Aula extra 2
Filas de Prioridade e Heaps

Estruturas de Dados 2018/1
Prof. Diego Furtado Silva

Fila de prioridade

São estruturas que armazenam itens contendo:

- Chave que define a prioridade
- Outras informações

Apesar do nome “fila”, não é FIFO

- O elemento acessado é sempre o de **maior prioridade** (que pode ser maior ou menor chave de todas)

Fila de prioridade

Operações principais:

- insere(F,I): insere o elemento I = (chave, info) na fila de prioridade F
- remove(F): remove (e retorna) o elemento com maior prioridade em F
- proximo(F): retorna (sem remover) o elemento com maior prioridade em F

Pode/deve ter outras operações: conta(F), vazia(F), alteraPrioridade(F,chave), etc

Fila de prioridade

Como implementar com o que vimos até agora?

- Lista estática ordenada
- Lista estática não-ordenada

Fila de prioridade

Como implementar com o que vimos até agora?

- Lista estática ordenada
- Lista estática não-ordenada

- Lista dinâmica ordenada
- Lista dinâmica não-ordenada

Fila de prioridade

Como implementar com o que vimos até agora?

- Lista estática ordenada
- Lista estática não-ordenada

- Lista dinâmica ordenada
- Lista dinâmica não-ordenada

Outras implementações: árvore balanceada (ainda vamos ver isso) e ...

Heap

Comumente traduzido como “pilha”

- Mas essa é uma péssima tradução para ED
- Melhor seria algo como “amontoado”

Money stack vs money heap

Heap

Heap

Heap

A tendência é o maior grão fique no topo

Não importa muito como estão os grãos abaixo disso

E o que isso tem a ver com árvores binárias?

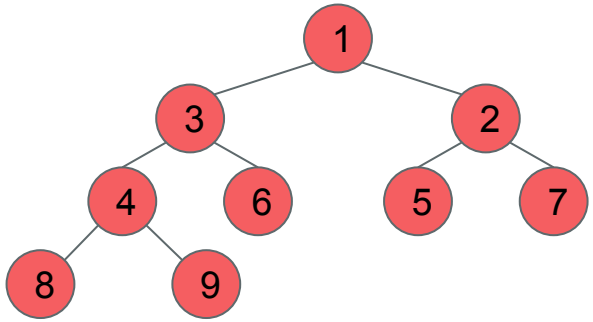
Heap

Uma heap é uma AB que respeita as seguintes propriedades:

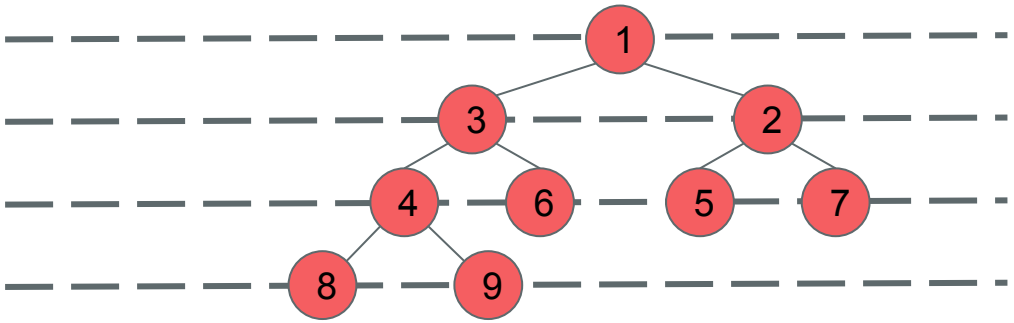
- **Ordem:** para cada nó *v*, exceto o nó raiz, temos que:
  chave(*v*) ≥ chave(pai(*v*)) - pode ser o contrário?

- **Completude:** uma AB é completa se (considerando altura *h*):
  * para *i = 0, …, h-1* existem 2^i nós de profundidade *i*
  * na altura *h*, os nós existentes estão à esquerda dos ausentes

Heap



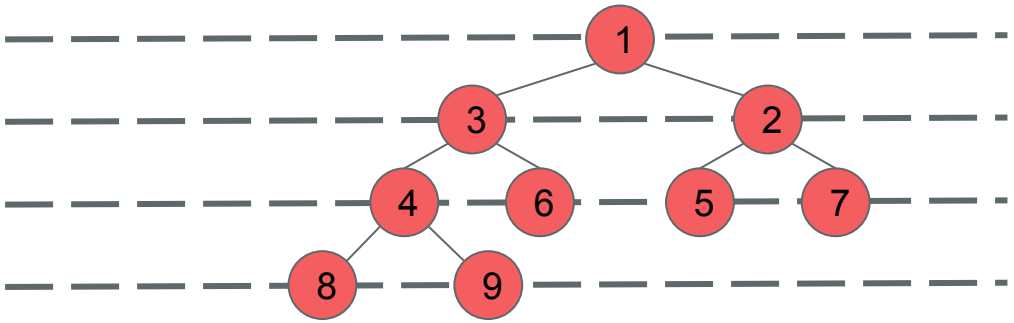
Heap



Profundidade	# chaves
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	2 (até 2^3)

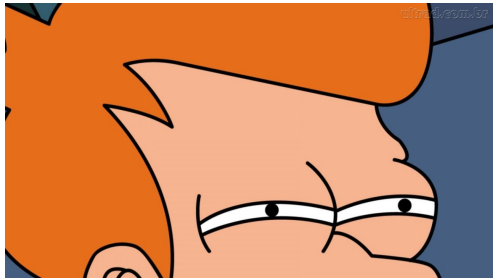
Uma heap armazenando *n* chaves possui altura *h* de ordem **O**($\log_2 n$)

Heap



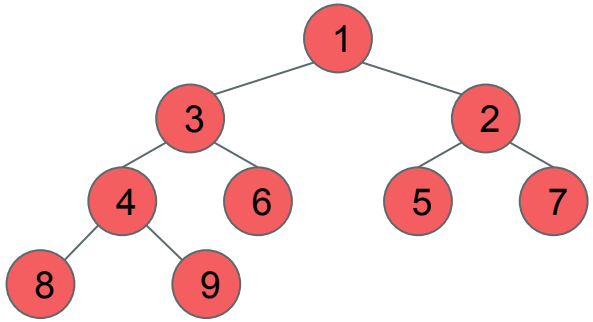
Profundidade	# chaves
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	2 (até 2^3)

Uma heap armazenando *n* chaves possui altura *h* de ordem **O**($\log_2 \boldsymbol{n}$)



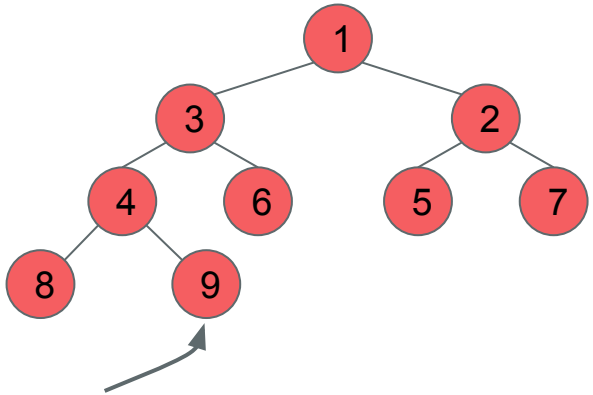
Ok, vamos provar isso

Heap



Pequena convenção: apesar de estar mostrando a chave direto dentro de cada nó, pode/deve haver um item com, ao menos, uma informação adicional à chave

Heap



Pequena convenção 2 : **último nó** é o mais à direita com profundidade *h*

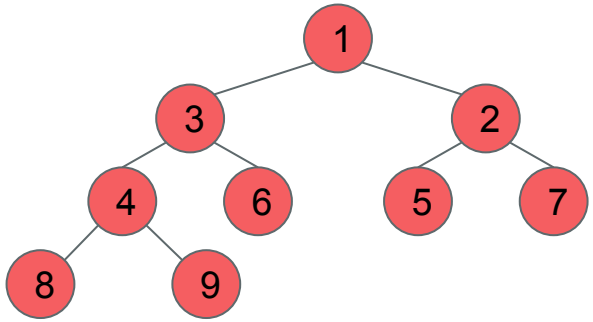
Fila de prioridade com heap

- Armazenamos a chave em cada nó e mantemos o controle sobre a **localização do último nó** (*w*) e, possivelmente, a onde será a próxima inserção (*z*)

- O **próximo item** (maior prioridade) estará **sempre na raiz**
  - **Maxheap** vs. **minheap**: vamos focar no *minheap* por facilidade

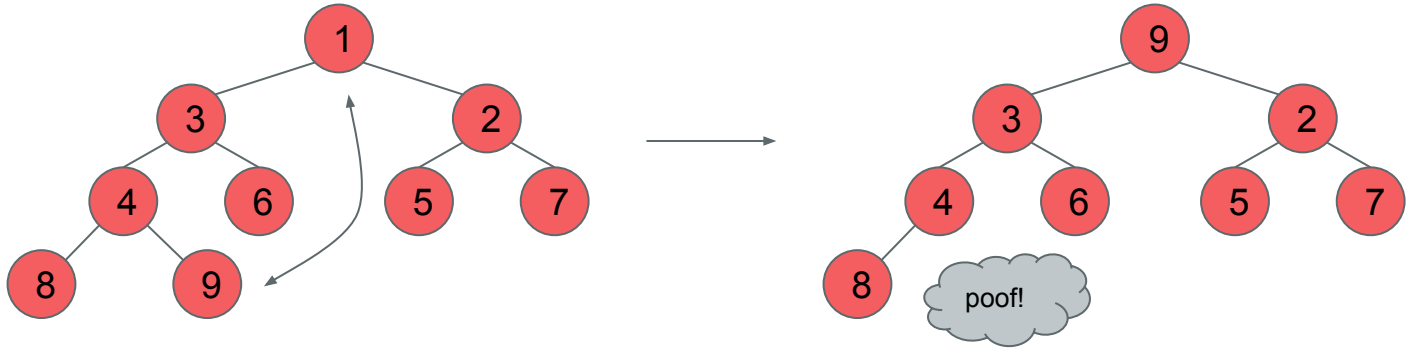
- Assim, remoção é sempre na raiz

Heap - remoção



Remove o 1... e depois?

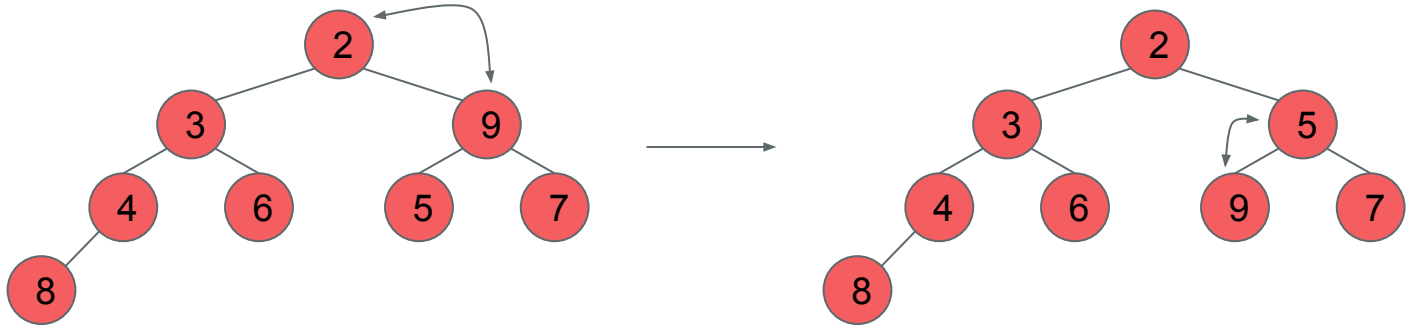
Heap - remoção



Primeiro, substituímos o conteúdo da raiz pelo conteúdo do último nó.
Então removemos o último.

E as propriedades da *heap*?

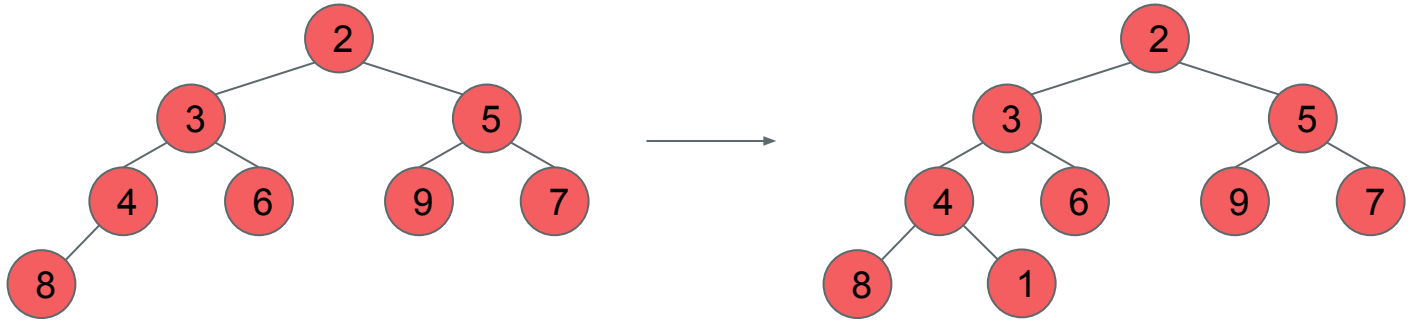
Heap - bubble-down



Trocamos a raiz com seu filho de menor valor. Recursivamente!

Exercício: remover todos os elementos da *heap* acima

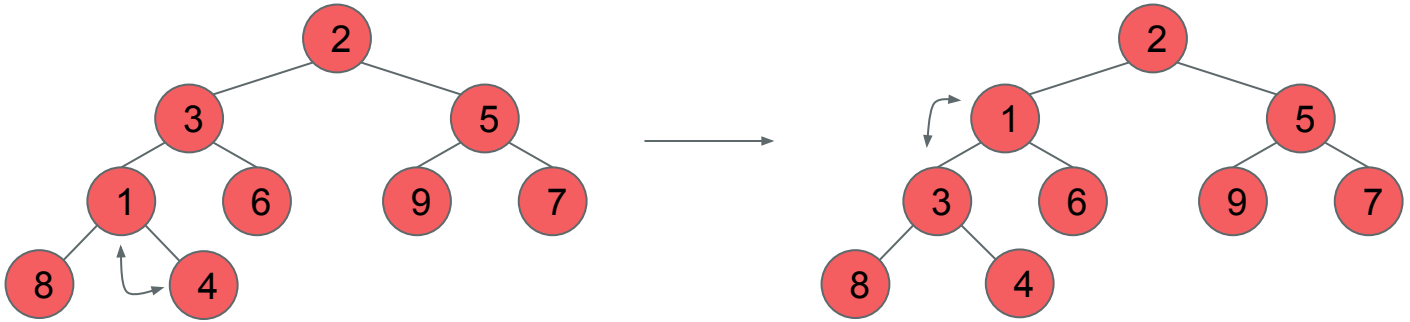
Heap - inserção



Encontra a posição a inserir (*z*) e atribui o item a ela.

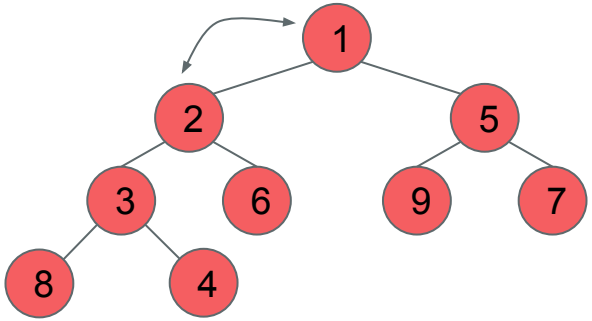
E as propriedades da *heap*?

Heap - bubble-up



Troca-se o item pelo seu pai até a raiz (ou até garantir a propriedade da heap)

Heap - bubble-up



Troca-se o item pelo seu pai até a raiz (ou até garantir a propriedade da heap)

Exercício: crie uma *heap* a partir das inserções de chaves 1, 9, 3, 5, 7, 8, 2, 6, 4

Heap - complexidade

A complexidade das operações ***bubble-up*** e ***bubble-down*** são diretamente relacionadas à altura da árvore, ou seja, **O(log*n*)**

Há outra etapa da inserção ou remoção que “domina” a complexidade?

Heap - implementação

Quero ideias!

Vocês falam e eu coloco no quadro

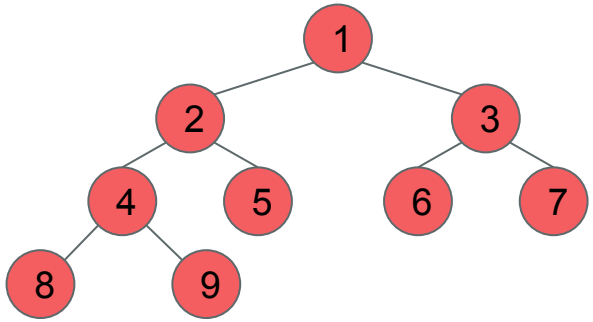
Heap - implementação com arranjos

Enganei vocês (?)

Heap - implementação com arranjos

Essa é a implementação mais comum.

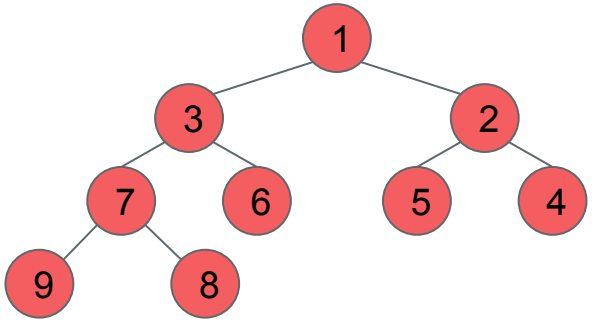
Há uma maneira bem interessante de se implementar AB usando arranjos.
Pense em como indexar cada nó de uma AB completa e ligar pais com filhos.



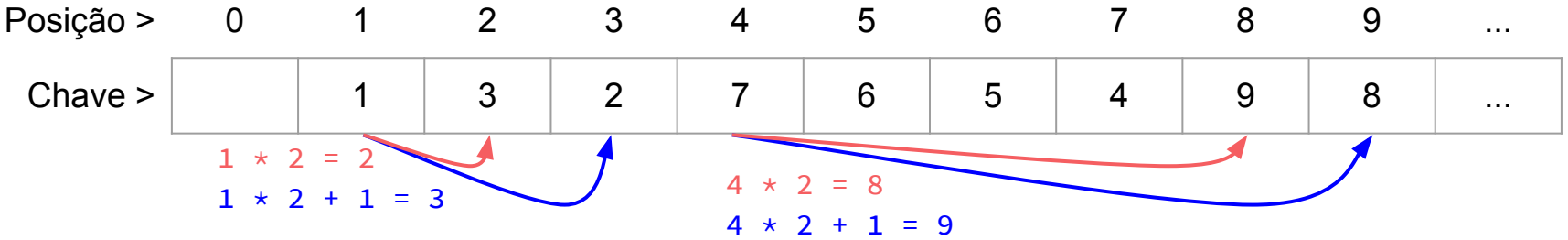
Qual é a relação do número 4 com 8 e 9?

E do número 3 com 6 e 7?

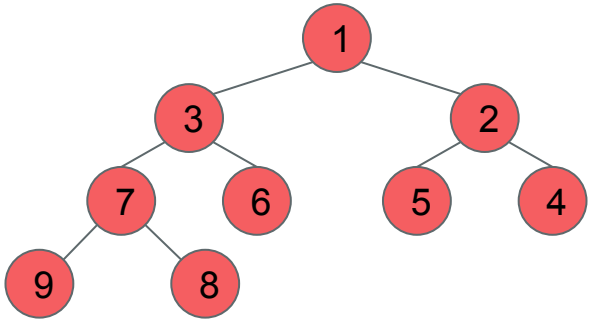
Heap - implementação estática



Posições dos filhos:
- Esquerda: pos*2
- Direita: pos*2 + 1



Heap - implementação com arranjos



Posição do pai:
floor(pos/2)

Posição >	0	1	2	3	4	5	6	7	8	9	...
Chave >		1	3	2	7	6	5	4	9	8	...

2/2 = 1

9/2 = 4.5
floor(4.5) = 4

Heap - implementação com arranjos

Vamos implementar

Filas de prioridade - Complexidade

Operação	**Tempo**		
	Lista não ordenada	Lista ordenada	Heap
próximo	O(n)	O(1)	O(1)
insere	O(1)	O(n)	O(log n)
remove	O(n)	O(1)	O(log n)

Filas de prioridade - Complexidade

Operação	Tempo		
	Lista não ordenada	Lista ordenada	Heap
próximo	O(n)	O(1)	O(1)
insere	O(1)	O(n)	O(log n)
remove	O(n)	O(1)	O(log n)

E o que aconteceria se usarmos uma implementação de AB dinâmica?

Filas de prioridade - aplicações

Muito usada em **algoritmos gulosos**

- Paradigma de resolução de problemas bastante comum
- Aplicações em grafos: caminhos mínimos (Djikstra), AGM (Prim / Kruskal)
- Soluções aproximadas para problemas complexos
- ...

Tudo isso (e mais) em PAA :]

Agora: **ordenação**

Filas de prioridade - Ordenação

Podemos ordenar elementos usando fila de prioridades seguindo:

1. Insira os elementos na fila um a um via uma série de **operações insere**

2. Retorne os elementos via uma série de **operações remove**

Complexidade depende da implementação da fila de prioridade

p.s. Construir uma heap em O(n): https://www.geeksforgeeks.org/time-complexity-of-building-a-heap/

Filas de prioridade - Ordenação

Usando uma *heap*, ordenamos ***n*** elementos em **O(n logn)**

- Muito mais rápido que algoritmos “elementares”, usualmente **O(n^2)**
  - Seleção, inserção, *bubble*, etc
  - Exceto para *n* bem pequenos

- Algoritmo apresentado chama-se ***heapsort***

Excelente visualização do algoritmo: https://visualgo.net/en/heap

Exercício

Ordene os seguintes valores usando o heapsort (ordenação com filas de prioridades)

2, 7, 26, 25, 19, 17, 1, 90, 3, 36

Filas de prioridade - Fim!