INTRODUÇÃO

Marcus Ritt

CMP 601 – Algorithms and Theory of Computation — Ago 2021

Outline

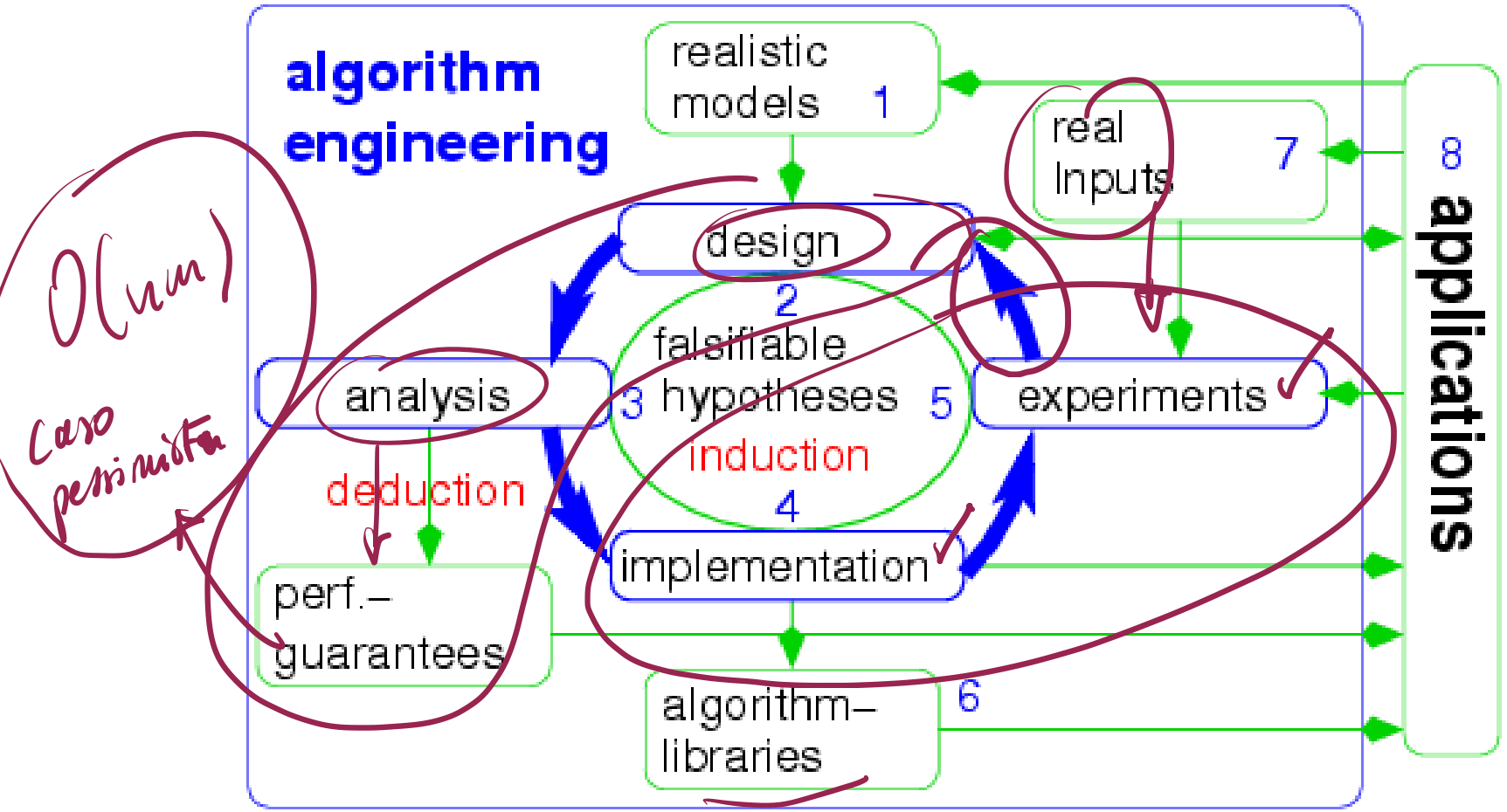
Filosofia: teoria e prática

Teoria

Seg : Algoritmos
Análise de compl.
Correctude

Prática

Qua: Laboratórios


algorithm engineering
realistic models 1
real inputs 7
8
applications
design
2
falsifiable hypotheses
3
5
analysis
experiments
induction
deduction
4
implementation
perf.– guarantees
algorithm– libraries
6
O(nm)
caso pessimista

$G = (V, A)$ $\quad G = (V(G), A(G))$ (Diestel, Graph theory)

grafo vértices arestas / arcos

Simples ( ) ; Loop

$n = |V|$
$m = |A|$

Não-dir.: $A \subseteq [V]^2$; $a = \{u, v\}$ $\quad uv = vu$

Dir. : $A \subseteq V^2$; $a = (u, v)$ $\quad uv \neq vu$

(v, v)

Não-dir. $\begin{cases} N(v) = \{u \mid \exists uv \in A\} & \text{vért. viz.} \\ N(v) = \{a \mid a \in A, a \cap v \neq \emptyset\} & \text{arestas incidentes} \end{cases}$

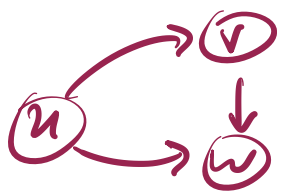
Dir. $\begin{cases} N^+(v) = \{a \mid a \in A, \exists w : a = (v, w)\} & \text{arcos sainte} \\ N^-(v) = \{a \mid a \in A, \exists u : a = (u, v)\} & \text{arcos entrantes} \end{cases}$

Não-dir. $d(v) = |N(v)|$

Dir. $d^+(v) = |N^+(v)|$

$d^-(v) = |N^-(v)|$

Ex



$d^+(u) = 2$
$d^-(u) = 0$

$V = \{u, v, w\}$, $A = \{(u, v), (v, v), (u, w)\}$

Grafos: notação

Estrutura de dados? $n = 234$ $(234)^2 = 400$

$[n] = \{1, 2, \dots, n\}$

$[n, m] = \{n, n+1, \dots, m\}$

Matriz de adjacência

$M = (m_{ij})_{i \in [n],\, j \in [n]}$

$$m_{ij} = \begin{cases} 1, & ij \in A \\ 0, & \text{c.c.} \end{cases}$$

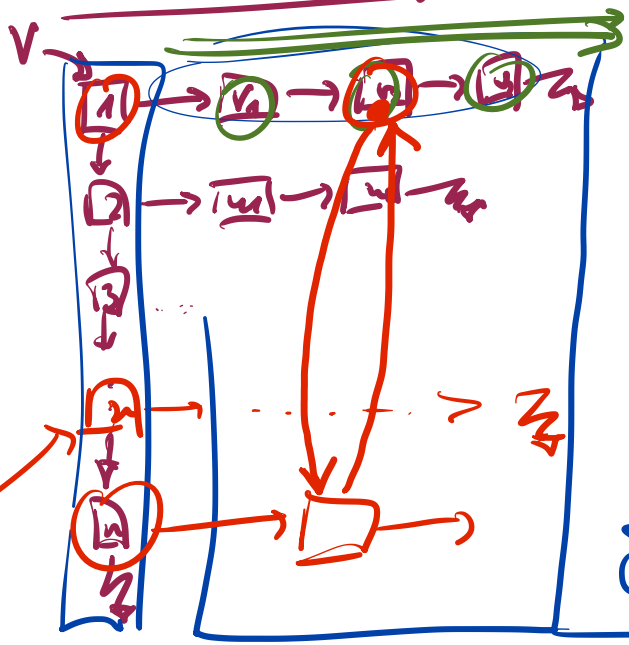
$O(1)$

Operações:
- inserir vértice / arestas / arcos
- remover
- testar $uv \in A$
- percorrer vértices / arestas / vizinhos
- grau de um vértice

$O(n^2)$

Mem.

Listas de adjacência



$O(n)$ $O(m)$

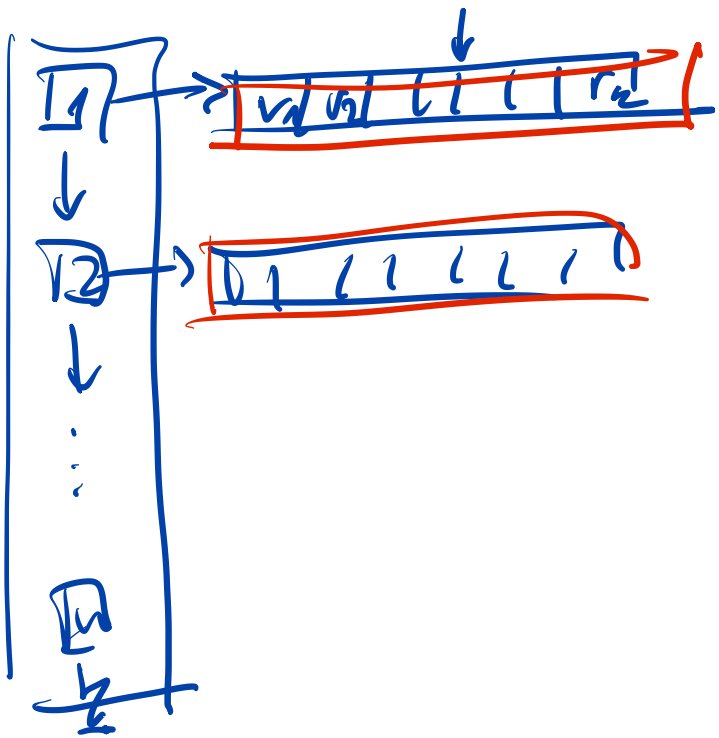
$O(d(v))$

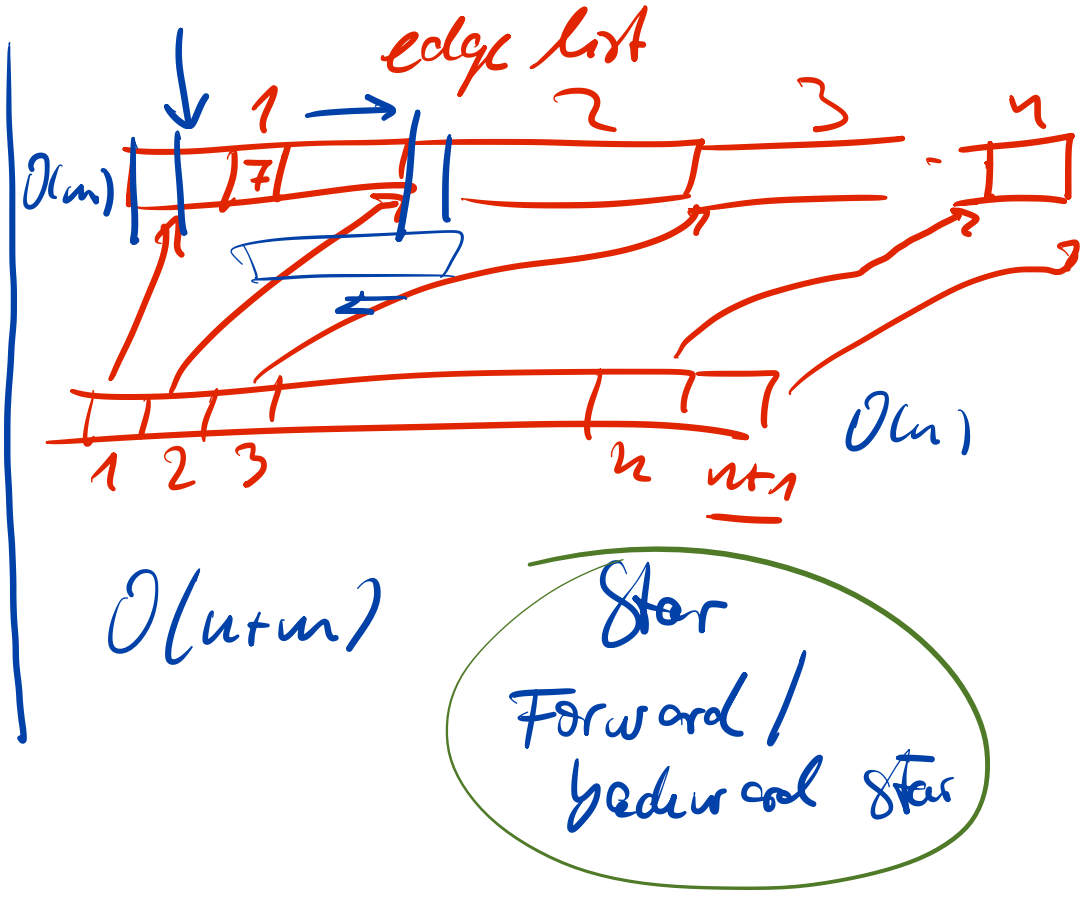
$\sum_{v \in V} \delta(v) = 2m$

$= O(m)$

Prática: $O(n+m)$ linear

Grafos: representação



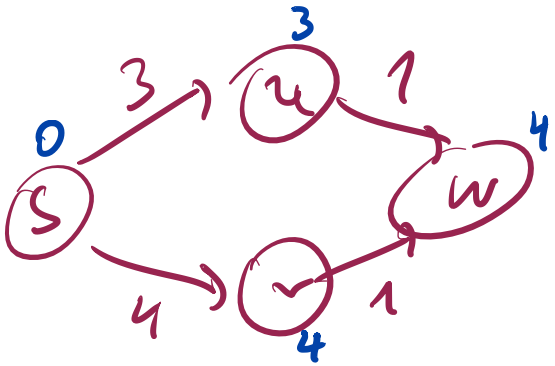


- Entrada: Grafo $G = (V, E)$ com pesos $c_a \geq 0$ nas arestas $a \in E$, e um vértice $s \in V$.
- Saida: A distância mínima d_v entre s e cada vértice $v \in V$.

Single source shortest path

(s, t)

All pairs shortest path



v	s	u	v	w
d_v	0	3	4	4

Algoritmo de Dijsktra (1956) - 1

Q: fila de prioridade

$d_s := 0; d_v := \infty, \forall v \in V \setminus \{s\}$
$\text{visited}(v) := \text{false}, \forall v \in V$
$Q := \emptyset$ ✓
$\text{insert}(Q, (s, 0))$ ✓

} O(n) ou O(1)

Θ

Algoritmo de Dijsktra (1956) - 2

```
while Q ≠ ∅ do
    v := deletemin(Q)
    visited(v) := true
    for u ∈ N+(v) do
        if not visited(u) then
            if d_u = ∞ then
                d_u := d_v + d_vu
                insert(Q, (u, d_u))
            else if d_v + d_vu < d_u
                d_u := d_v + d_vu
                update(Q, (u, d_u))
            end if
        end if
    end for
end while
```

$v \in Q \iff d_v < \infty$

$O(1)$.

$O(n)$

$O(\delta^+(v))$

$\sum_{v \in V} \delta^+(v) = m$

$O(m)$

$O(1)$

$n \times$ insert

$m \times$ update

$O(n) + n \times \text{insert} + m \times \text{update} + n \times \text{deletemin}$.

Ideal: insert, deletemin, update: $O(1)$

$O(n+m) \Rightarrow$ tempo linear

$\Rightarrow O(\text{insert} + \text{deletemin}) = \Omega(\log n)$.

Ord. $\Omega(n \log_2 n)$

$O(n \log n + m \times \text{update})$

$O(1)$?

$n \log n + m$ $m \log n$

Tipo abstrato de dados. Operações:

- insere um elemento de dada prioridade
- teste: fila vazia?
- remove o elemento de maior prioridade
- atualiza prioridade de um elemento

Adicionalmente:

- encontra o elemento de maior prioridade
- remove um elemento arbitrário
- une duas filas

Teste: elemento na fila?

Implementações

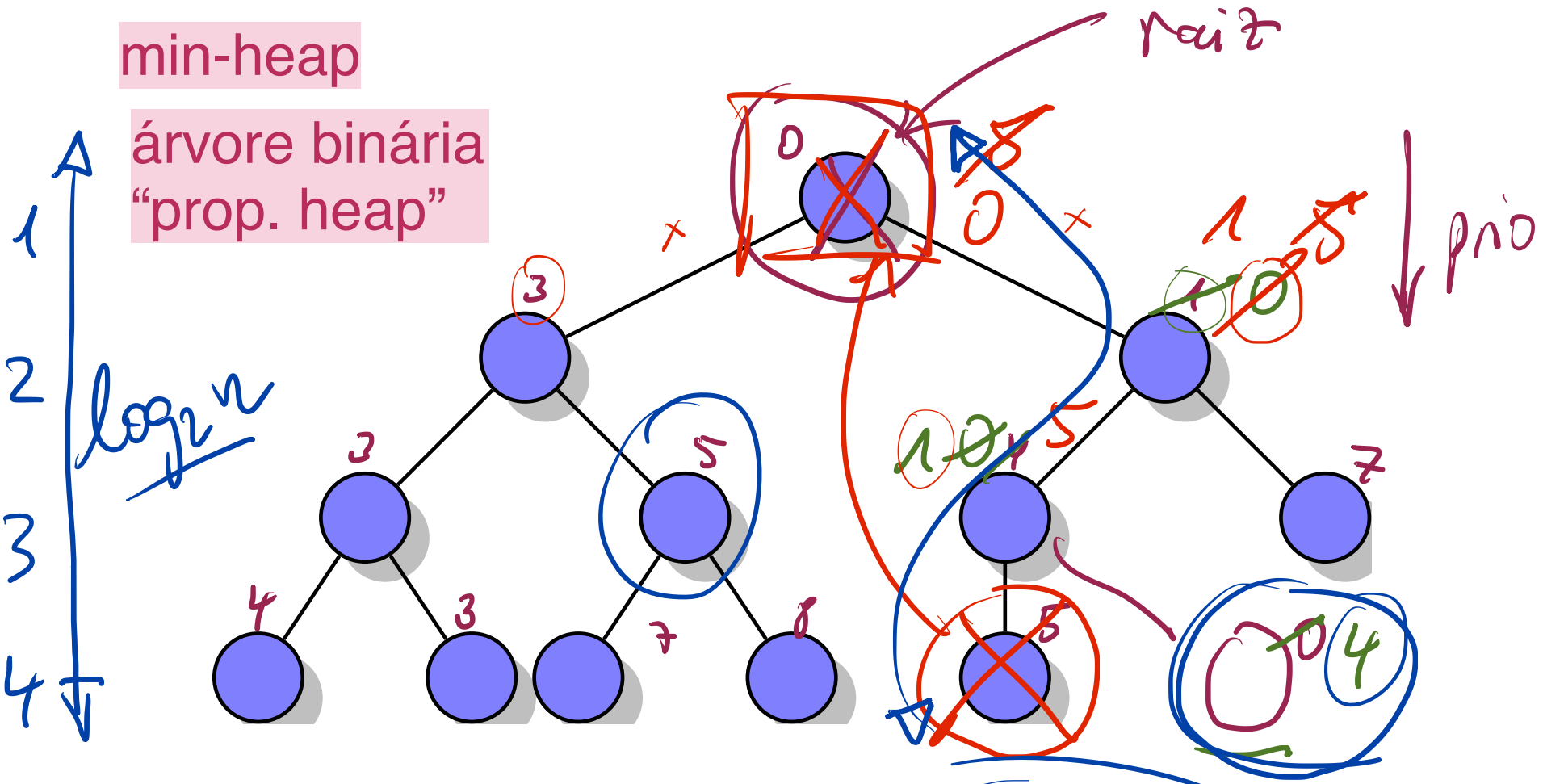
		insert	test	deletemin	update
Vetor	ord.	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Listas	ord.	$O(n)$	$O(1)$	$O(1)$	$\Theta(n)$
Vetor não ord.					

$O(n^2 + n) = O(n^2)$

$\Theta(n^2)$

min-heap

árvore binária
"prop. heap"



insert,deletemin,update: $O(\log_2 n)$

$\Rightarrow$ Dijkstra : $O((n+m)\log n)$

```
insert(H,c) :=
        insere c na última posição p
        heapify-up(H,p)

heapify-up(H,p) :=
        if root(p) return
        if key(parent(p))>key(p) then
                swap(key(parent(p)),key(p))
                heapify-up(H,parent(p))
        end if
```

Th. Para um min-heap T: depois um decremento da prioridade de um elemento p, e depois de uma chamada de heapify-up, temos novamente um min-heap.

Ideia: Indução sobre profundidade k do elemento p.

Base: $k=1$: p é raiz. ✓

Passo: $k>1$: a) chave de pai(p) < chave de p. ✓

Heap binário: deleção

delete(H,p):=

troca última posição com p

heapify-down(H,p)

heapify-down(H,p):=

if p não possui filhos return

if p possui um filho then

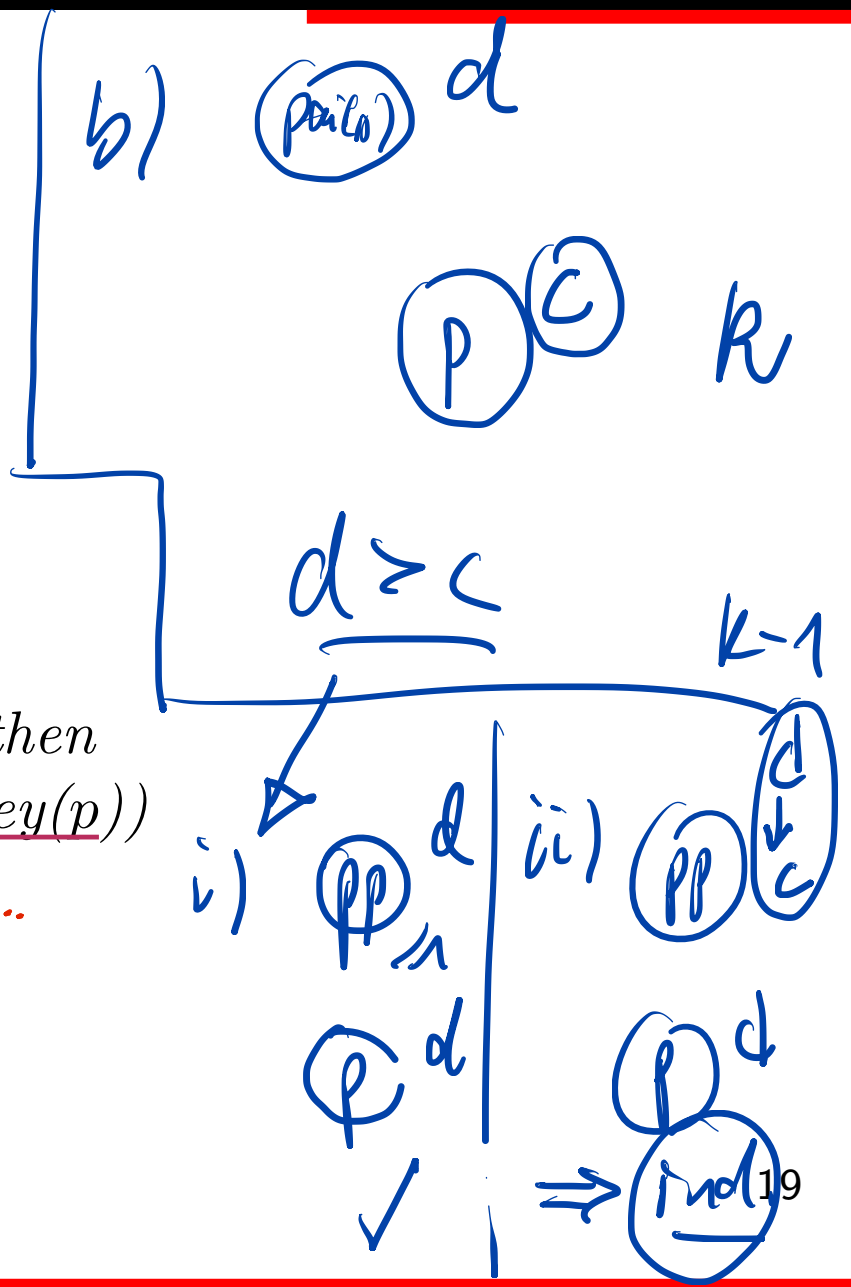
if key(left(p))<key(p)) then

swap(key(left(p)),key(p))

return

end if

heap-down..



Heap binário: deleção

```
... heapify-down(H,p):=

    { p possui dois filhos }
    if key(p)>key(left(p)) or key(p)>key(right(p)) then
        if (key(left(p))<key(right(p)) then
            swap(key(left(p)),key(p))
            heapify-down(H,left(p))
        else
            swap(key(right(p)),key(p))
            heapify-down(H,right(p))
        end if
    end if
```

Heap binário: atualização

$$
\begin{array}{l}
update(H,p,v) := \\
\quad if\ v < key(p)\ then \\
\quad\quad key(p) := v \\
\quad\quad heapify\text{-}up(H,p) \\
\quad else \\
\quad\quad key(p) := v \\
\quad\quad heapify\text{-}down(H,p) \\
\quad end\ if
\end{array}
$$

$root(p) := return\ p = 0$
$parent(p) := return\ \lfloor (p-1)/2 \rfloor$
$key(p) := return\ v[p]$
$left(p) := return\ 2p + 1$
$right(p) := return\ 2p + 2$
$numchildren(p) := return\ \max(\min(n - \text{left}(p), 2), 0)$