Aula 29: Manipulação de listas de prioridades

- Inserção em listas de prioridades
- Remoção em listas de prioridades
- Construção de listas de prioridades e ordenação

Operações básicas

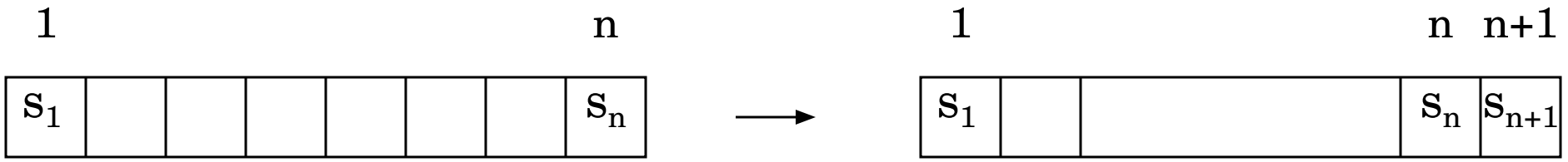
- Descrever métodos para as operações de
  - inserção
  - remoção
  - construção

em listas de prioridades

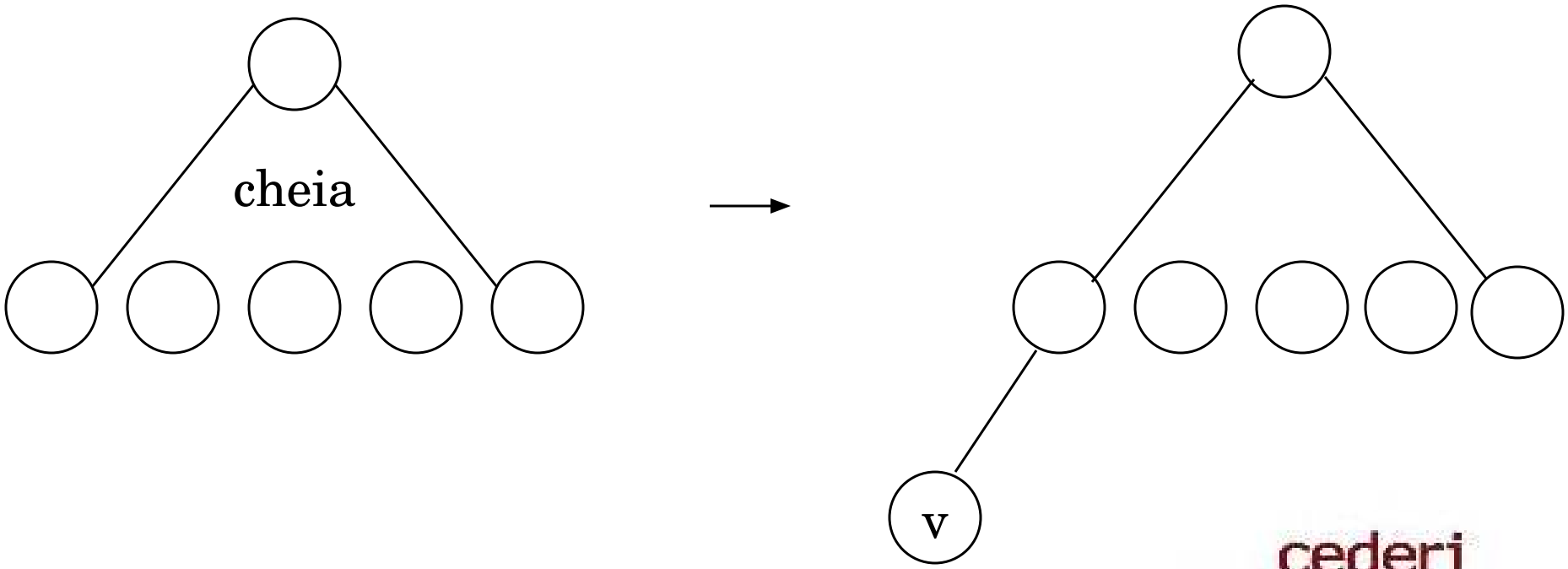
Inserção em lista de prioridades

- Seja v o nó a ser inserido, com uma dada prioridade, num heap com n nós.
- Inserir v, ao final do heap. Isto é, na posição n+1.
- Na árvore completa T, equivalente ao heap, caso T seja cheia, o nó v será incluído como o filho mais à esquerda da folha mais à esquerda de T. Caso contrário, v se transformará em uma nova folha, imediatamente à direita da folha mais à direita de T.

Inserção em lista de prioridades

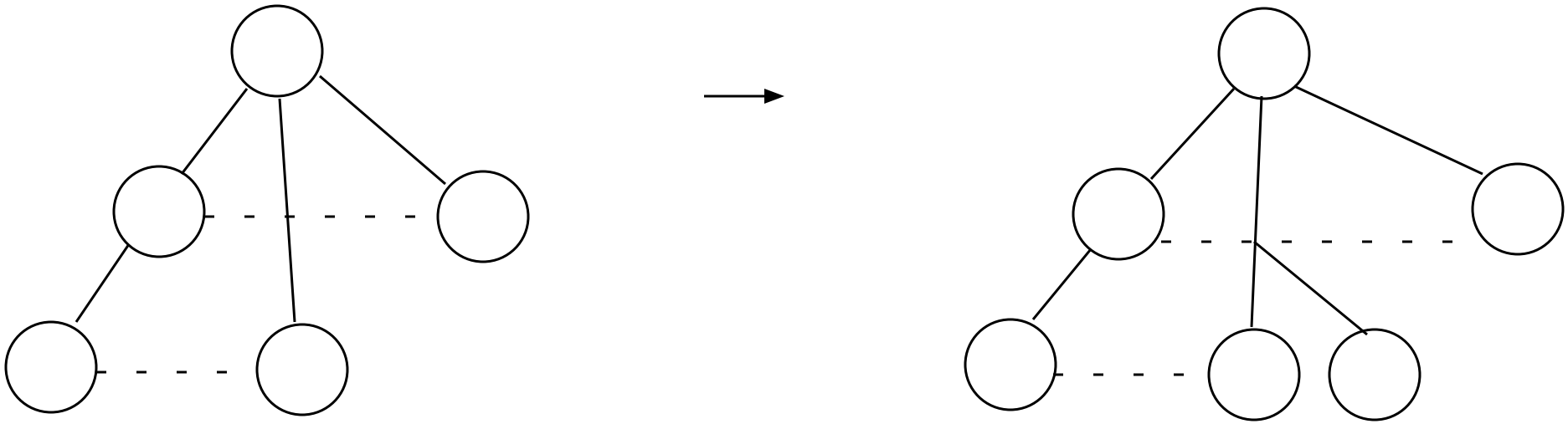


$v = s_{n+1}$



Inserção em lista de prioridades

ou



Inserção e aumento de prioridades

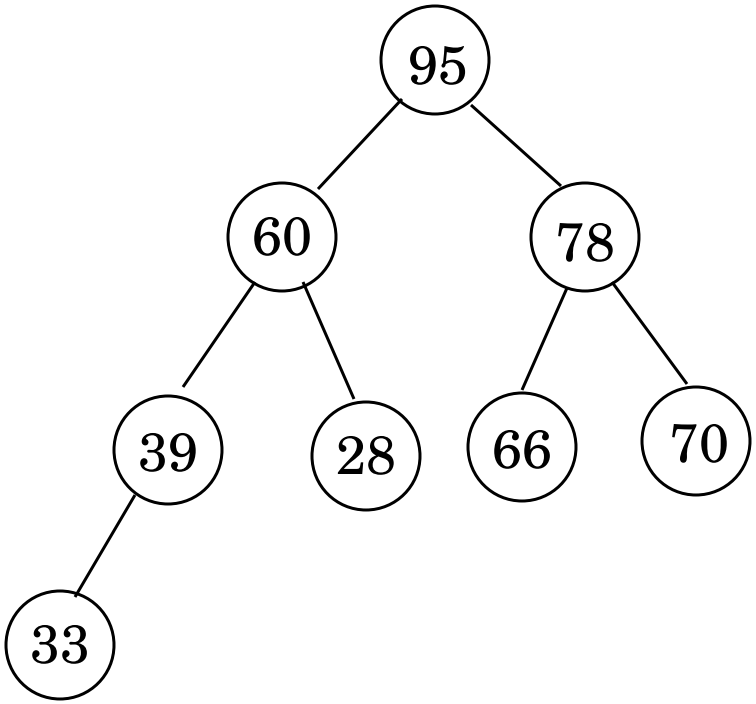
- Após a inserção de v na posição n+1, é necessário verificar se a condição das prioridades do heap continua válida.

- Em caso negativo, é necessário efetuar uma reordenação.

- Para tal, basta supor que o elemento n+1 do heap teve sua prioridade aumentada. Nesse caso, a solução é utilizar o procedimento de aumento de prioridade.

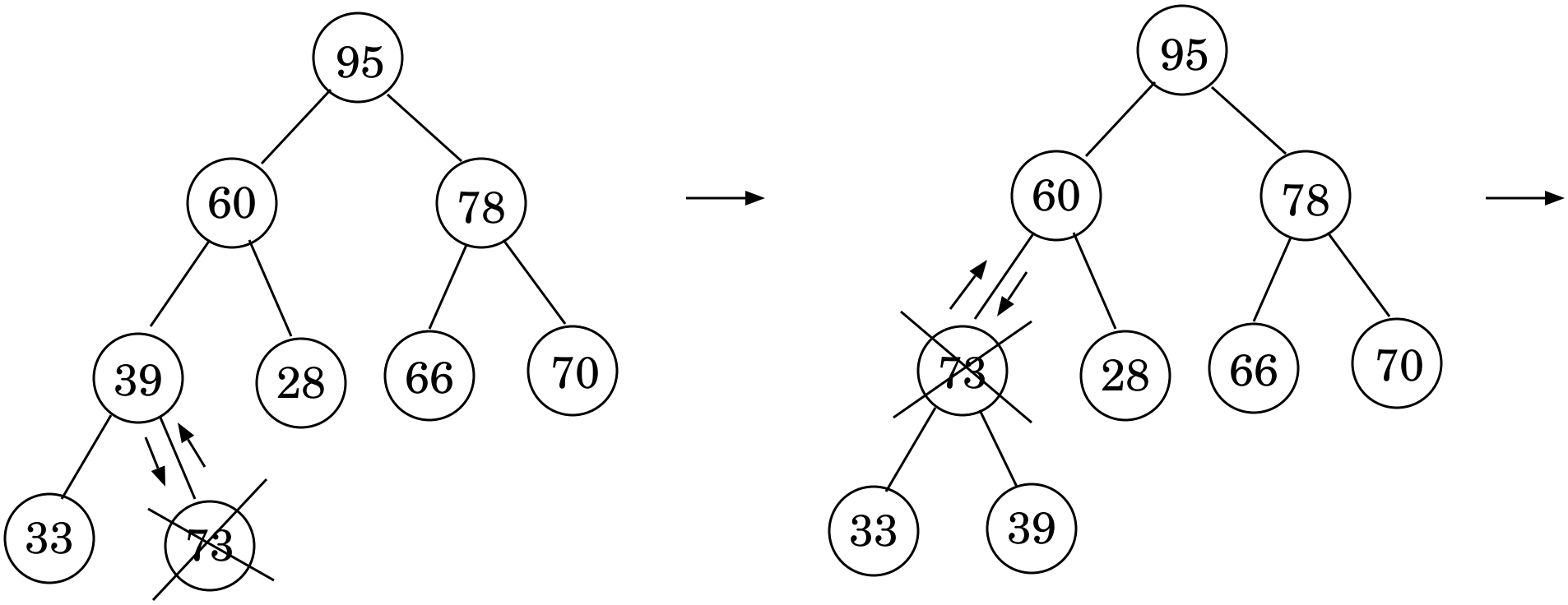
Inserção e aumento de prioridades

Exemplo:

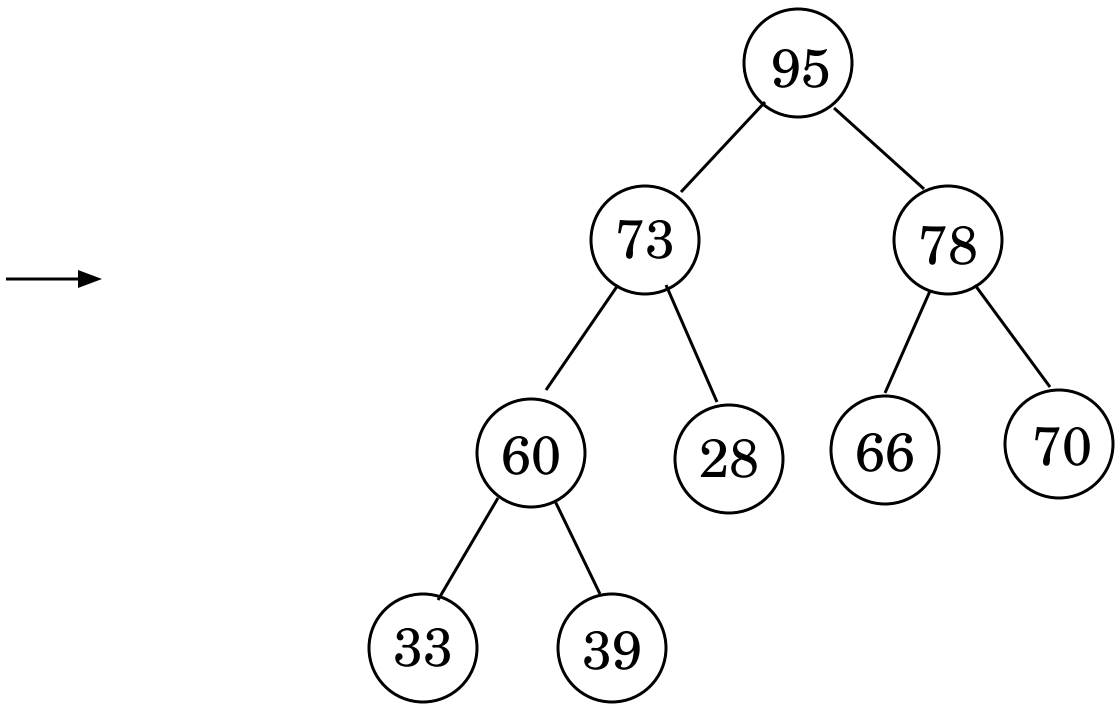


Inserir nó com prioridade 73

Inserção e aumento de prioridades



Inserção e aumento de prioridades



Algoritmo de inserção

Algoritmo: inserção em uma lista de prioridades

```
se n < M então

    T [n+1] := novo
    n := n+1
    subir (n)
senão   overflow
```

Algoritmo de inserção

- O heap está armazenado na tabela T
- M é o tamanho total da memória disponível
- A variável novo representa o nó a ser inserido
- Complexidade: O (log n)

Algoritmo de inserção

- Exercício

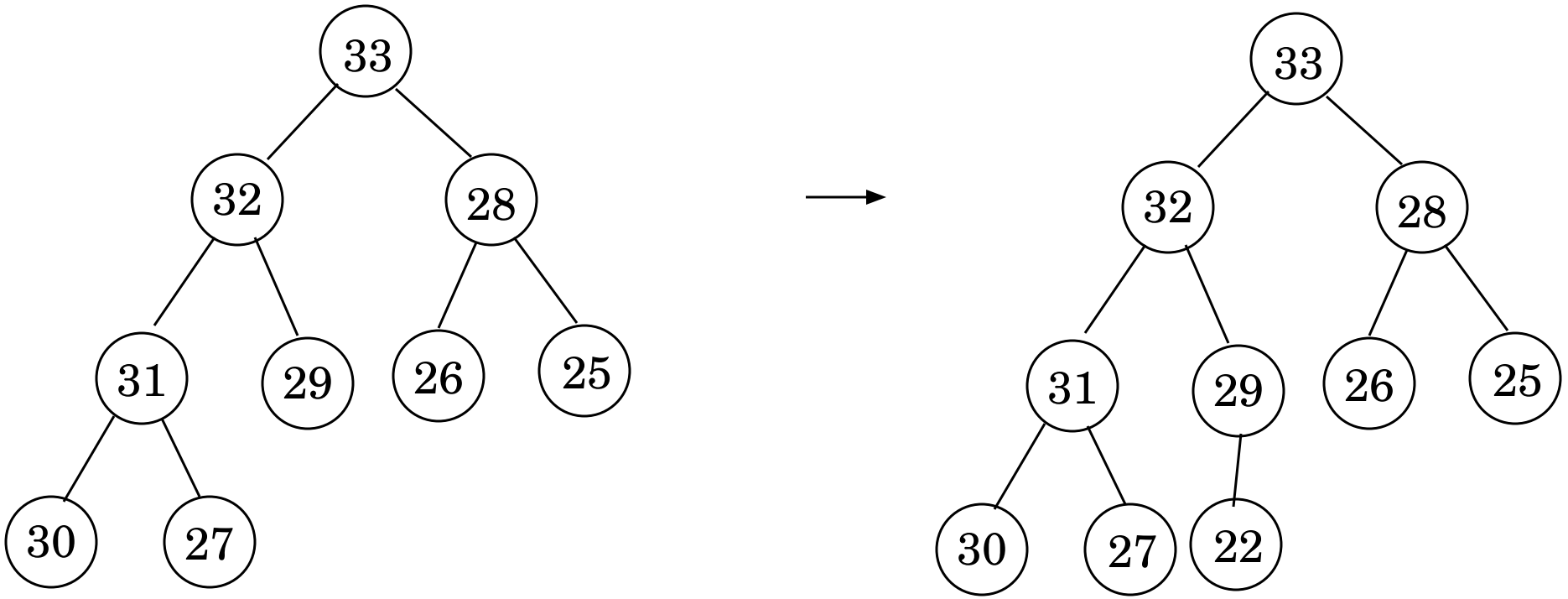
- Inserir um novo nó, com prioridade igual a 22 no heap

33 32 28 31 29 26 25 30 27

Tempo: 1 minuto

Algoritmo de inserção

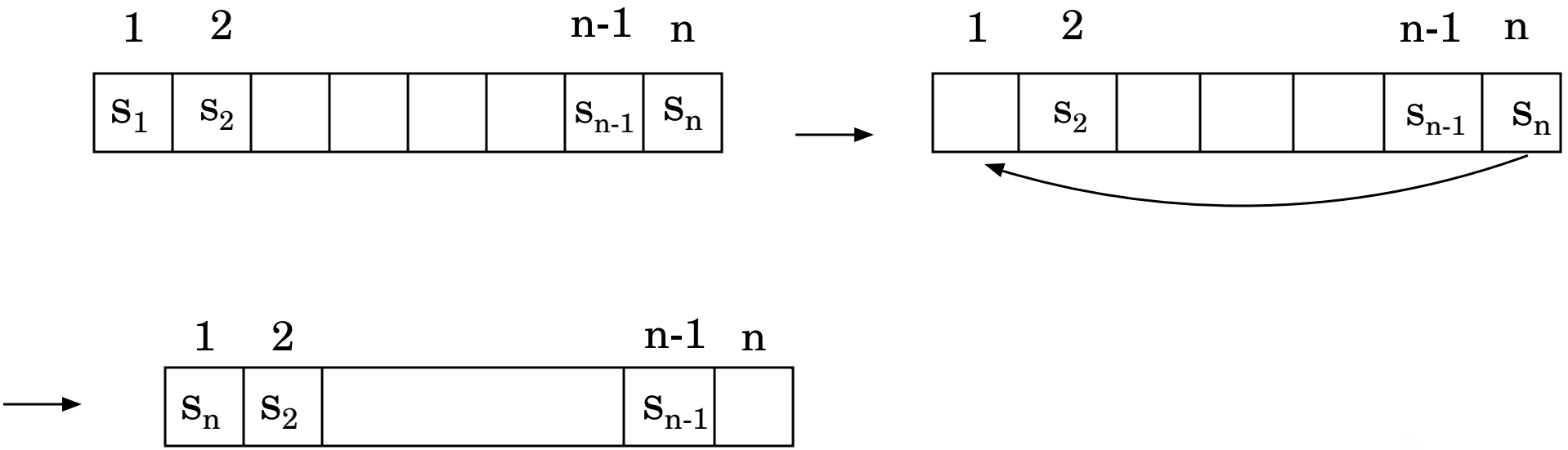
Solução



Remoção em listas de prioridades

- Remover o nó de maior prioridade, isto é, o primeiro nó do heap.

- Após a remoção, mover o último nó para a posição 1. A lista torna-se de tamanho n-1.

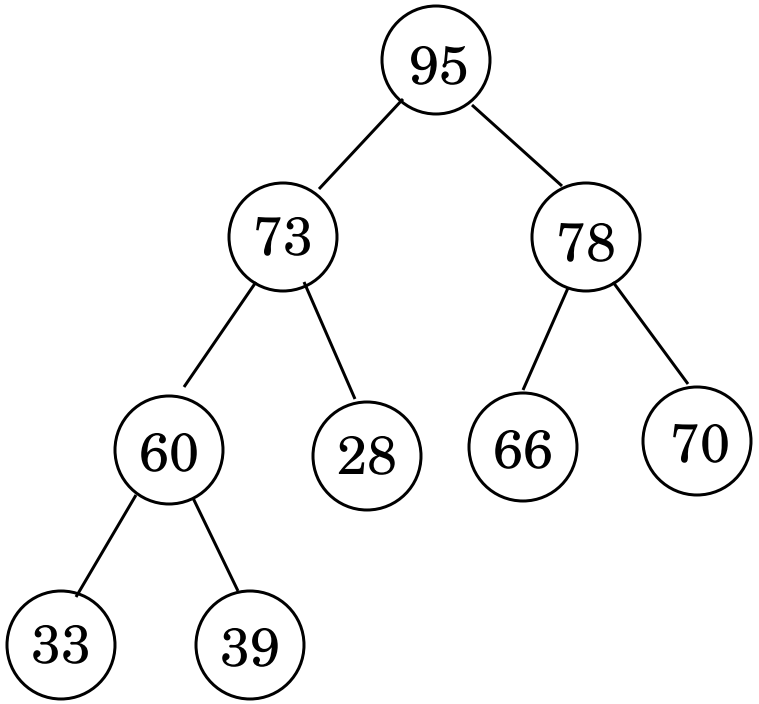


Remoção e diminuição de prioridades

- Após a remoção do nó de prioridade maior e a mudança do último elemento para a posição 1, é necessário verificar se a condição das prioridades dos heaps continua válida.
- Em caso negativo, é necessário efetuar uma reordenação.
- Para tal, basta supor que o elemento 1 do heap teve a sua prioridade diminuida. Nesse caso, a solução é utilizar o procedimento de diminuição de prioridade.

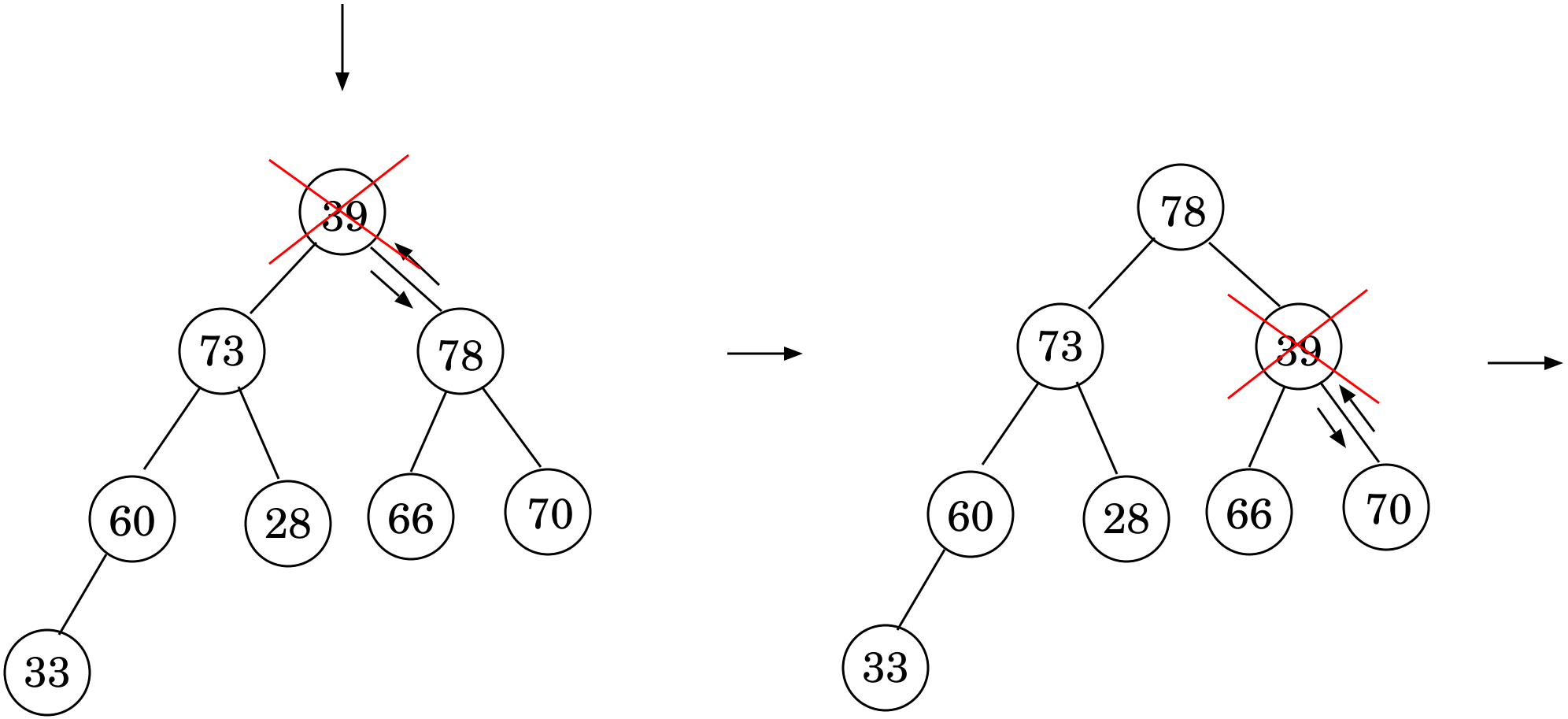
Remoção e diminuição de prioridades

Exemplo

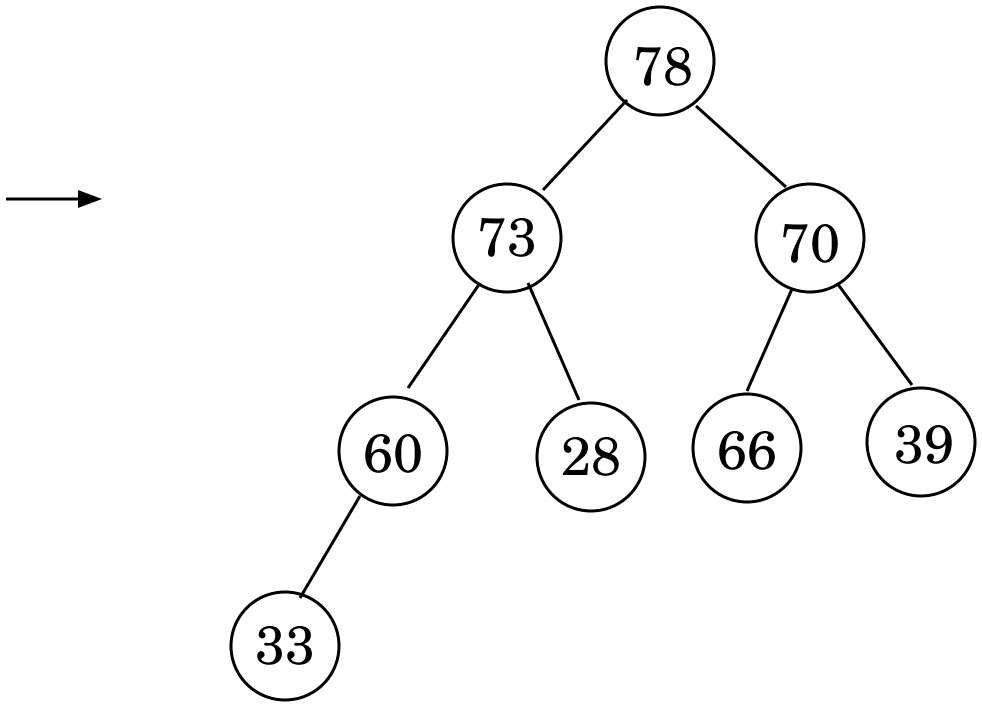


Remover elemento de maior prioridade

Remoção e diminuição de prioridades



Remoção e diminuição de prioridades



Algoritmo de remoção

- Algoritmo: remoção do elemento de maior prioridade

```
se n≠0  então
        agir (T[1])
        T [1] := T [n]
        n := n-1
        descer (1, n)
senão   underflow
```

- O heap está armazenado na tabela T.
- O procedimento agir consiste na utilização do elemento de maior prioridade, de acordo com a aplicação.
- Complexidade: O (log n)

Algoritmo de remoção

- Exercício

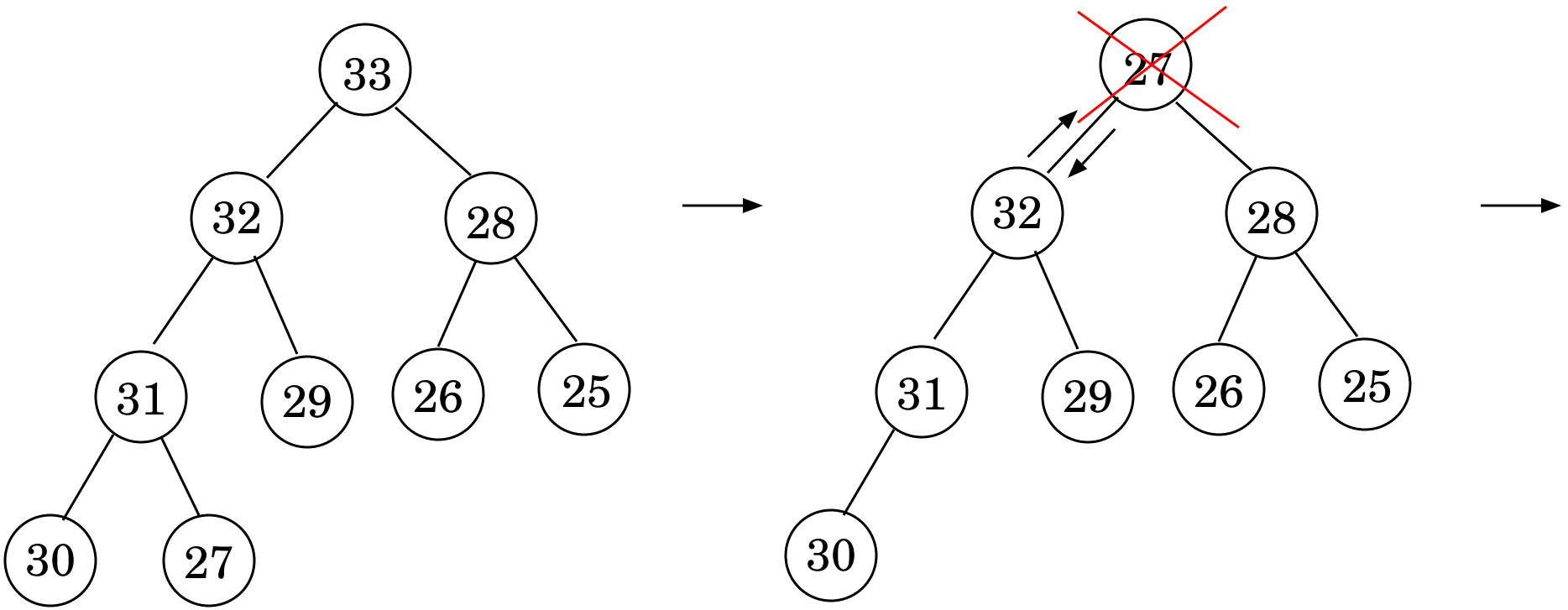
- Remover o elemento de maior prioridade do heap

33 32 28 31 29 26 25 30 27

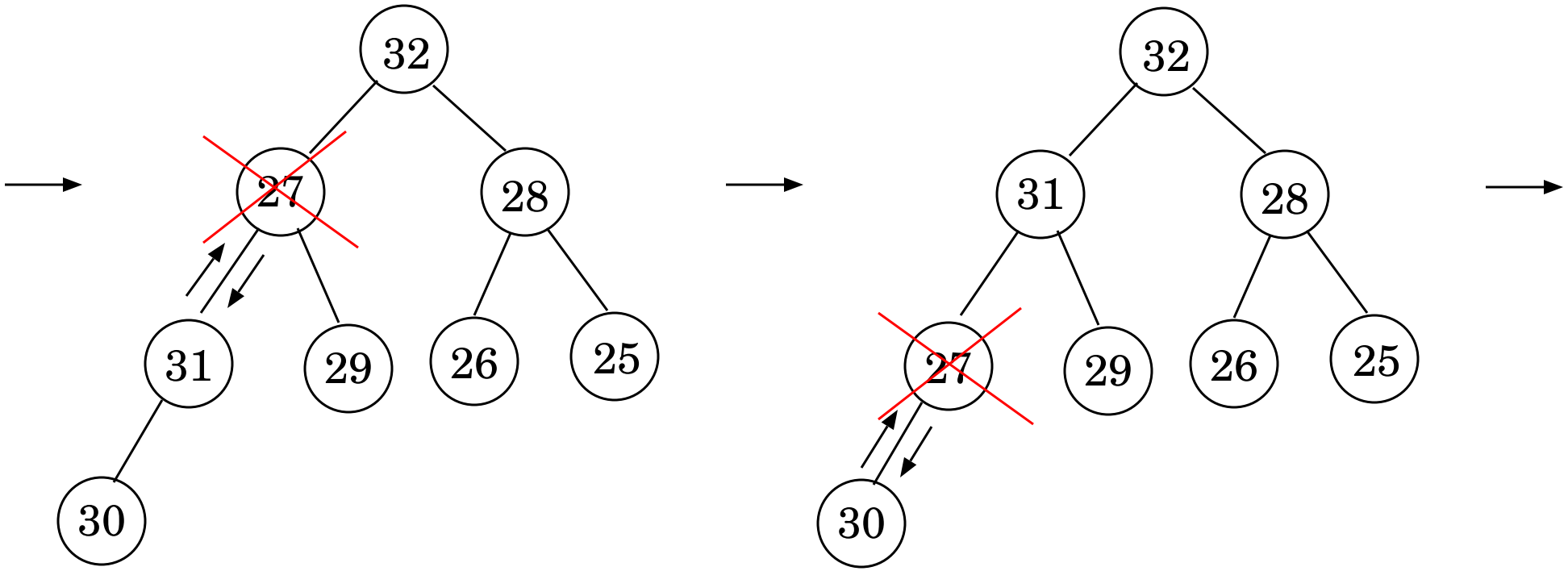
Tempo: 2 minutos

Algoritmo de remoção

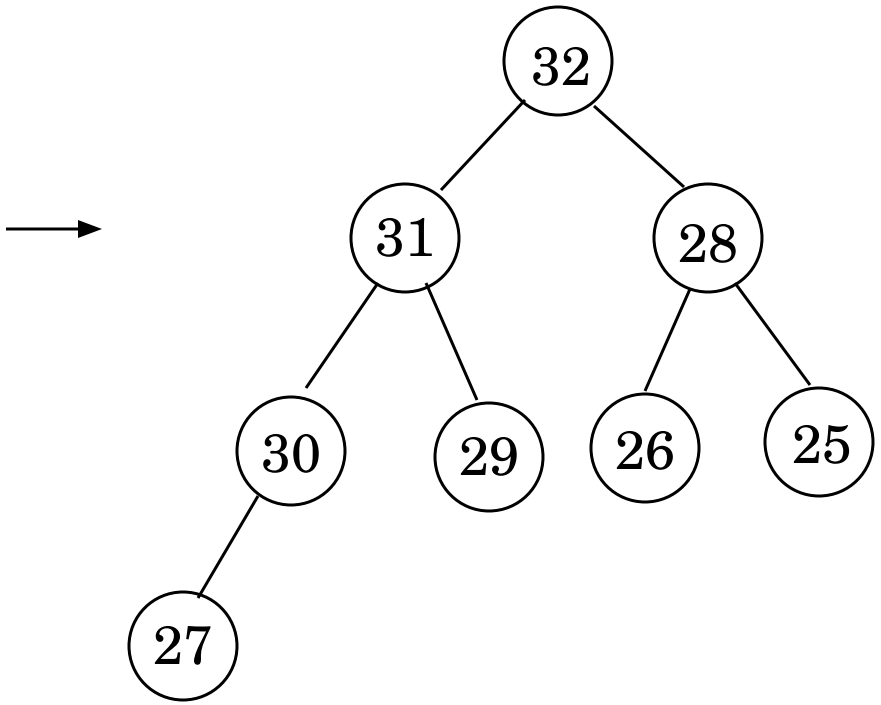
Solução



Algoritmo de remoção



Algoritmo de remoção



Construção de um heap

- Objetivo: Descrever métodos para construir heaps.

- Solução 1:

  Uma lista ordenada constitui um heap.
  Logo, um heap pode ser construído, simplesmente, ordenando-se uma lista.

- Complexidade: O (n log n)

Construção de um heap

Solução 2:

Seja S uma lista dada, para a qual se deseja construir um heap. Para que S seja um heap, basta corrigir a posição de seus nós, isto é, considerar cada um de seus nós, como sendo uma nova inserção.

Algoritmo: Construção de um heap

```
para i = 2, ..., n  faça
        subir (i)
```

Complexidade: O (n log n)

Construção de um heap

Exercício

Seja uma lista dada pelas prioridades a seguir:

18 25 41 34 14 10 52 50 48

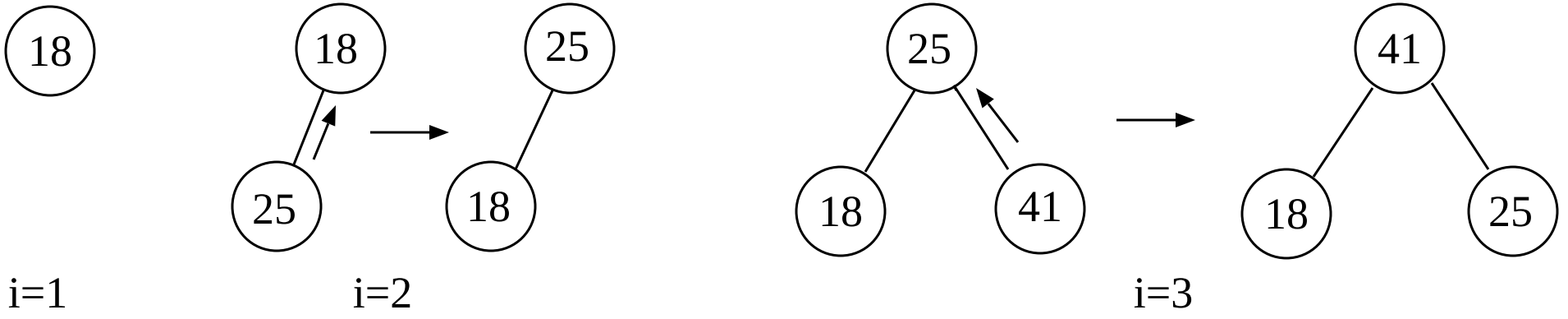
Construir um heap para esta lista, utilizando o algoritmo da solução 2.

Tempo: 8 minutos

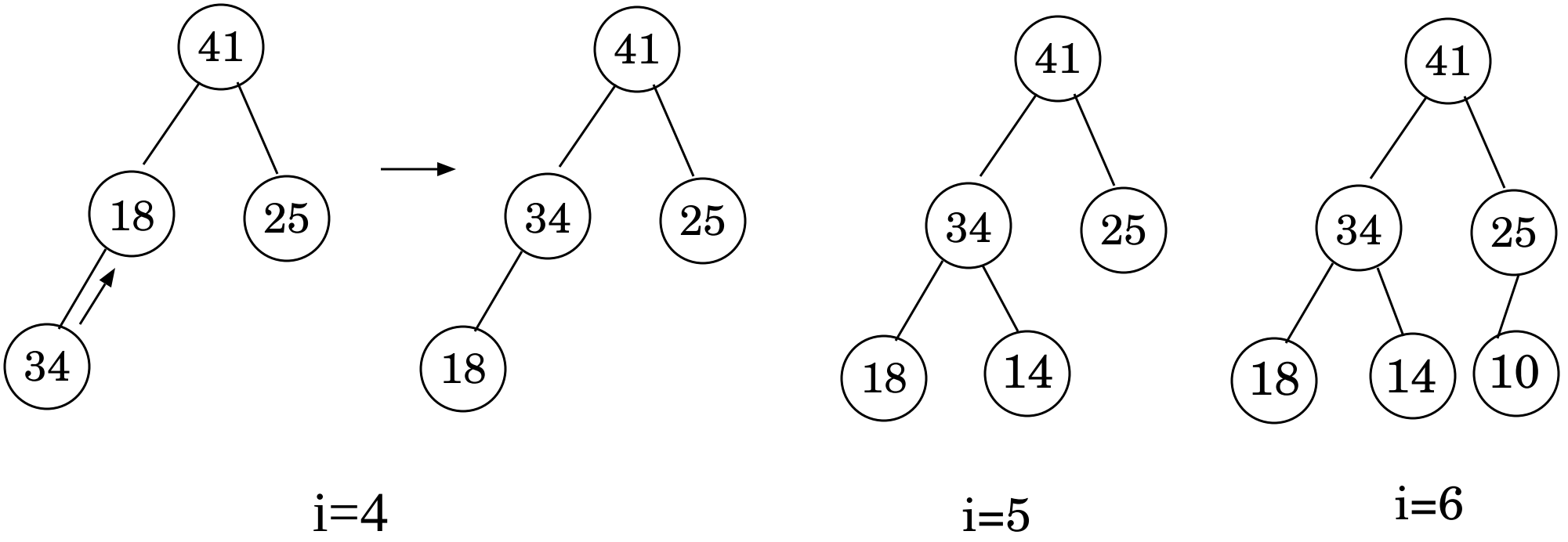
Construção de um heap

Solução

i	1	2	3	4	5	6	7	8	9
	18	25	41	34	14	10	52	50	48

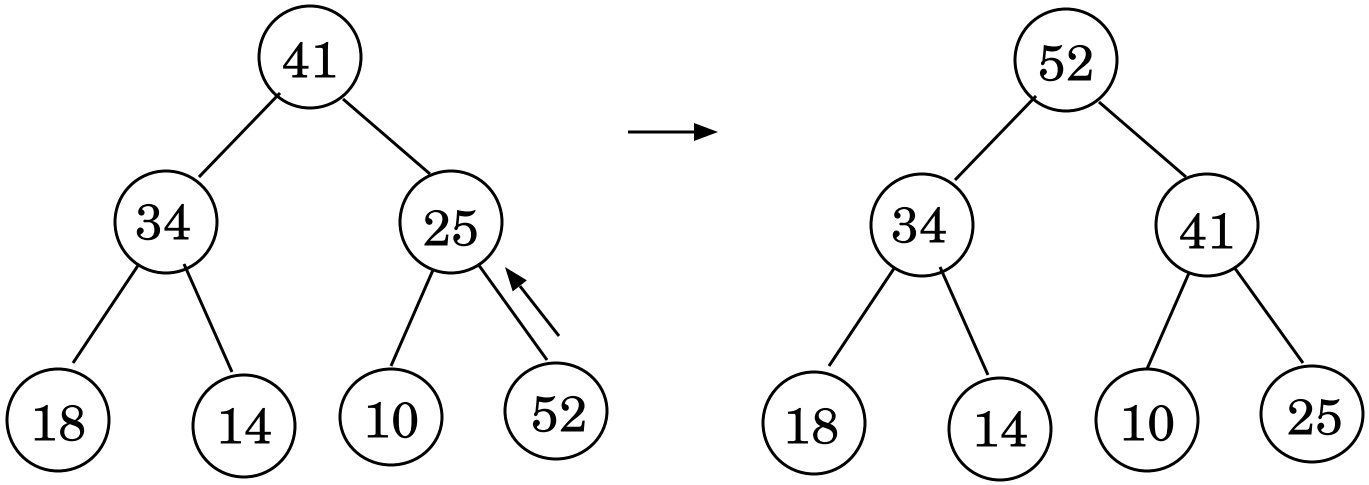


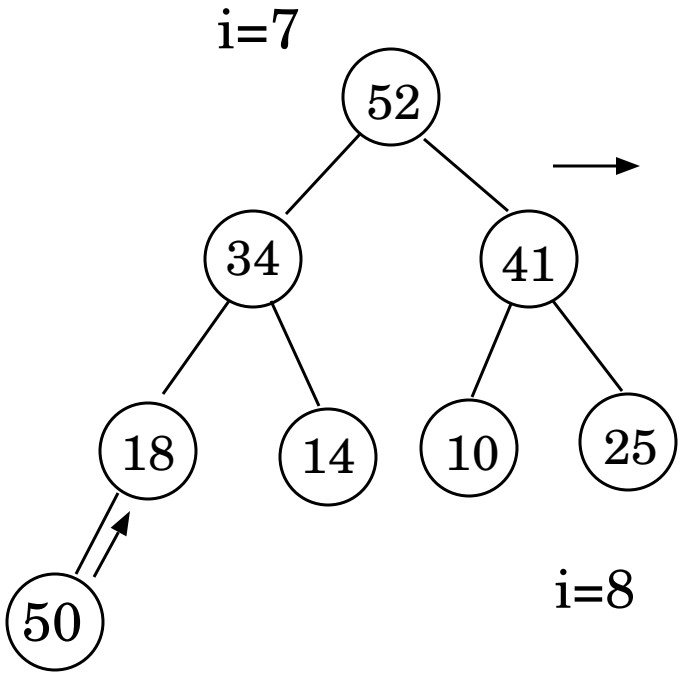
Construção de um heap



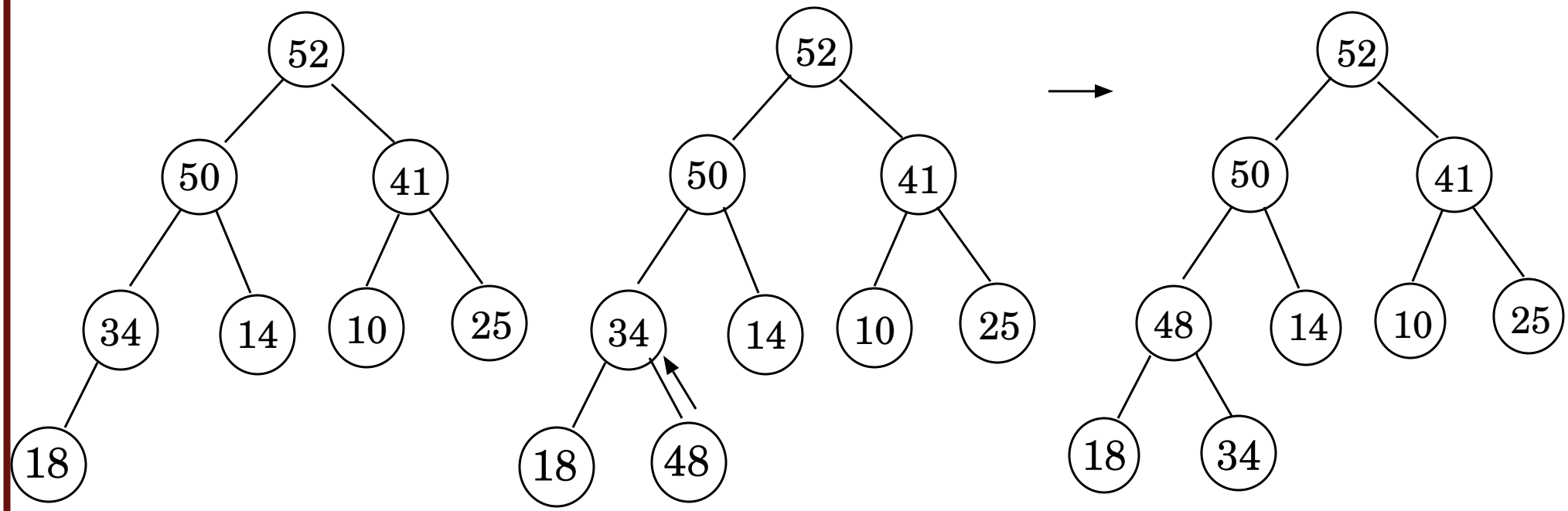
i=4 i=5 i=6

Construção de um heap





Construção de um heap



Construção de um heap

Solução 3:

- A propriedade das prioridades dos heaps é sempre, trivialmente, satisfeita pelas folhas. No caso, pelos nós alocados a partir da posição $\lfloor n/2 \rfloor + 1$.
- Assim, na construção dos heaps, os únicos nós relevantes, sob o ponto de vista de análise, são os nós interiores. Estes devem ter suas prioridades verificadas e acertadas, a partir dos níveis mais baixos da árvore.

Construção de um heap

Algoritmo: Construção de heaps

```
procedimento arranjar (n)
    para i = ⌊n/2⌋, ..., 1 faça
        descer (i, n)
```

Complexidade: O (n)

Aplicação: ordenação por heaps

- Seja T uma lista que se deseja ordenar.
- Construir um heap para T, tomando-se a prioridade igual à chave de ordenação desejada.
- Iterativamente, remover o elemento de maior prioridade e corrigir o heap.
- O método acima produzirá a ordenação da lista em ordem não decrescente de suas chaves de ordenação.

Algoritmo de ordenação

Algoritmo: Ordenação de uma lista T

```
arranjar ( n )
m := n
enquanto m > 1 faça
        T [1] ⇔ T [m]
        m := m - 1
        descer ( 1, m )
```

- O procedimento arranjar (n) constrói um heap para T, segundo a solução 3.
- A notação T [1] ⇔ T [m] significa a troca de posição em T, entre T [1] e T [m].
- Complexidade: O (n log n)

Algoritmo de ordenação

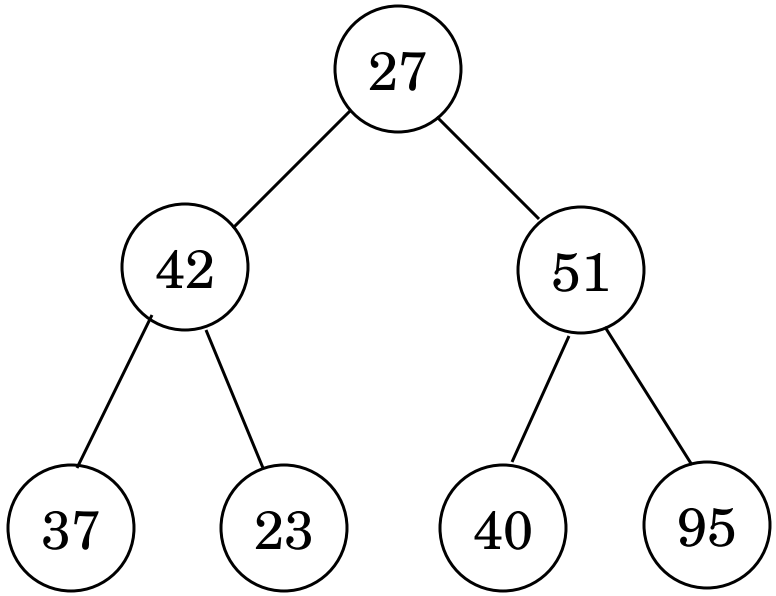
Exemplo

Seja a lista: 40 37 95 42 23 51 27

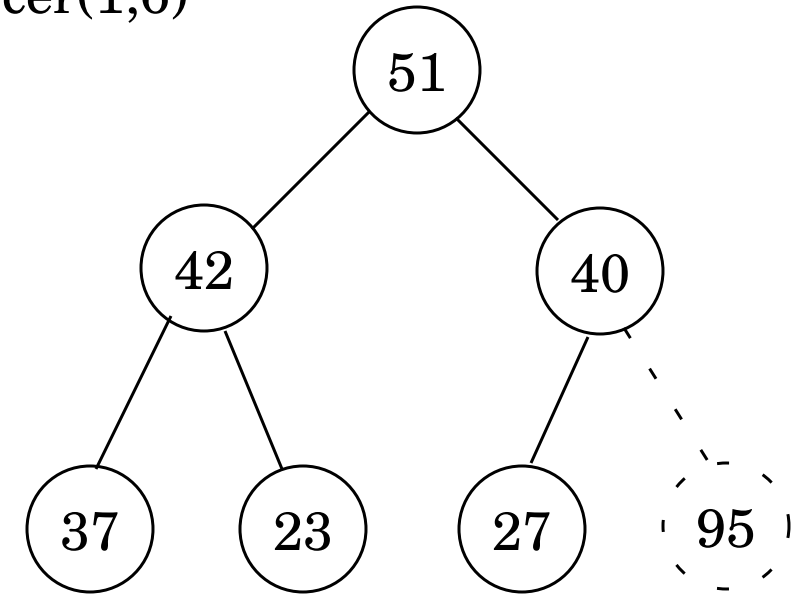
O método da ordenação por heaps, aplicado a esta lista, corresponderá às seguintes operações.

Algoritmo de ordenação

trocar(TB[1],TB[7])

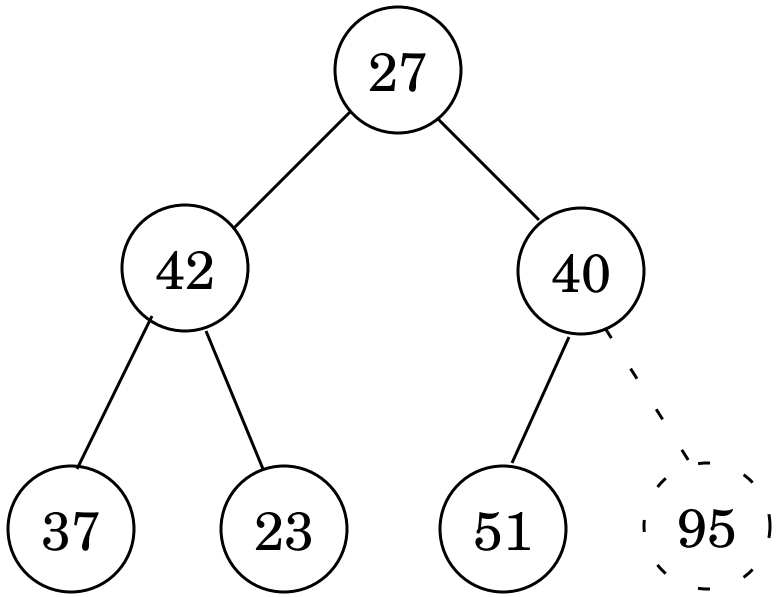


m = 6
descer(1,6)

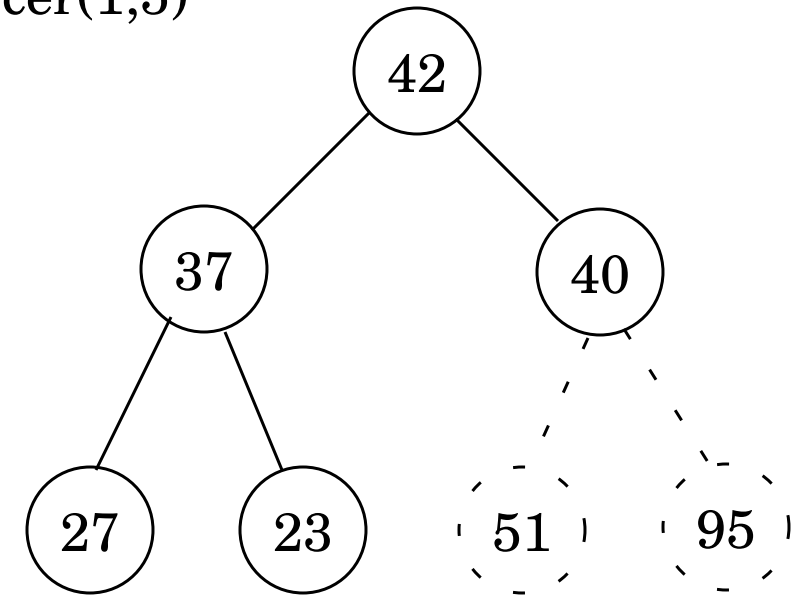


Algoritmo de ordenação

trocar(TB[1],TB[6])

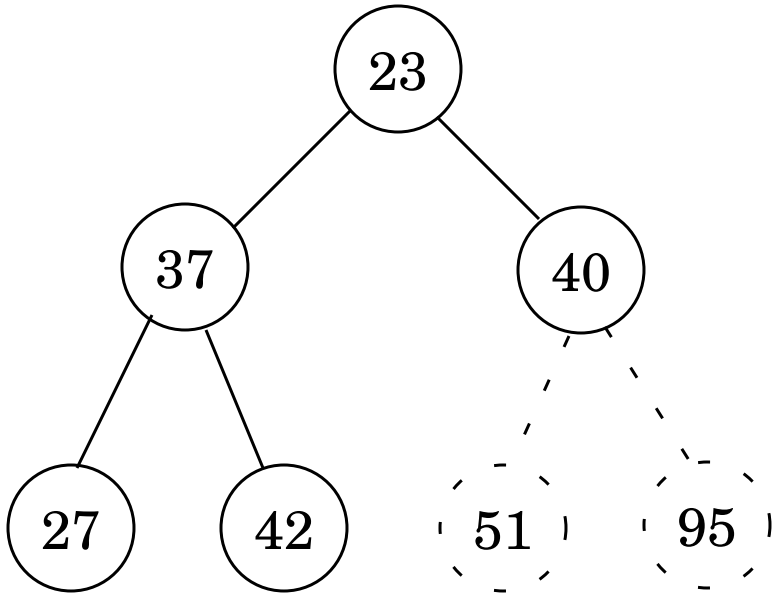


m = 5
descer(1,5)

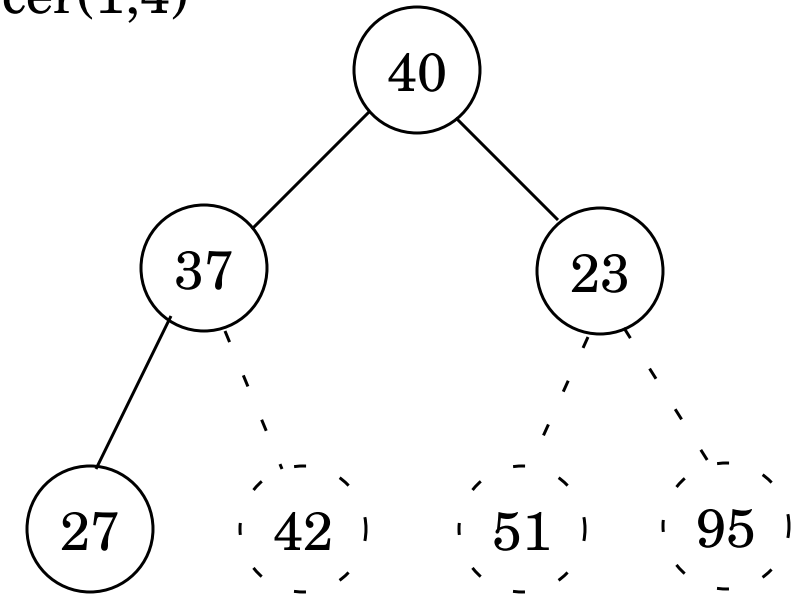


Algoritmo de ordenação

trocar(TB[1],TB[5])

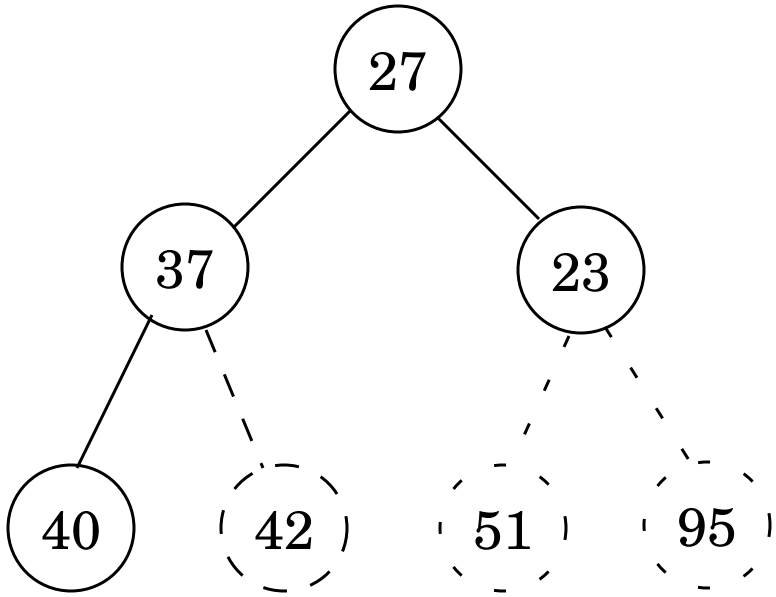


m = 4
descer(1,4)

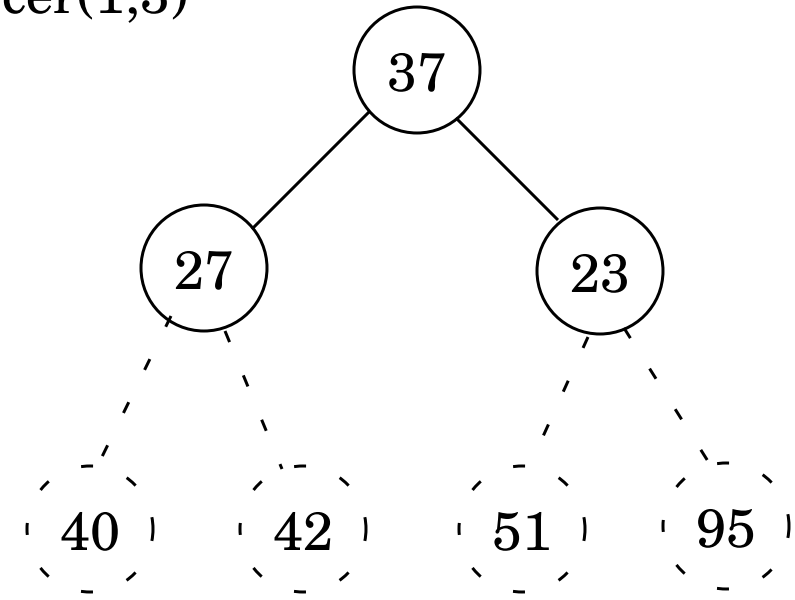


Algoritmo de ordenação

trocar(TB[1],TB[4])

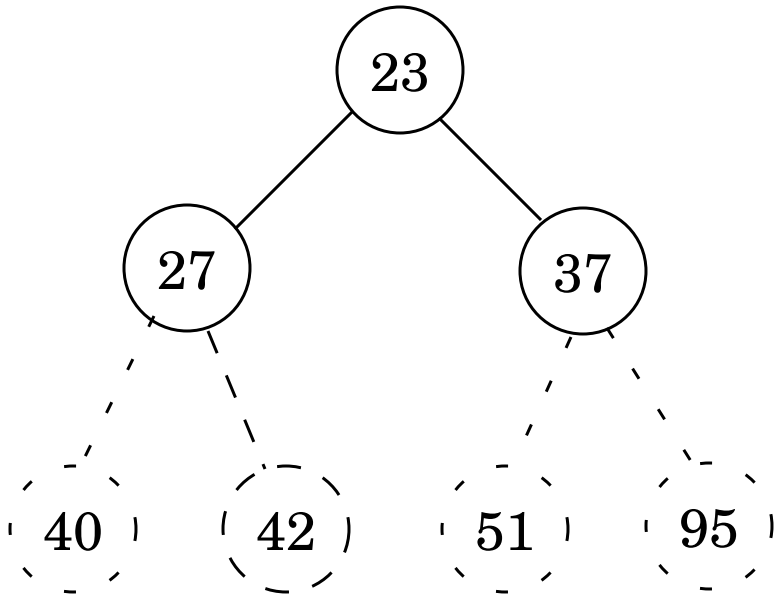


m = 3
descer(1,3)

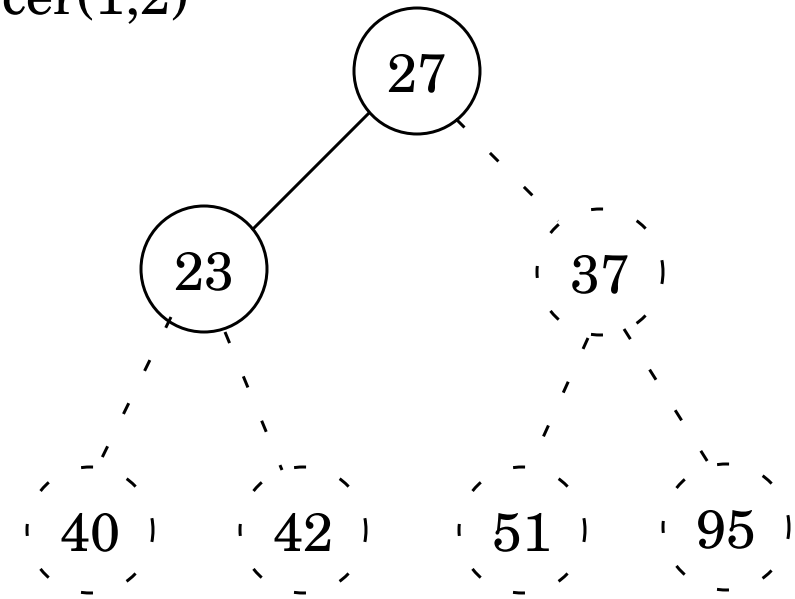


Algoritmo de ordenação

trocar(TB[1],TB[3])

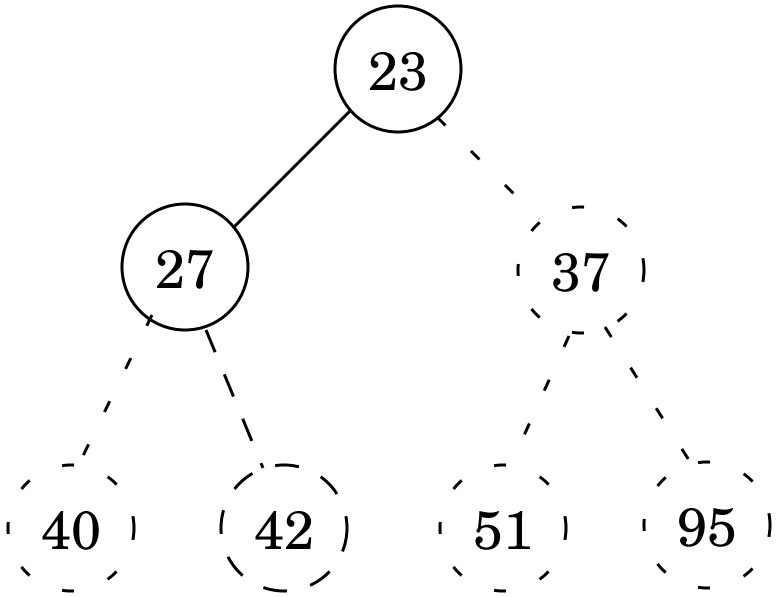


m = 2
descer(1,2)

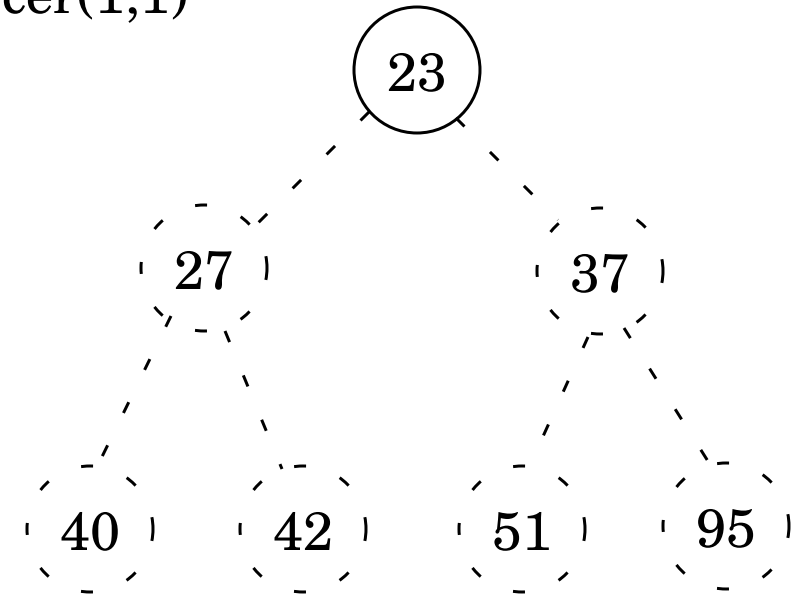


Algoritmo de ordenação

trocar(TB[1],TB[2])



m = 1
descer(1,1)



Algoritmo de ordenação

Exercício Final

Provar que o algoritmo da solução 3, para a construção de um heap, termina em O (n) passos.