Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 343
Desafios de Programação

João Batista Ribeiro
joao42lbatista@gmail.com

Slides baseados no material do prof. Guilherme C. Pena

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Estruturas de Dados Elementares

Estruturas de Dados Elementares

Árvores:

Listas, pilhas, filas são todas estruturas de dados lineares.

No caso das árvores, temos uma estrutura de dados bidimensional formada por uma série de nós.

Quando existe, a árvore inicia em um nó que denominamos **raiz**.

Estruturas de Dados Elementares

Árvores (Propriedades):

- Uma árvore com zero nós é dita uma árvore vazia;
- Todo nó de uma árvore é raiz de uma subárvore;
- O número de subárvores de um nó é denominado o **grau** daquele nó;
- O **grau da árvore** é definido como o maior grau dentre todos os seus nós. Em particular, uma árvore de grau 2 é denominada uma árvore binária;

Estruturas de Dados Elementares

Árvores (Propriedades):

- Os nós de grau zero são denominados ***folhas*** da árvore.

- Um nó y que está imediatamente “**abaixo**” de um nó x é dito ser um ***descendente direto ou filho*** do nó x. Reciprocamente, o nó x é dito ser o ***pai*** do nó y;

Estruturas de Dados Elementares

Árvores (Propriedades):

- A raiz da árvore está no nível 1. Os filhos de um nó no nível *i* estão no nível (*i+1*);

- A altura da árvore é definida como sendo o nível máximo dentre todos os seus nós;

- O número máximo de nós numa árvore de grau d e altura h, indicado por $N_d(h)$ é:

$$N_d(h) = \sum_{i=0}^{h-1} d^i$$

Estruturas de Dados Elementares

Árvores:

Existem muitas representações de árvores na computação.

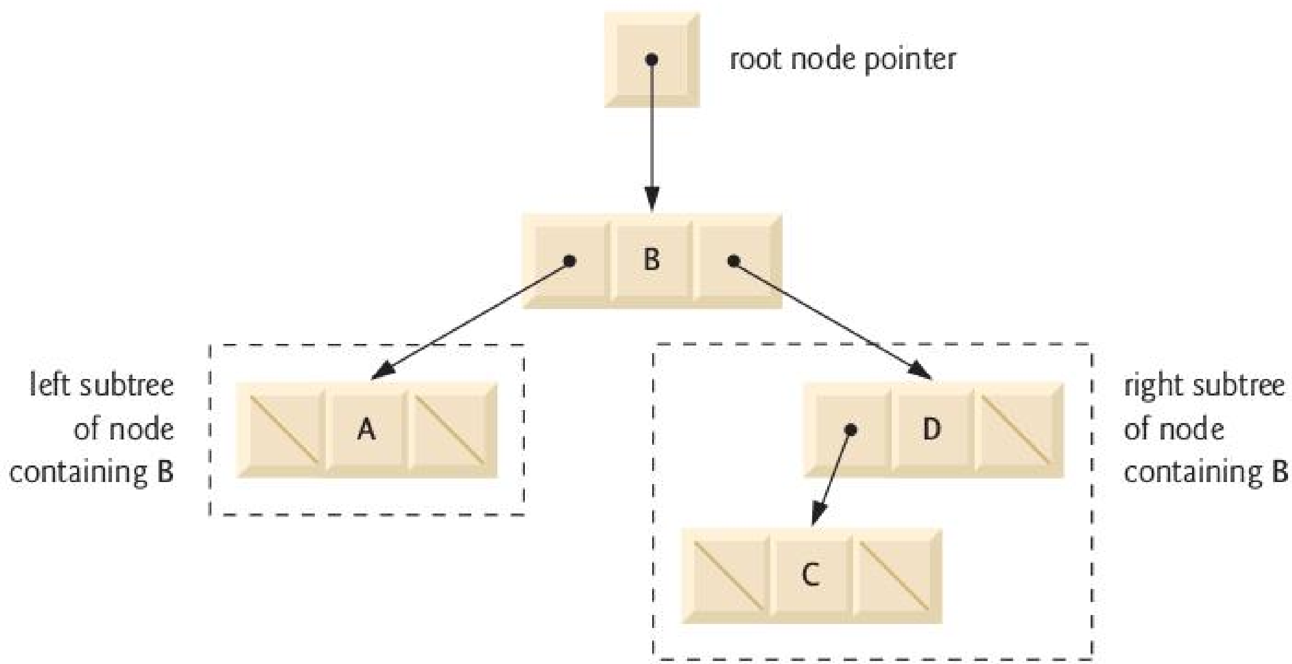
Por enquanto vamos começar com a mais simples delas:

- **Árvore Binária de Pesquisa**

Estruturas de Dados Elementares

Árvore Binária de Pesquisa:

Uma ABP (*BST – Binary Search Tree*) é um tipo especial de árvore em cada nó da árvore possui de **0 a 2 nós-filhos**.



Estruturas de Dados Elementares

Árvore Binária de Pesquisa:

Uma ABP (*BST – Binary Search Tree*) também possui as características:

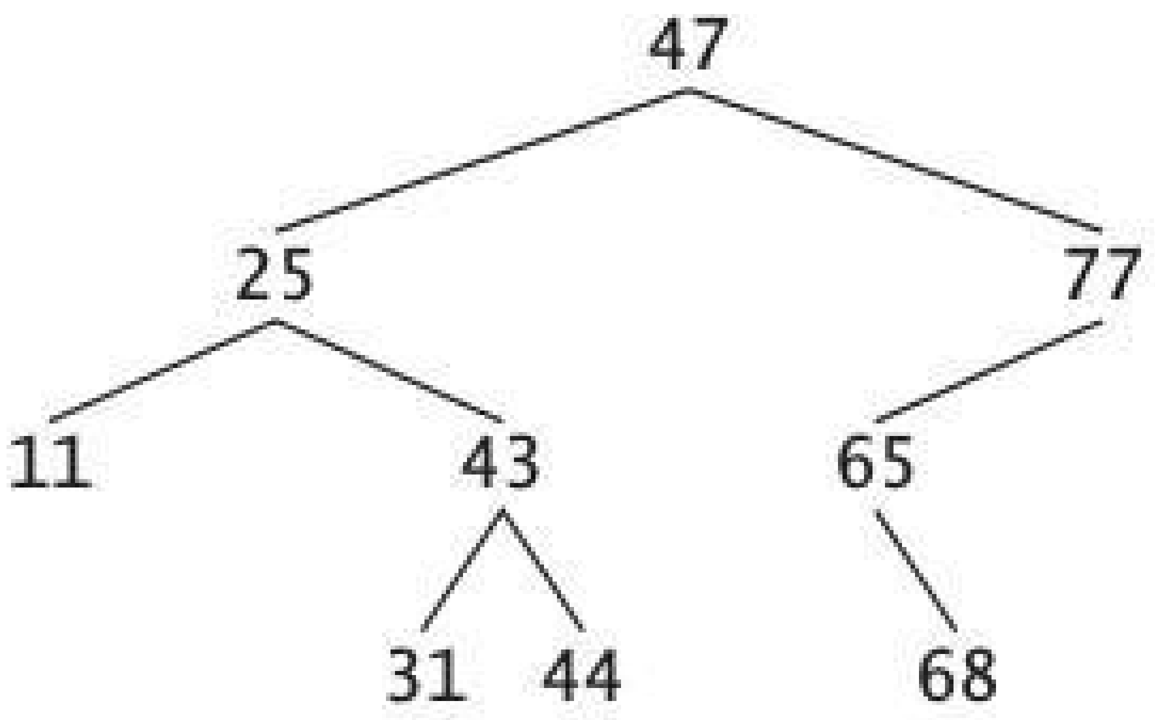
- de que os todos os valores são únicos;

- de que os valores em qualquer sub-árvore esquerda são menores que o valor em seu nó-pai e;

- de que os valores em qualquer sub-árvore direita são maiores que o valor em seu nó-pai;

Estruturas de Dados Elementares

Árvore Binária de Pesquisa:



Estruturas de Dados Elementares

Árvore Binária de Pesquisa:

Casos especiais:

a) árvore binária degenerada: todos os nós, exceto as folhas, têm um único descendente;

b) árvore binária completa: todos os nós, exceto as folhas, têm dois descendentes e todas as folhas estão num mesmo nível .

Estruturas de Dados Elementares

Árvore Binária de Pesquisa (Operações):

- **CriaArvore() :** cria uma árvore binária vazia;
- **localiza (k,T) :** retorna uma referência para o nó da árvore T que contém a chave igual a k; caso não exista nenhum item com esta chave, retorna (NULL);
- **insere(x,T) :** insere o item x na árvore T;
- **elimina(k,T) :** elimina da árvore T, o item cuja chave é igual a k.

Estruturas de Dados Elementares

Árvore Binária de Pesquisa (Operações):

- **max(T) :** retorna uma referência para o nó que contém o item com a maior chave na árvore T;
- **min(T) :** retorna uma referência para o nó que contém o item com a menor chave na árvore T;
- **estaVazia(T) :** retorna verdadeiro caso a árvore T esteja vazia e retorna falso caso contrário.
- **imprime(T) :** imprime (em ordem crescente) os elementos armazenados na árvore

Estruturas de Dados Elementares

Árvore Binária de Pesquisa (Implementação):

Ver arquivos:

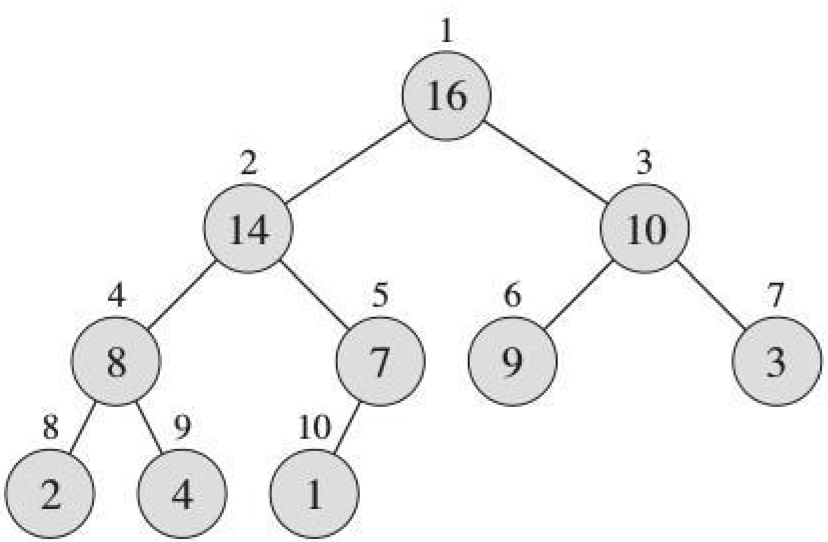
ArvBinPesq.h e TestaArvore.cpp

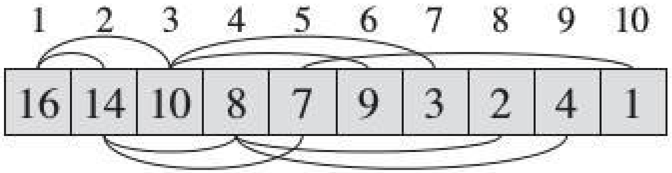
Estruturas de Dados Elementares

Heap (Binário):

A estrutura de dados *heap (binário)* é um objeto arranjo que pode ser vista como uma árvore binária quase completa com dois atributos:

A.comprimento e **A.tamanho-do-heap**





Estruturas de Dados Elementares

Heap (Binário):

A raiz da árvore é A[1] e, dado o índice i de um nó, podemos calcular facilmente os índices de seu pai, do filho à esquerda e do filho à direita:

Pai(i) : return [i/2]

Fesq(i): return [2i]

Fdir(i): return [2i+1]

Estruturas de Dados Elementares

Heap (Binário):

Existem dois tipos de heaps binários: ***Heaps de Máximo*** e ***Heaps de Mínimo***.

Heap de Máximo:

A ***propriedade de heap de máximo*** é que, para todo ***nó i*** exceto a raiz,

A[Pai(i)] >= A[i]

Isto é, o valor de um nó é no máximo o valor de seu pai. Assim, o maior elemento está na raiz.

Estruturas de Dados Elementares

Heap (Binário):

Existem dois tipos de heaps binários: ***Heaps de Máximo*** e ***Heaps de Mínimo***.

Heap de Mínimo:

A ***propriedade de heap de mínimo*** é que, para todo ***nó i*** exceto a raiz,

A[Pai(i)] <= A[i]

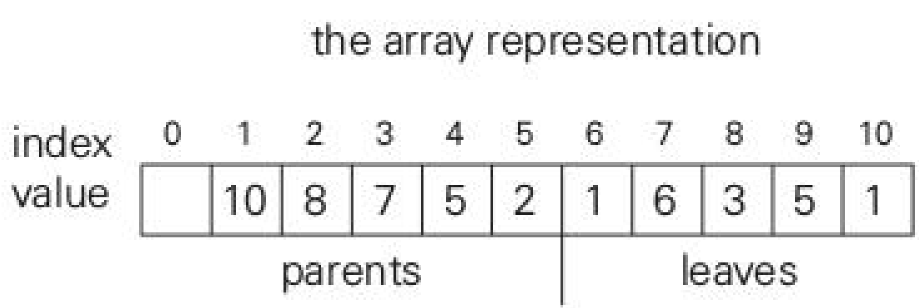
Isto é, o valor de um nó é no mínimo o valor de seu pai. Assim, o menor elemento está na raiz.

Estruturas de Dados Elementares

Heap (Binário):

Ideia geral de como construir um heap a partir de um vetor (***heap de máximo neste exemplo***):

Passo 1: Comece com a posição pai n / 2;
Passo 2: Verifique a condição de heap para a posição pai atual. Se a condição não for satisfeita, siga trocando o elemento desta posição com o maior de seus filhos até que a condição seja satisfeita;
Passo 3: Repita o passo 2 para a posição pai seguinte (atual-1) até que não haja mais posições pais para processar.



Estruturas de Dados Elementares

Heap (Binário):

MAX-HEAPIFY(A, i)

```
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

```
void heap(int *h, int ini, int fim) {
    int k, v;
    bool heap;//Satisfaz Propriedade
    for(int i=fim/2; i>=ini; i--) {
        k = i;
        v = h[k];
        heap = false;
        while(!heap && 2*k <= fim) {
            int j = 2*k;
            if(j<fim) //Maior Filho
                if(h[j] < h[j+1])
                    j++;
            if(v >= h[j])
                heap = true;
            else {
                h[k] = h[j];
                k = j;
            }
        }

        h[k] = v;
    }
}
```

Estruturas de Dados Elementares

Fila de Prioridades:

Uma das aplicações mais populares dos ***heaps*** são as ***filas de prioridade***.

Da mesma forma que os heaps, existem as filas de prioridade **máxima** e **mínima**.

Estruturas de Dados Elementares

Fila de Prioridades (Operações):

Supondo uma ***fila de prioridades máxima***, temos as seguintes operações:

- **Insere(S, x) :** insere o elemento x no conjunto S;
- **Maximum (S) :** Devolve o elemento de S que tem a maior chave;
- **Remove-Max(S) :** Remove e devolve o elemento de S que tem a maior chave;
- **Aumenta-Chave(S, x, k) :** aumenta o valor da chave do elemento x até o novo valor k, que admite-se ser, pelo menos, tão grande quando o valor da chave atual de x.

Estruturas de Dados Elementares

Fila de Prioridades (Operações):

- **Remove-Max(S) :**

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size − 1
MAX-HEAPIFY(A, 1)
return max
```

Estruturas de Dados Elementares

Fila de Prioridades (Operações):

- **Aumenta-Chave(S, x, k) :**

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

- **Insere(S, x) :**

```
MAX-HEAP-INSERT(A, key)
A.heap-size = A.heap-size + 1
A[A.heap-size] = -∞
HEAP-INCREASE-KEY(A, A.heap-size, key)
```

Estruturas de Dados Elementares

Fila de Prioridades (Implementação):

Ver arquivo: **p_queue.cpp**

Estruturas de Dados Elementares

Fila de Prioridades (C++):

#include <queue>

C++ traz uma implementação de uma **fila de prioridades de máximo.**

http://www.cplusplus.com/reference/queue/priority_queue/

Estruturas de Dados Elementares

Fila de Prioridades (C++):

```
#include <iostream>
#include <queue>

using namespace std;

int main (){
  priority_queue<int> mypq;
  mypq.push(30);
  mypq.push(100);
  mypq.push(25);
  mypq.push(40);

  while (!mypq.empty()) {

     cout << ' ' << mypq.top();
     mypq.pop();
  }
  cout << '\n';
  return 0;
}
```

```
100 40 30 25
```

Estruturas de Dados Elementares

Árvores AVL:

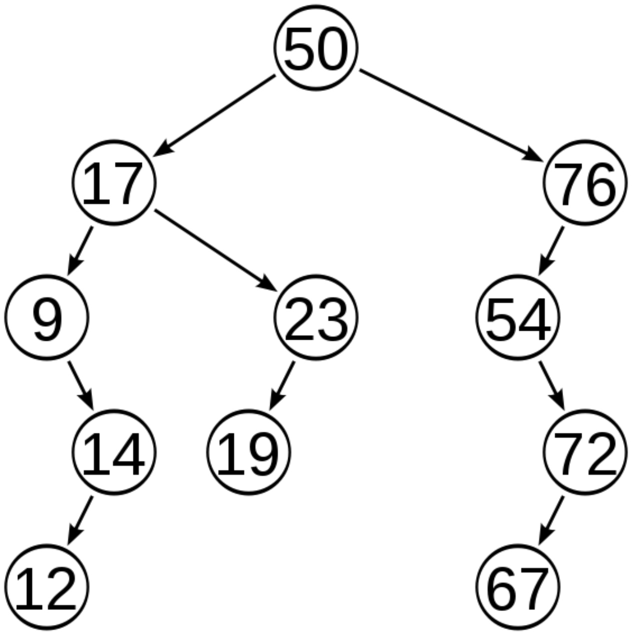
Uma ***árvore AVL*** é uma árvore binária de pesquisa de altura balanceada.

Para cada nó x, o módulo da diferença entre as alturas das sub-árvores à esquerda e à direita é no máximo 1.

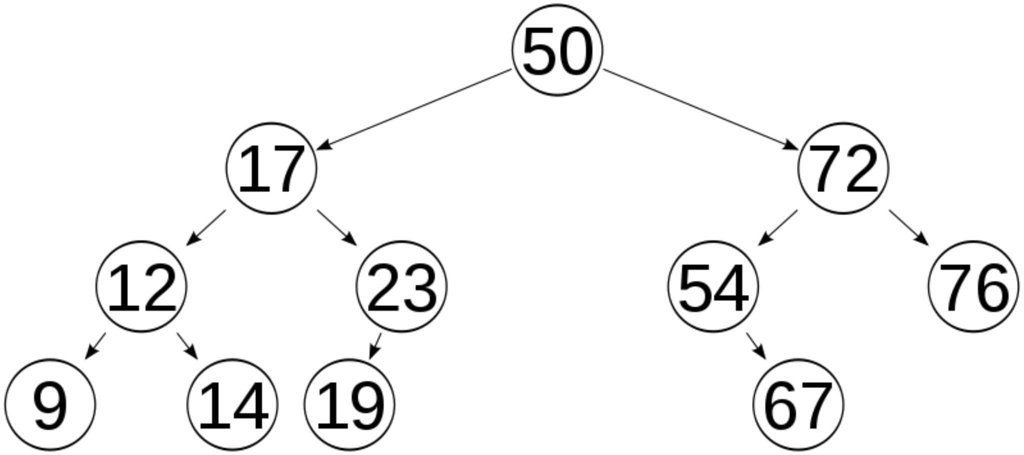
Chamamos essa característica de ***fator de balanço***, que no caso pode ser -1, 0 ou 1 em cada nó.

Estruturas de Dados Elementares

Árvores AVL:



Não Balanceada.



Balanceada.

Estruturas de Dados Elementares

Árvores AVL:

Para transformarmos uma árvore não-balanceada em uma balanceada, precisamos fazer uma série de **rotações**.

A **rotação** é uma operação local feita na árvore com o objetivo de balanceá-la preservando a propriedade de ABP.

Estruturas de Dados Elementares

Árvores AVL:

Para garantirmos as propriedades da árvore AVL rotações devem ser feitas conforme necessário após operações de ***remoção*** ou ***inserção***.

Seja ***P*** o nó pai, ***FE*** o filho da esquerda de P e ***FD*** o filho da direita de P podemos definir 2 categorias de rotação: ***Rotações Simples*** e ***Rotações Duplas.***

Estruturas de Dados Elementares

Árvores AVL:

Uma **rotação simples** ocorre quando um nó está desbalanceado e seu filho estiver no mesmo sentido da inclinação, formando uma linha reta.

Uma **rotação dupla** ocorre quando um nó estiver desbalanceado e seu filho estiver inclinado no sentido inverso ao pai, formando um "*joelho*".

Estruturas de Dados Elementares

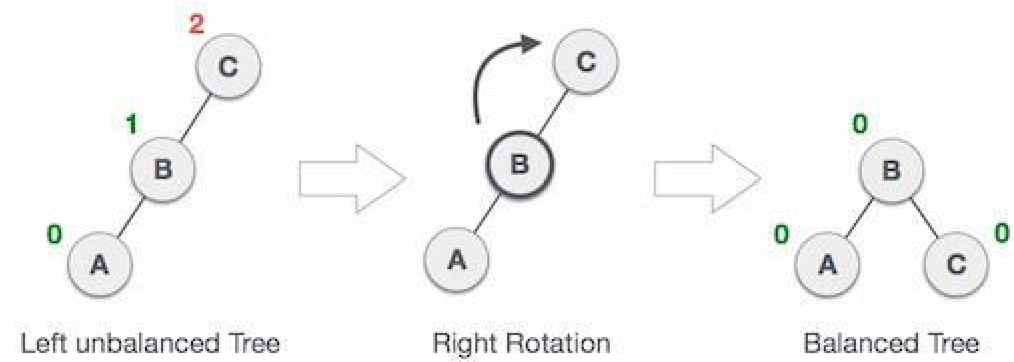
Árvores AVL:

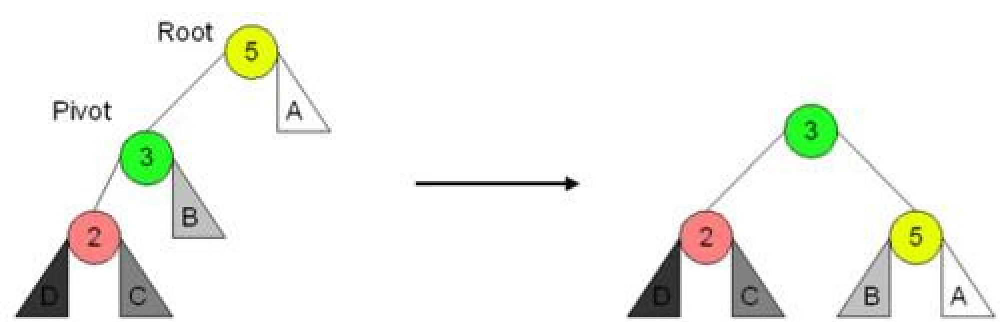
Rotação Simples à direita (Right-Rotation-RR):

A raiz vira um filho direito.

RR(): Pseudocódigo

1. AUX = C
2. raiz = B
3. AUX.esq = B.dir
4. B.dir = AUX



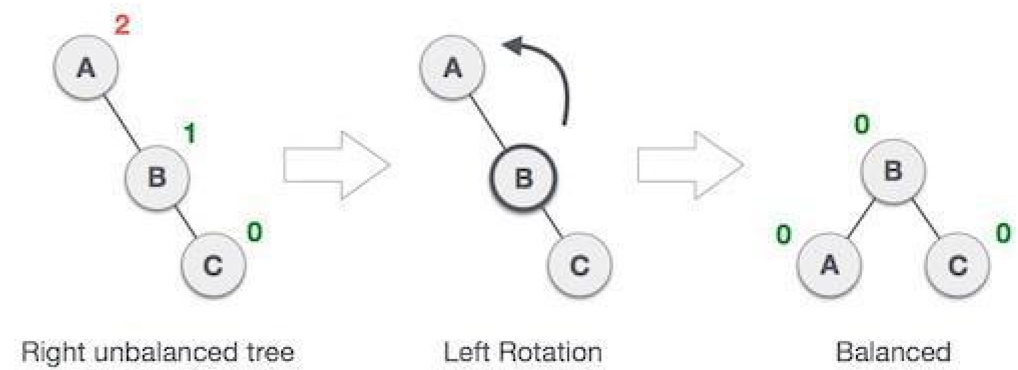


Estruturas de Dados Elementares

Árvores AVL:

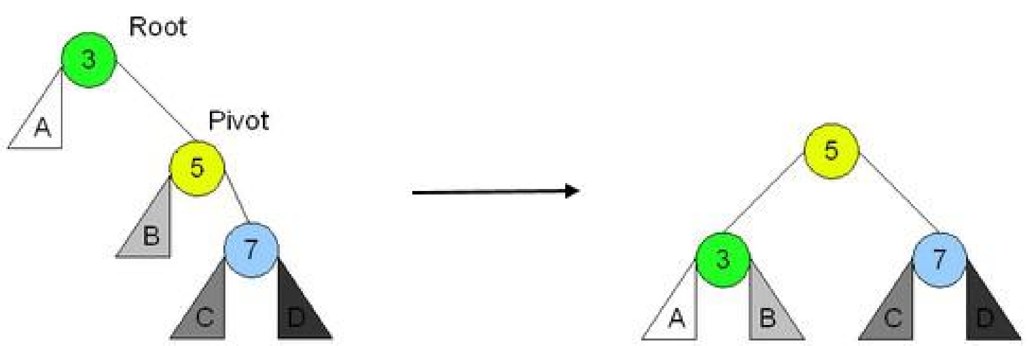
Rotação Simples à esquerda (Left-Rotation-LL):

A raiz vira um filho esquerdo.



LL(): Pseudocódigo

1. AUX = A
2. raiz = B
3. AUX.dir = B.esq
4. B.esq = AUX



Estruturas de Dados Elementares

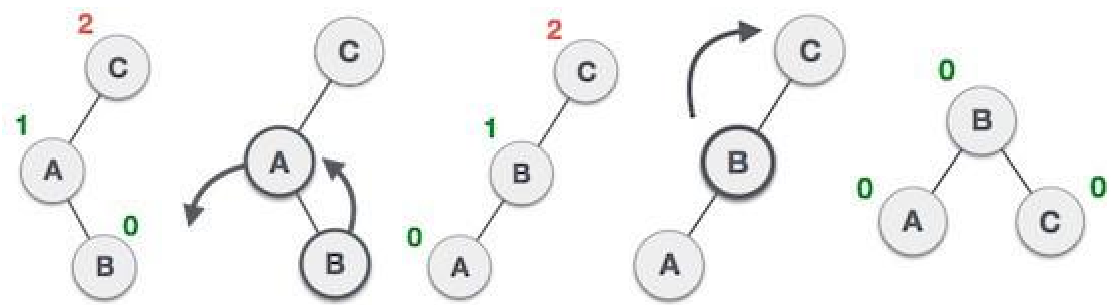
Árvores AVL:

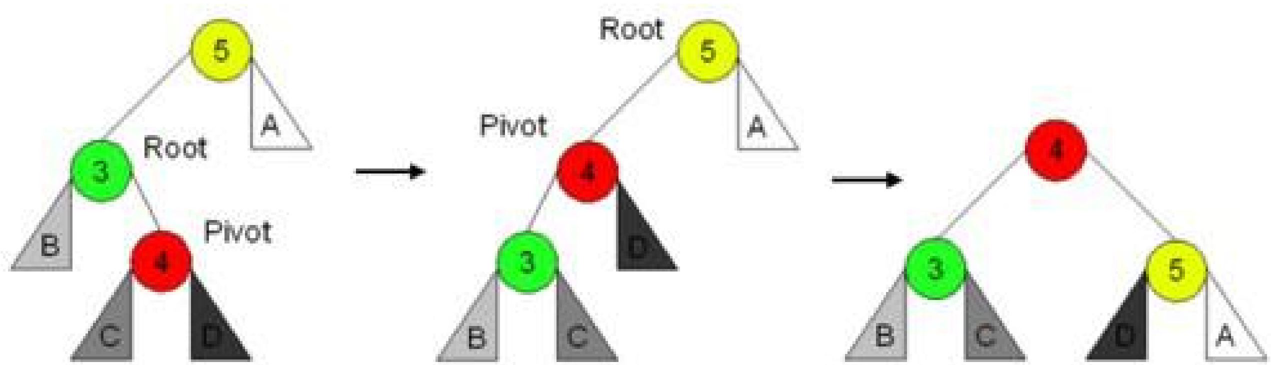
Rotação Dupla à direita (Left-Right Rotation (LR):

A raiz vira um filho direito na segunda rotação.

LR(): Pseudocódigo

1. AUX1 = C
2. AUX2 = A
2. raiz = B
3. AUX1.esq = B.dir
4. AUX2.dir = B.esq
5. B.esq = AUX2
6. B.dir = AUX1





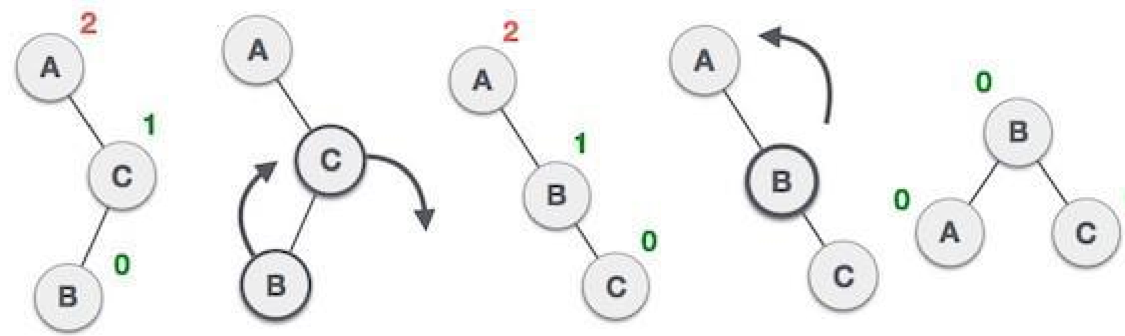
SIN343 - Desafios de Programação

Estruturas de Dados Elementares

Árvores AVL:

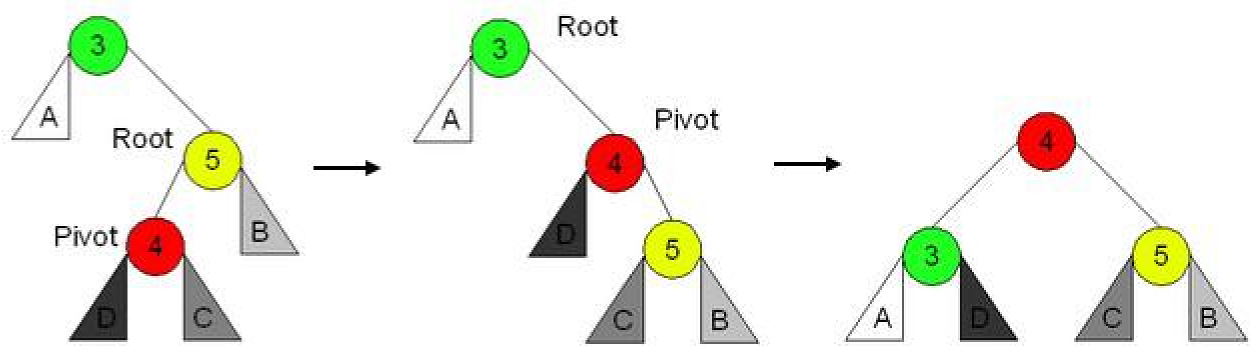
Rotação Dupla à esquerda (Right-Left Rotiation (RL):

A raiz vira um filho esquerdo na segunda rotação.



RL(): Pseudocódigo

1. AUX1 = A
2. AUX2 = C
2. raiz = B
3. AUX1.dir = B.esq
4. AUX2.esq = B.dir
5. B.esq = AUX1
6. B.dir = AUX2



Estruturas de Dados Elementares

Árvore AVL (Implementação):

Ver arquivos:

avl.h e TestaAVL.cpp

Estruturas de Dados Elementares

Treap (ABP + Heap):

O Treap é uma árvore binária de pesquisa balanceada onde a cada nó além da ***chave*** é definida uma ***prioridade***.

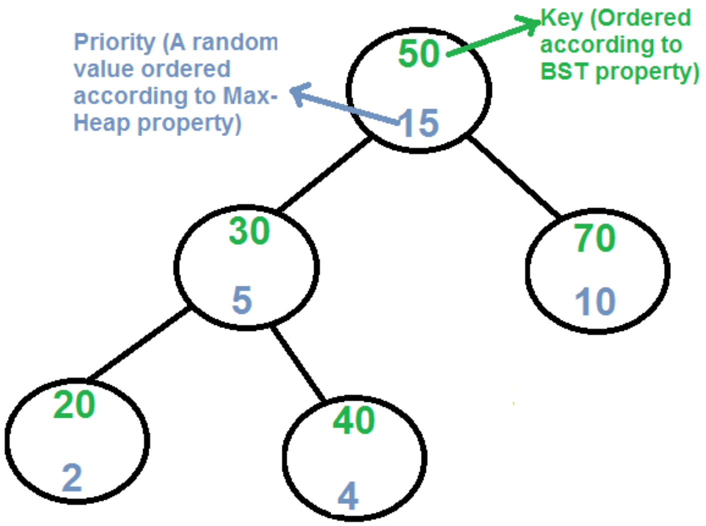
A ordenação dos nós obedecem as **propriedades de uma ABP** e as **propriedades de um heap** (de máximo neste exemplo):

- Se **v** é um filho esquerdo de **u**, ***v.chave < u.chave***
- Se **v** é um filho direito de **u**, ***v.chave > u.chave***
- Se **v** é um filho de **u**, então ***v.prioridade < u.prioridade***

Estruturas de Dados Elementares

Treap (ABP + Heap):

A ideia básica usa uma randomização sobre as prioridades e a propriedade de heap para manter o balanço. A complexidade esperada da busca, inserção e remoção é ***O(Log n)***.



Estruturas de Dados Elementares

Treap (ABP + Heap):

O ***Treap*** também usa rotações para manter a propriedade de heap máximo durante as inserções e remoções.

Estruturas de Dados Elementares

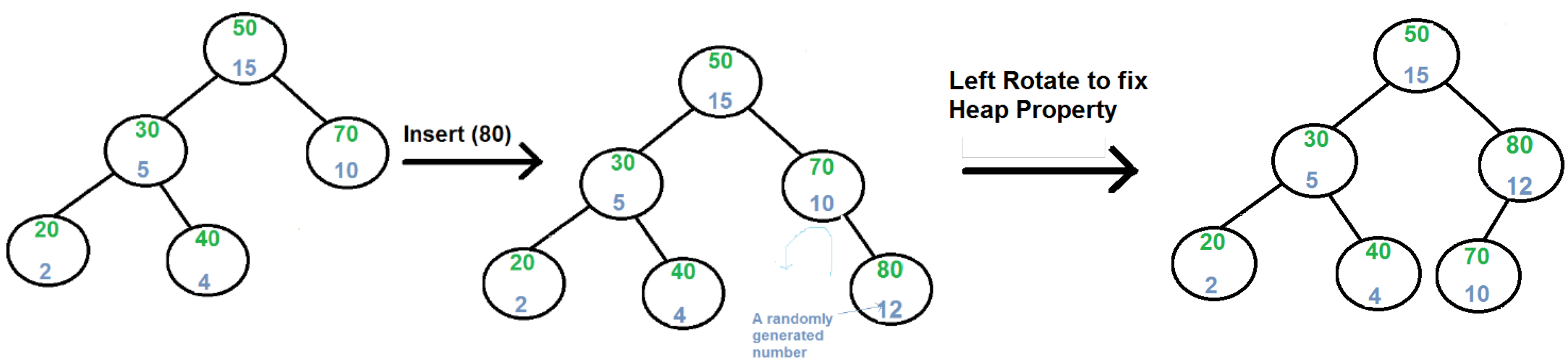
Treap (ABP + Heap):

Insert(x):

1)Cria um novo nó com chave x e prioridade *rand*.

2)Insere normalmente numa ABP.

3)Usa as rotações para garantir que a prioridade do novo nó segue a proriedade do heap.



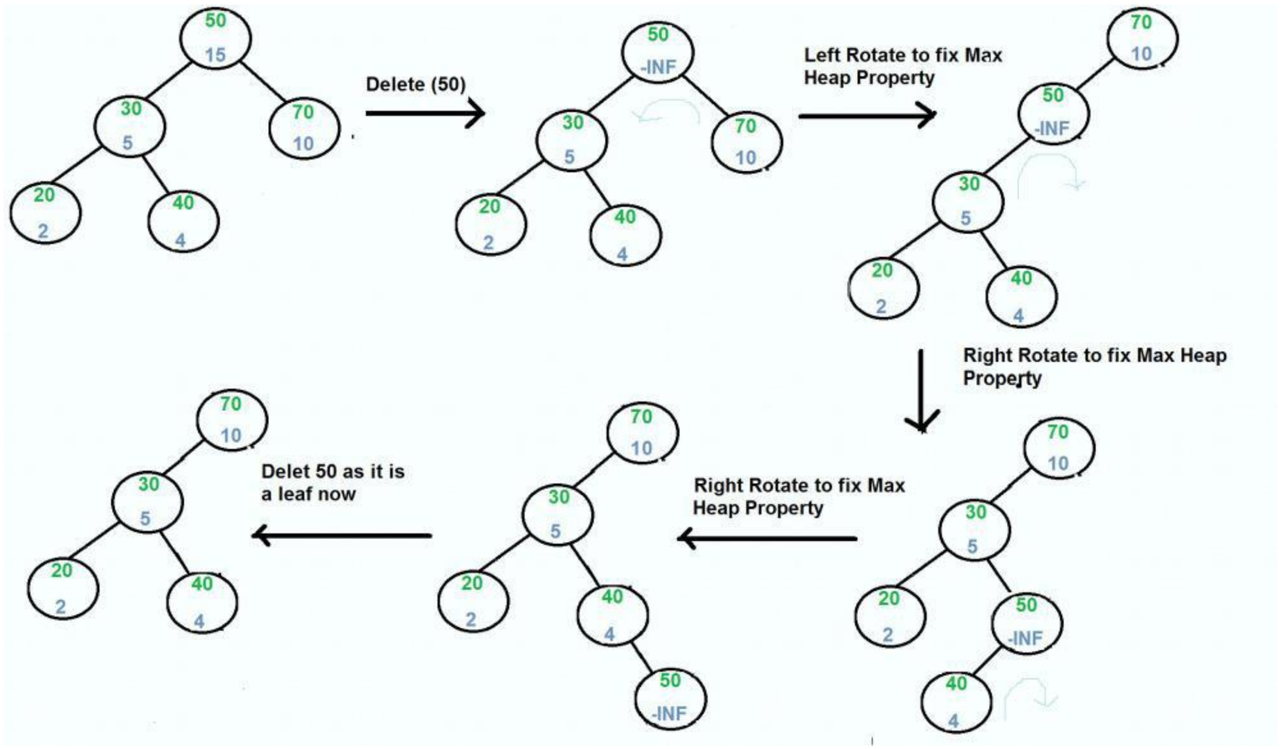
Estruturas de Dados Elementares

Treap (ABP + Heap):

Delete(x):

1) Se o nó a ser deletado é uma folha, delete-o.

2) Se não, reduza a prioridade dele para um valor muito pequeno e realize as rotações até se tornar uma folha.



Estruturas de Dados Elementares

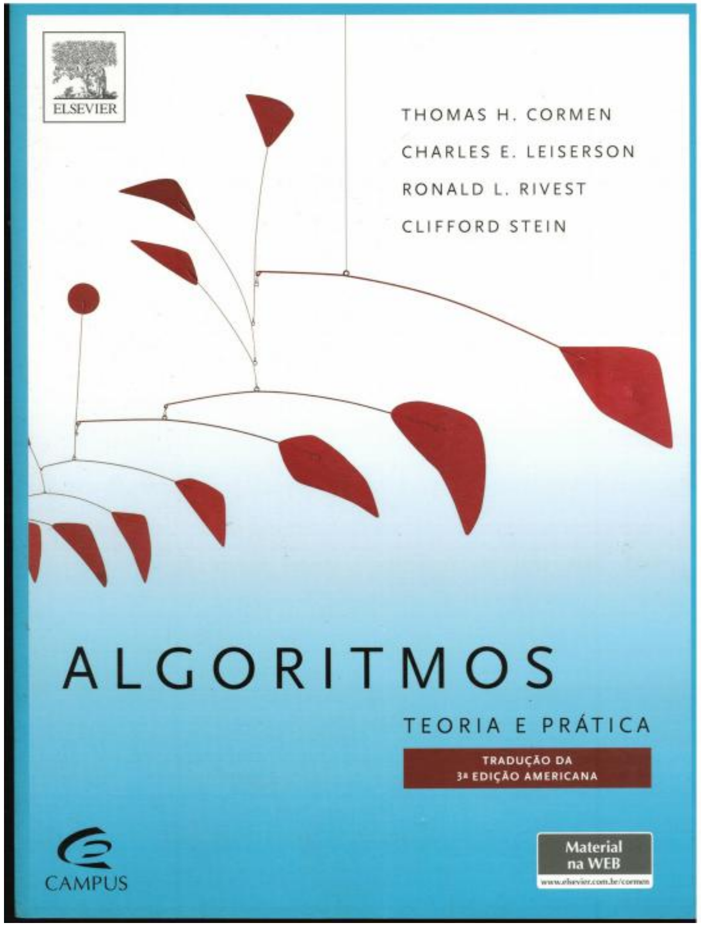
Treap (Implementação):

Ver arquivo:

Treap.cpp

Referência Bibliográfica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. Tradução da 2. ed. Americana. Rio de Janeiro: Campus, 2002.



Referência Bibliográfica

- SKIENA, S.S. REVILLA, M. A. Programming challenges: the programming contest training manual. Springer, 2003.