15. 10 has left child 16, right child 20.

Below each tree is its height: $h: 6$, $h: 2$, $h: 4$, and $h: 6$ respectively.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
24 Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.

Caso pior
A árvore binária de pesquisa é construída de tal modo que só existe um dos ramos.
Neste caso o desempenho do método é $O(n)$, sendo n o número de elementos da BST

Caso mais favorável
A árvore binária de pesquisa é construída de tal modo que todos os níveis estão completamente preenchidos (a árvore é equilibrada).
Neste caso o desempenho do método é $O(\log n)$, sendo n o número de elementos da BST

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.

exemplo: lista {10,16,20,6,15,5,12}, elemento a procurar: $x = 3$

caso-pior
A árvore binária de pesquisa é construída de tal modo que só existe um dos ramos.

Neste caso o desempenho do método é $O(n)$
sendo n o número de elementos da BST

x : 3

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.

exemplo: lista {10,16,20,6,15,5,12}, elemento a procurar: $x = 3$

caso-melhor
A árvore binária de pesquisa é construída de tal modo que todos os níveis estão completamente preenchidos (a árvore é **equilibrada**).

Neste caso o desempenho do método é $O(\log n)$
sendo n o número de elementos da BST

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

27

utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.

exemplo: lista {10,16,20,6,15,5,12}, elemento a procurar: $x = 3$

caso-melhor:
A árvore binária de pesquisa é construída de tal modo que todos os níveis estão completamente preenchidos (a árvore é equilibrada).

Neste caso, o desempenho do método é $O(\log n)$, onde n é o número de elementos da BST.

Quando a árvore binária de pesquisa é equilibrada, todas as suas operações são $O(\log n)$

```
graph TD; 12((12)) --- 6((6)); 12 --- 16((16)); 6 --- 5((5)); 6 --- 10((10)); 16 --- 15((15)); 16 --- 20((20))
```

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

28

utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma árvore binária de pesquisa (BST – Binary Search Tree)

O desempenho do método depende da altura h da árvore binária: $O(h)$.
A altura da árvore depende do seu equilíbrio.

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

29

utad
Pedro Melo-Pinto 2022

02 PESQUISA BINÁRIA

divisão-e-conquista

Este método é normalmente efectuado recorrendo a uma BST.

O desempenho do método depende da altura h da árvore binária: $O(h)$.

A altura da árvore depende do seu equilíbrio.

Estratégias de equilíbrio de árvores binárias

1. Não equilibrar

Podemos acabar por ter uma árvore com uma profundidade elevada.
O caso pior acontece com alguma frequência, deteriorando fortemente o desempenho da BST.
2. Equilíbrio rigoroso

Caso ideal em que árvore deve permanecer sempre perfeitamente equilibrada.
No entanto é demasiado exigente na sua manutenção.
3. (Bom) equilíbrio

Admite-se que a árvore possa apresentar algum desequilíbrio, numa situação que não se afaste do caso óptimo.
Este relaxamento na exigência do equilíbrio pode significar um ganho apreciável na exigência em termos de manutenção, sem perda significativa no desempenho.



Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

31

utad
Pedro Melo-Pinto 2022

A pesquisa binária é normalmente efectuada recorrendo a uma BST.
O desempenho do método depende da altura h da árvore binária: $O(h)$.
A altura da árvore depende do seu equilíbrio.

Metodologias para o equilíbrio de árvores binárias

- A** Aleatoriedade
- B** Amortização
- C** Optimização

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

32

utad
Pedro Melo-Pinto 2022

A pesquisa binária é normalmente efectuada recorrendo a uma BST.
O desempenho do método depende da altura h da árvore binária: $O(h)$.
A altura da árvore depende do seu equilíbrio.

Metodologias para o equilíbrio de árvores binárias

- A** Aleatoriedade

Introduz-se aleatoriedade no algoritmo.
O objectivo é reduzir fortemente a probabilidade de ocorrência do caso-pior
(independentemente dos dados de entrada).

BSTs aleatórias

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

33

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

34

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

35

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

36

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

37

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```
graph TD; 2((2)) --- 1((1)); 2 --- 3((3))
```

38

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```
graph TD; 2((2)) --- 1((1)); 2 --- 4((4)); 4 --- 3((3)); 4 --- 5((5))
```

39

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```
graph TD
    7((7)) --- 1((1))
    7 --- 4((4))
    4 --- 3((3))
    4 --- 5((5))
```

40

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```
graph TD
    6((6)) --- 2((2))
    6 --- 4((4))
    2 --- 1((1))
    2 --- 3((3))
    4 --- 5((5))
```

41

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

42

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
end if
pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

43

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (randomized BSTs)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
    end if
    pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```

graph TD
    6((6)) --- 2((2))
    6 --- 9((9))
    2 --- 1((1))
    2 --- 4((4))
    4 --- 3((3))
    4 --- 5((5))
    6 --- 8((8))
    
```

44

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade - BSTs Aleatórias (randomized BSTs)
- ➋ Amortização
- ➌ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
    end if
    pass out: BT
end function
```

exemplo: construção da BST a partir da lista {1,2,3,4,5,6,7,8,9}

```

graph TD
    6((6)) --- 2((2))
    6 --- 9((9))
    2 --- 1((1))
    2 --- 4((4))
    4 --- 3((3))
    4 --- 5((5))
    6 --- 8((8))
    
```

45

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (randomized BSTs)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
    end if
    pass out: BT
end function
```

exemplo: inserção de elementos na BST → inserir x = 5

46

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (randomized BSTs)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $\frac{1}{n+1}$), inserir um elemento na raiz da BST.

Algoritmo

```
function insert
    pass in: BTREE INT BT, INT x
    var: INT n, r
    n ← call: length(BT)
    r ← call: random(0, n)
    if r = n
        return call: insertAtRoot(BT, x)
    else if x < dataKey(BT)
        return call: insert(Left(BT), x)
    else
        return call: insert(Right(BT), x)
    end if
    end if
    pass out: BT
end function
```

exemplo: inserção de elementos na BST → inserir x = 5

47

utad
Pedro Melo-Pinto 2022

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $1/(n+1)$), inserir um elemento na raiz da BST.

Análise do desempenho

Qual a altura da BST resultante?

Se construirmos a BST a partir de uma lista ordenada, devemos escolher o elemento do meio para ser a raiz (para equilibrar ao máximo a BST).

Desta forma, e como $n \leq 2^{h+1} - 1$, em que h é a altura da BST

$$h \leq \lfloor \log_2(n + 1) - 1 \rfloor \leq \lfloor \log_2 n \rfloor$$

A altura da BST neste caso é
 $O(\log n)$

48

utad
Pedro Melo-Pinto 2022

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- Ⓐ Aleatoriedade - BSTs Aleatórias (*randomized BSTs*)
- Ⓑ Amortização
- Ⓒ Optimização

A ideia é que a sua construção seja tal que, sem ser necessário que os elementos sejam introduzidos de forma aleatória, a topologia da árvore resultante seja equivalente.

Uma forma de o conseguir é, de modo aleatório (geralmente com uma probabilidade de $1/(n+1)$), inserir um elemento na raiz da BST.

Análise do desempenho

Qual a altura da BST resultante?

Se construirmos a BST a partir de uma lista, e os elementos forem introduzidos na BST de forma aleatória (ao escolhermos o elemento i a subárvore da esquerda terá $i - 1$ elementos e a subárvore da direita $n - i$) a altura da árvore resultante é:

$$h = 1 + \max(h_{left\ subtree}, h_{right\ subtree}) = 1 + \max(h_{i-1}, h_{n-i})$$

Se qualquer elemento tem a mesma probabilidade de ser escolhido, o valor final esperado será a média de todos os casos:

$$E[h_n] = \frac{1}{n} \sum_{i=1}^n [1 + \max(h_{i-1}, h_{n-i})]$$

Demonstra-se que
 $E[h_n] = O(\log n)$

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

49

A pesquisa binária é normalmente efectuada recorrendo a uma BST.
O desempenho do método depende da altura h da árvore binária: $O(h)$.
A altura da árvore depende do seu equilíbrio.

Metodologias para o equilíbrio de árvores binárias

B Amortização

Faz-se trabalho extra em determinada ocasião para evitar mais trabalho depois.
O objectivo é garantir um limite superior eficaz do custo médio por operação.

Splay trees

A ideia é utilizar as rotações utilizadas na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.
Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.
Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

50

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.
Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.
Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se Q é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

51
utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se Q é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zig – rotação à direita

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

52
utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se Q é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zag – rotação à esquerda

53

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se O é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zig zig - rotação dupla à direita

54

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se O é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zag zag - rotação dupla à esquerda

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

55

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se O é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zig zag – rotação à direita e à esquerda

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

56

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

A operação de *splay* depende de:

1. Se O é o filho direito ou esquerdo de P
2. Se P é a raiz. Se não
3. Se P é o filho esquerdo ou direito de G.

zag zig – rotação à esquerda e à direita

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

57

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- Aleatoriedade
- Amortização - *Splay trees*
- Optimização

```

function splay
    pass in: BTREE INT BT, INT x
    var: INT n, r
    repeat while x ≠ BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call: rightRotation(Parent(x))
            else
                call: leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call: rightRotation(Parent(Parent(x)))
            call: rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call: leftRotation(Parent(Parent(x)))
            call: leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call: leftRotation(Parent(x))
            call: rightRotation(Parent(x))
        else //Left(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call: rightRotation(Parent(x))
            call: leftRotation(Parent(x))
        end if
    end if
    end if
    end repeat
    pass out: BT
end function

```

Algoritmo: splay

A operação de *splay* corresponde a mover um elemento para a raiz.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

58

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- Aleatoriedade
- Amortização - *Splay trees*
- Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmo

```

function splay
    pass in: BTREE INT BT, INT x
    var: INT n, r
    repeat while x ≠ BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call: rightRotation(Parent(x))
            else
                call: leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call: rightRotation(Parent(Parent(x)))
            call: rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call: leftRotation(Parent(Parent(x)))
            call: leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call: leftRotation(Parent(x))
            call: rightRotation(Parent(x))
        else
            call: rightRotation(Parent(x))
            call: leftRotation(Parent(x))
        end if
    end if
    end if
    end repeat
    pass out: BT
end function

```

exemplo: splay do elemento x = 1

Algoritmia (e Estruturas de Dados)

ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

59

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmo

```

function splay
    pass in BTREE INT BT, INT x
    var INT n, r
    repeat while x < BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call:rightRotation(Parent(x))
            else
                call:leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:rightRotation(Parent(Parent(x)))
            call:leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(x))
            call:rightRotation(Parent(x))
        else
            call:rightRotation(Parent(x))
            call:leftRotation(Parent(x))
        end if
    end if
    end ip
    end repeat
    pass out BT
end function

```

exemplo: splay do elemento x = 1

Algoritmia (e Estruturas de Dados)

ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

60

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmo

```

function splay
    pass in BTREE INT BT, INT x
    var INT n, r
    repeat while x < BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call:rightRotation(Parent(x))
            else
                call:leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:rightRotation(Parent(Parent(x)))
            call:leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(x))
            call:rightRotation(Parent(x))
        else
            call:rightRotation(Parent(x))
            call:leftRotation(Parent(x))
        end if
    end if
    end ip
    end repeat
    pass out BT
end function

```

exemplo: splay do elemento x = 1

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

61

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmo

```

function splay
    pass in BTREE INT BT, INT x
    var INT n, r
    repeat while x < BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call:rightRotation(Parent(x))
            else
                call:leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:rightRotation(Parent(Parent(x)))
            call:leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else
            call:rightRotation(Parent(x))
            call:leftRotation(Parent(x))
        end if
    end if
    end ip
    end repeat
    pass out BT
end function

```

exemplo: splay do elemento x = 1

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

62

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Algoritmo

```

function splay
    pass in BTREE INT BT, INT x
    var INT n, r
    repeat while x < BT
        if Parent(Parent(x)) = NULL
            if Left(Parent(x)) = x
                call:rightRotation(Parent(x))
            else
                call:leftRotation(Parent(x))
            end if
        else if Left(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:rightRotation(Parent(Parent(x)))
            call:leftRotation(Parent(x))
        else if Right(Parent(x)) = x and Right(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else if Right(Parent(x)) = x and Left(Parent(Parent(x))) = Parent(x)
            call:leftRotation(Parent(Parent(x)))
            call:rightRotation(Parent(x))
        else
            call:rightRotation(Parent(x))
            call:leftRotation(Parent(x))
        end if
    end if
    end ip
    end repeat
    pass out BT
end function

```

exemplo: splay do elemento x = 1

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

63

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Pesquisar um elemento
 a pesquisa de elementos é feita como numa BST normal;
 depois, ao nó encontrado aplica-se uma operação de *splay* (o nó vai ficar na raiz).

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

64

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Pesquisar um elemento
 a pesquisa de elementos é feita como numa BST normal;
 depois, ao nó encontrado aplica-se uma operação de *splay* (o nó vai ficar na raiz).

Algoritmo <pre> function search pass in: BTREE INT BT, INT key if BT = NULL or key = Data(BT) pass out: BT end if if key < Data(BT) pass out: call search(Left(BT), key) end if pass out: call search(Right(BT), key) end function </pre>	exemplo: pesquisa do elemento x = 4 pesquisada feita como numa BST normal
--	---

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

65

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Pesquisar um elemento
 a pesquisa de elementos é feita como numa BST normal;
 depois, ao nó encontrado aplica-se uma operação de *splay* (o nó vai ficar na raiz).

Algoritmo

```
function find
    pass in: BTREE INT BT, INT key
    var:   INT x
    x ← call: search(BT, key)
    call: splay(BT, x)
    pass out: BT
end function
```

exemplo: pesquisa do elemento x = 4

pesquisa feita como numa BST normal splaying do elemento encontrado

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

66

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Pesquisar um elemento
 a pesquisa de elementos é feita como numa BST normal;
 depois, ao nó encontrado aplica-se uma operação de *splay* (o nó vai ficar na raiz).

A operação de *splay* também se aplica quando o elemento pesquisado não foi encontrado.
 Neste caso, a operação de *splay* é aplicada ao último dos elementos acedidos.

67

utad
Pedro Melo-Pinto 2022

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: **Inserir um elemento**
 a inserção de elementos é feita como numa BST normal;
 depois, ao nó inserido aplica-se uma operação de *splay* (o nó vai ficar na raiz).

Existem outras variantes.

68

utad
Pedro Melo-Pinto 2022

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: **Inserir um elemento**
 a inserção de elementos é feita como numa BST normal;
 depois, ao nó inserido aplica-se uma operação de *splay* (o nó vai ficar na raiz).

Algoritmo

```

function insert
    pass in: BTREE INT BT, INT x
    var: BTREE INT temp
    temp ← BT
    repeat while temp != NULL
        if x < temp
            temp ← Left(temp)
        else
            temp ← Right(temp)
        end if
    end repeat
    temp ← x
    call splay(BT, x)
    pass out: BT
end function

```

69

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Inserir um elemento

a inserção de elementos é feita como numa BST normal;
depois, ao nó inserido aplica-se uma operação de *splay* (o nó vai ficar na raiz).

exemplo: inserir o elemento $x = 8$

The diagram shows four stages of a BST insertion:

- inserção feita como numa BST normal:** A BST with root 6. Inserting 8 makes 8 a leaf node under 7.
- splaying do elemento inserido:** After inserting 8, the tree is rotated so that 8 becomes the new root (the splay operation).

70

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Deste modo mantemos os elementos acedidos recentemente perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Construção da árvore

a construção da árvore é feita pela sucessão de inserções dos elementos.

71

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Eliminar um elemento

encontra-se o elemento e aplica-se-lhe a operação de *splay*;

elimina-se este nó, ficando a árvore dividida em duas subárvore;

encontra-se o máximo dos nós da subárvore esquerda (antecessor do nó a eliminar), aplica-se-lhe a operação de *splay* e reúne-se com a subárvore da direita ficando este nó como a raiz da árvore.

Existem outras variantes.

72

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ Amortização - *Splay trees*
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

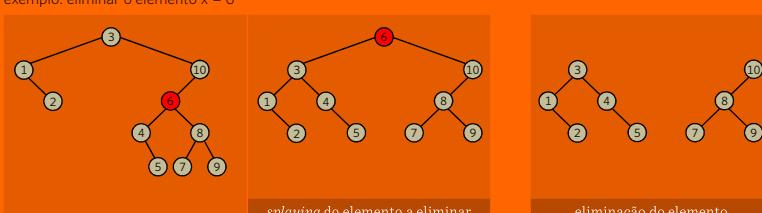
Operações: Eliminar um elemento

encontra-se o elemento e aplica-se-lhe a operação de *splay*;

elimina-se este nó, ficando a árvore dividida em duas subárvore;

encontra-se o máximo dos nós da subárvore esquerda (antecessor do nó a eliminar), aplica-se-lhe a operação de *splay* e reúne-se com a subárvore da direita ficando este nó como a raiz da árvore.

exemplo: eliminar o elemento x = 6



splaying do elemento a eliminar

eliminação do elemento

73

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Operações: Eliminar um elemento

encontra-se o elemento e aplica-se-lhe a operação de *splay*;
 elimina-se este nó, ficando a árvore dividida em duas subárvore;

encontra-se o máximo dos nós da subárvore esquerda (antecessor do nó a eliminar), aplica-se-lhe a operação de *splay* e reúne-se com a subárvore da direita ficando este nó como a raiz da árvore.

exemplo: eliminar o elemento $x = 6$

máximo dos elementos da subárvore esquerda *splaying* do máximo da esquerda reunião das duas subárvores

74

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Análise do desempenho

Análise geral

Os elementos acedidos mais frequentemente ficam próximos da raiz.

A árvore reorganiza-se após cada operação.

Um elemento move-se para a raiz com uma operação de *splay* após ser acedido.

Caso-pior das operações de inserção, remoção e pesquisa é $O(n)$.

Complexidade algorítmica temporal amortizada destas operações é $O(\log n)$.

75

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Análise do desempenho

Amortização

A complexidade é analisada para uma sequência de operações e não para uma determinada operação.

A motivação para este tipo de análise reside no facto de ao considerarmos somente o caso-pior na nossa análise, podermos estar a ser pessimistas.

Operações individuais de grande custo mas de pouca frequência de ocorrência poderão não ter grande impacto no custo geral do conjunto de operações efectuadas, se existirem outras operações mais frequentes e de menor custo.

76

utad
Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Análise do desempenho: análise com amortização

Para a análise com amortização, define-se o seguinte, para cada elemento x :

- Um valor constante (peso) arbitrário no âmbito da análise a fazer
 $w(x) > 0$
- A soma destes pesos (size)
 $s(x) = \sum_{y \in \text{subárvore}(x)} w(y)$
 em que $\text{subárvore}(x)$ é a subárvore com raiz em x , incluindo x
- rank
 $r(x) = \log s(x)$
 utilizamos $r(x)$ como potencial de um nó (elemento)
- Função potencial depois de i operações
 $\phi(i) = \sum_{x \in \text{árvore}} r(x)$

77

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Análise do desempenho: análise com amortização

Lema 1: O custo amortizado de um passo da operação de *splay* no nó x é $\leq 3(r'(x) - r(x)) + 1$, em que r é o rank antes da operação de *splay* e r' é o rank depois da operação de *splay*

Além disso, o custo amortizado das rotações de *splay* rr, ll, lr e rl é $\leq 3(r'(x) - r(x))$.

Corolário: O custo amortizado da operação de *splay* no nó x é $O(\log n)$

nota : rr, ll, lr e rl são duplas rotações à direita-direita, esquerda-esquerda, esquerda-direita e direita-esquerda

78

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Pedro Melo-Pinto 2022

Árvores Equilibradas

- ➊ Aleatoriedade
- ➋ **Amortização - Splay trees**
- ➌ Optimização

A ideia é utilizar rotações na metodologia de inserção de elementos na raiz, de modo a garantir que estas rotações equilibrem, de algum modo, a BST.

Quando um nó é acedido, a operação de *splay* corresponde a movê-lo para a raiz.

Desse modo os elementos acedidos recentemente ficam perto da raiz (mais rapidamente acessíveis), enquanto a árvore fica aproximadamente equilibrada.

Pros

- Codificação mais simples do que árvores AVL (que vamos ver em seguida).
- Custo amortizado médio é aproximadamente igual ao de árvores (perfeitamente) equilibradas.
- Não necessita de manter informação adicional (como outras árvores equilibradas).
- As rotações aplicadas em caminhos de acesso longos tendem a reduzir o custo das futuras pesquisas.
- A maioria dos acessos ocorrem perto da raiz, minimizando o número de rotações envolvidas.

Cons

- A profundidade da árvore pode degradar-se até ser linear.
- A árvore muda quando ocorre uma pesquisa, o que pode ser problemático em sistemas concorrentes em que múltiplos processos tentem aceder à mesma árvore.

79

Algoritmia (e Estruturas de Dados)

ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Pedro Melo-Pinto 2022

A pesquisa binária é normalmente efectuada recorrendo a uma BST.

O desempenho do método depende da altura h da árvore binária: $O(h)$.

A altura da árvore depende do seu equilíbrio.

Metodologias para o equilíbrio de árvores binárias

c Optimização

Tem-se o trabalho sempre para garantir eficácia em todas as operações.

Os algoritmos exigem que se acrescente informação estrutural às BSTs.

Árvores AVL

Cada nó tem informação extra que representa o equilíbrio da árvore de que esse nó é a raiz.

Cada nó pode ainda ter uma *flag* para a implementação da “*lazy deletion*”.

O algoritmo de pesquisa é o mesmo das BST normais
(logo não há perda de eficácia nesta operação devido à informação extra).

80

Algoritmia (e Estruturas de Dados)

ED02

ÁRVORES AVL

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

81 utad Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvore da esquerda e da direita difere, no máximo, em **1**

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

82 utad Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvore da esquerda e da direita difere, no máximo, em **1**

```

graph TD
    Root((depth 0 height 3)) --> Node1((depth 1 height 1))
    Root --> Node2((depth 1 height 2))
    Node1 --> Leaf1((depth 2 height 0))
    Node1 --> Leaf2((depth 2 height 0))
    Node2 --> Leaf3((depth 2 height 1))
    Node2 --> Leaf4((depth 3 height 0))

    %% Legend
    %% root node (dark grey)
    %% inner node (light grey)
    %% leaf node (white)
    %% depth (text above node)
    %% height (text below node)
  
```

A **profundidade de um nó** é o número de ligações da raiz ao nó.
A **altura de um nó** é o número de ligações no caminho mais longo desde esse nó até uma folha.
A **profundidade de uma árvore** é a profundidade da folha mais funda da árvore.
A **altura de uma árvore** é a altura da raiz.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

83

utad
Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvores da esquerda e da direita difere, no máximo, em 1

Ajusta no acesso
Auto-ajustável

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

84

utad
Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvores da esquerda e da direita difere, no máximo, em 1

Uma árvore AVL é uma BST com uma condição de equilíbrio.
A designação AVL vem dos seus inventores: Georgy Adelson-Velskii and Evgenii Landis.
Uma árvore AVL aproxima uma árvore perfeita (árvore perfeitamente equilibrada).
Uma árvore AVL mantém a sua altura perto do mínimo possível.

É mantida informação da altura em cada nó $\text{balanceFactor} = \text{height}(\text{rightSubtree}(X)) - \text{height}(\text{leftSubtree}(X))$

O factor de equilíbrio de um nó é a diferença entre a altura da sua subárvore direita e a altura da sua subárvore esquerda. Um nó com um factor de equilíbrio de 1, 0, or -1 considera-se equilibrado

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

85 utad Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvore da esquerda e da direita difere, no máximo, em 1

Exemplo:

The diagram shows two binary search trees side-by-side. The left tree, labeled "SIM", has a root node 13 with children 8 and 17. Node 8 has children 4 and 11, with 4 having children 2 and 6. The height of this tree is labeled $h=4$. The right tree, labeled "NÃO", has the same structure but node 4 has a right child 1 instead of 6. This creates an unbalanced structure where the left subtree of the root has height 3 and the right subtree has height 1, so the height of the entire tree is labeled $h=2$. A vertical arrow on the right indicates the height of the tree.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

86 utad Pedro Melo-Pinto 2022

Definição:

Uma árvore AVL é uma árvore binária de pesquisa tal que, para qualquer nó da árvore, a altura das suas subárvore da esquerda e da direita difere, no máximo, em 1

Exemplo:

The diagram shows two binary search trees side-by-side. The left tree, labeled "SIM", has a root node 13 with children 8 and 17. Node 8 has children 4 and 11, with 4 having children 2 and 6. The height of this tree is labeled $h=4$. The right tree, labeled "NÃO", has the same structure but node 4 has a right child 1 instead of 6. This creates an unbalanced structure where the left subtree of the root has height 3 and the right subtree has height 1, so the height of the entire tree is labeled $h=1$. A vertical arrow on the right indicates the height of the tree.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

87

utad
Pedro Melo-Pinto 2022

Propriedades:

A profundidade de uma árvore AVL típica anda muito perto do valor óptimo $\log n$. Consequentemente, as operações de pesquisa numa árvore AVL tem limites superiores do caso-pior logarítmicos.

Uma actualização (entrada ou saída) numa árvore AVL pode destruir o seu equilíbrio, e deve ser reequilibrada antes da operação poder ser considerada completa.

Depois de uma inserção, só os nós no caminho do ponto de inserção até à raiz poderão ter tido o seu factor de equilíbrio alterado.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

88

utad
Pedro Melo-Pinto 2022

Propriedade (equilíbrio):

A altura de uma árvore AVL com n nós é $O(\log n)$.

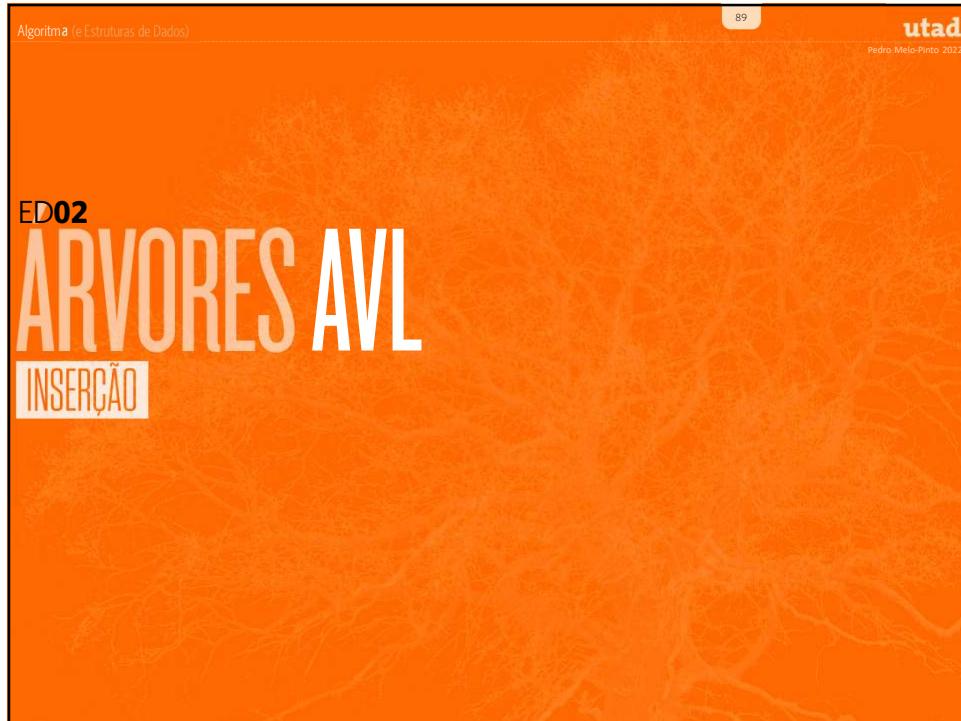
DEMONSTRAÇÃO

Seja $N(h)$ o número mínimo de nós de uma árvore AVL com altura h .
Seja r a raiz desta árvore.

Assuma-se, sem perda de generalidade, que a subárvore da esquerda é maior que a subárvore da direita.
Podemos representar $N(h)$ a partir de :

$N(h-1)$ número mínimo de nós na subárvore da esquerda de r
 $N(h-2)$ número mínimo de nós na subárvore da direita de r

$N(h) = 1 + N(h-1) + N(h-2)$
 $N(h) > 2^{h/2}$
 $h < \log N(h)$ A altura de uma árvore AVL é $O(\log n)$

A presentation slide with an orange background. The title 'Inserção' is at the top. Below it, several text boxes contain the following information:

- Inserir um nó numa árvore AVL T envolve a mudança da altura de alguns dos seus nós
- Se a inserção provocar o desequilíbrio de T, vamos subindo na árvore desde o nó inserido até encontrar o primeiro nó cujo avô seja um nó desequilibrado
- Precisamos de reequilibrar a árvore T
- Depois de cada inserção, são necessárias no máximo duas rotações para restabelecer o equilíbrio da árvore

The slide has a header 'Algoritmia (e Estruturas de Dados)' and a footer 'utad Pedro Melo-Pinto 2022'. A small number '90' is in the top right corner.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

91

utad
Pedro Melo-Pinto 2022

Inserção

Inserir um nó numa árvore AVL T envolve a mudança da altura de alguns dos seus nós

antes da inserção: casos possíveis (nó de altura $k+1$; subárvores do nó de altura k ou $k-1$)
não esquecer que a árvore tem de respeitar a propriedade AVL

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

92

utad
Pedro Melo-Pinto 2022

Inserção

Inserir um nó numa árvore AVL T envolve a mudança da altura de alguns dos seus nós

antes da inserção: casos possíveis (nó de altura $k+1$; subárvores do nó de altura k ou $k-1$)
a árvore tem de respeitar a propriedade AVL

violação da propriedade AVL

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

93

utad
Pedro Melo-Pinto 2022

Inserção

Inserir um nó numa árvore AVL T envolve a mudança da altura de alguns dos seus nós

Se a inserção provocar o desequilíbrio de T, vamos subindo na árvore desde o nó inserido até encontrar o primeiro nó cujo avô seja um nó desequilibrado

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

94

utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

O equilíbrio é recuperado através de rotações na árvore

Designemos o nó a ser reequilibrado por X
Existem 4 possíveis casos a necessitar de atenção
(dois são simétricos dos outros dois):

- ① Uma inserção na subárvore esquerda do filho esquerdo de X,
- ② Uma inserção na subárvore direita do filho esquerdo de X,
- ③ Uma inserção na subárvore esquerda do filho direito de X, ou
- ④ Uma inserção na subárvore direita do filho direito de X.

(inserção externa) casos 1 e 4 são simétricos
necessitam da mesma operação para reequilibrar: *rotação simples*.
(inserção interna) casos 2 e 3 são simétricos
necessitam da mesma operação para reequilibrar: *rotação dupla*.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

95
utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO SIMPLES

Uma rotação simples troca os papéis de pai e filho enquanto mantém a ordem de percurso (*inorder*)

A rotação ocorre entre um nó e o seu pai

O filho torna-se o pai

O pai fica o filho direito na rotação à direita, e o filho esquerdo na rotação à esquerda

rotação simples à direita

rotação simples à esquerda

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

96
utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO SIMPLES

Uma rotação simples troca os papéis de pai e filho enquanto mantém a ordem de percurso (*inorder*)

A rotação ocorre entre um nó e o seu filho

O filho torna-se o pai

O pai fica o filho direito na rotação à direita, e o filho esquerdo na rotação à esquerda

APLICA-SE NOS CASOS

- ① Uma inserção na subárvore esquerda do filho esquerdo de X,
- ④ Uma inserção na subárvore direita do filho direito de X.

O resultado é uma BST que satisfaz a propriedade de uma árvore AVL

exemplo: novo elemento, 1

Uma rotação é suficiente para reparar os casos 1 e 4.
A rotação simples mantém a altura original (antes da inserção)

altura

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

97 utad Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO SIMPLES

Uma rotação simples troca os papéis de pai e filho enquanto mantém a ordem de percurso (*inorder*). A rotação ocorre entre um nó e o seu filho

- O filho torna-se o pai
- O pai fica o filho direito na rotação à direita, e o filho esquerdo na rotação à esquerda

APLICA-SE NOS CASOS

- ① Uma inserção na subárvore esquerda do filho esquerdo de X,
- ② Uma inserção na subárvore direita do filho direito de X.

O resultado é uma BST que satisfaz a propriedade de uma árvore AVL

exemplo: novo elemento, 1

Uma rotação é suficiente para reparar os casos 1 e 4.
A rotação simples mantém a altura original (antes da inserção)
Logo, é suficiente percorrer o caminho do nó inserido para a raiz, e aplicar uma rotação simples no primeiro nó que esteja desequilibrado.

altura ↑

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

98 utad Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO SIMPLES

Uma rotação simples troca os papéis de pai e filho enquanto mantém a ordem de percurso (*inorder*). A rotação ocorre entre um nó e o seu filho

- O filho torna-se o pai
- O pai fica o filho direito na rotação à direita, e o filho esquerdo na rotação à esquerda

APLICA-SE NOS CASOS

- ① Uma inserção na subárvore esquerda do filho esquerdo de X,
- ② Uma inserção na subárvore direita do filho direito de X.

O resultado é uma BST que satisfaz a propriedade de uma árvore AVL

exemplo: novo elemento, 1

NÃO → SIM

ANÁLISE

Uma rotação é suficiente

- ① inserir o elemento na BST – $O(\log n)$
- ② procurar o nó desequilibrado – $O(\log n)$
- ③ rodar à esquerda/direita – $O(1)$

ROTAÇÃO SIMPLES
inserção na AVL é $O(\log n)$

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

99 utad Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO DUPLA
Nalguns casos não é suficiente uma rotação simples para reequilibrar

100 utad Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO DUPLA
Uma rotação dupla é equivalente a uma sequência de duas rotações simples

APLICA-SE NOS CASOS

- ② Uma inserção na subárvore direita do filho esquerdo de X,
- ③ Uma inserção na subárvore esquerda do filho direito de X.

O resultado é uma BST que satisfaz a propriedade de uma árvore AVL

exemplo: novo elemento, 7

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

101
utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO DUPLA

01. Rotação dupla esquerda-direita

Uma rotação dupla esquerda-direita é equivalente a uma sequência de duas rotações simples:
 uma rotação na árvore original: rotação à esquerda entre o filho e neto esquerdos de X
 seguida de uma rotação na árvore resultante: rotação à direita entre X e o seu novo filho esquerdo
 (em que X é o primeiro nó desequilibrado, no caminho do nó inserido para a raiz)

exemplo: novo elemento, 7

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

102
utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO DUPLA

02. Rotação dupla direita-esquerda

Uma rotação dupla direita-esquerda é equivalente a uma sequência de duas rotações simples:
 uma rotação na árvore original: rotação à direita entre o filho e neto direitos de X
 seguida de uma rotação na árvore resultante: rotação à esquerda entre X e o seu novo filho direito
 (em que X é o primeiro nó desequilibrado, no caminho do nó inserido para a raiz)

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

103
utad
Pedro Melo-Pinto 2022

Inserção | Reequilibrar

ROTAÇÃO DUPLA

01 Rotação dupla esquerda-direita
Uma rotação dupla esquerda-direita é equivalente a uma sequência de duas rotações simples: uma rotação na árvore original: rotação à esquerda entre o filho e neto esquerdos de X, seguida de uma rotação na árvore resultante: rotação à direita entre X e o seu novo filho esquerdo.

02 Rotação dupla direita-esquerda
Uma rotação dupla direita-esquerda é equivalente a uma sequência de duas rotações simples: uma rotação na árvore original: rotação à esquerda entre o filho e neto esquerdos de X, seguida de uma rotação na árvore resultante: rotação à direita entre X e o seu novo filho esquerdo.

exemplo: novo elemento, 7

ANÁLISE

- ① inserir o elemento na BST – $O(\log n)$
- ② procurar violação do equilíbrio – $O(1)$
- ③ rodar à esquerda – $O(1)$
- ④ rodar à direita – $O(1)$

ROTAÇÃO DUPLA
inserção na AVL é $O(\log n)$

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

104
utad
Pedro Melo-Pinto 2022

Inserção | Algoritmo & complexidade

Insert the new node on the BST

Check each of the node's ancestors for consistency with the AVL rules

For each node checked,

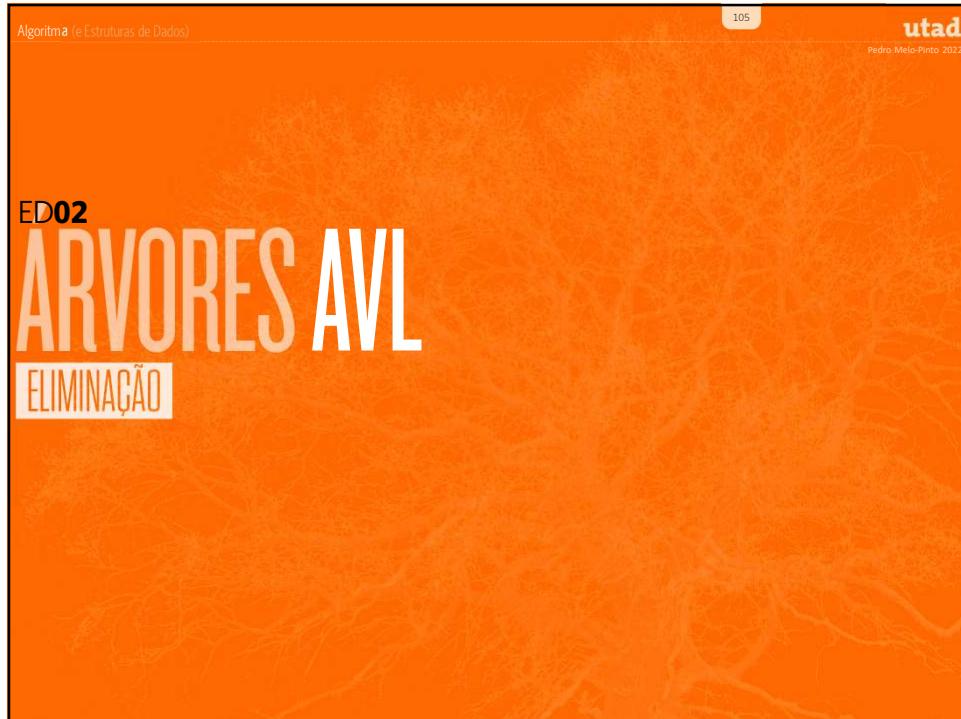
- If the balance factor remains 1, 0, or -1
then no rotations are necessary
- Else
it's unbalanced

After each insertion, at most two tree rotations are needed to restore the entire tree

Qual é a complexidade do caso-pior da operação de inserção?

Inserir o nó : percorrer desde a raiz até onde ficará a nova folha e acrescentá-la - $O(\log n)$.
 Verificar desequilíbrio : percorrer o caminho desde o nó inserido até à raiz, verificando as violações da AVL, reparando-as com rotações;
 as operações feitas ao longo do caminho têm complexidade constante - $O(1)$,
 o tempo total é proporcional ao comprimento do caminho, que é $O(\log n)$.
 (assumindo que a determinação da altura de um nó é feita em tempo $O(1)$ – p.e. tendo um campo altura em cada nó)

A única altura em que a altura de um nó muda, é quando uma folha é acrescentada por baixo dele, ou quando ele está envolvido numa rotação. Assim, todas as mudanças de altura encontram-se no caminho da raiz até à folha inserida.

A presentation slide with an orange background. At the top, there is navigation text: 'Algoritma (e Estruturas de Dados)', 'ESTRUTURAS DE DADOS', 'AVRORES BINARIAS EQUILIBRIO', '106', 'utad', and 'Pedro Melo-Pinto 2022'. The main title 'Eliminação' is centered at the top. Below it, a text box contains the following content:

A eliminação de nós é mais complicada

Poderemos necessitar de mais do que um reequilíbrio, no caminho do nó eliminado até à raiz

De recordar que cada nó da árvore AVL tem associado um factor de equilíbrio menor, igual ou maior do que zero, consoante a sua subárvore esquerda tenha altura maior, igual ou menor do que a sua subárvore direita

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

107

utad
Pedro Melo-Pinto 2022

Eliminação

A eliminação de nós é mais complicada

Poderemos necessitar de mais do que um reequilíbrio, no caminho do nó eliminado até à raiz

A **eliminação lazy** é melhor se as eliminações são pouco frequentes:
Não elimina verdadeiramente os nós
Marca-os como eliminados

A ideia é não ter de implementar todo o processo de eliminação.
Para cada nó é mantido um valor que indica se este está *activo* ou *não*.
Quando um nó é eliminado declaramo-lo como não activo, permanecendo na árvore. Se existir uma inserção posterior desse nó, regressa à condição de activo.
Após um conjunto (alargado) de eliminações/inserções deve ser executada uma operação de *garbage collection*.

Torna-se necessária a existência de uma *flag* extra (*dead/alive*) em cada nó.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

108

utad
Pedro Melo-Pinto 2022

Eliminação

A eliminação de nós é mais complicada

Poderemos necessitar de mais do que um reequilíbrio, no caminho do nó eliminado até à raiz

A **eliminação lazy** é boa?
Se o número de nós activos for semelhante ao dos nós não activos, a profundidade da AVL é ligeiramente (uma constante pequena) maior (em média).

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

109

utad
Pedro Melo-Pinto 2022

Eliminação

Se o nó é uma folha, retira-se

Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)

Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó

O nó que substitui o eliminado tem, no máximo, uma subárvore

Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

Após cada eliminação, pode ser necessária uma rotação em cada nó deste caminho

eliminar o nó x como numa BST normal

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

110

utad
Pedro Melo-Pinto 2022

Eliminação

A eliminação de nós é mais complicada

Poderemos necessitar de mais do que um reequilíbrio, no caminho do nó eliminado até à raiz

lembrar que a eliminação “lazy” é muito mais simples e frequentemente suficiente na prática

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

111 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

A eliminação de nós é mais complicada

A eliminação de um nó X de uma AVL requere as mesmas ideias básicas, incluindo as rotações simples e duplas, utilizadas na inserção

- ① Uma eliminação envolvendo a subárvore esquerda do filho esquerdo de X,
- ② Uma eliminação envolvendo a subárvore direita do filho esquerdo de X,
- ③ Uma eliminação envolvendo a subárvore esquerda do filho direito de X, ou
- ④ Uma eliminação envolvendo a subárvore direita do filho direito de X.

casos 1 e 4 são simétricos
necessitam da mesma operação para reequilibrar: *rotação simples*.
casos 2 e 3 são simétricos
necessitam da mesma operação para reequilibrar: *rotação dupla*.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

112 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 01: SEM ROTAÇÃO

Depois da eliminação do nó, a árvore permanece equilibrada ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita no máximo em 1

exemplo: eliminar 2

ELIMINAÇÃO

Se o nó é uma folha, retira-se
Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó

Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

altura

OK

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

113 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 02 : SEM ROTAÇÃO

Depois da eliminação do nó, a árvore permanece equilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita no máximo em 1

exemplo: eliminar 2

ELIMINAÇÃO

- Se o nó é uma folha, retira-se
- Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
- Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó

Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

altura

OK

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

114 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 03 : SEM ROTAÇÃO

Depois da eliminação do nó, a árvore permanece equilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita no máximo em 1

exemplo: eliminar 4

ELIMINAÇÃO

- Se o nó é uma folha, retira-se
- Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
- Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó

Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

altura

OK

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

115 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 04 : 1X ROTAÇÃO SIMPLES

Depois da eliminação do nó, a árvore fica desequilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita em mais de 1

ELIMINAÇÃO

- Se o nó é uma folha, retira-se
- Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
- Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó
- Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

exemplo: eliminar 15

rotação simples

Nó X : primeiro nó desequilibrado no caminho do pai do nó eliminado até à raiz.
Nó y : filho do nó X de maior altura
Nó z : filho do nó y de maior altura

altura

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

116 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 04 : 1X ROTAÇÃO SIMPLES

Depois da eliminação do nó, a árvore fica desequilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita em mais de 1

ELIMINAÇÃO

- Se o nó é uma folha, retira-se
- Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
- Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó
- Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

exemplo: eliminar 15

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

117 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 05 : ROTAÇÃO DUPLA

Depois da eliminação do nó, a árvore fica desequilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita em mais de 1

ELIMINAÇÃO

Se o nó é uma folha, retira-se
Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó
Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

exemplo: eliminar 4

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

118 utad Pedro Melo-Pinto 2022

Eliminação | Reequilibrar

CASO 05 : ROTAÇÃO DUPLA

Depois da eliminação do nó, a árvore fica desequilibrada
ou seja, para qualquer nó da árvore, a altura da subárvore esquerda difere da da direita em mais de 1

ELIMINAÇÃO

Se o nó é uma folha, retira-se
Se o nó tem um filho, liga-se o pai desse nó à subárvore (não vazia)
Se o nó tem dois filhos, substitui-se pelo maior elemento da sua subárvore esquerda, ou pelo elemento menor da sua subárvore direita, eliminando-se o nó
Após a eliminação, percorre-se o caminho entre o pai do nó eliminado e a raiz, ajustando os factores de equilíbrio se necessário

exemplo: eliminar 4

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

119

utad
Pedro Melo-Pinto 2022

Eliminação | Algoritmo & complexidade

Delete a node x as in an ordinary BST

Check each of the node's ancestors for consistency with the AVL rules

For each node checked,

- If** the balance factor remains 1, 0, or -1
then no rotations are necessary
- Else**
it's unbalanced - perform the appropriate rotation

After each deletion, at most $\log n$ rotations (1 per node checked) are needed to restore the entire tree

Qual é a complexidade do caso-pior da operação de eliminação?

Eliminar o nó: eliminar como numa BST normal ($O(h)$, em que h é o nível do nó – caso-pior : $O(\log n)$).
 Verificar desequilíbrios: no caminho até à raiz procurar violações da propriedade AVL, reparando-as com rotações. temos de acrescentar o custo de uma (possível) rotação em cada nó.
 o tempo total é, no máximo, $O(\log n)$.
 (assumindo que a determinação da altura de um nó é feita em tempo $O(1)$ – p.e. tendo um campo altura em cada nó)

Reequilibrar a AVL depois de uma eliminação pode requerer efectuar $O(\log n)$ rotações extra, mas como as rotações são operações $O(1)$, não afectam o tempo global $O(\log n)$ da eliminação.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

120

utad
Pedro Melo-Pinto 2022

Inserção & Eliminação | Análise

Uma eliminação pode precisar $1.44 \log(n+2)$ reequilíbrios no caso-pior, enquanto uma inserção necessita no máximo de um reequilíbrio.

O caso médio leva $\log(n) + 0.25$ pesquisas, o que reduz o número de rotações, no caso das eliminações, para $\log(n) + 0.25$.

Testes indicam que 78% das eliminações não necessitam de reequilibrar, enquanto só 25% das inserções não provocam desequilíbrio.

As eliminações mais demoradas ocorrem menos frequentemente.

A eliminação lazy é melhor se as eliminações forem pouco frequentes.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

121

utad
Pedro Melo-Pinto 2022

Pesquisa

A pesquisa numa árvore AVL funciona da mesma maneira do que numa BST não equilibrada.

Tendo em atenção o equilíbrio da AVL, a operação de pesquisa tem complexidade temporal de $O(\log n)$.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

122

utad
Pedro Melo-Pinto 2022

Pros e Cons

Argumentos favoráveis às árvores AVL:

1. Todas as operações apresentam um caso-pior logarítmico , uma vez que as árvores envolvidas são sempre equilibradas.
2. Este equilíbrio não acrescenta mais do que um factor constante ao custo das operações de inserção e eliminação.

Argumentos desfavoráveis às árvores AVL:

1. Difíceis de programar e depurar
2. Necessitam de espaço extra para a altura (e para a flag no caso da eliminação *lazy*)
3. São assintoticamente rápidas, mas o reequilíbrio leva um pouco de tempo
4. A maioria das pesquisas em grandes conjuntos é feita em sistema de tipo base de dados em disco, e utilizam outras estruturas (p.e. B-trees)
5. Se o tempo logarítmico amortizado for suficiente, utilize árvores *splay*

The slide has a light blue header with the title 'Metodologias para o equilíbrio de árvores binárias'. Below the title, there is a section titled 'c Optimização' with the following text:
A pesquisa binária é normalmente efectuada recorrendo a uma BST.
O desempenho do método depende da altura h da árvore binária: $O(h)$.
A altura da árvore depende do seu equilíbrio.

The slide has a light blue header with the title 'ED02 ARVORES RUBINEGRAS'. The text 'ED02' is in a smaller font above 'ARVORES RUBINEGRAS'.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

125 utad Pedro Melo-Pinto 2022

Definição:

Uma árvore RubiNegra é uma árvore binária de pesquisa que cumpre as seguintes regras:

01. Aplica-se a regra das BSTs e os nós têm côn. Rubi/Negra
02. A raiz é Negra.
03. Todos os nós externos (NULL) são Negros.
04. Se um nó é Rubi então ambos os seus filhos são Negros.
05. O número de nós Negros entre um nó e um qualquer nó externo (NULL) é o mesmo independentemente do caminho (descendente, se existir).

A representação (habitual) de uma árvore RubiNegra considera os filhos NULL como nós externos da árvore.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

126 utad Pedro Melo-Pinto 2022

Propriedade (mais importante):

Uma árvore RubiNegra com n nós internos tem uma altura máxima h_{\max} de $2 \log(n+1)$

01. Aplica-se a regra das BSTs e os nós têm côn. Rubi/Negra
02. A raiz é Negra.
03. Todos os nós externos (NULL) são Negros.
04. Se um nó é Rubi então ambos os seus filhos são Negros.
05. O número de nós Negros entre um nó e um qualquer nó externo (NULL) é o mesmo independentemente do caminho (descendente, se existir).

O número de nós de côn. Negra num caminho raiz-null é limitado pelo número de nós no caminho raiz-null mais curto.

Consideremos que o caminho raiz-null mais curto tem profundidade k :
 existem $k+1$ nós no caminho raiz-null mais curto
 existe uma subárvore binária perfeita de profundidade k

Suponhamos que a subárvore perfeita tem n nós.
 O número de nós no caminho raiz-null ($k+1$) é logarítmico em n :

$$\begin{aligned} n &= 2^{k+1} - 1 \\ n + 1 &= 2^{k+1} \\ \log_2(n + 1) &= \log_2(2^{k+1}) = k+1 \log_2 2 = k+1 \end{aligned}$$

atendendo a 05:
 o número máximo de nós Negros num caminho raiz-null é $\log_2(n+1)$

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

127 utad Pedro Melo-Pinto 2022

Propriedade (mais importante):

Uma árvore RubiNegra com n nós internos tem uma altura máxima h_{\max} de $2 \log(n+1)$

01. Aplica-se a regra das BSTs e os nós têm cor Rubi/Negra
02. A raiz é Negra.
03. Todos os nós externos (NULL) são Negros.
04. Se um nó é Rubi então ambos os seus filhos são Negros.
05. O número de nós Negros entre um nó e um qualquer nó externo (NULL) é o mesmo independentemente do caminho (descendente, se existir).

O número máximo de nós Negros num caminho raiz-null é $\log_2(n+1)$
O caminho raiz-null mais longo terá $\log_2(n+1)$ nós Negros.
E quantos nós Rubi poderá ter?

Atendendo a 04, um nó Rubi tem 2 filhos Negros, o que restringe o número de nós Rubi.
O número máximo de nós Rubi num caminho raiz-null fica assim limitado pelo número máximo de nós Negros - $\log_2(n+1)$ - no mesmo caminho.

Assim:

$$\begin{aligned} h_{\max} &= \max \text{ nós Negros} + \max \text{ nós Rubi} \\ &= \log_2(n+1) + \log_2(n+1) \\ &= 2 \times \log_2(n+1) \end{aligned}$$

A altura da árvore é $O(\log n)$



128 utad Pedro Melo-Pinto 2022

ED02

ÁRVORES RUBINEGRAS

INSCRIÇÃO

Inserção

OK 01. Regra das BSTs
02. Raiz Negra.
OK 03. Todos os nós externos (NULL) são Negros.
04. Nó Rubi → Filhos Negros.
05. Nº nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

Inserção

OK 01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Assegurar as propriedades de uma árvore RubiNegra.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

131

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Assegurar as propriedades de uma árvore RubiNegra.
Depende da cor dos vizinhos.

OK 01. Regra das BST's
02. Raiz Negra.
03. Todos os nós externos (NULL) são Negros.
04. Nó Rubi → Filhos Negros.
05. Nº nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

132

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Dependendo da cor dos vizinhos:
02.1 O novo nó N é a raiz da BST.
Neste caso é-lhe mudada a cor para Negro, de modo a satisfazer a propriedade 02.

OK 01. Regra das BST's
02. Raiz Negra.
03. Todos os nós externos (NULL) são Negros.
04. Nó Rubi → Filhos Negros.
05. Nº nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

133

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Depende da cor dos vizinhos:
02.2 O pai do nó N é Negro.
Não é necessária nenhuma outra operação.

OK 01. Regra das BST's
OK 02. Raiz Negra.
OK 03. Todos os nós externos (NULL) são Negros.
OK 04. Nô Rubi → Filhos Negros.
OK 05. N° nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

134

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Depende da cor dos vizinhos.
02.3 Tanto o pai P como o tio T são Rubi (existem seguramente um avô e um tio).
Ambos mudam para Negro e o avô A muda para Rubi.
Mas A pode ficar a violar as propriedade 04 ou 02. Para o ultrapassar aplica-se 2.1, 2.2 ou 2.3 a A.

OK 01. Regra das BST's
OK 02. Raiz Negra.
OK 03. Todos os nós externos (NULL) são Negros.
OK 04. Nô Rubi → Filhos Negros.
OK 05. N° nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

neste caso aplicou-se 2.3, seguida de 2.1 ao avô de A

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

135 utad Pedro Melo-Pinto 2022

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Depende da cor dos vizinhos.
02.4 O pai P é Rubi mas o tio T é Negro.
O nó N é o filho esquerdo de P, enquanto este é o filho esquerdo de A.
Neste caso aplica-se uma rotação para a direita em P, trocando em seguida as cores de P e A.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

136 utad Pedro Melo-Pinto 2022

Inserção

01. Inserir o nó como se fosse uma BST e atribuir-lhe cor Rubi.
Contudo, em vez de inserir um nó externo, devemos substituir o nó externo por este nó, adicionando-lhe dois nós externos Negros.

02. Depende da cor dos vizinhos.
02.5 O pai P é Rubi mas o tio T é Negro.
O nó N é o filho direito de P, enquanto P é o filho esquerdo de A.
Rotação para a esquerda de N, seguida da situação prevista em 2.4.

rotação à direita em N,
seguida de troca de cores entre N e 5

Algoritmia (e Estruturas de Dados)
ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

137

utad
Pedro Melo-Pinto 2022

Construção

A construção é feita pela inserção sucessiva de nós.

exemplo: construir a árvore a partir da lista {A, S, E, R, C, H, I, N, G, X}

OL. Regra das BST's
02. Raiz Negra.
03. Todos os nós externos (NULL) são Negros.
04. Nó Rubi → Filhos Negros.
05. N° nós Negros entre um qualquer nó e um nó externo (NULL) é sempre o mesmo.

Algoritmia (e Estruturas de Dados)

138

utad
Pedro Melo-Pinto 2022

ED02

ARVORES RUBINEGRAS

ELIMINAÇÃO

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

139 utad Pedro Melo-Pinto 2022

Eliminação

00. Nós NIL.

Cada nó sem 2 filhos tem 1|2 nós NIL. Estes são sempre de cor Negra. Não é necessário representá-los.

Na operação de eliminação de elementos têm um papel importante:
quando eliminamos um nó Negro que se torne um nó NIL, é marcado como Duplo-Negro (DB).

01. Regra das BST's
02. Raiz Negra.
03. Todos os nós externos (NIL) são Negros.
04. Nô Rubi → Filhos Negros.
05. N° nós Negros entre um qualquer nó e um nó externo (NIL) é sempre o mesmo.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

140 utad Pedro Melo-Pinto 2022

Eliminação

A operação não é simples:

00. Regras.

Case #	Check condition	Action
1	If node to be deleted is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

01. Regra das BST's
02. Raiz Negra.
03. Todos os nós externos (NIL) são Negros.
04. Nô Rubi → Filhos Negros.
05. N° nós Negros entre um qualquer nó e um nó externo (NIL) é sempre o mesmo.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

141 utad Pedro Melo-Pinto 2022

Eliminação

Começa-se sempre por
A. Eliminar o nó como se fosse uma BST.

Existem 3 casos a considerar:

1. Eliminar um nó sem filhos (folha)
Basta eliminar o nó da árvore.
2. Eliminar um nó com um filho
Substitui-se esse nó pelo seu filho (mantendo este as suas sub-árvore).
3. Eliminar um nó com dois filhos
Substitui-se pelo seu antecessor ou sucessor R (ao percorrer a árvore utilizando o método *in-order*), eliminando este da sua posição anterior.

Quando se faz a eliminação de um nó numa BST, acabamos sempre a eliminar um nó sem filhos ou com um filho só (no caso 3, a operação de eliminação do successor/antecessor é um caso 1 ou 2)

OL. Regra das BST's

O2. Raiz Negra.

O3. Todos os nós externos (NIL) são Negros.

O4. Nô Rubi → Filhos Negros.

O5. N° nós Negros entre um qualquer nó e um nó externo (NIL) é sempre o mesmo.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

142 utad Pedro Melo-Pinto 2022

Eliminação

Exemplo 01:
eliminar o nó 17 (Rubi).

A. Eliminar o nó como se fosse uma BST.
substitui-se pelo seu sucessor,
eliminando este da sua posição anterior

B. Ver caso da tabela.
caso 1 da tabela

Case #	Check condition	Action
1	If node to be deleted is a red leaf	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and DB and DB's sibling's children are black	(a) Remove the DB or null DB then delete the node and for other nodes remove the DB (b) Make DB's sibling red. (c) If DB's parent is black, make it DB, else swap color of DB's parent with DB's sibling
4	If DB's sibling is red	(a) Swap color of DB's parent with DB's sibling (b) Perform rotation at parent node in the direction of DB node (c) Check which case can be applied to this new situation
5	(a) DB's sibling is black (b) DB's sibling's child which is far from DB is red (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's next child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

143 utad Pedro Melo-Pinto 2022

Eliminação

Exemplo 02:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como duplo-negro

B. Ver caso da tabela.
caso 3 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction opposite of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black, (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction opposite of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

144 utad Pedro Melo-Pinto 2022

Eliminação

Exemplo 03:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como duplo-negro

B. Ver caso da tabela.
caso 3 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction opposite of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction opposite of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

145 utad Pedro Melo-Pinto 2022

Eliminação

Exemplo 03:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como duplo-negro

B. Ver caso da tabela.
caso 3 da tabela
caso 3 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction opposite of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black, (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

146 utad Pedro Melo-Pinto 2022

Eliminação

Exemplo 03:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como duplo-negro

B. Ver caso da tabela.
caso 3 da tabela
caso 3 da tabela
caso 2 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritmia (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

Eliminação

Exemplo 04:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como **duplo-negro**

B. Ver caso da tabela.
caso 4 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction opposite of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black, and (b) DB's sibling's child is red (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black

Algoritmia (e Estruturas de Dados)

ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

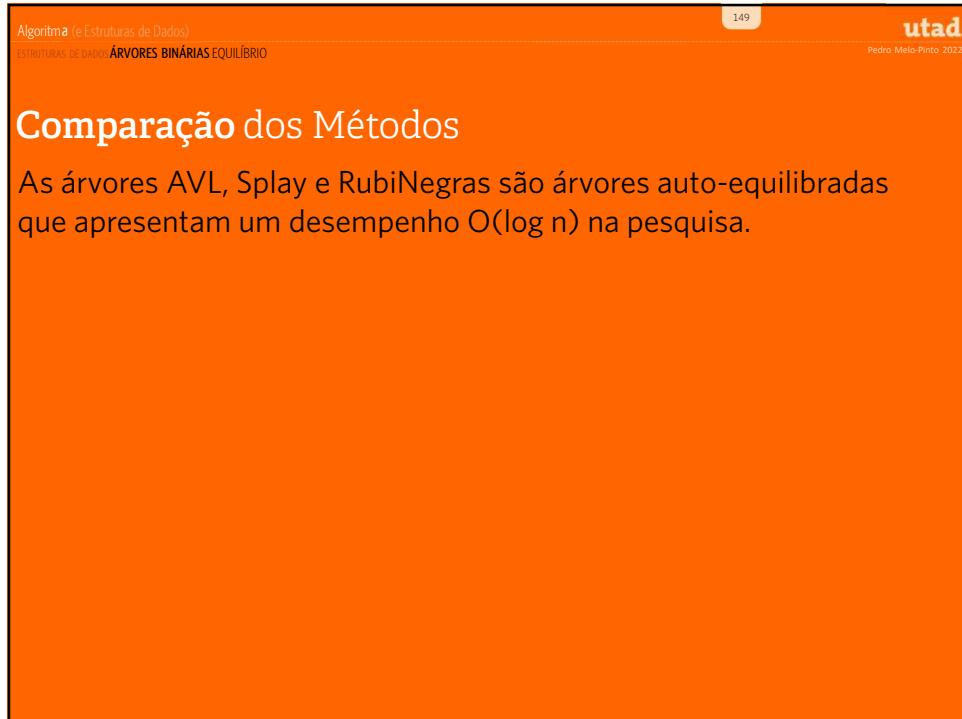
Eliminação

Exemplo 04:
eliminar o nó 15 (Negro).

A. Eliminar o nó como se fosse uma BST.
basta eliminá-lo (não tem filhos),
fica como **duplo-negro**

B. Ver caso da tabela.
caso 4 da tabela
caso 3 da tabela

Case #	Check condition	Action
1	If node to be delete is a red leaf node	Just remove it from the tree
2	If DB node is root	Remove the DB and root node becomes black.
3	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is Black, make it DB, else make it black
4	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in the direction opposite of DB node (c) Check which case can be applied to this new tree and perform that action
5	(a) DB's sibling is black, and (b) DB's sibling's child is red (c) DB's sibling's child which is far from DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	(a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB node (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black



149

utad

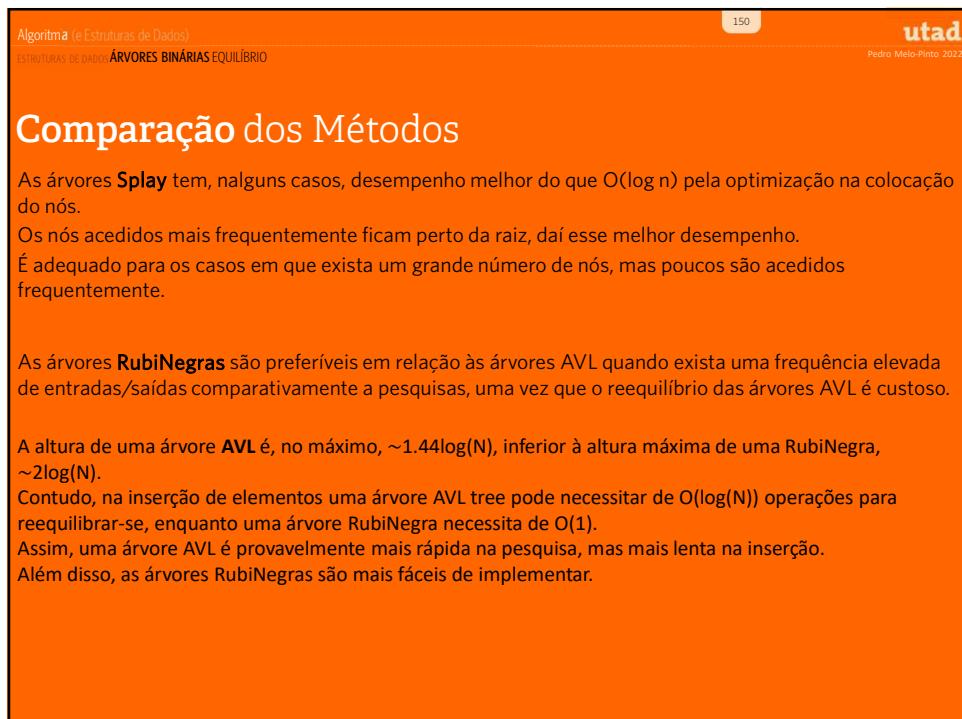
Pedro Melo-Pinto 2022

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Comparação dos Métodos

As árvores AVL, Splay e RubiNegras são árvores auto-equilibradas que apresentam um desempenho $O(\log n)$ na pesquisa.



150

utad

Pedro Melo-Pinto 2022

Algoritma (e Estruturas de Dados)

ESTRUTURAS DE DADOS: ÁRVORES BINÁRIAS EQUILÍBRIO

Comparação dos Métodos

As árvores **Splay** tem, na alguns casos, desempenho melhor do que $O(\log n)$ pela optimização na colocação do nós.

Os nós acedidos mais frequentemente ficam perto da raiz, daí esse melhor desempenho.

É adequado para os casos em que existe um grande número de nós, mas poucos são acedidos frequentemente.

As árvores **RubiNegras** são preferíveis em relação às árvores AVL quando exista uma frequência elevada de entradas/saídas comparativamente a pesquisas, uma vez que o reequilíbrio das árvores AVL é custoso.

A altura de uma árvore **AVL** é, no máximo, $\sim 1.44\log(N)$, inferior à altura máxima de uma RubiNegra, $\sim 2\log(N)$.

Contudo, na inserção de elementos uma árvore AVL pode necessitar de $O(\log(N))$ operações para reequilibrar-se, enquanto uma árvore RubiNegra necessita de $O(1)$.

Assim, uma árvore AVL é provavelmente mais rápida na pesquisa, mas mais lenta na inserção.

Além disso, as árvores RubiNegras são mais fáceis de implementar.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

151 utad Pedro Melo-Pinto 2022

B-Trees

São árvores equilibradas de pesquisa desenhadas para trabalhar em memórias secundárias de acesso directo.

São semelhantes às Red-Black BSTs com melhor desempenho nas operações de entrada/saída (I/O)

Cada nó pode ter mais do que dois filhos (podem atingir milhares). Este número depende das características do dispositivo de memória em causa.

Cada nó tem:

- o número de chaves armazenadas no nó ($n(x)$); estas chaves estão armazenadas ascendenteamente.
- $n(x) + 1$ apontadores para os seus filhos.

Todas as folhas tem a mesma profundidade.

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

152 utad Pedro Melo-Pinto 2022

Data Structure	Time Complexity								Space Complexity
	Average				Worst				
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Stack	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Queue	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Singly-Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Doubly-Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Skip List	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$
Hash Table	N/A	$O(1)$	$O(1)$	$O(1)$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Binary Search Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Cartesian Tree	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
B-Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
Red-Black Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
Splay Tree	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
AVL Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
KD Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Algoritma (e Estruturas de Dados)
ESTRUTURAS DE DADOS ÁRVORES BINÁRIAS EQUILÍBRIO

153 utad Pedro Melo-Pinto 2022

Custo das operações de construção da árvore e de pesquisa utilizando BSTs

N: número de elementos da lista

N	construction						search misses					
	B	T	T*	S	R	L	B	T	T*	S	R	L
1250	20	28	28	10	14	15	11	10	10	10	10	16
2500	10	36	40	24	25	21	15	12	11	12	11	19
5000	22	33	65	145	42	35	26	26	26	27	19	46
12500	90	128	197	267	92	145	75	74	86	80	60	145
25000	167	569	492	215	181	385	175	180	212	195	158	355
50000	381	648	1105	1125	430	892	420	415	505	433	359	878
100000	1004	2593	2656	1148	1190	3257	1047	1041	1357	1113	861	2094
200000	2628	6121	6341	2784	2936	7493	2553	2573	2893	2649	2114	5109

B : Standard BST
T : BST built by root insertion
T* : Randomized BST
S : Splay BST
R : Red–black BST
L : Skip list

Algoritma (e Estruturas de Dados)
Aprendemos PESQUISA

154 utad Pedro Melo-Pinto 2022

Custo das operações de pesquisa (utilizando ou não BSTs)

N: número de elementos da lista

implementation	worst-case cost (after N inserts)			average-case (after N random inserts)			ordered iteration?
	search	insert	delete	search hit	insert	delete	
sequential search (unordered list)	N	N	N	N/2	N	N/2	no
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes
red-black BST	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	$1.00 \lg N$	yes

Conseguimos melhorar?
Sim, mas com um tipo diferente de acesso aos dados.

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

155

utad
Pedro Melo-Pinto 2022

Leitura Adicional:

- ① Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009.
Introduction to Algorithms 3rd Edition. **CAPÍTULOS 11-13**
- ② Sedgewick R. and Wayne, K., 2011
Algorithms 4th Edition. **CAPÍTULO 3**
- ③ Adrego da Rocha A., 2014
Estruturas de Dados e Algoritmos em C 3^a Edição. **CAPÍTULOS 4, 5.1-5.3**
- ④ Adrego da Rocha A., 2014
Análise da Complexidade de Algoritmos. **SUBCAPÍTULOS 2.1-2.2**

