

Árvores

Estruturas de Dados e Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

- 1 Introdução
- 2 Árvores Binárias
- 3 Percurso em Árvores
- 4 Árvores Binárias de Pesquisa
- 5 Links



Sumário

1 Introdução



Árvores

- Árvores são EDs utilizadas para resolver muitos problemas.
- Podemos ter vários tipos diferentes de árvore.
- São de natureza recursiva e hierárquica.



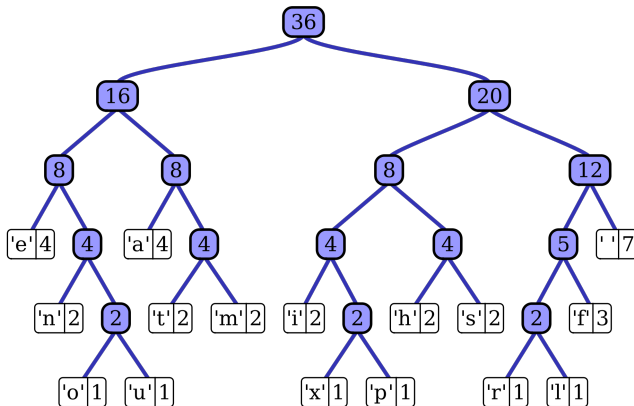
Árvores

Árvores de Huffman

- Utilizadas em compressão de dados.
- Organizam os símbolos mais frequentes próximo da raiz.
- Códigos menores para os símbolos mais frequentes.



Árvores de Huffman





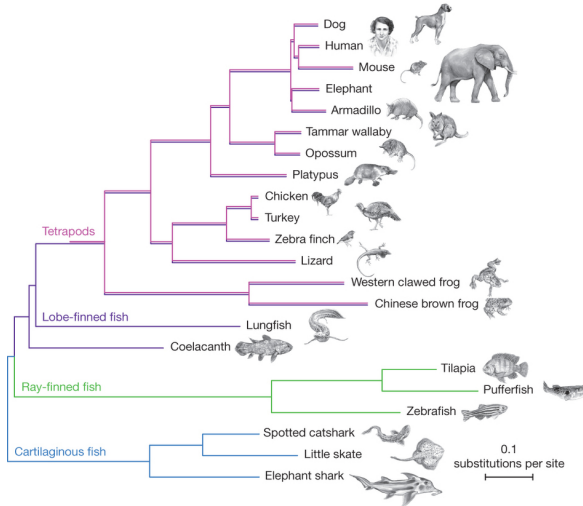
Árvores

Árvores Filogenéticas

- Utilizadas em Biologia Computacional.
- Estimam a distância evolutiva de organismos.



Árvores Filogenéticas





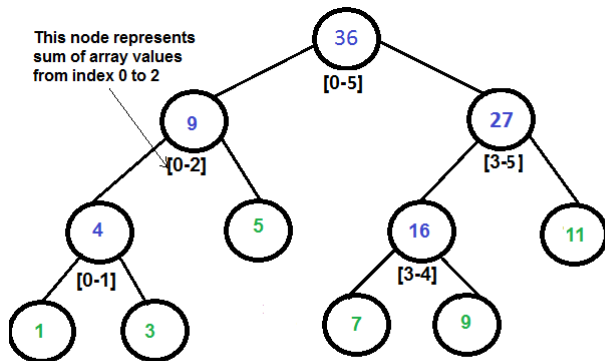
Árvores

Árvores de Segmentos

- Utilizadas em problemas diversos.
- Armazenam alguma propriedade sobre um dado intervalo $[l, r]$.



Árvores de Segmentos



Segment Tree for input array {1, 3, 5, 7, 9, 11}



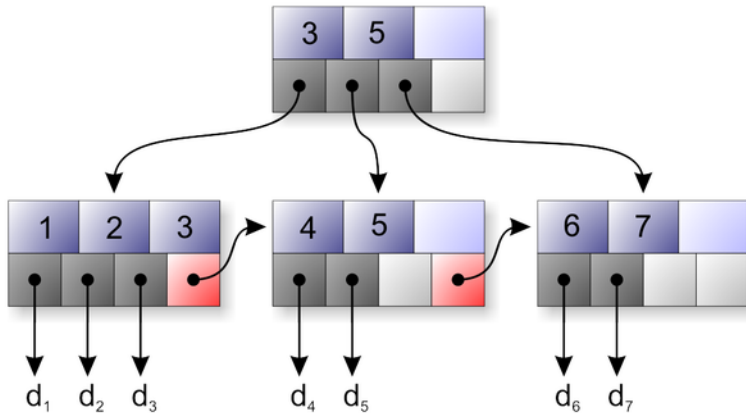
Árvores

Árvores B e B+

- Utilizadas em Sistemas de Arquivos.
- Convertem endereço de bloco de arquivo para endereço de disco.



Árvores B e B+





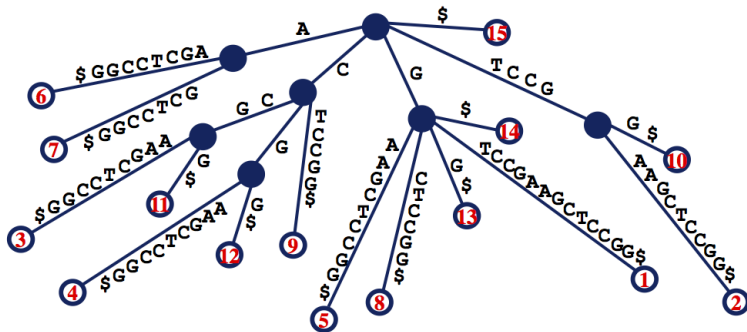
Árvores de Sufixos

Árvores de Sufixos

- Utilizadas em processamento de palavras.
- Codificam todos os sufixos de um texto de maneira compacta.
- Resolvem vários problemas sobre strings.



Árvores de Sufixos





Árvores

- Podemos citar várias outras:
 - ▶ K^2 -Tree: representação compacta de relações binárias.
 - ▶ Árvore-Rubro-Negra: árvore ordenada.
 - ▶ Fenwick Tree: calcula eficientemente soma de prefixos.
 - ▶ Tries: utilizadas em casamento de múltiplos padrões.
- Já deu para entender a infinidade de problemas que podemos resolver com árvores, certo?



Árvores

- No nosso curso, estudaremos uma das famílias mais básicas de árvores.
- As árvores binárias.
- Estamos trabalhando só com a ponta do *iceberg*.
- No entanto, servirá de alicerce ao estudar estruturas mais complexas posteriormente.



Sumário

2 Árvores Binárias



Sumário

2 Árvores Binárias

- Terminologia
- Estrutura



Terminologia

- Antes de elaborar qualquer algoritmo sobre árvore binárias, precisamos entender a sua representação.

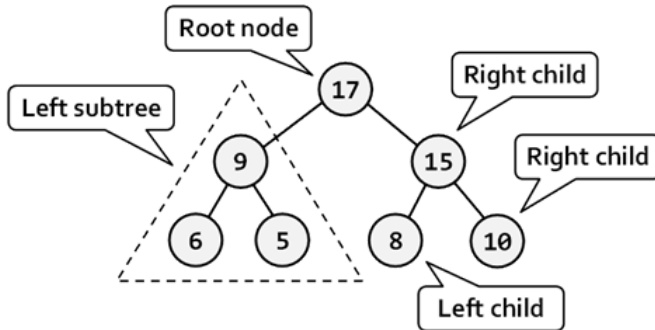


Terminologia

- **Raiz:** corresponde ao topo de uma árvore.
- **Pai:** nó que precede imediatamente um segundo em um caminho partindo da raiz.
- **Filho:** Nó que ocorre imediatamente após o outro em um caminho partindo da raiz.
- **Filho da esquerda:** se x é pai de y e y ocorre imediatamente após x ao seguir para esquerda, então y é o filho da esquerda de x .
- **Filho da direita:** se x é pai de y e y ocorre imediatamente após x ao seguir para direita, então y é o filho da direita de x .
- **Folha:** nó que não possui nenhum descendente.



Terminologia



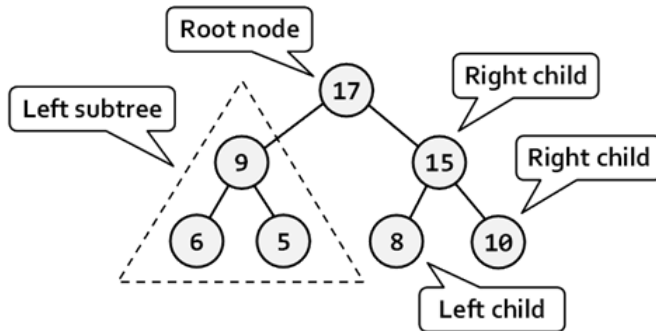


Terminologia

- **Altura:** a altura do nó x corresponde a maior distância de x a uma folha.
- **Grau:** quantidade de descendentes de um nó.
- **Nível:** conjunto de nós que estão na mesma altura em relação a raiz.



Terminologia





Árvore Binária

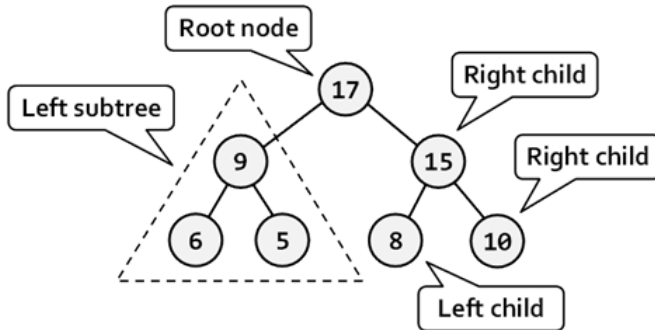
Definição (Árvore Binária)

Uma árvore binária é composta de:

- Um potencial nó denominado de raiz.
- Caso a raiz exista:
 - ▶ Uma subárvore da esquerda.
 - ▶ Uma subárvore da direita.



Terminologia





Árvores Binárias

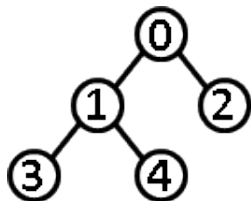
- Repare que a definição de uma árvore binária atua sobre ela própria.
- É uma definição recursiva!
- É natural que os algoritmos que atuem em árvores também sejam recursivos.



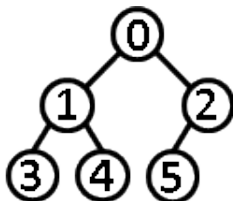
- Uma árvore binária pode ser:
 - ▶ Completa ou incompleta.
 - Uma árvore binária é completa quando todos os níveis dela estão preenchidos exceto pelo último, no qual os nós devem se encontrar mais a esquerda possível.
 - Pode ser representada através de um simples vetor.
 - ▶ Cheia ou não cheia.
 - Uma árvore binária é cheia se todo nó possui grau 0 ou 2.
 - ▶ Perfeita ou imperfeita.
 - Uma árvore binária é perfeita quando é cheia e todas as folhas possuem o mesmo nível.



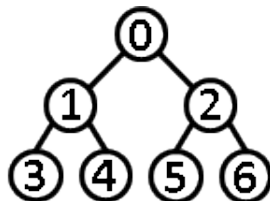
Terminologia



**full
binary tree**



**complete
binary tree**

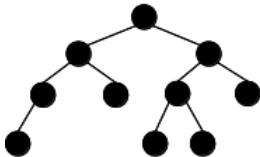


**perfect
binary tree**

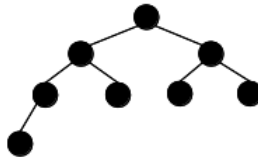


Terminologia

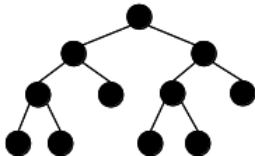
Neither complete nor full



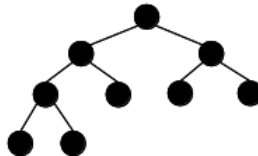
Complete but not full



Full but not complete



Complete and full





Sumário

2 Árvores Binárias

- Terminologia
- Estrutura



Estrutura

- Como visto, uma árvore binária possui, uma subárvore da esquerda e da direita.
- Podemos utilizar uma definição recursiva para representar essa estrutura computacionalmente.
- Como C não suporta tipos recursivos, emulamos essa característica através de ponteiros.



Estrutura

```
typedef struct tree_node{
    void* data; /* Dado da árvore */
    struct tree_node* left; /* Ponteiro para subárvore da esquerda */
    struct tree_node* right; /* Ponteiro para subárvore da direita */
}tree_node;

typedef struct arvore{
    tree_node* root; /* Raiz da Arvore */
}arvore;
```




Exemplos

Figura: É uma árvore binária?



Exemplos

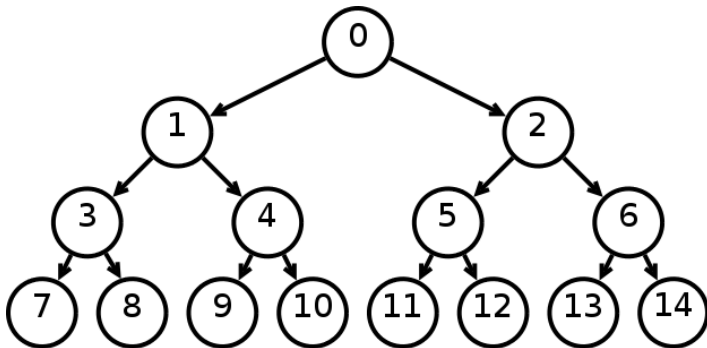


Figura: É uma árvore binária?



Exemplos

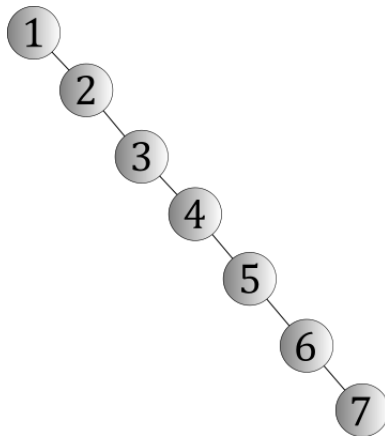


Figura: É uma árvore binária?



Exemplos

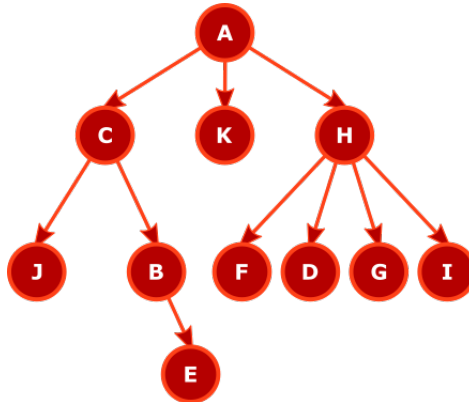


Figura: É uma árvore binária?



Árvores Binárias

- Árvores representam dados de maneira hierárquica.
- Se a árvore não tiver uma certa forma, sua utilidade pode ser questionada.



Exemplos

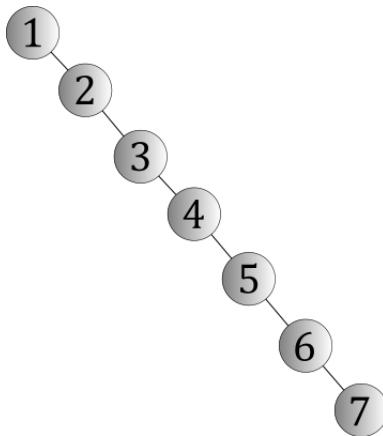


Figura: Árvore ou Lista?



Árvores Binárias

- Agora que conhecemos a terminologia e definições, podemos nos concentrar em aprender algo novo e útil.



Sumário

3 Percurso em Árvores



Percurso em Árvores

- Um dos problemas fundamentais sobre esta estrutura de dados é percorrê-la.
- Qual estratégia adotar?
- Busca em largura?
- Busca em profundidade?
 - ▶ Pré-ordem?
 - ▶ Em-ordem?
 - ▶ Pós ordem?



Percurso em Árvores

- Examinaremos agora cada uma destas abordagens.



Sumário

- 3 Percurso em Árvores
 - Busca em Largura
 - Busca em profundidade



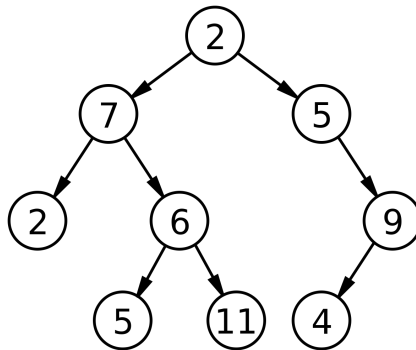
Percurso em Árvores

- Busca em largura ou amplitude.
 - ▶ Breath-first-search.
- Parte de um nó específico.
- Visita os vizinhos deste nó.
- Visita os vizinhos dos vizinhos do nó inicial.
- Visita os vizinhos dos vizinhos dos vizinhos do nó inicial.
- ...



Percurso em Árvores

Exemplo

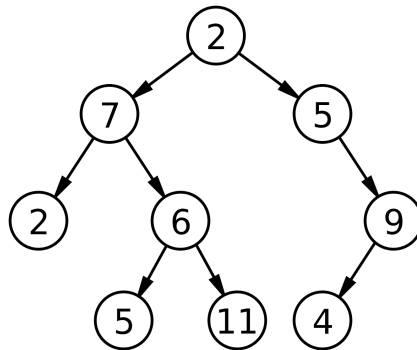


Qual a ordem dos nós a serem visitados ao partir do nó raiz?



Percurso em Árvores

Exemplo



2, 7, 5, 2, 6, 9, 5, 11, 4



Busca em largura

- Como implementar uma busca em largura?



Busca em largura

- Colocamos o nó inicial em uma **fila**.
- Enquanto a fila não for vazia:
 - ▶ Processo o nó que está na frente da fila.
 - ▶ Insira todos os seus vizinhos no fim da fila.
 - ▶ Retire o nó da fila.



Busca em Largura



Sumário

- 3 Percurso em Árvores
 - Busca em Largura
 - Busca em profundidade



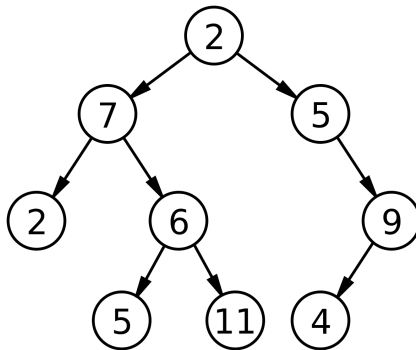
Busca em profundidade

- A busca em profundidade (depth-first-search), difere da busca em largura pois ela busca em um ramo inteiro da árvore antes de olhar para o outro ramo.
- Basicamente um nó é visitado e a busca procede recursivamente para o vizinho imediato.



Percurso em Árvores

Exemplo

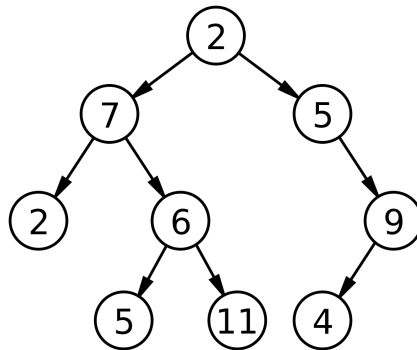


Qual a ordem dos nós a serem visitados ao partir do nó raiz?



Percurso em Árvores

Exemplo



2, 7, 2, 6, 5, 11, 5, 9, 4



Busca em profundidade

- Como implementar uma busca em profundidade?
- Substituindo a fila da busca em largura por uma pilha.



Busca em Profundidade



Busca em profundidade

- Implementação recursiva é mais simples, mais elegante e até mais rápida.
- Usamos uma pilha implícita quando chamamos a função recursivamente.



Busca em Profundidade

```
void dfs(tree_node* node){  
    if(node!=NULL){  
        processa(node);  
        dfs(node->left);  
        dfs(node->right);  
    }  
}
```



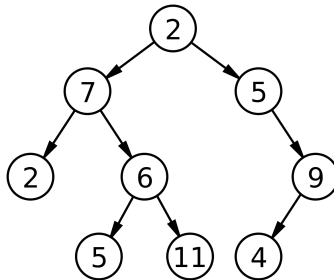
Busca em profundidade

- A busca em profundidade possui algumas variações.
- Pré-ordem: visitamos o nó antes de proceder recursivamente aos vizinhos.
- Em-ordem: procedemos recursivamente à esquerda, visitamos o nó, e procedemos recursivamente à direita.
- Pós-ordem: procedemos recursivamente à esquerda, procedemos recursivamente à direita, visitamos o nó.



Percurso em Árvores

Exemplo

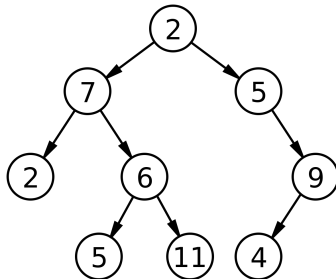


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em pré-ordem?



Percurso em Árvores

Exemplo

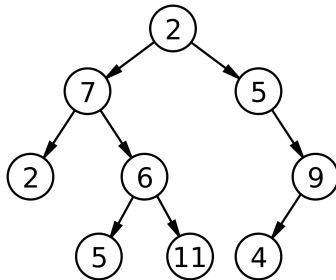


2, 7, 2, 6, 5, 11, 5, 9, 4



Percurso em Árvores

Exemplo

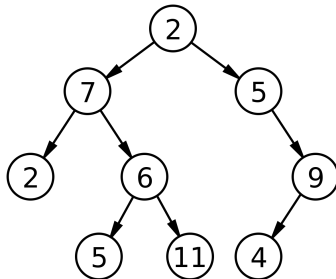


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em-ordem?



Percurso em Árvores

Exemplo



2, 7, 5, 6, 11, 2, 5, 4, 9



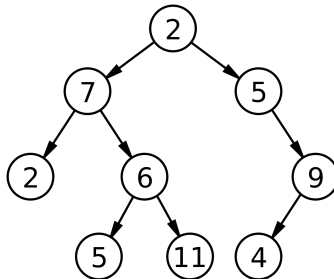
Busca em Profundidade

```
void dfs(tree_node* node){  
    if(node!=NULL){  
        dfs(node->left);  
        processa(node);  
        dfs(node->right);  
    }  
}
```



Percurso em Árvores

Exemplo

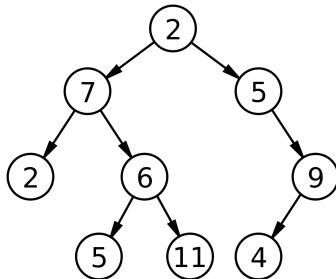


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em pós-ordem?



Percurso em Árvores

Exemplo



2, 5, 11, 6, 7, 4, 9, 5, 2



Busca em Profundidade



Sumário

4 Árvores Binárias de Pesquisa



Sumário

4 Árvores Binárias de Pesquisa

- Introdução
- Árvores AVL
- Treaps



Árvores Binárias de Pesquisa

- Uma árvore binária de pesquisa (BST ou Binary Search Tree) organiza os nós pela sua chave.
- Facilita a busca de uma determinada nó através da chave.



Árvores Binárias de Pesquisa

Definição (Árvore Binária de Pesquisa)

Uma árvore binária de pesquisa (BST) ordenada de acordo com a chave x de seus nós possui:

- Uma potencial raiz com valor de chave y .
- Caso a raiz exista, uma subárvore da esquerda com nós que possuem chaves menores que y .
- Caso a raiz exista, uma subárvore da direita com nós que possuem chaves maiores ou iguais a y .



Árvores Binárias de Pesquisa


 figuras/bst.png

Figura: Busca Pelo 19.



Árvores Binárias de Pesquisa

- Durante uma busca por um nó com chave x em uma BST, temos os seguintes casos:
 - ▶ A árvore é vazia: o nó não encontra-se na árvore.
 - ▶ A raiz possui a chave buscada: o nó buscado foi encontrado.
 - ▶ A chave buscada é menor que a chave da raiz: procede-se recursivamente para a subárvore da esquerda.
 - ▶ Caso contrário, procede-se recursivamente para a subárvore da direita.



Árvores Binárias de Pesquisa


 figuras/bst.png

Figura: Busca Pelo 19.



Árvores Binárias de Pesquisa

 figuras/bst.png

Figura: Busca Pelo 54.



Árvores Binárias de Pesquisa


 figuras/bst.png

Figura: Busca Pelo 18.



Árvores Binárias de Pesquisa

- A forma como uma BST é organizada lembra alguma técnica?



Árvores Binárias de Pesquisa

- A forma como uma BST é organizada lembra alguma técnica?
- **Busca binária.**



Árvores Binárias de Pesquisa

- A forma como uma BST é organizada lembra alguma técnica?
- **Busca binária.**
- A diferença agora é que a BST permite inserir e remover elementos sem precisar ordenar todo o vetor, no caso da busca binária.



Inserção em BSTs

- BSTs são estruturas de dados dinâmicas, permitem inserções e remoções.
- Todas as inserções são feitas nas folhas.



Árvores Binárias de Pesquisa

Inserção em uma BST

Durante a inserção de um nó com chave x em uma BST, temos os seguintes casos:

- A raiz é vazia: o novo nó passa a ser a raiz.
- A árvore é vazia: o nó é inserido.
- O nó a ser inserido possui a chave menor que a da raiz: procede-se recursivamente para a subárvore da esquerda.
- Caso contrário, procede-se recursivamente para a subárvore da direita.



Árvores Binárias de Pesquisa

► [BST Applet](#)



Árvores Binárias de Pesquisa

Remoção em uma BST

- Como visto, as inserções são simples, pois são sempre feitas nas folhas.
- No entanto, a remoção de um nó pode ocorrer em um nó interno.
- Como proceder?



Árvores Binárias de Pesquisa

Remoção em uma BST

Caso 1: o nó a ser removido é uma folha.

- Este é o caso mais simples.
- O nó é removido sem complicações.



Árvores Binárias de Pesquisa

► [BST Applet](#)



Árvores Binárias de Pesquisa

Remoção em uma BST

Caso 2: o nó a ser removido possui apenas um filho.

- Também pode ser lido facilmente.
- O filho é transplantado no lugar do pai.
 - ▶ O avô do filho passa a apontar diretamente para o filho.



Árvores Binárias de Pesquisa

Remoção em uma BST

Caso 3: o nó a ser removido possui dois filhos.

- Este é o caso mais difícil.
- Solução: transformar em um caso fácil.
- Obrigatoriamente o nó possui subárvores não vazias.
- Trocamos o valor do nó pelo seu antecessor.
 - ▶ O antecessor encontra-se no nó mais à direita da subárvore da esquerda.
- Procede-se recursivamente para remover o valor do nó.



Árvores Binárias de Pesquisa

► [BST Applet](#)



Árvores Binárias de Pesquisa

Problemas com BSTs

- BSTs oferecem um meio de organizar a informação e facilitar a busca.
- Problema: dependendo da ordem de inserções/remoções, a BST subjacente pode tornar-se degenerada.
- Operações começam a levar tanto tempo quanto em listas. . .



Exemplos

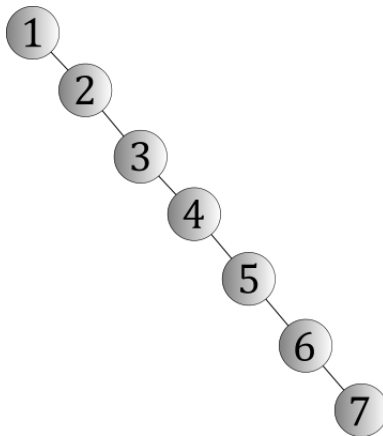


Figura: Árvore ou Lista?



Árvores Binárias de Pesquisa

► [BST Applet](#)



Árvores Binárias de Pesquisa

Solução

- Árvores Binárias de Pesquisa **Balanceadas**;
- Continuam sendo BSTs, mas utilizam operações que tentam espalhar os itens da árvore de modo a deixá-la balanceada de acordo com algum critério, como altura, peso , *etc* ...
- Em nosso curso, veremos duas delas:
 - ▶ Árvore AVL;
 - ▶ Treap.



Sumário

4 Árvores Binárias de Pesquisa

- Introdução
- Árvores AVL
- Treaps



Árvores AVL

- As árvores AVL foram as primeiras BST balanceadas da literatura.
- Nomeada de acordo com os seus desenvolvedores: Georgy **Adelson-Velsky** e Evgenii **Landis**.
- Possuem tempo de consulta/inserção/remoção limitado a $\Theta(\lg n)$.
- São estruturas que são auto-balanceáveis, tentando deixar as alturas dos nós de cada nível próximas.



Árvore AVL

`figuras/AVL-tree.pdf`



Árvore AVL

- Antes de expor os algoritmos que atuam sobre esta estrutura de dados, é necessário formalizar alguns conceitos.



Árvore AVL

Definição (Altura)

A altura de um nó x é dada pela maior distância dele até uma folha.

Esta altura é denotada por $height(x)$.

A altura de uma árvore vazia é 0.



Árvore AVL

Definição (Fator de Equilíbrio)

Sejam T_1 e T_2 as subárvores da esquerda e da direita de uma árvore com raiz no nó x . Sejam t_1 e t_2 as respectivas raízes de T_1 e T_2 .

O fator de equilíbrio (balance factor) de x é dado como:

$$bf(x) := height(t_1) - height(t_2)$$



Árvore AVL

`figuras/AVL-tree.pdf`



Árvore AVL

Fator de Equilíbrio

- Em árvores AVL, o fator de equilíbrio de cada nó está limitado a três possíveis valores: $\{-1, 0, 1\}$.
- Esta restrição que possibilita operações de inserção/remoção/busca serem efetuadas em tempo $\Theta(\lg n)$.



Árvore AVL

`figuras/AVL-tree.pdf`



Operações em Árvores AVL

- Como árvores AVL são especializações de BST, os procedimentos de inserção, remoção e busca são rigorosamente iguais.
- O problema é que, após uma inserção e remoção, a árvore pode ficar desbalanceada, isto é, o fator de equilíbrio de algum nó está fora do intervalo $\{-1, 0, 1\}$.
- Assim, é necessário rebalancear a árvore.

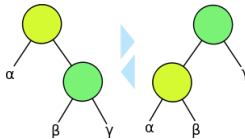


Balanceamento em Árvores AVL

- O rebalanceamento é feito através de rotações, que podem ser:
 - ▶ Rotações para a esquerda.
 - ▶ Rotações para a direita.
- Uma rotação não interfere na propriedade de BST, isto é, a subárvore da esquerda continua os elementos com chaves menores do que a raiz e a subárvore da direita continua com os elementos com chaves maiores ou iguais a da raiz.

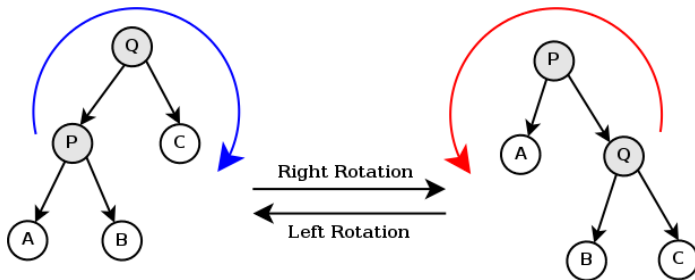


Rotações





Rotações





Árvores AVL

► Tree Rotation



Árvores AVL

- Ao final de cada inserção/remoção, a árvore deve ser atualizada.
- Para cada nó ao longo do caminho percorrido, deve-se computar:
 - ▶ A nova altura.
 - ▶ O novo fator de equilíbrio.
- Rotações para a esquerda/direita são feitas de modo a preservar os fatores de balanceamento no intervalo $\{-1, 0, 1\}$.
 - ▶ Rotação para esquerda aumenta em 1 o fator de equilíbrio.
 - ▶ Rotação para direita diminui em 1 o fator de equilíbrio.



Balanceamento de Árvores AVL

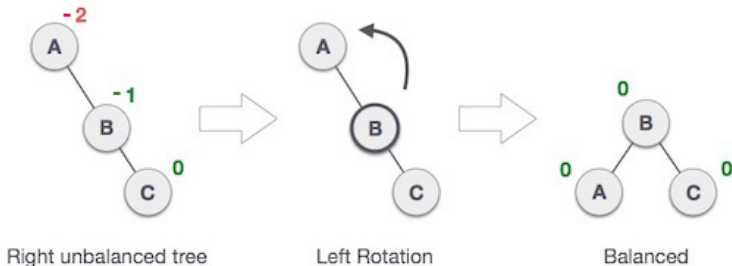
- O balanceamento das árvores AVL concentram-se em 4 casos:
 - ▶ Rotação para esquerda (L).
 - ▶ Rotação para direita (R).
 - ▶ Rotação para esquerda seguida de rotação para direita (LR).
 - ▶ Rotação para direita seguida de rotação para esquerda (RL).



Balanceamento de Árvores AVL

Caso 1 (L)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da direita possui fator de equilíbrio ≤ 0 , uma rotação à esquerda é suficiente para balancear a árvore.





Balanceamento de Árvores AVL

Caso 2 (R)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio ≥ 0 , uma rotação à direita é suficiente para balancear a árvore.

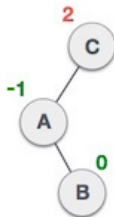
`figuras/avl-tree-case-2.jpg`



Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

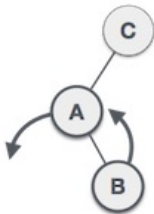




Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

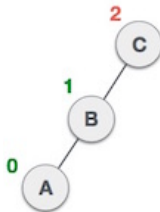




Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

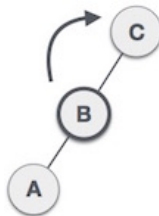




Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

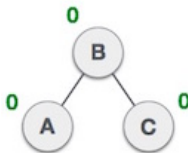




Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a árvore com raiz em x tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

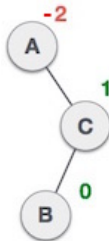




Balanceamento de Árvores AVL

Caso 4 (RL)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

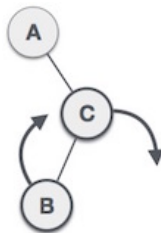




Balanceamento de Árvores AVL

Caso 4 (RL)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

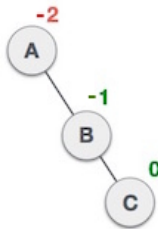




Balanceamento de Árvores AVL

Caso 4 (RL)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

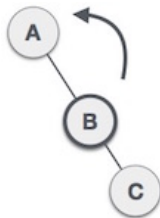




Balanceamento de Árvores AVL

Caso 4 (RL)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.





Balanceamento de Árvores AVL

Caso 4 (RL)

- Se a árvore com raiz em x tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0 , uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

`figuras/avl-tree-case-4-5.jpg`



Comparação Árvores AVL

Operação	Inserção	Remoção	Busca
Árvore	Complexidade (Pior caso)		
BST	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
AVL	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$



Árvores Binárias de Pesquisa

► AVL Applet



Sumário

4 Árvores Binárias de Pesquisa

- Introdução
- Árvores AVL
- Treaps

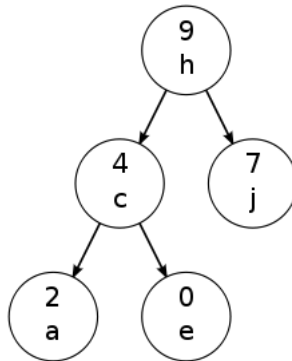


Treap

- $\text{Treap} = \text{BST} + \text{Heap}$.
- Uma Treap é uma estrutura de dados **probabilística**.
- Também é uma BST, mas agora cada nó tem duas informações:
 - ▶ Valor da chave.
 - ▶ Prioridade.
- Durante a inserção de um nó, o campo de prioridade é gerado aleatoriamente.
- Para conservar a propriedade de Heap, rotações podem ser feitas, ocasionado uma árvore relativamente balanceada com **alta probabilidade**.



Treap





Treap

- Definiremos a estrutura formalmente para que possamos elaborar os algoritmos sobre ela com precisão.



Treap

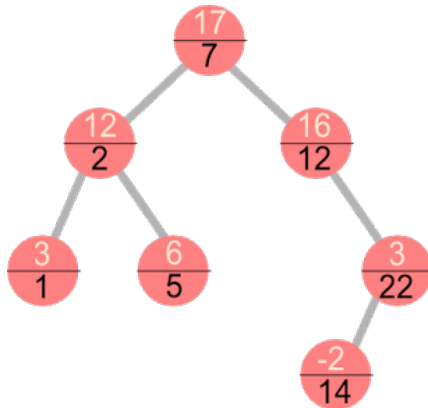
Definição (Treap)

Uma treap possui:

- Uma potencial raiz, com chave e prioridade.
- Caso a raiz exista, uma subtreap da esquerda, de modo que todo elemento possua chave menor do que a chave da raiz e prioridade menor ou igual a prioridade da raiz.
- Caso a raiz exista, uma subtreap da direita, de modo que todo elemento possua chave maior do que a chave da raiz e prioridade menor ou igual a prioridade da raiz.



Treap





Treap: Busca

- A busca de um nó pela sua chave em uma Treap é idêntica a de uma BST.
- As prioridades são ignoradas completamente.

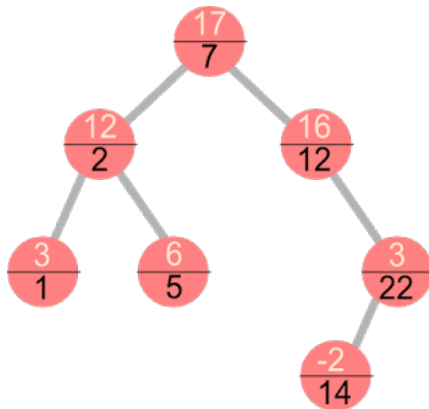


Treap: Busca

► [Treap Applet](#)



Treap



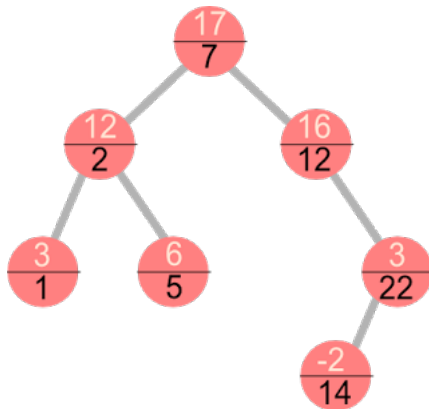


Treap: Inserção

- A inserção é feita como em uma BST.
- No fim o nó é inserido e corresponde a uma folha.
- A este nó é gerado uma prioridade.
- Rotações são feitas de modo a conservar a propriedade de heap da Treap, isto é, enquanto o nó inserido tiver prioridade maior que o seu pai, uma rotação é feita.



Treap





Treap: Inserção

► [Treap Applet](#)

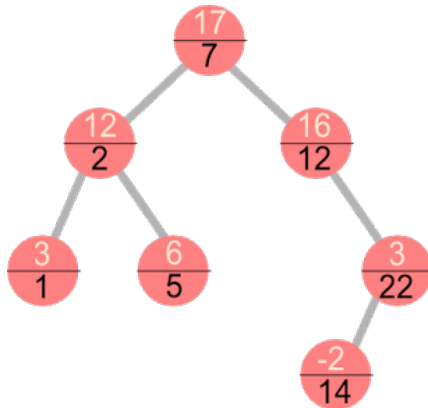


Treap: Remoção

- É feita como em uma BST.
- Caso 1: se o nó a ser removido é uma folha, remova-o sem complicações.
- Caso 2: se o nó a ser removido possui apenas um filho, transplante o filho no lugar do pai.
- Caso 3: se o nó a ser removido possui dois filhos, troque-o de lugar com o filho de maior prioridade através de uma rotação.
 - ▶ Eventualmente esta operação fará com que o nó a ser removido recaia no caso 1 ou no caso 2.
 - ▶ Como a rotação troca com o filho de maior prioridade, a propriedade de heap é conservada após a remoção.



Treap





Treap: Inserção

► [Treap Applet](#)



Sumário

5 Links



Links

- Applet para visualização de EDs:
<https://people.ksp.sk/~kuko/gnarley-trees/>.