HEAP BINÁRIO

Algoritmos e Estrutura de Dados II

Prof. André Vignatti

ESTRUTURA DE DADOS

Estrutura de dados: forma de organizar dados para manipulação eficiente

- estrutura de dados fornece funções para manipulação (API)
  - ex: inserção, remoção, busca, ...
- objetivo: eficiência
  - sabedoria comum: não dá pra ser eficiente em todas as operações, por isso existem diversas estruturas de dados (adequada para diversas situações)

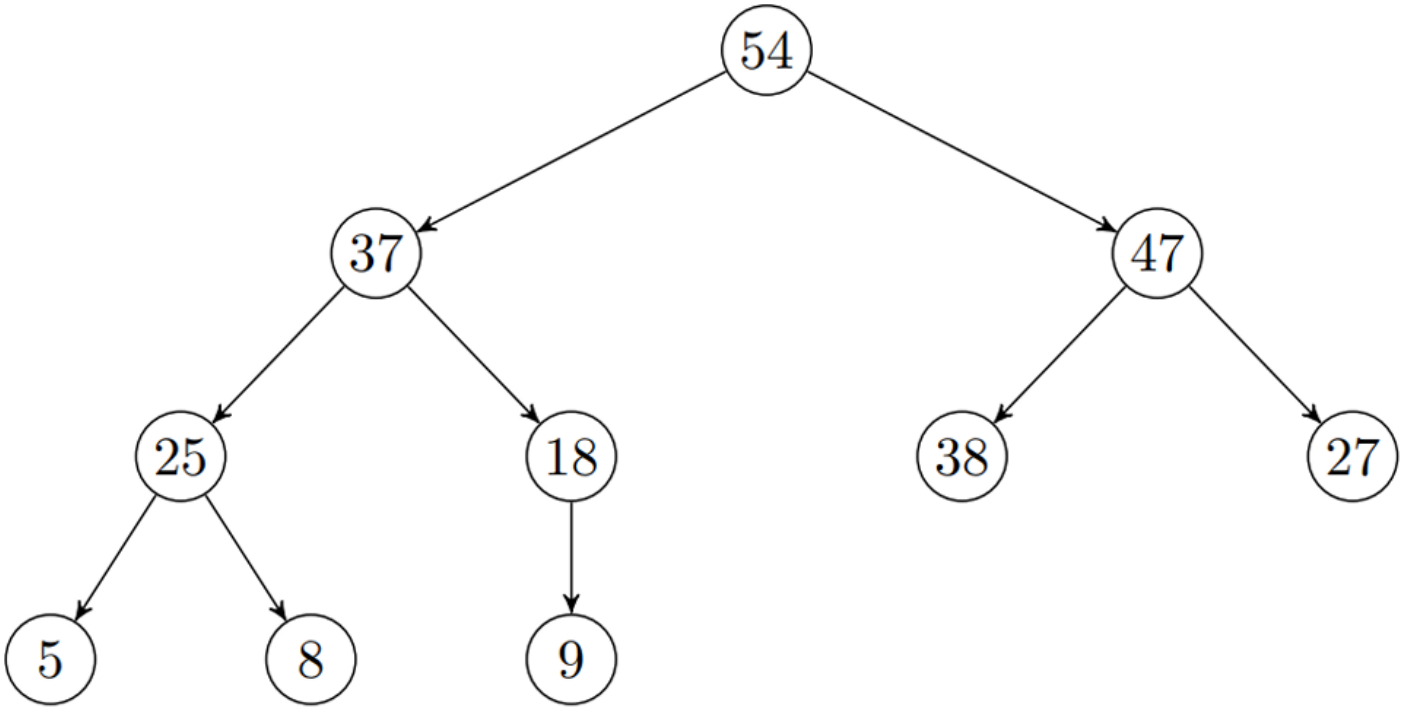
HEAP

Heap: estrutura de dados especializada na obtenção do maior (menor) elemento

- heaps estruturam os dados usando árvores
- há vários heaps diferentes, veremos o chamado **heap binário**
- heap binário usa árvores binárias quase completas

Propriedade de heap : $v[\text{pai}(i)] \geq v[i]$, para todo nodo i, exceto a raiz

Propriedade de heap : $v[\text{pai}(i)] \geq v[i]$, para todo nodo i, exceto a raiz



ARRUMANDO O HEAP

Problema: Arrumar Heap

Instância: (v, i, n), onde

- $v[1..n]$ é um vetor que representa uma árvore binária quase completa
- $1 \leq i \leq n$
- as sub-árvores enraizadas em direita(i) e esquerda(i) são heaps

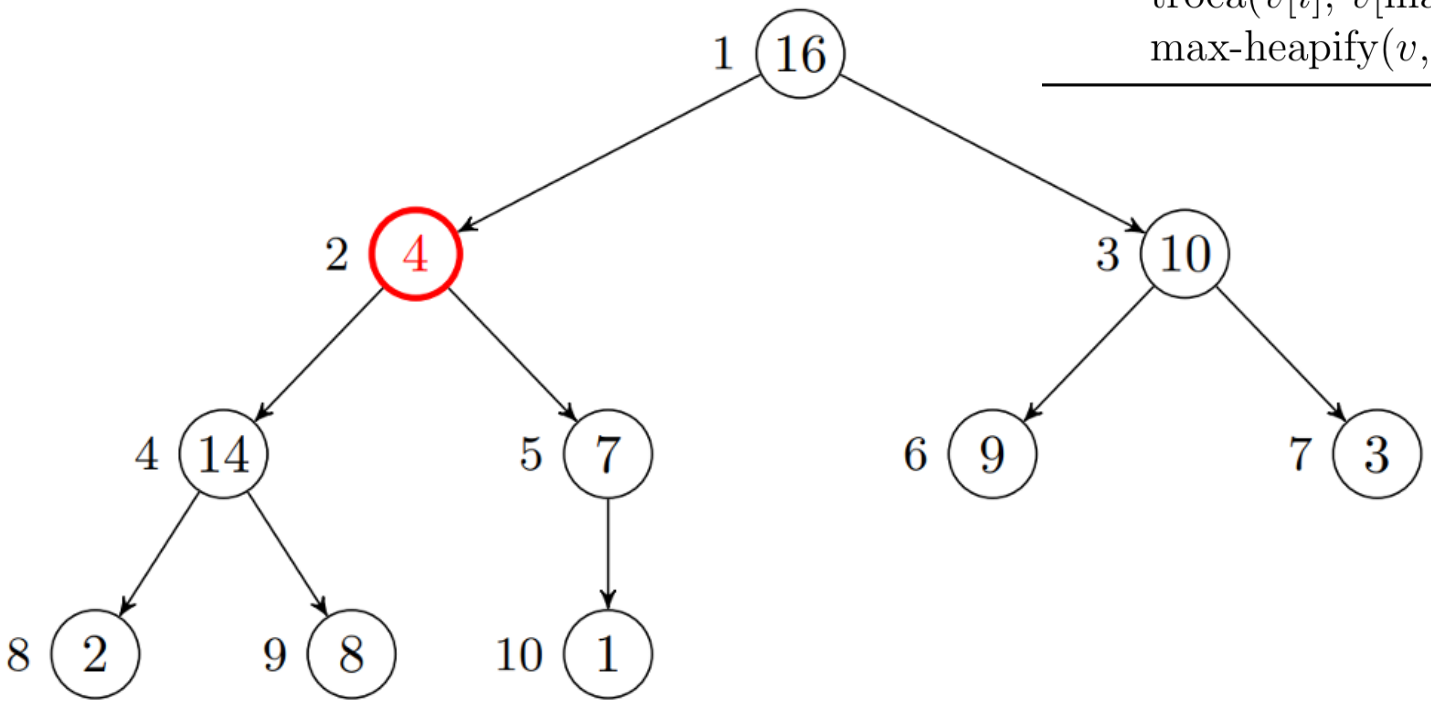
Resposta: sub-árvore enraizada em i é heap.

max-heapify(v, i, n)

esq $\leftarrow$ esquerda(i)
dir $\leftarrow$ direita(i)
se $esq \leq n$ **e** $v[esq] > v[i]$ **então**
    maior $\leftarrow$ esq
senão
    maior $\leftarrow i$
se $dir \leq n$ **e** $v[dir] > v[maior]$ **então**
    maior $\leftarrow$ dir
se $maior \neq i$ **então**
    troca$(v[i], v[\text{maior}])$
    max-heapify(v, maior, n)

Exemplo: executar max-heapify$(v, 2)$

```
max-heapify(v, i, n)
  esq ← esquerda(i)
  dir ← direita(i)
  se esq ≤ n e v[esq] > v[i] então
      maior ← esq
  senão
      maior ← i
  se dir ≤ n e v[dir] > v[maior] então
      maior ← dir
  se maior ≠ i então
      troca(v[i], v[maior])
      max-heapify(v, maior, n)
```



ANÁLISE DO PIOR CASO

No pior caso, sempre temos que maior $\neq i$, e as chamadas recursivas só terminam quando chegamos em uma folha.

$C(k)$: número de comparações com elementos do vetor para nó com altura k

$$C(k) = \begin{cases} 0, & \text{se } k = 0 \\ 2 + C(k-1), & \text{se } k > 0 \end{cases}$$

Resolvendo a recorrência, temos que $C(k) = 2k$

CONSTRUINDO UM HEAP

Problema: Construir Heap
Instância: (v, n) onde v é um vetor indexado de 1 a n. **Resposta:** os elementos de v reorganizados como heap.

build-max-heap(v, n)
para $i \leftarrow \lfloor n/2 \rfloor$ **até** 1 **faça** max-heapify(v, i)

build-max-heap(v, n)

para $i \leftarrow \lfloor n/2 \rfloor$ **até** 1 **faça**
max-heapify(v, i)

	1	2	3	4	5	6	7	8	9	10
v	15	35	40	10	20	5	45	70	25	90