MC-202 — Aula 18
Escolhendo uma estrutura de dados

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.

Mas, afinal, qual usar?

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.
Mas, afinal, qual usar?

Resposta: Depende

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.
Mas, afinal, qual usar?

Resposta: Depende de quê?

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.
Mas, afinal, qual usar?

Resposta: Depende de quê?

Analisando um problema

Sempre que vamos escolher uma estrutura de dados para resolver um problema, temos que perguntar:

- quantos elementos?
- quantas vezes vamos utilizá-lo?
- que operações queremos realizar?
- queremos alguma garantia?
- que restrições?

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.
Mas, afinal, qual usar?

Resposta: Depende de quê?

Analisando um problema

Sempre que vamos escolher uma estrutura de dados para resolver um problema, temos que perguntar:

- quantos elementos?
- quantas vezes vamos utilizá-lo?
- que operações queremos realizar?
- queremos alguma garantia?
- que restrições?
- que forma deve ter a estrutura: matriz, árvore, etc.?

Árvores, árvores e mais árvores

Já vimos diversas árvores e variantes.
Mas, afinal, qual usar?

Resposta: Depende de quê?

Analisando um problema

Sempre que vamos escolher uma estrutura de dados para resolver um problema, temos que perguntar:

- quantos elementos?
- quantas vezes vamos utilizá-lo?
- que operações queremos realizar?
- queremos alguma garantia?
- que restrições?
- que forma deve ter a estrutura: matriz, árvore, etc.?
- ...

Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

F_0

Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

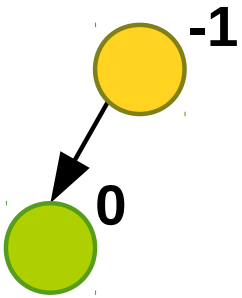
F_1

Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

F_2

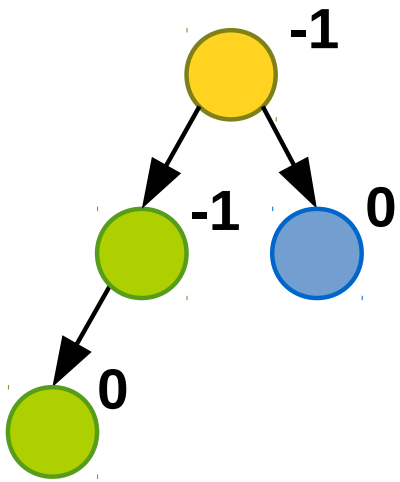


Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

F_3

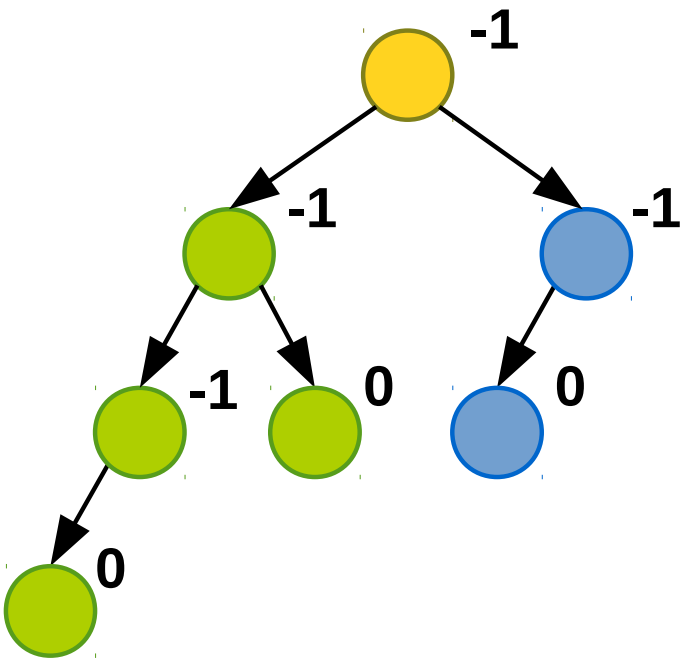


Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?

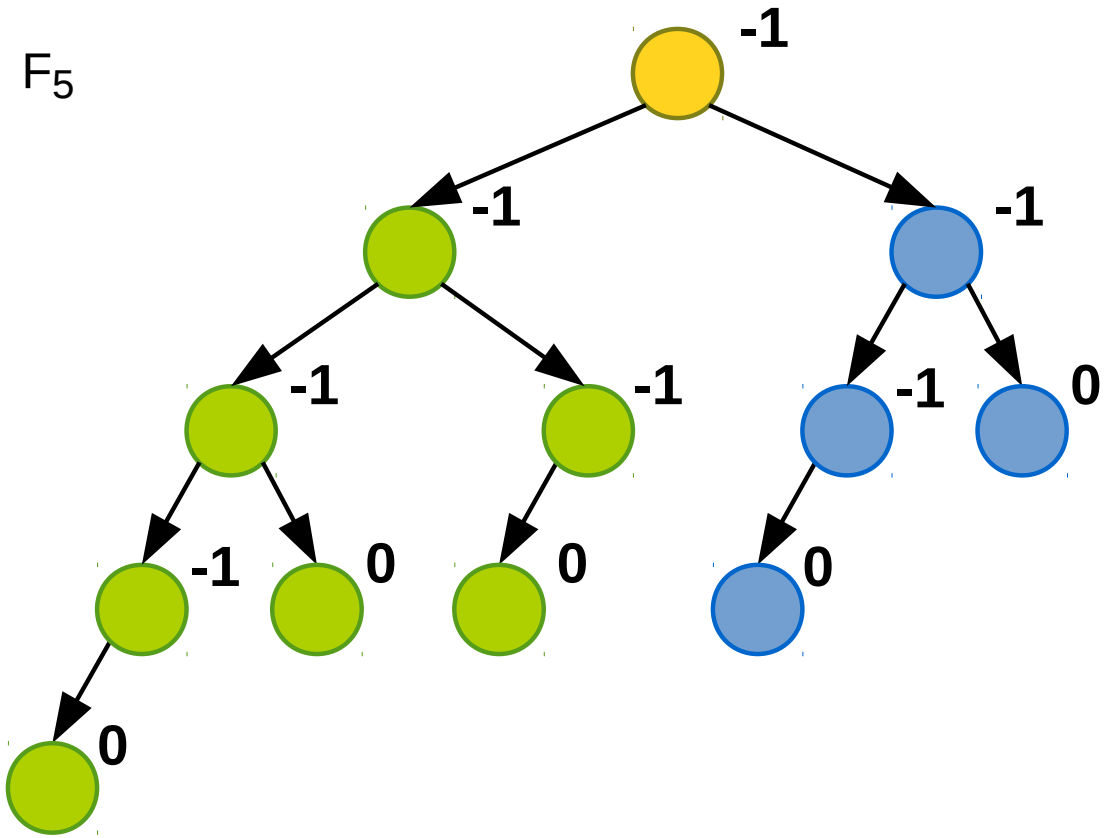
F_4



Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

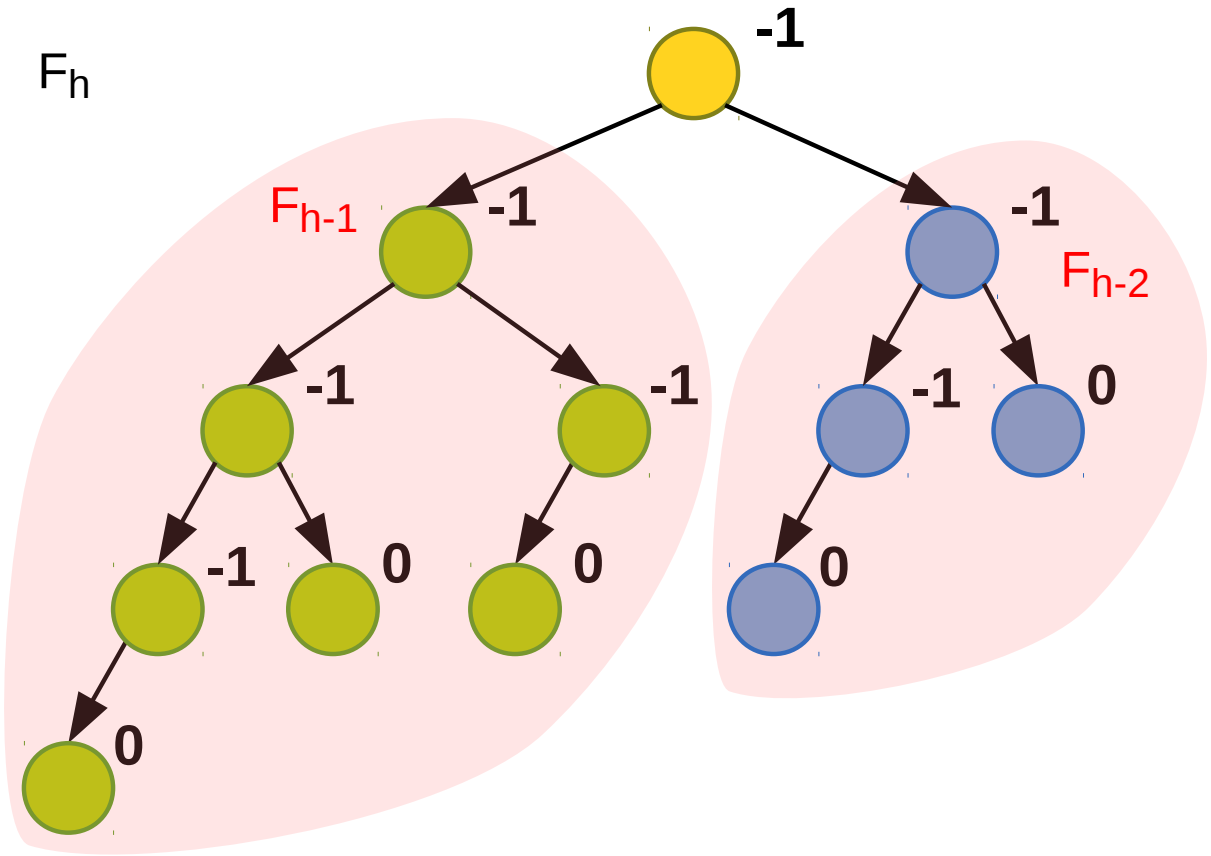
Vamos verificar: Qual a menor árvore com uma certa altura?



Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

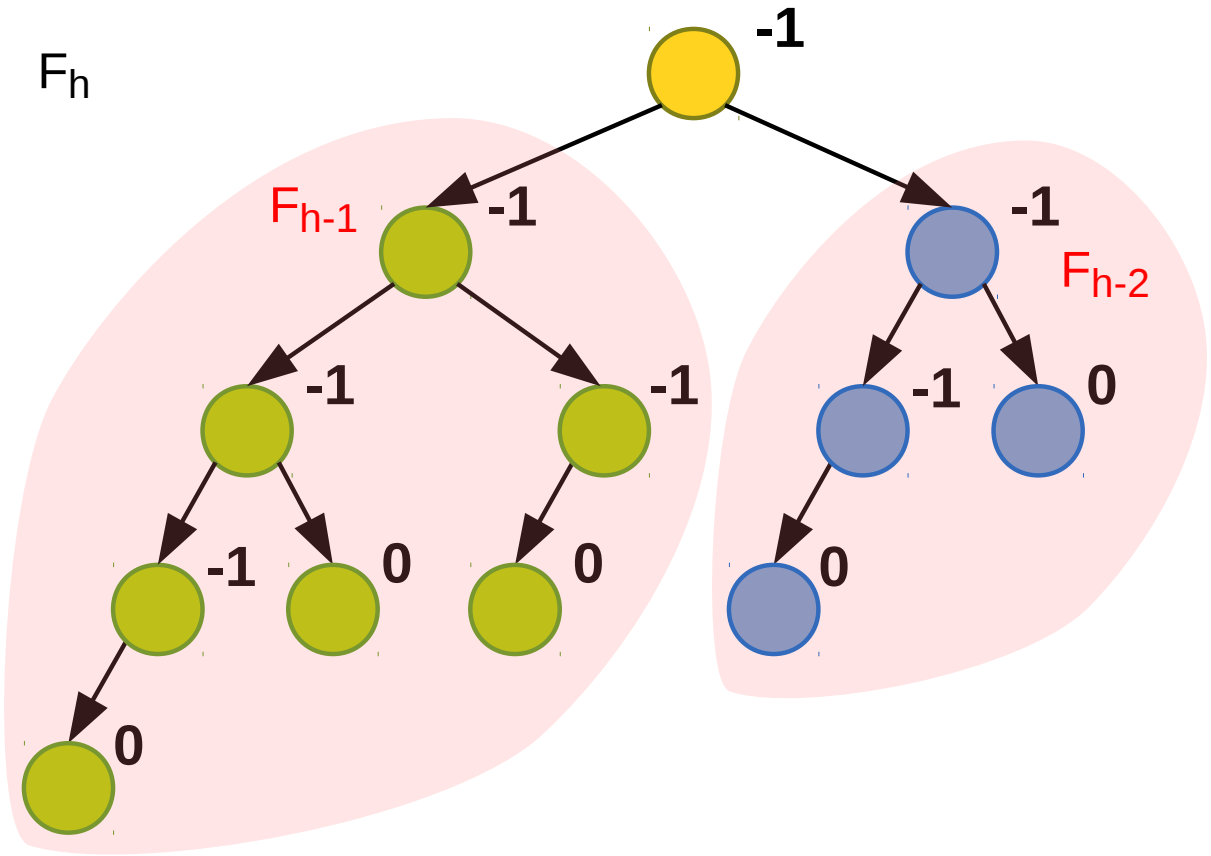
Vamos verificar: Qual a menor árvore com uma certa altura?



Exemplo: AVL, uma estrutura "garantidamente" boa

Uma busca sempre acessa no máximo $1.44 \log_2 n + C$ nós!

Vamos verificar: Qual a menor árvore com uma certa altura?



Exercício: mostrar o limitante acima.

Outro exemplo: *Treap*, não "garantidamente" boa

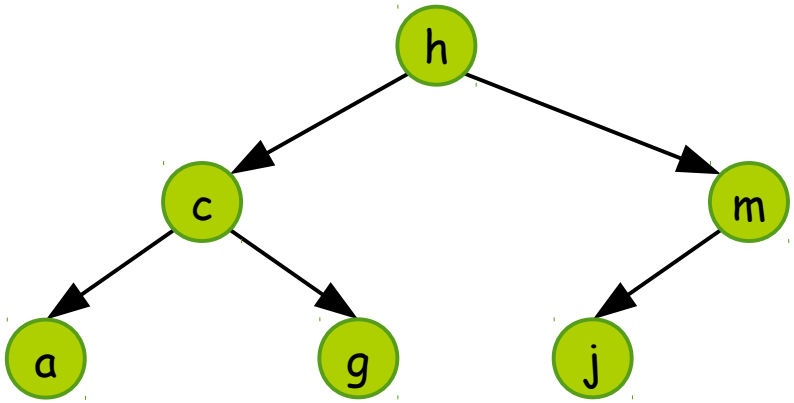
Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

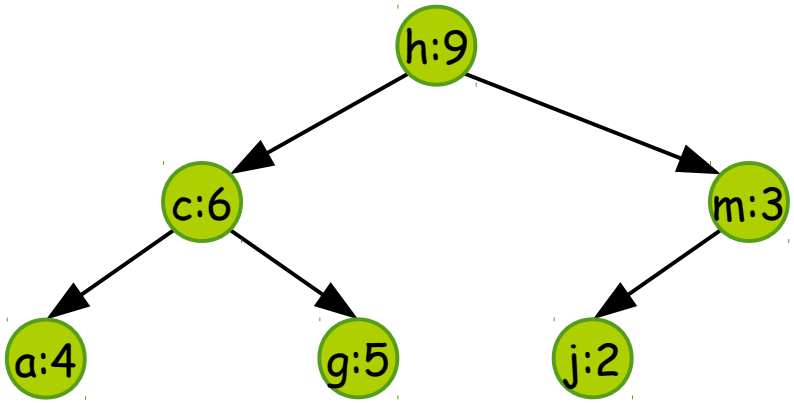
- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

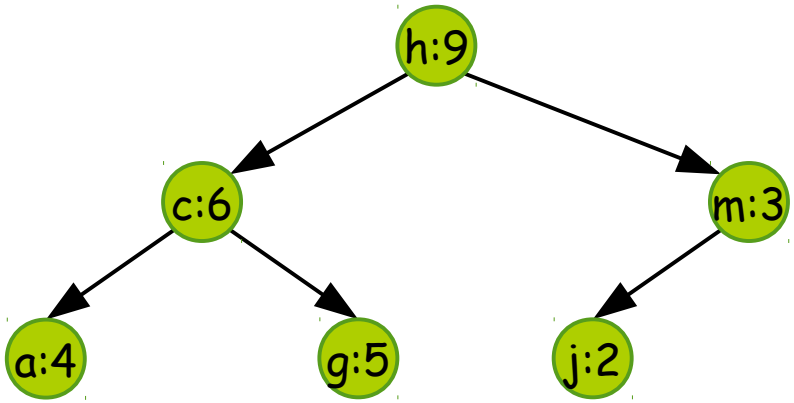
- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)

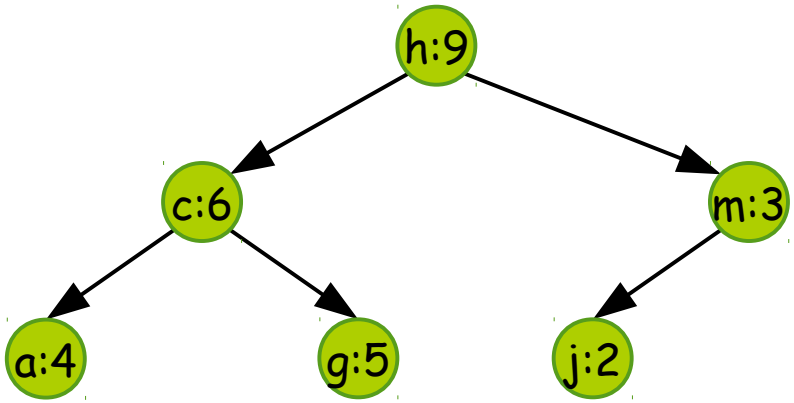


Propriedades

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



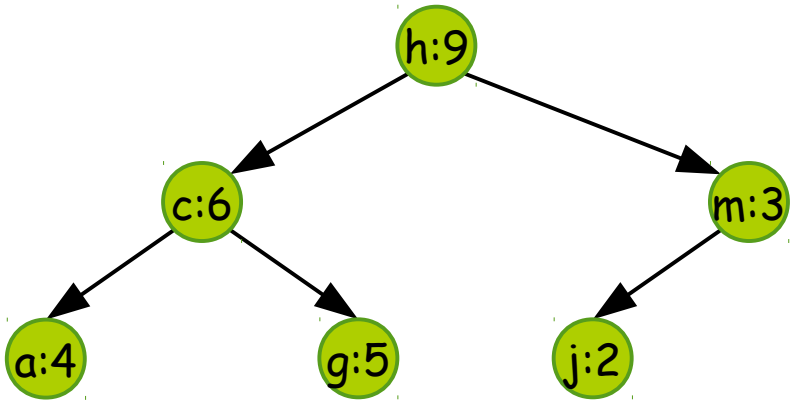
Propriedades

- árvore de busca comum

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



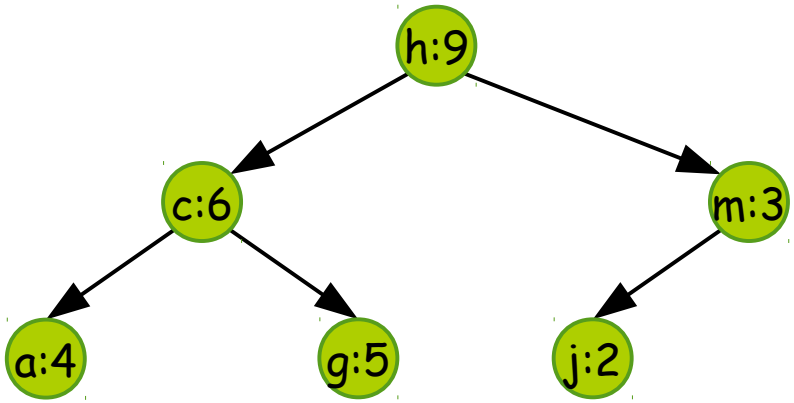
Propriedades

- árvore de busca comum
- cada nó tem uma prioridade aleatória

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



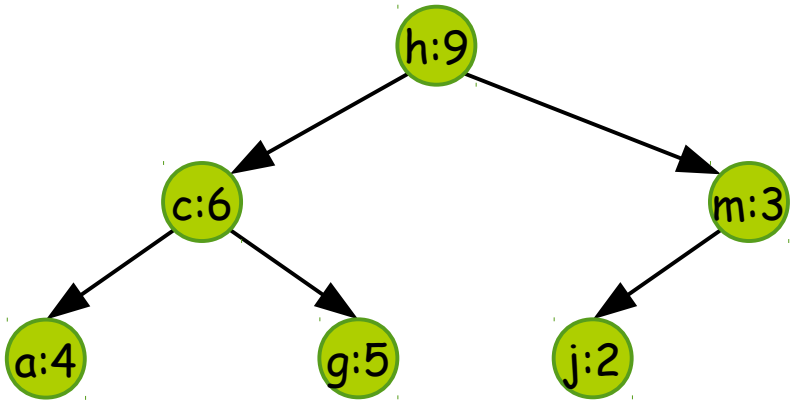
Propriedades

- árvore de busca comum
- cada nó tem uma prioridade aleatória
- os nós respeita a propriedade de *max-heap*

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)



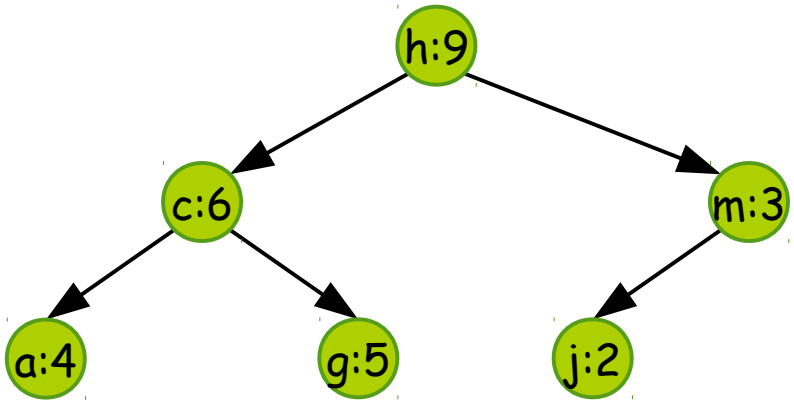
Propriedades

- árvore de busca comum
- cada nó tem uma prioridade aleatória
- os nós respeita a propriedade de *max-heap*
- garantia de pior caso: $O(n)$

Outro exemplo: *Treap*, não "garantidamente" boa

Mais uma árvore: *tree* + *heap*

- introduzida por Cecilia R. Aragon and Raimund Seidel (1989)

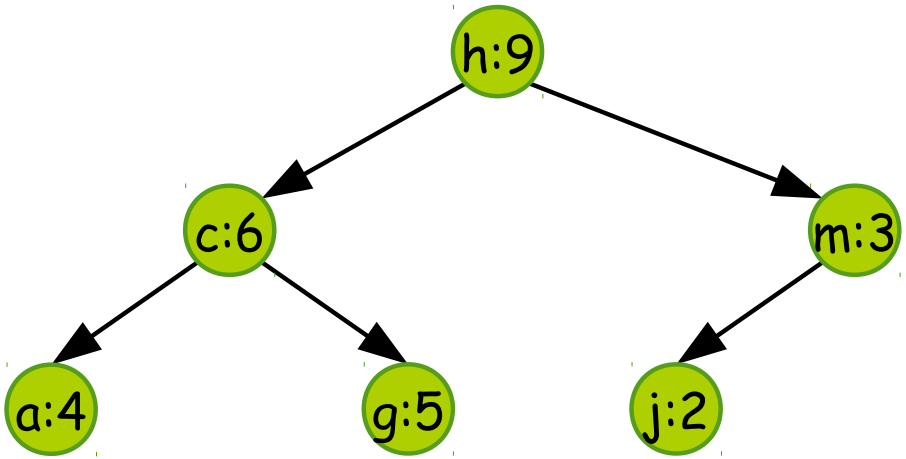


Propriedades

- árvore de busca comum
- cada nó tem uma prioridade aleatória
- os nós respeita a propriedade de *max-heap*
- garantia de pior caso: $O(n)$
- mas executa em $O(\log n)$ com alta prioridade

Inserindo em uma *Treap*

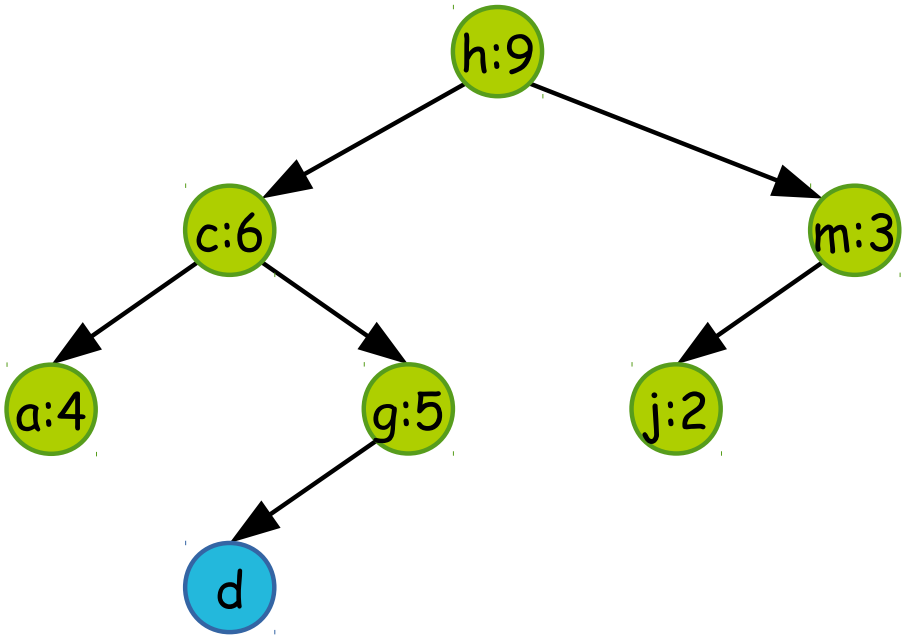
Algoritmo



Inserindo em uma *Treap*

Algoritmo

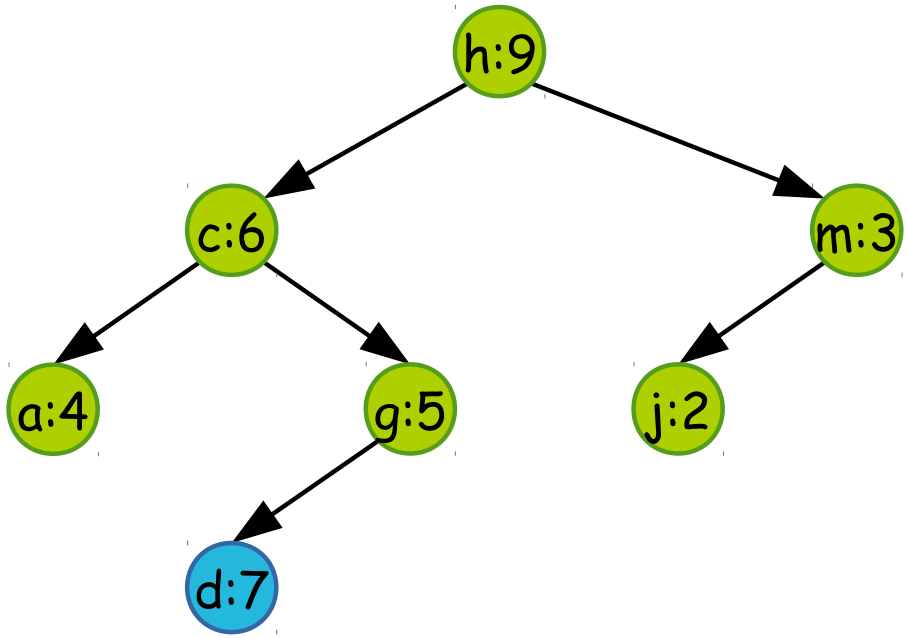
1. inserimos normalmente



Inserindo em uma *Treap*

Algoritmo

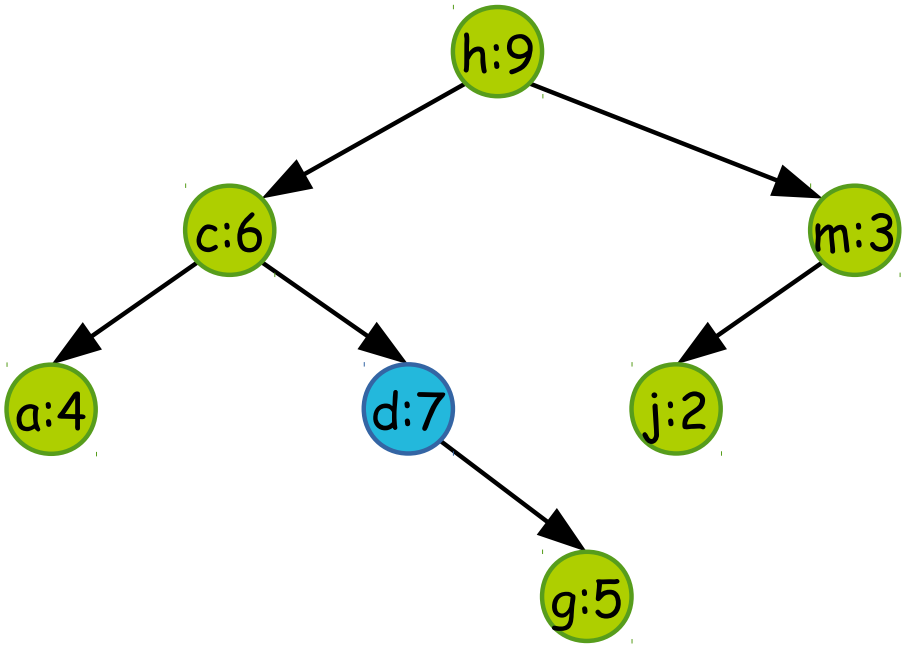
1. inserimos normalmente
2. adicionamos uma prioridade aleatória



Inserindo em uma *Treap*

Algoritmo

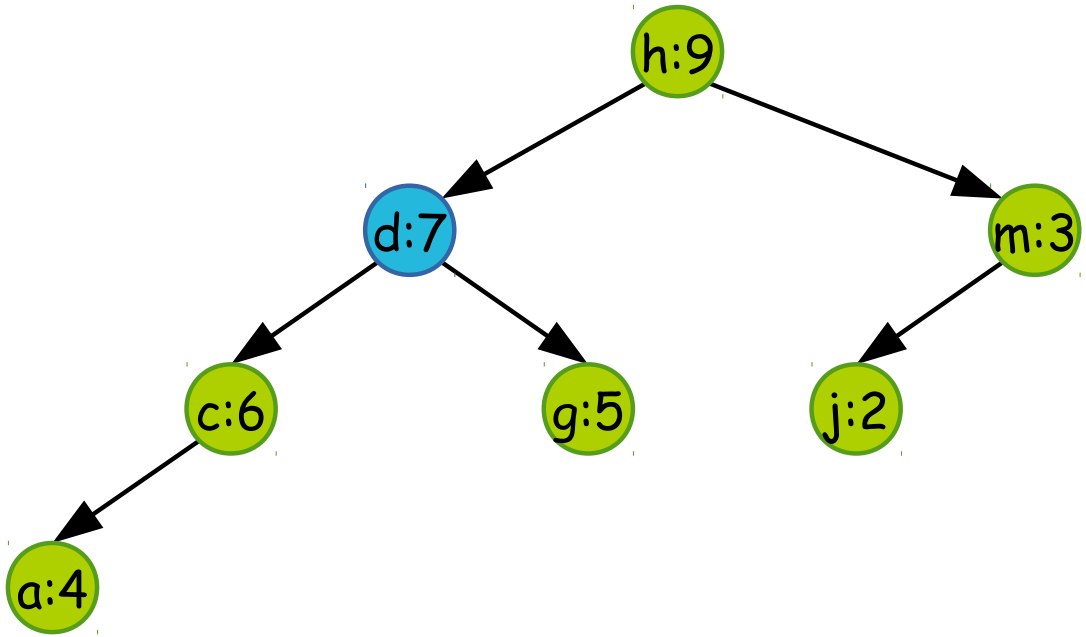
1. inserimos normalmente
2. adicionamos uma prioridade aleatória
3. subimos rotacionando



Inserindo em uma *Treap*

Algoritmo

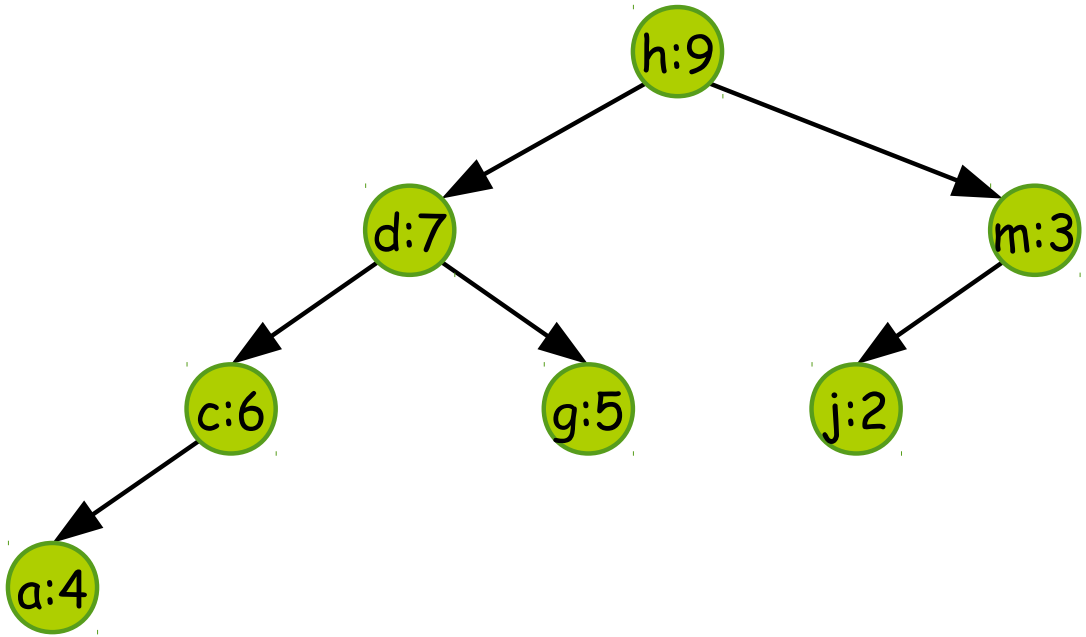
1. inserimos normalmente
2. adicionamos uma prioridade aleatória
3. subimos rotacionando



Inserindo em uma *Treap*

Algoritmo

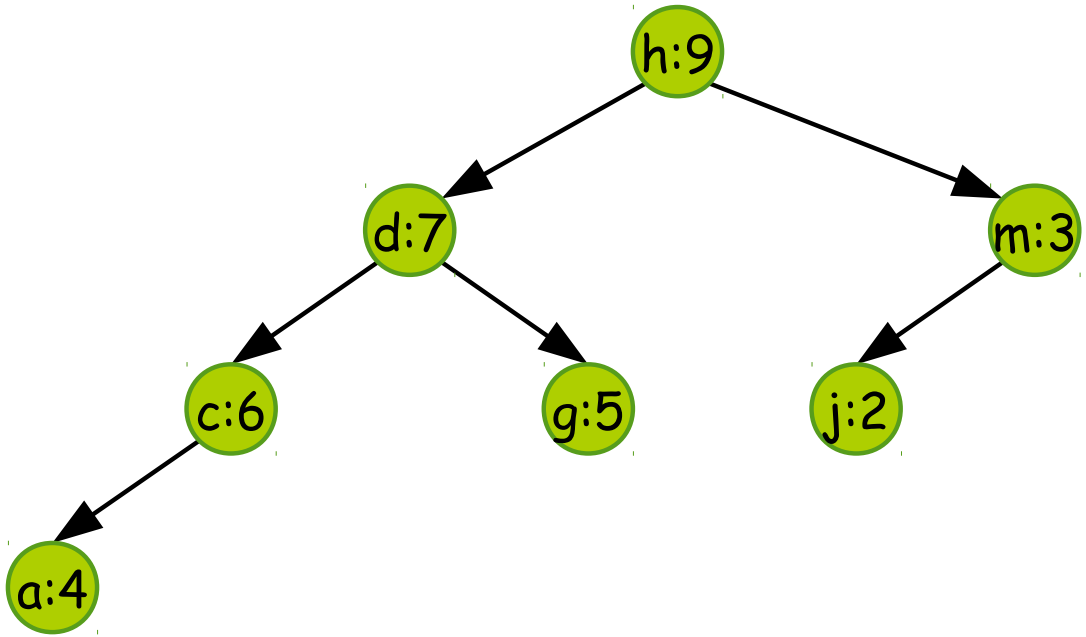
1. inserimos normalmente
2. adicionamos uma prioridade aleatória
3. subimos rotacionando



Inserindo em uma *Treap*

Algoritmo

1. inserimos normalmente
2. adicionamos uma prioridade aleatória
3. subimos rotacionando



Variante: aumentamos a prioridade dos elementos mais acessados

Então?

Então?

O que obtemos dos exemplos

- Algumas têm garantias sempre
- Outras são boas em situações específica
- Outras estruturas têm aplicações específicas
- ...

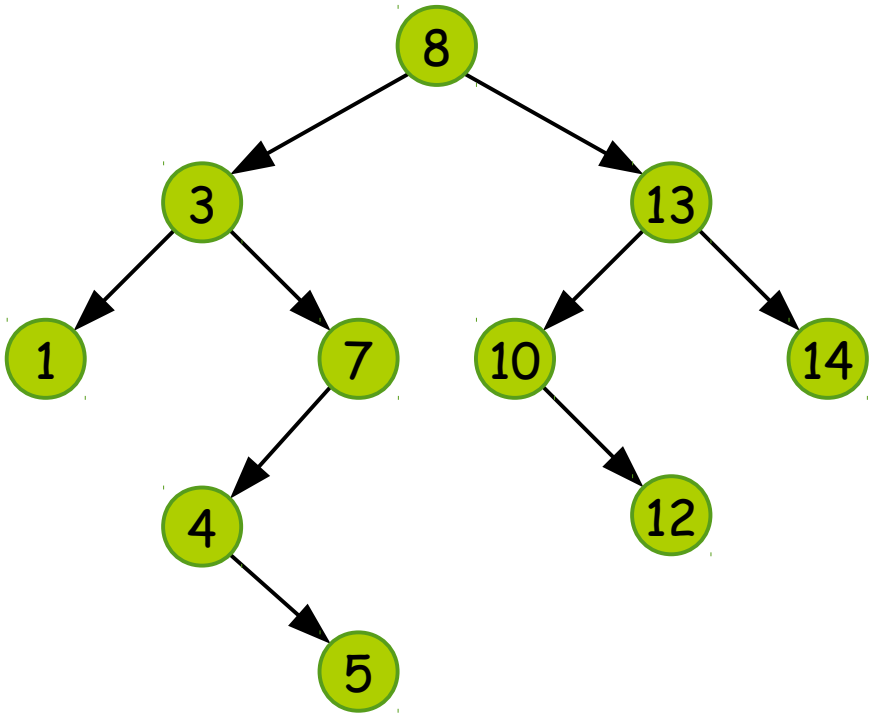
Então?

O que obtemos dos exemplos

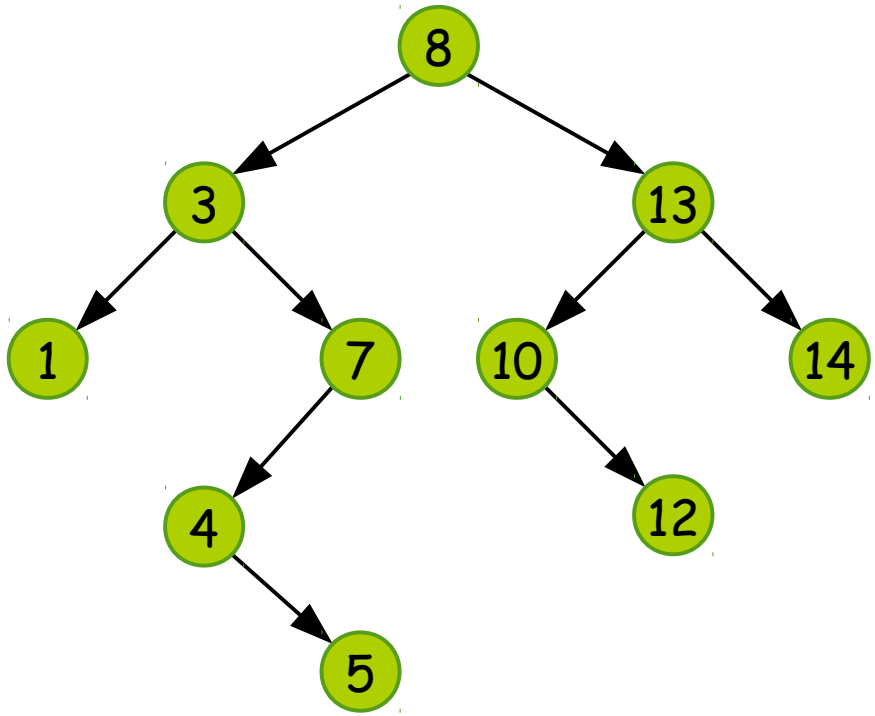
- Algumas têm garantias sempre
- Outras são boas em situações específica
- Outras estruturas têm aplicações específicas
- ...

Revisando...

Árvore de Busca Comum



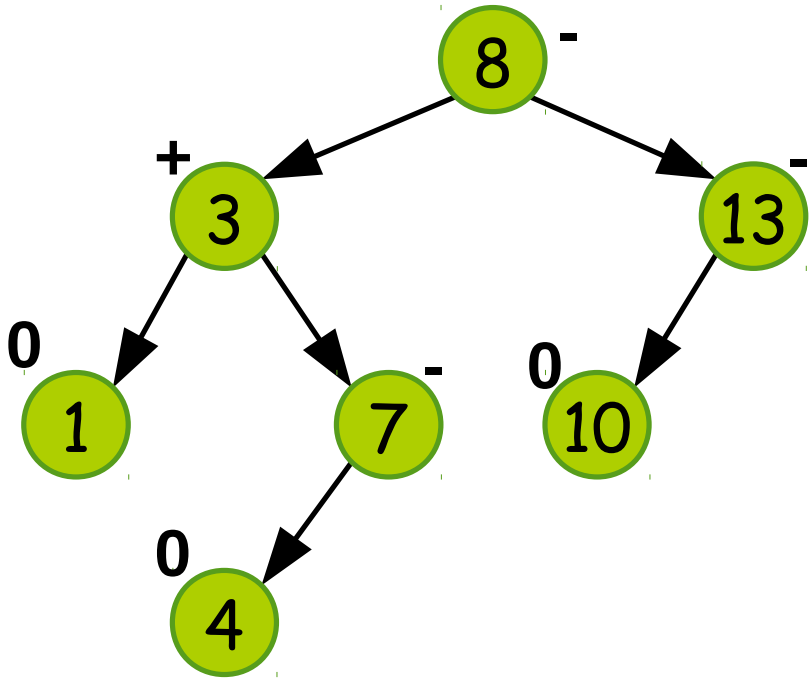
Árvore de Busca Comum



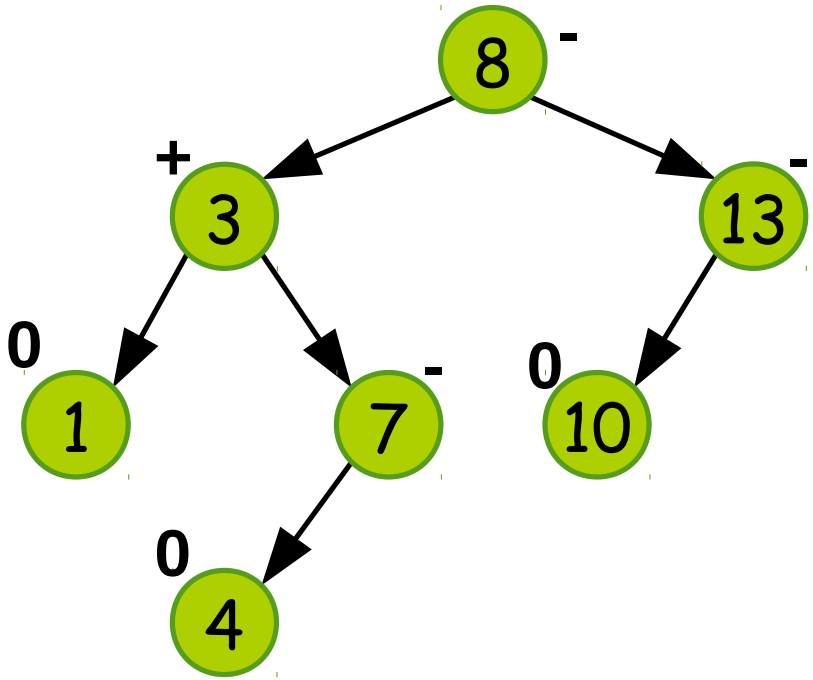
Objetivos: Organizar objetos "ordenáveis"

Problema: Desbalanceamento

Árvore Balanceada AVL



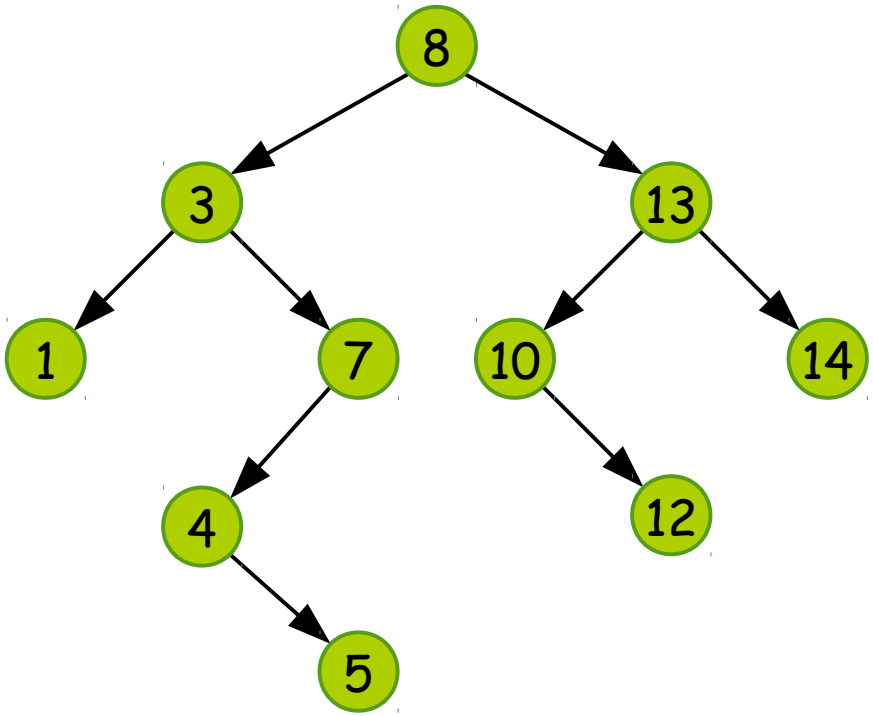
Árvore Balanceada AVL



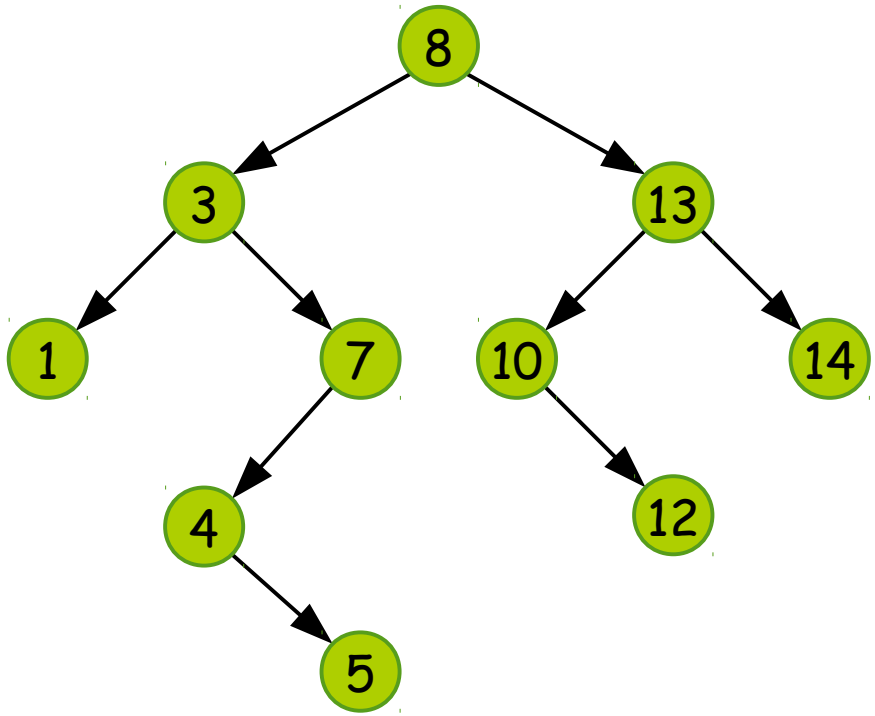
Objetivos: Garantir desbalanceamento

Problema: Muitas rotações

Árvore de Afunilamento



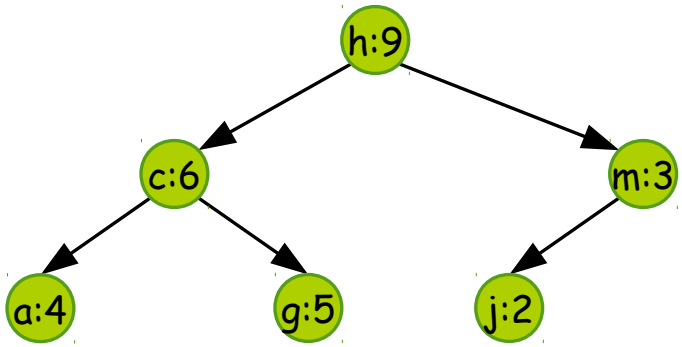
Árvore de Afunilamento



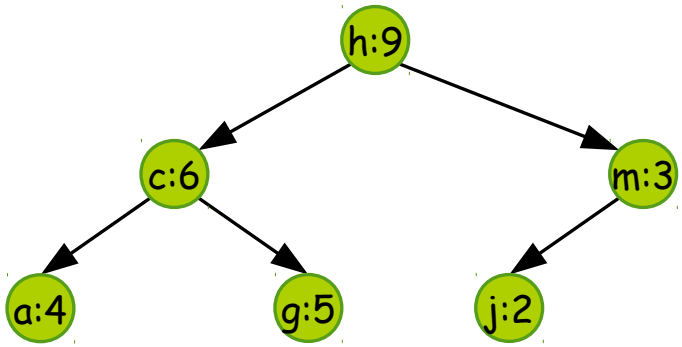
Objetivos: Aproveitar propriedade de referência e localidade

Problema: Pior caso

Árvore Treap



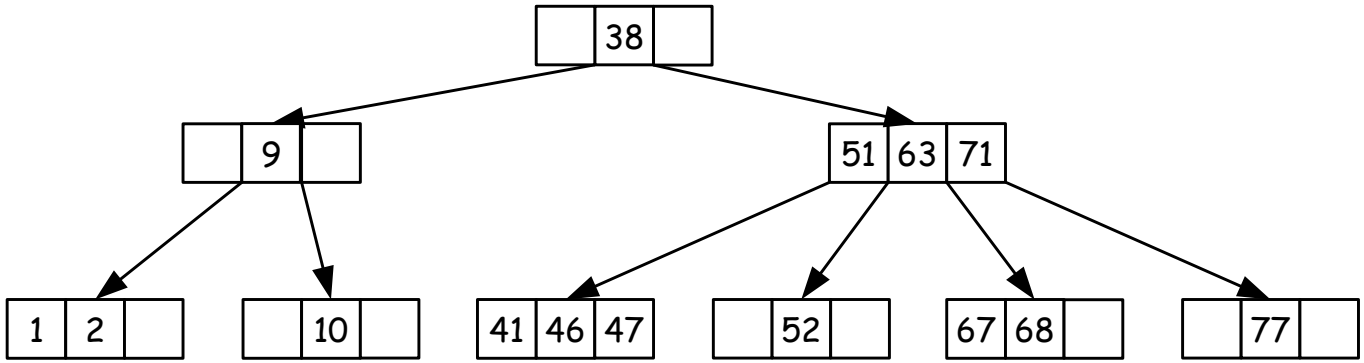
Árvore Treap



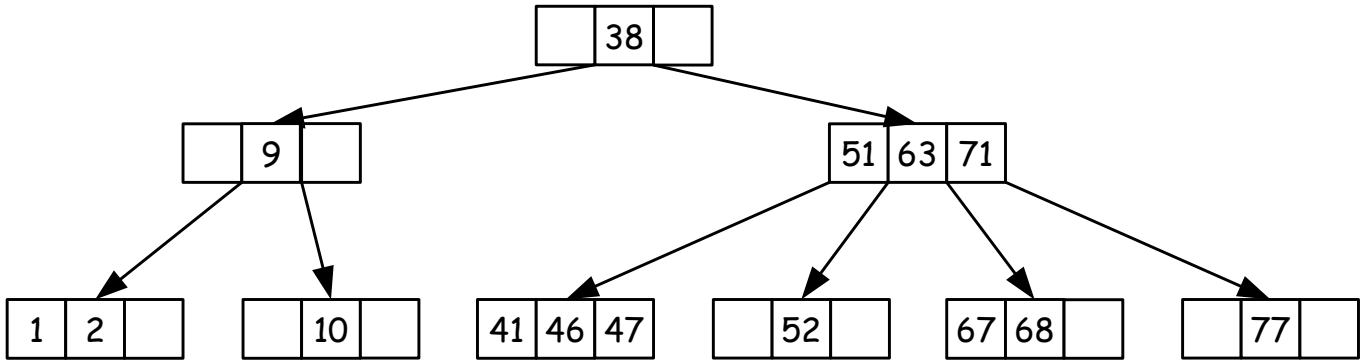
Objetivos: Balancear árvore

Problema: Pior caso

Árvore-B



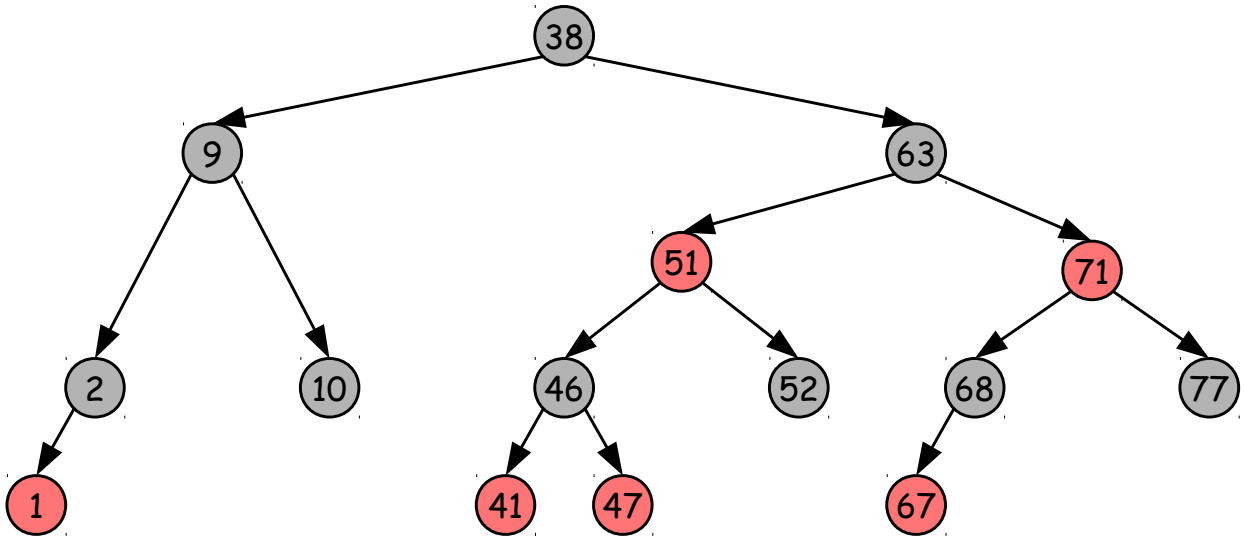
Árvore-B



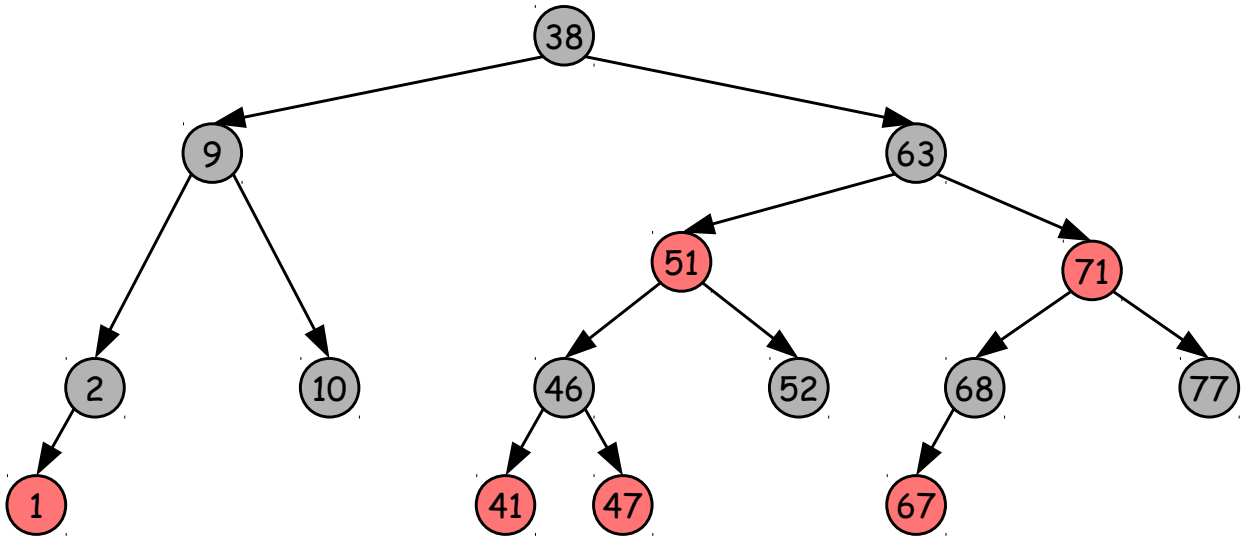
Objetivos: Altura da árvore "praticamente" constante

Observação: Para armazenamento externo

Árvore Rubro-Negra



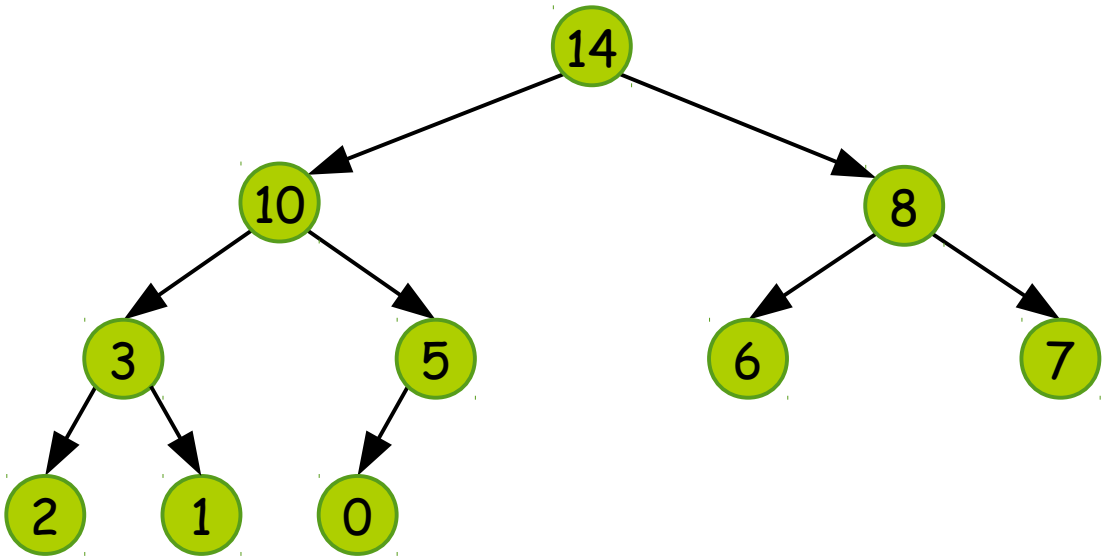
Árvore Rubro-Negra



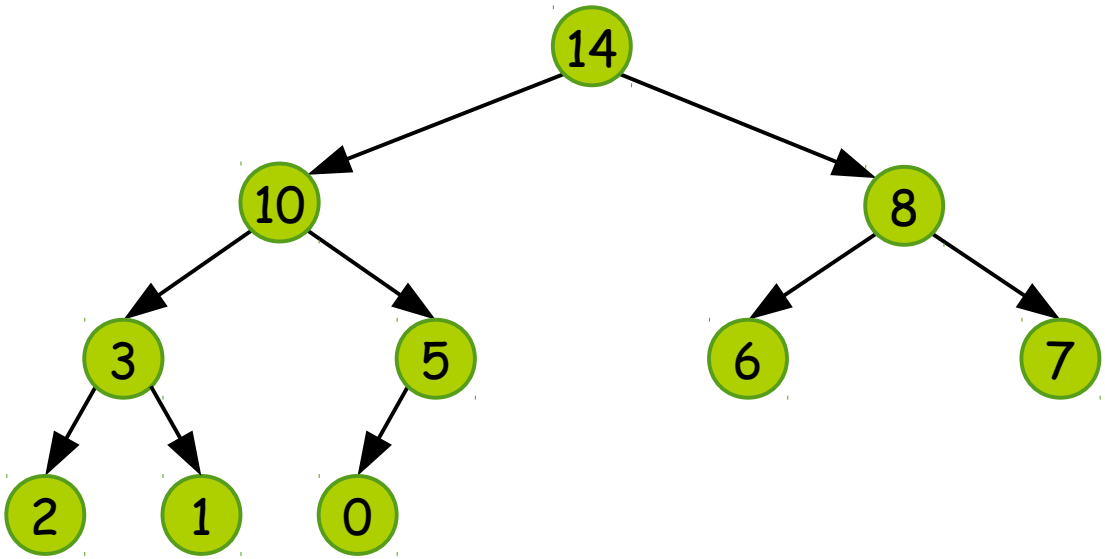
Objetivos: Balancear árvore binária

Problema: Altura em geral pior que AVL

Fila de Prioridade



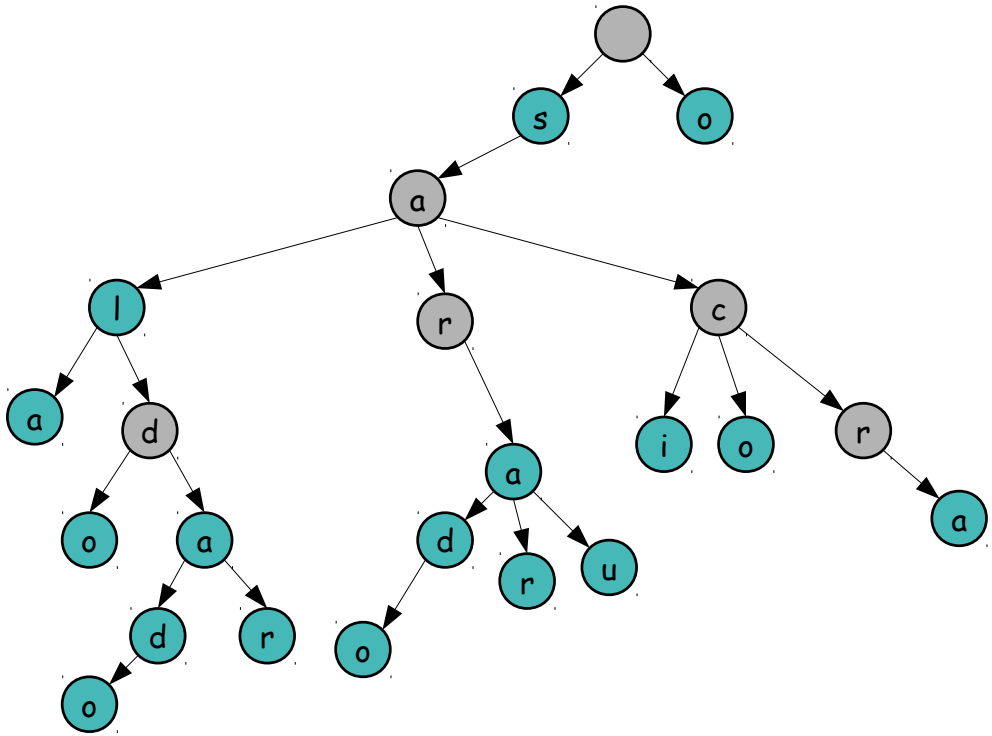
Fila de Prioridade



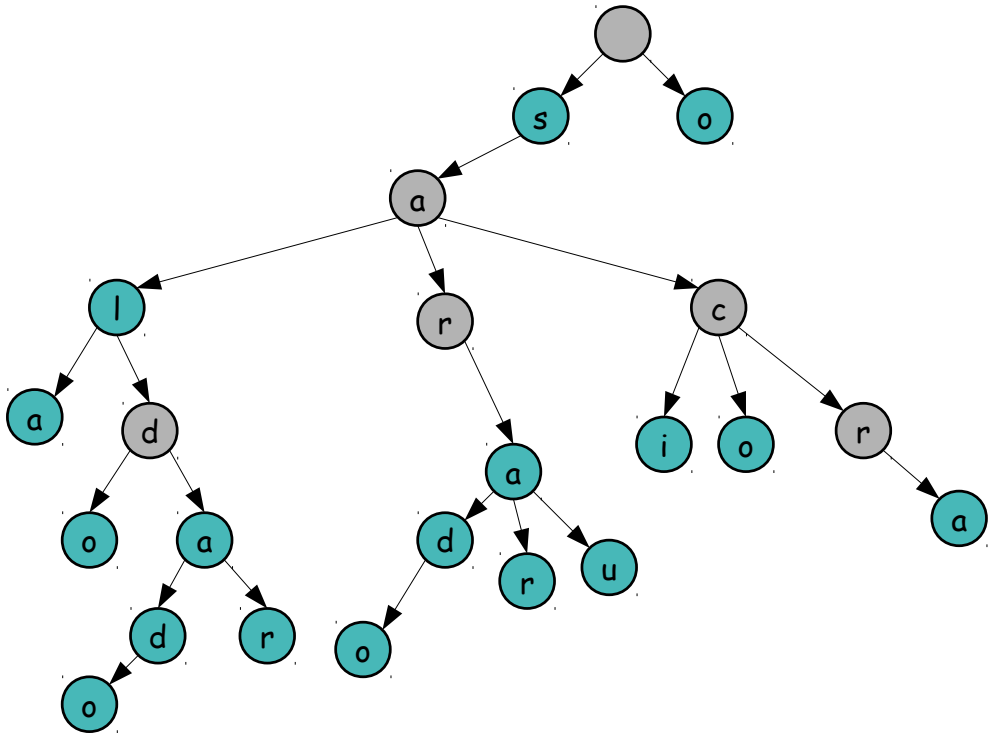
Objetivos: Encontrar o próximo maior rapidamente

Observação: Não serve para busca

Árvore de Prefixos



Árvore de Prefixos



Objetivos: Dicionário de palavras

Problema: Memória ocupada

Situação 1

Campeonato

Uma competição tem vários jogadores. Os jogadores são separados em níveis: série A, B, ..., Z (sendo A os de elite e Z os recém inscritos). A cada jogo, o jogador pode ganhar ou perder pontos. Em cada torneio, os primeiros de um grupo sobem de nível e os últimos descem. Qual a melhor estrutura de dados?

Situação 2

Estatísticas de universidade

Em uma universidade, os seus vários alunos são classificados pelo CR. Um aluno, que acha essa classificação injusta, fez a seguinte afirmação:

- meu CR é baixo, mas cursei 100% de disciplinas de “exatas”;
- os alunos de CR alto só cursam disciplinas de “humanas”.

Situação 2

Estatísticas de universidade

Em uma universidade, os seus vários alunos são classificados pelo CR. Um aluno, que acha essa classificação injusta, fez a seguinte afirmação:

- meu CR é baixo, mas cursei 100% de disciplinas de "exatas";
- os alunos de CR alto só cursam disciplinas de "humanas".

É claro que a afirmação do aluno é incoerente. Pra mostrar isso, você tem que criar uma estrutura de dados que, dados quaisquer pares a e b, retorne eficientemente o percentual de disciplinas de exatas cursadas por todos os alunos com CR entre a e b.

Exercícios em dupla

1. Compare *treaps* e *árvore de afunilamento*. Leia a seguinte afirmação e se concordar, explique porque é válida, ou, se não, diga porque ela não está correta: "Suponha que, em uma treap T, sempre que inserirmos um nó, ele tem a prioridade maior que todos os outros já inseridos. Se criarmos uma árvore de afunilamento S usando a mesma sequência de inserções, então T e S serão iguais."
2. Você têm o seguinte problema: "Em uma pesquisa, diversos dados são coletados. Cada dado é um número que corresponde a um vetor no espaço R^2 (isso é, é um par (x, y) de números). Uma vez coletados, é necessário realizar diversos experimentos. Escolha a estrutura de dados mais eficiente, justifique sua escolha (por que essa é adequada? e por que outras são menos apropriadas?) e escreva o algoritmo para as seguintes situações.
   - Se queremos realizar repetidas buscas pelo par (x, y) com maior valor de x.
   - Se quisermos iteradamente remover os pares de pontos mais distantes.