Listas de Prioridade

Fonte de consulta: Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Capítulo 6

Prioridade

- Algumas aplicações precisam recuperar rapidamente um dado de maior prioridade

- Exemplo: lista de tarefas
  - A cada momento, deve-se executar a tarefa que tem mais prioridade
  - Selecionar a tarefa mais prioritária de uma lista e retirá-la da lista
  - Prioridades podem mudar
  - Novas tarefas podem chegar e precisam ser acomodadas

Lista de Prioridades

- Tabela onde cada registro está associado a uma prioridade
- Prioridade: valor numérico armazenado em um dos campos do registro

- Operações sobre listas de prioridade:
  - Seleção do elemento de maior prioridade
  - Inserção de novo elemento
  - Remoção do elemento de maior prioridade
  - Alteração da prioridade de um determinado elemento

Implementação de Listas de Prioridade

- Lista não ordenada
- Lista ordenada
- Heap

Implementação de Listas de Prioridade

- **Lista não ordenada**
- Lista ordenada
- Heap

Implementação por Lista Não Ordenada

- **Inserção e Construção**: elementos (registros) podem ser colocados na tabela em qualquer ordem
- **Remoção**: implica em percorrer a tabela sequencialmente em busca do elemento de maior prioridade
- **Alteração**: não implica em mudança na estrutura da tabela, mas exige busca do elemento a ser alterado
- **Seleção**: idem à Alteração

Complexidade

- Para uma tabela com n elementos
  - Seleção: O(n)
  - Inserção: O(1)
  - Remoção: O(n)
  - Alteração: O(n)
  - Construção: O(n)

Implementação de Listas de Prioridade

- Lista não ordenada
- **Lista ordenada**
- Heap

Implementação por Lista Ordenada

- **Remoção e Seleção**: imediata, pois elemento de maior prioridade é o primeiro da tabela
- **Inserção**: exige percorrer a tabela para encontrar a posição correta de inserção
- **Alteração**: semelhante a uma nova inserção
- **Construção**: exige ordenação prévia da tabela

Complexidade

- Para uma tabela com n elementos
  - Seleção: O(1)
  - Inserção: O(n)
  - Remoção: O(1)
  - Alteração: O(n)
  - Construção: O(n log n) (complexidade da ordenação)

Implementação de Listas de Prioridade

- Lista não ordenada
- Lista ordenada
- **Heap**

Implementação por Heap

- Mais eficiente na atualização do que as alternativas anteriores

Heap

- Lista linear composta de elementos com chaves $s_1, ..., s_n$, tal que $s_i \leq s_{i/2}$ para $1 < i \leq n$
  - Chaves representam as prioridades
  - Não existem dois elementos com a mesma prioridade

- Exemplo

  95 60 78 39 28 66 70 33

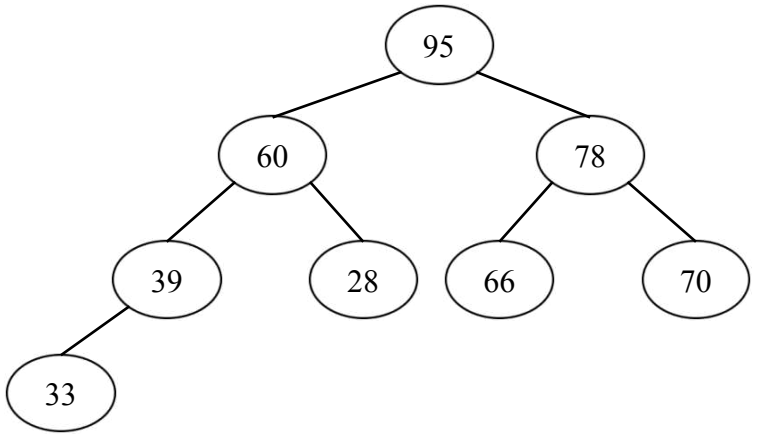
Heap

- Lista linear composta de elementos com chaves $s_1, ..., s_n$, tal que $s_i \leq s_{i/2}$ para $1 < i \leq n$
  - Chaves representam as prioridades
  - Não existem dois elementos com a mesma prioridade

- Exemplo

  95 60 78 39 28 66 70 33

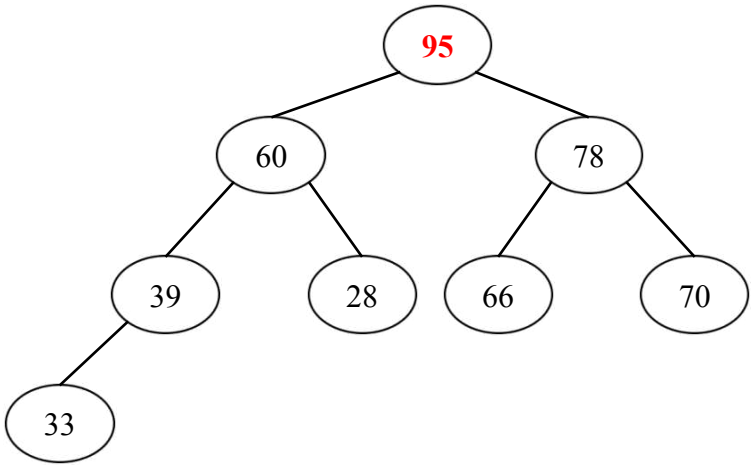
Lista representada como uma árvore binária completa



Montagem da Árvore Binária

- Nós são numerados sequencialmente, da raiz para os níveis mais baixos, da esquerda para a direita

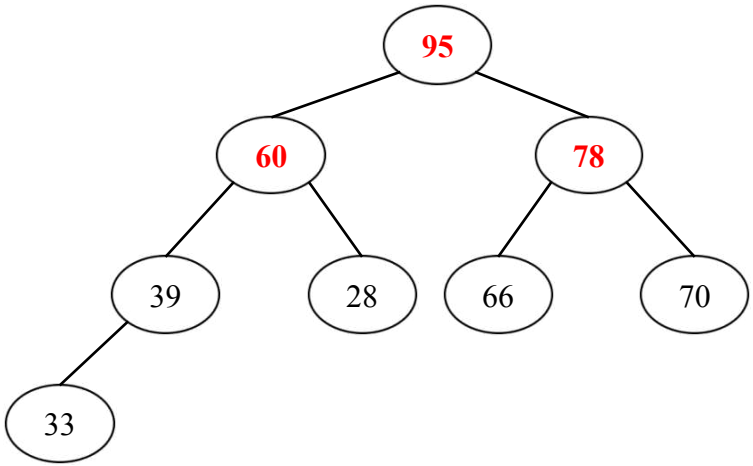
95 60 78 39 28 66 70 33



Montagem da Árvore Binária

- Nós são numerados sequencialmente, da raiz para os níveis mais baixos, da esquerda para a direita

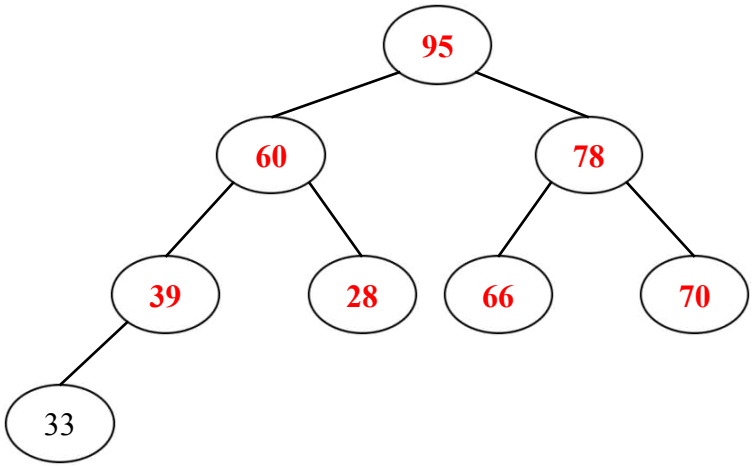
95 60 78 39 28 66 70 33



Montagem da Árvore Binária

- Nós são numerados sequencialmente, da raiz para os níveis mais baixos, da esquerda para a direita

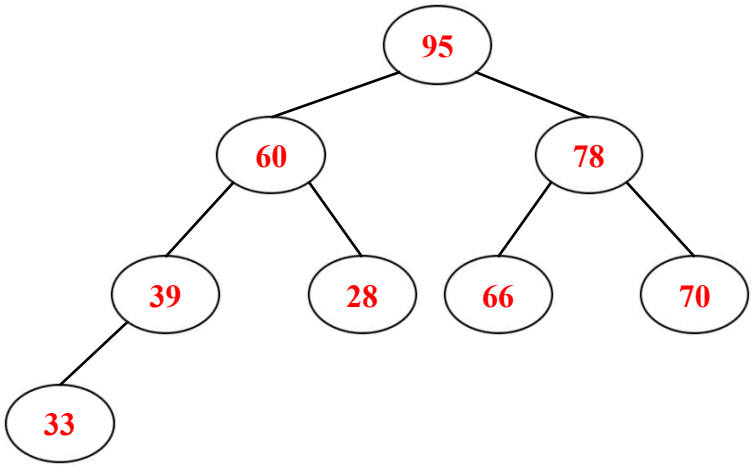
95 60 78 39 28 66 70 33



Montagem da Árvore Binária

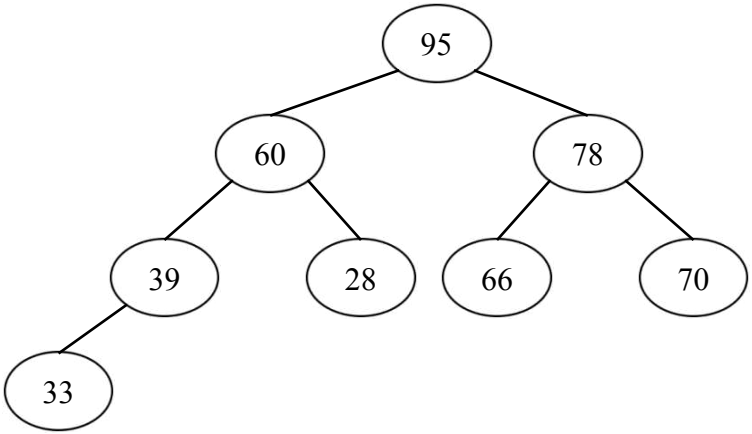
- Nós são numerados sequencialmente, da raiz para os níveis mais baixos, da esquerda para a direita

95 60 78 39 28 66 70 33



Propriedades

- Cada nó possui prioridade maior do que seus dois filhos
- O elemento de maior prioridade é sempre a raiz da árvore



- A representação em memória pode ser feita usando uma lista linear sequencial (vetor)

Complexidade

- **Seleção**: imediata, pois elemento de maior prioridade é o primeiro da tabela
- **Inserção**, **Alteração e Remoção**: podem ser feitos em O(log n) (veremos detalhes mais adiante)
- **Construção**: pode ser feita em O(n) (melhor que no cenário anterior, que exigia ordenação total)

Complexidade

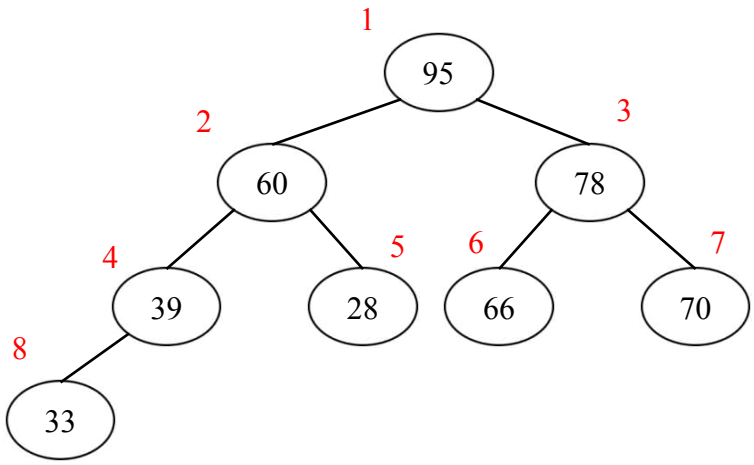
- Para uma tabela com n elementos
  - Seleção: O(1)
  - Inserção: O(log n)
  - Remoção: O(log n)
  - Alteração: O(log n)
  - Construção: O(n)

Alteração de Prioridade

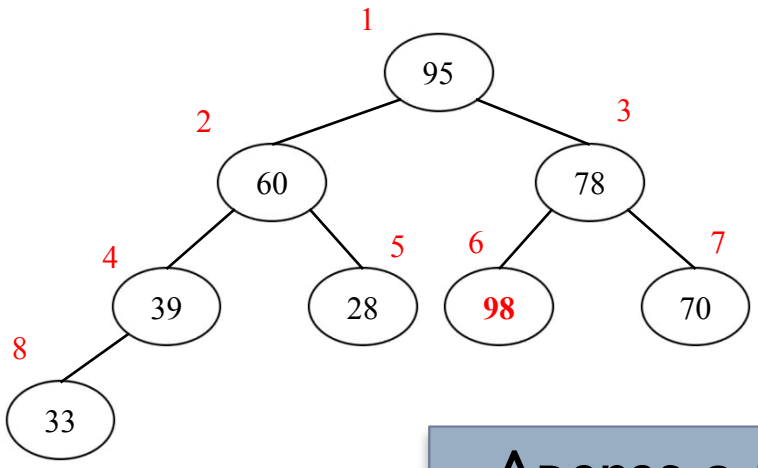
- Ao alterar a prioridade de um nó, é necessário rearrumar a heap para que ela respeite as prioridades
  - Um nó que tem a prioridade aumentada precisa "subir" na árvore
  - Um nó que tem a prioridade diminuída precisa "descer" na árvore

Exemplo:

Aumentar a prioridade do nó 6 de 66 para 98

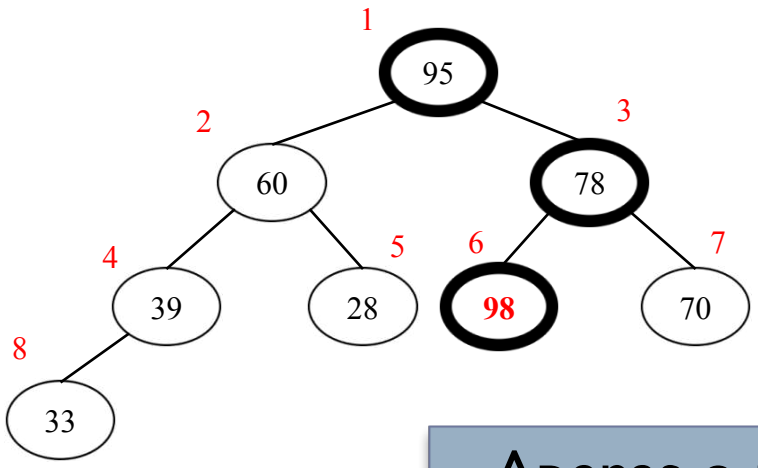


Exemplo:
Aumentar a prioridade do nó 6 de 66 para 98



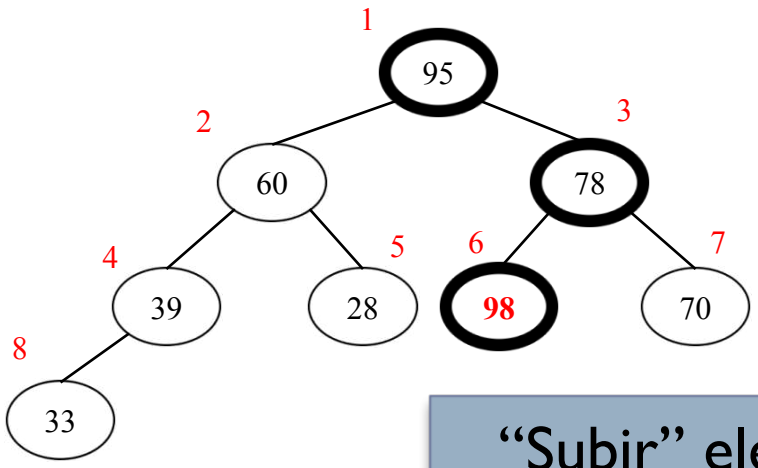
Apenas o ramo que vai do nó atualizado até a raiz é afetado – o restante da árvore permanece inalterado

Exemplo:
Aumentar a prioridade do nó 6 de 66 para 98



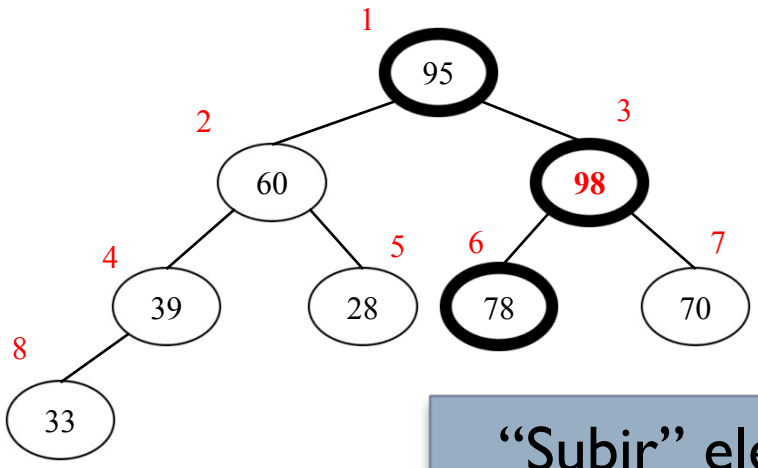
Apenas o ramo que vai do nó atualizado até a raiz é afetado – o restante da árvore permanece inalterado

Exemplo:
Aumentar a prioridade do nó 6 de 66 para 98



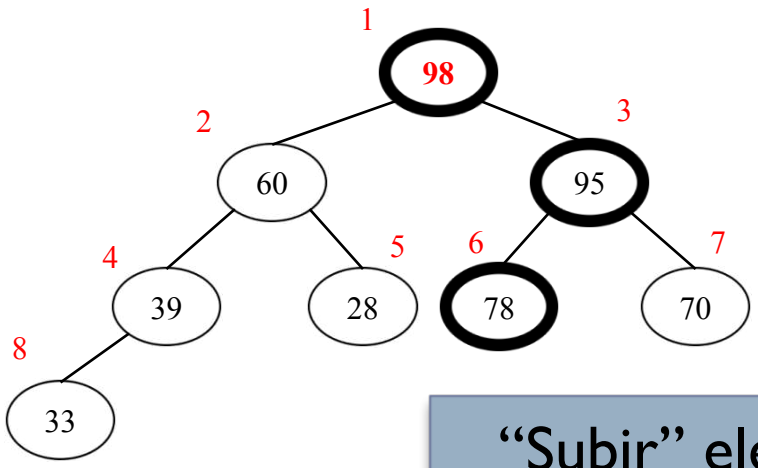
"Subir" elemento alterado na árvore, fazendo trocas com o nó pai, até que a árvore volte a ficar correta (todo nó pai tem prioridade maior que seus filhos)

Exemplo:
Aumentar a prioridade do nó 6 de 66 para 98



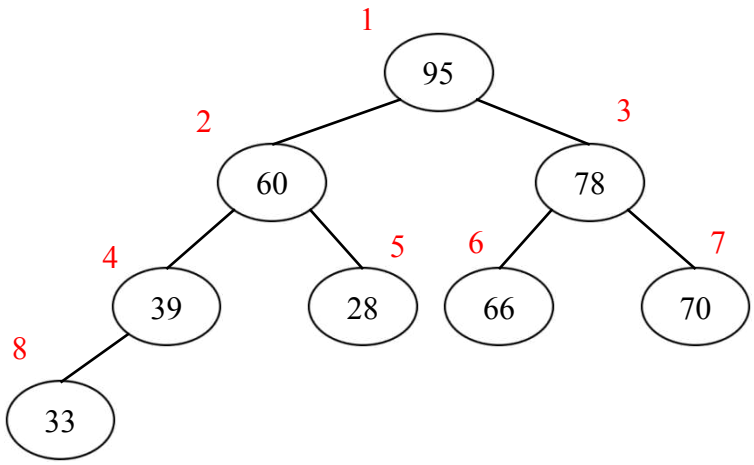
"Subir" elemento alterado na árvore, fazendo trocas com o nó pai, até que a árvore volte a ficar correta (todo nó pai tem prioridade maior que seus filhos)

Exemplo:
Aumentar a prioridade do nó 6 de 66 para 98

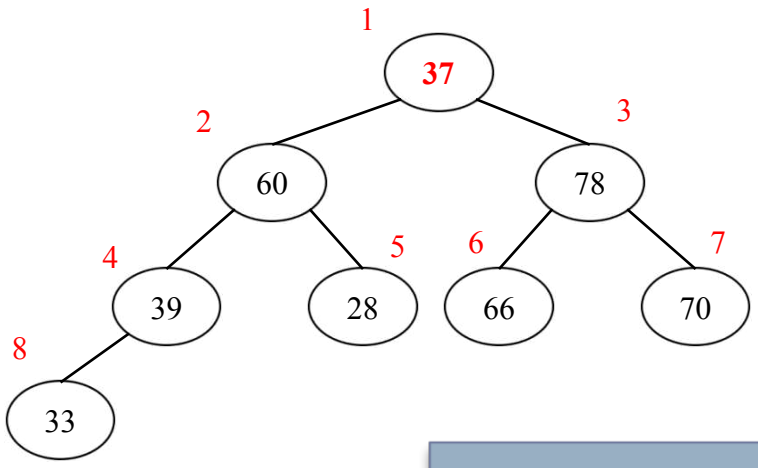


"Subir" elemento alterado na árvore, fazendo trocas com o nó pai, até que a árvore volte a ficar correta (todo nó pai tem prioridade maior que seus filhos)

Exemplo:
Diminuir a prioridade do nó 1 para 37

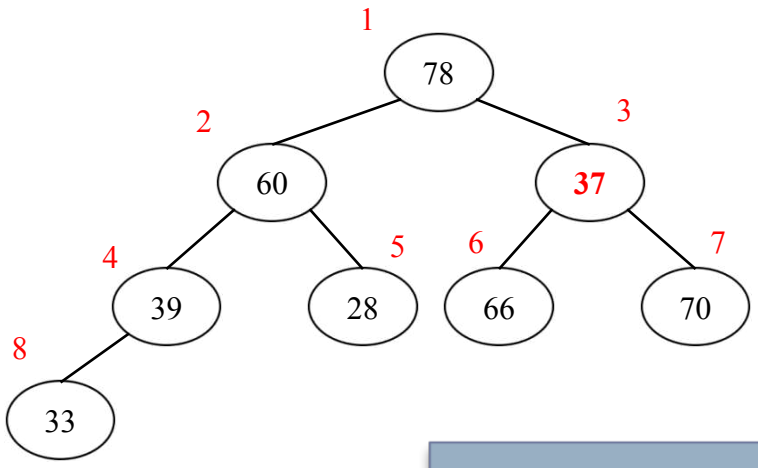


Exemplo:
Diminuir a prioridade do nó 1 para 37



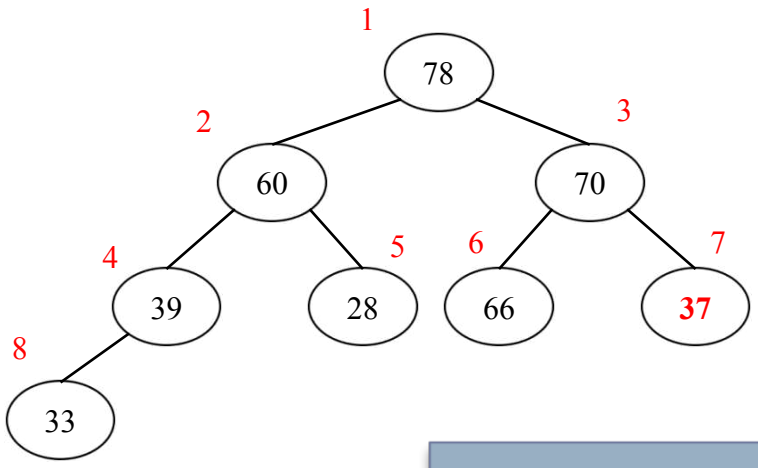
"Descer" elemento alterado na árvore, fazendo trocas **com o nó filho de maior prioridade**, até que a árvore volte a ficar correta

Exemplo:
Diminuir a prioridade do nó 1 para 37



"Descer" elemento alterado na árvore, fazendo trocas **com o nó filho de maior prioridade**, até que a árvore volte a ficar correta

Exemplo:
Diminuir a prioridade do nó 1 para 37



"Descer" elemento alterado na árvore, fazendo trocas **com o nó filho de maior prioridade**, até que a árvore volte a ficar correta

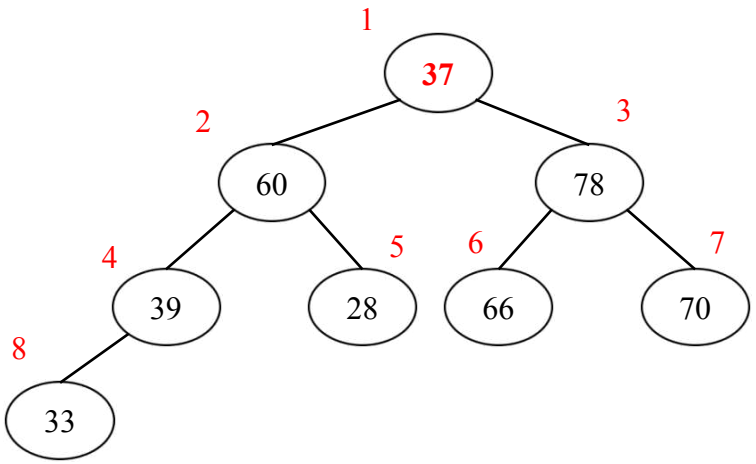
Algoritmo
Subir por um caminho da árvore

```
/* i indica a posição do elemento a ser revisto
    campo chave armazena a prioridade do nó
    A tabela está armazenada em T
    Notação T[i] <-> T[j] representa a troca de
     posição entre os nós i e j na tabela
*/
 procedimento subir(i)
   j := floor(i/2) //arredonda o valor da divisão
      para baixo
   se j ≥ 1 então
      se T[i].chave > T[j].chave então
         T[i] <-> T[j]
         subir(j)
```

Algoritmo

Descer por um caminho da árvore

```
/* i indica a posição do elemento a ser revisto
   n é o número de elementos da tabela
   campo chave armazena a prioridade do nó
   A tabela está armazenada em T
   Notação T[i] <-> T[j] representa a troca de
    posição entre os nós i e j na tabela
*/
 procedimento descer(i, n)
   j := 2 * i
   se j ≤ n então
      se j < n então
         se T[j+1].chave > T[j].chave então
            j := j + 1
      se T[i].chave < T[j].chave então
         T[i] <-> T[j]
         descer(j, n)
```

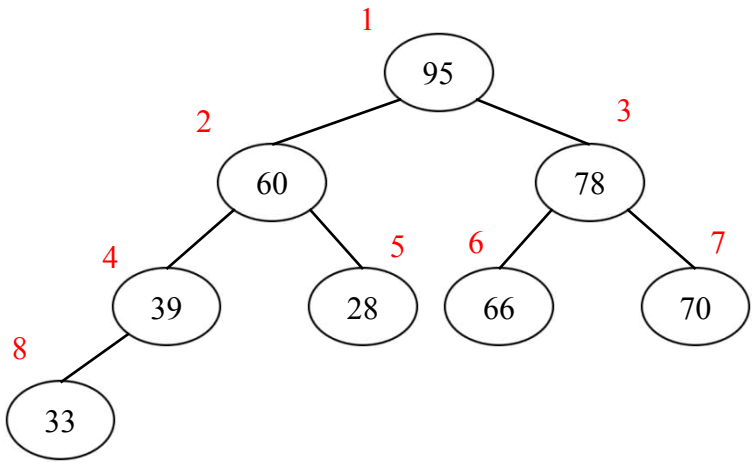


Inserção

- Tabela com n elementos
- Inserir novo elemento na posição n+1 da tabela
- Assumir que esse elemento já existia e teve sua prioridade aumentada
- Executar algoritmo de subida na árvore para corrigir a prioridade e colocar o novo elemento na posição correta

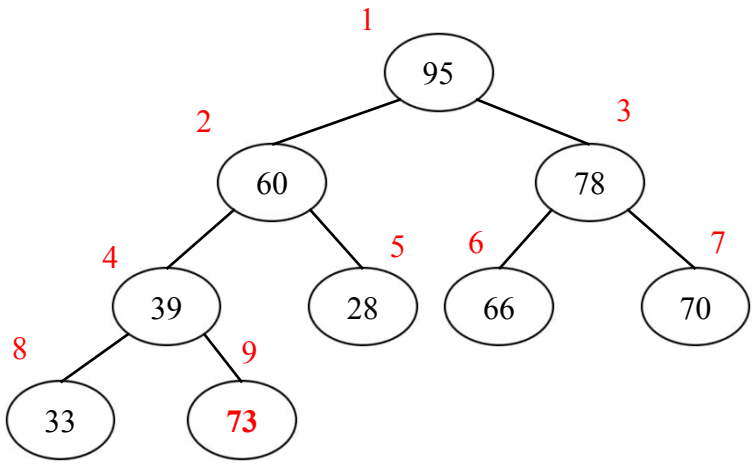
Exemplo:
Inserir elemento 73

95	60	78	39	28	66	70	33



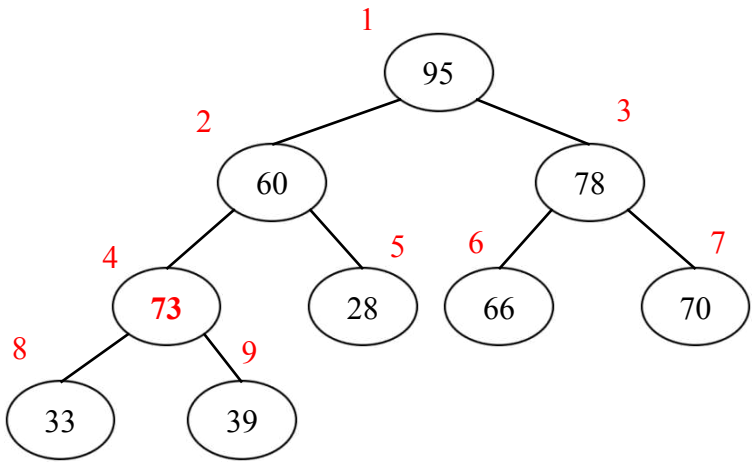
Exemplo:
Inserir elemento 73

95	60	78	39	28	66	70	33	**73**



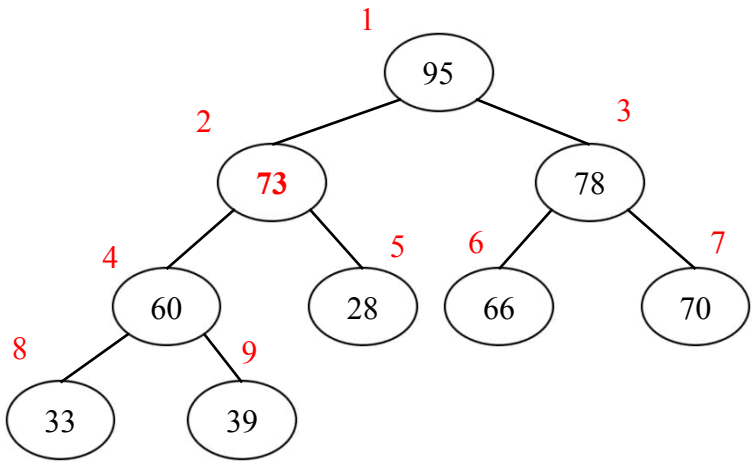
Exemplo:
Inserir elemento 73

95	60	78	**73**	28	66	70	33	**39**



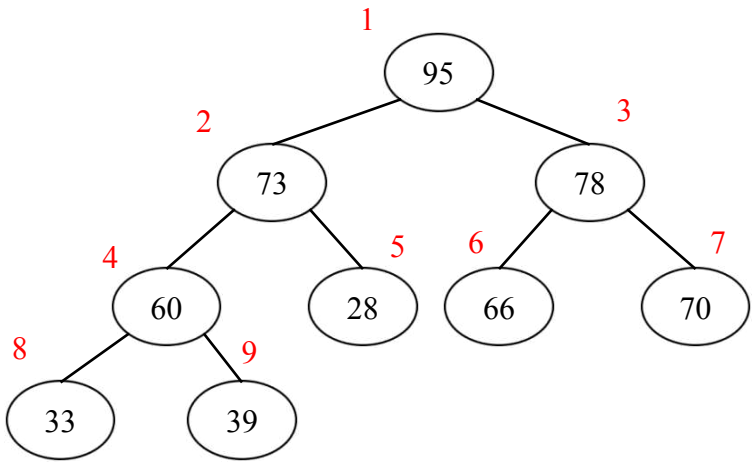
Exemplo:
Inserir elemento 73

95	**73**	78	**60**	28	66	70	33	39



Exemplo:
Inserir elemento 73

95	**73**	78	60	28	66	70	33	39



Algoritmo
Inserção em Lista de Prioridade

```
/* n é o número de elementos da tabela
   novo é o novo elemento a ser inserido
*/
 procedimento inserir(n, novo)
   T[n+1] := novo
   n := n + 1
   subir(n)
```

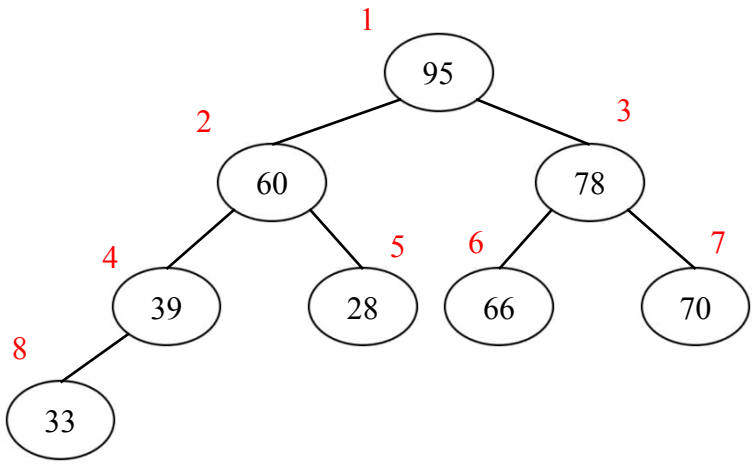
Remoção do Elemento mais Prioritário

- Remover o primeiro elemento da tabela
- Preencher o espaço vazio deixado por ele com o último elemento da tabela
- Executar o algoritmo de descida na árvore para corrigir a prioridade desse elemento

Exemplo

Remover Elemento mais Prioritário

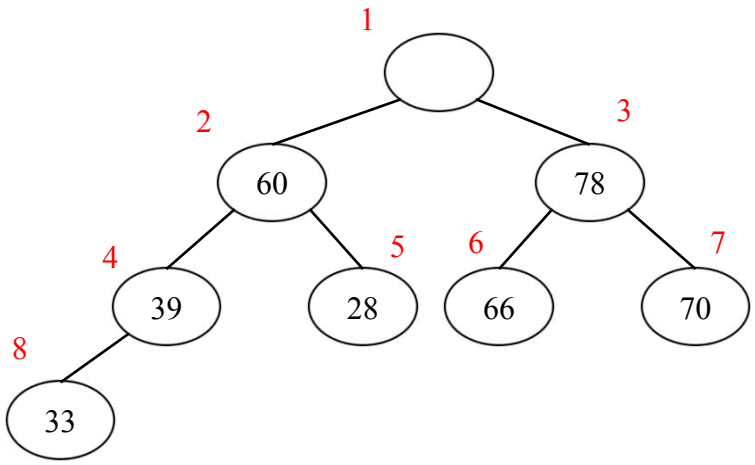
95	60	78	39	28	66	70	33



Exemplo

Remover Elemento mais Prioritário

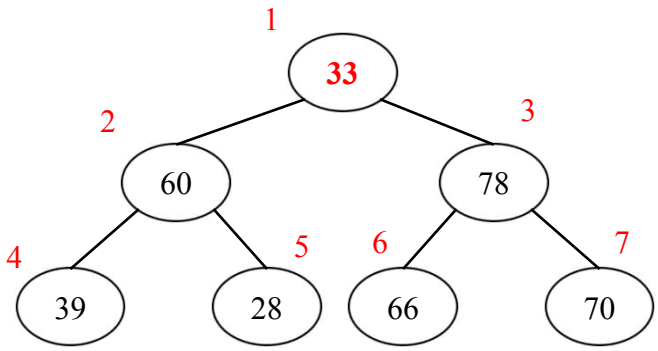
	60	78	39	28	66	70	33



Exemplo

Remover Elemento mais Prioritário

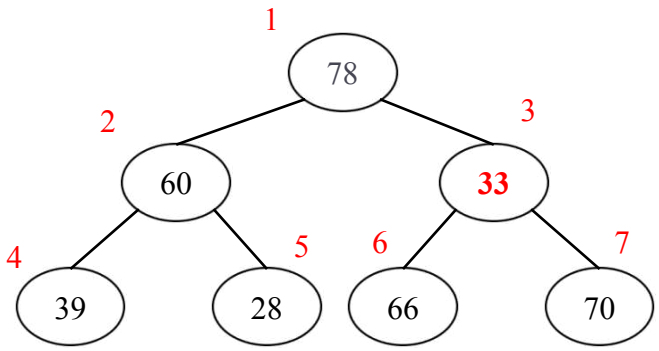
33	60	78	39	28	66	70



Exemplo

Remover Elemento mais Prioritário

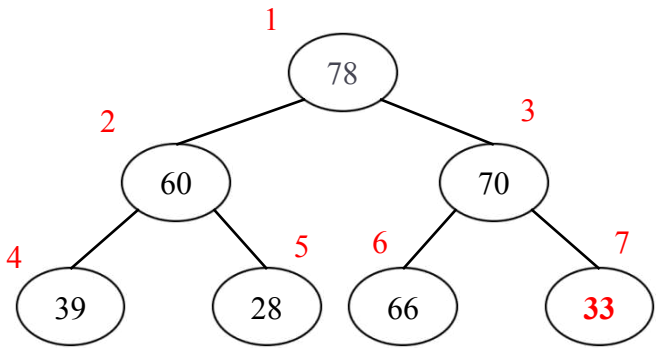
78	60	**33**	39	28	66	70



Exemplo

Remover Elemento mais Prioritário

78	60	**70**	39	28	66	**33**



Algoritmo
Remoção em Lista de Prioridade

```
/* n é o número de elementos da tabela
   agir implementa quaisquer operações que a
    aplicação precise realizar com o elemento
    mais prioritário
*/
 procedimento remover( )
   agir(T[1])
   T[1] := T[n]
   n := n - 1
   descer(1, n)
```

Construção de Lista de Prioridades

- Dada uma lista L de elementos para a qual se deseja construir uma heap H, há duas alternativas

1) Considerar uma heap vazia e ir inserindo os elementos de L um a um em H

2) Considerar que a lista L é uma heap, e corrigir as prioridades.

  - Assumir que as prioridades das folhas estão corretas (pois eles não têm filhos, então satisfazem à propriedade de terem prioridade maior que seus filhos)
  - Acertar as prioridades dos nós internos realizando descidas quando necessário

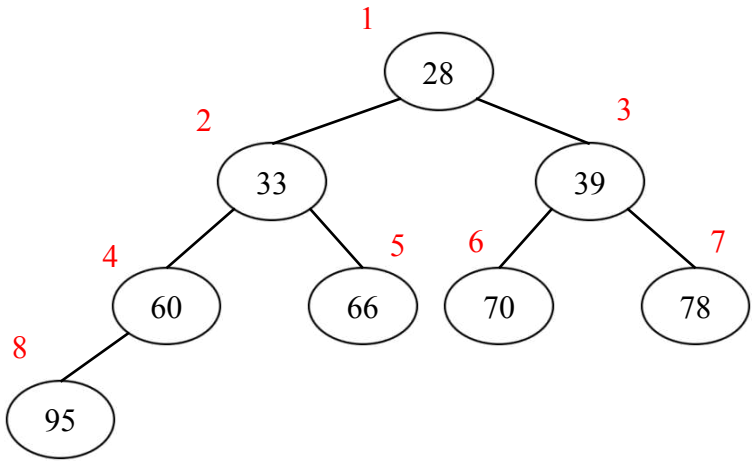
Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

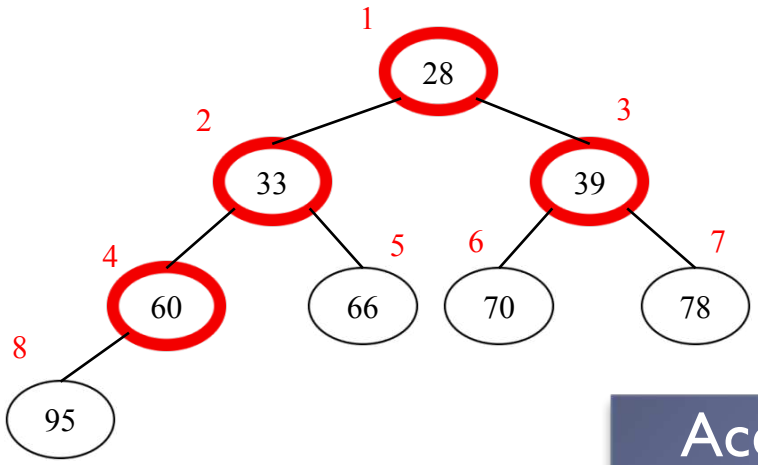
28	33	39	60	66	70	78	95



Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	39	60	66	70	78	95

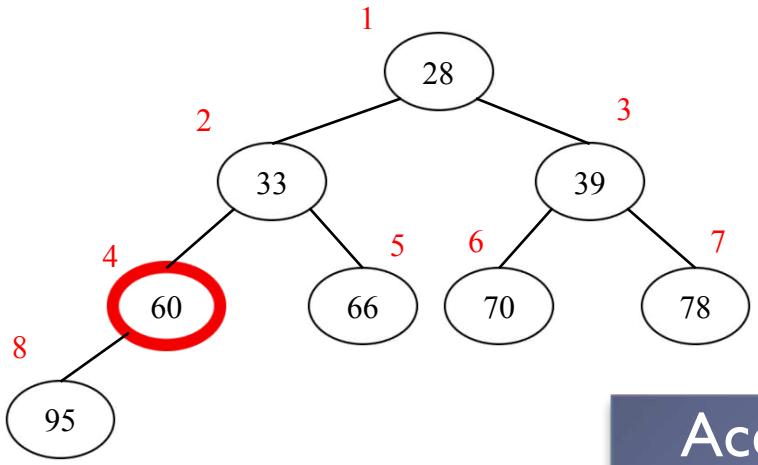


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	39	60	66	70	78	95

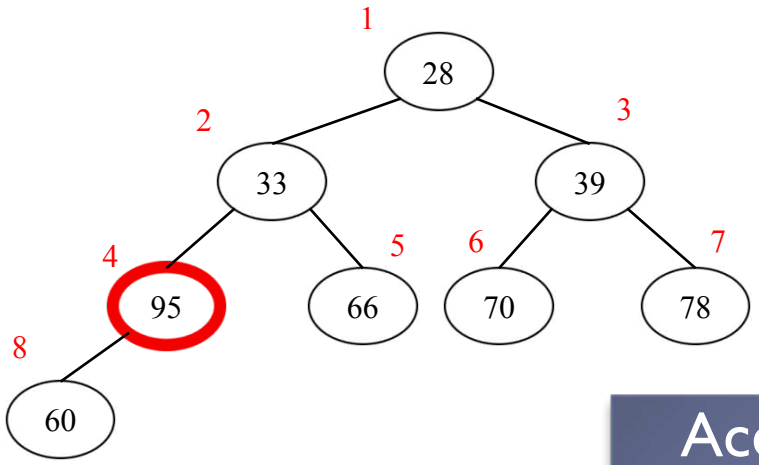


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	39	95	66	70	78	60

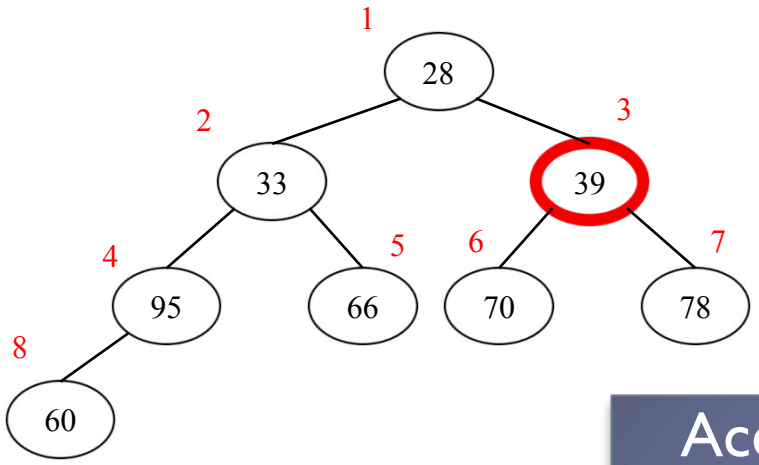


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	39	95	66	70	78	60

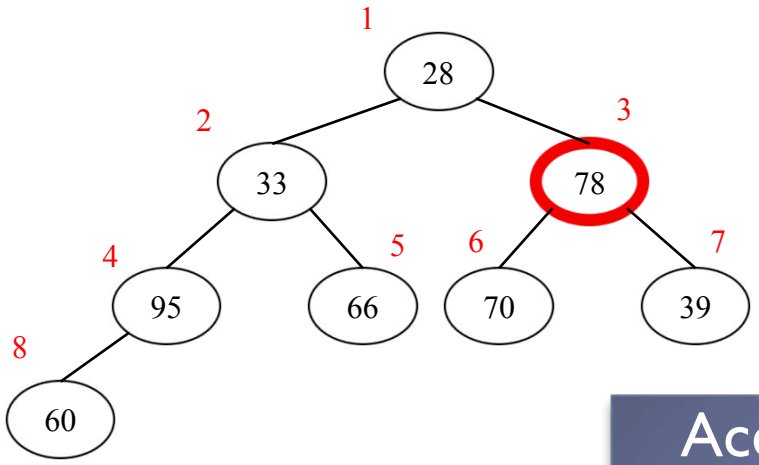


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	78	95	66	70	39	60

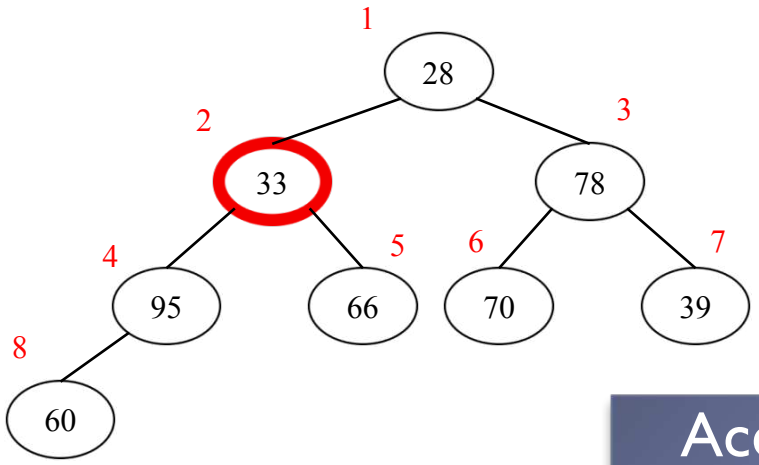


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	33	78	95	66	70	39	60

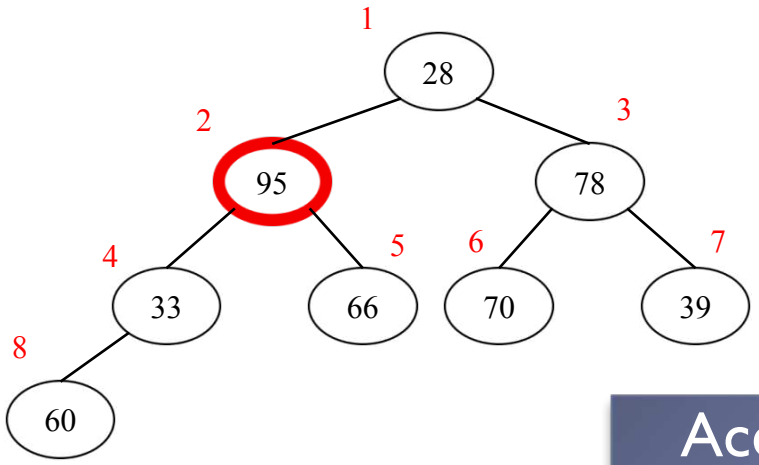


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	95	78	33	66	70	39	60

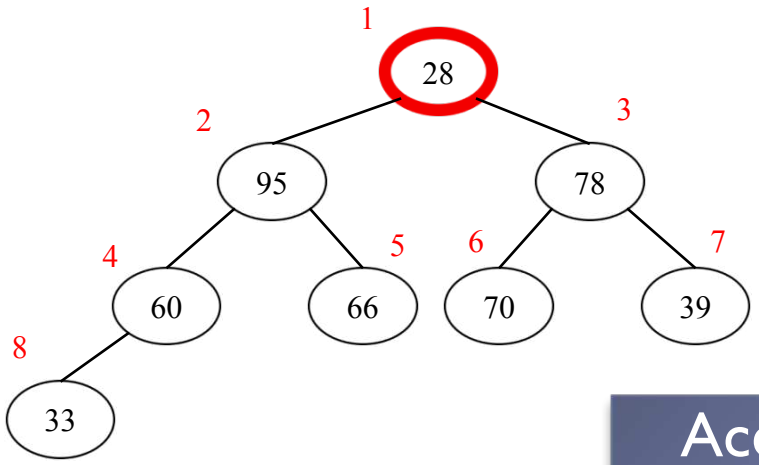


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

28	95	78	60	66	70	39	33

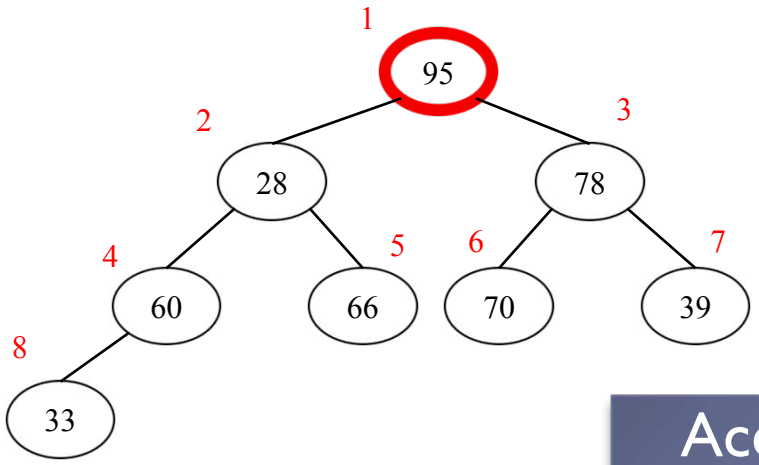


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

95	28	78	60	66	70	39	33

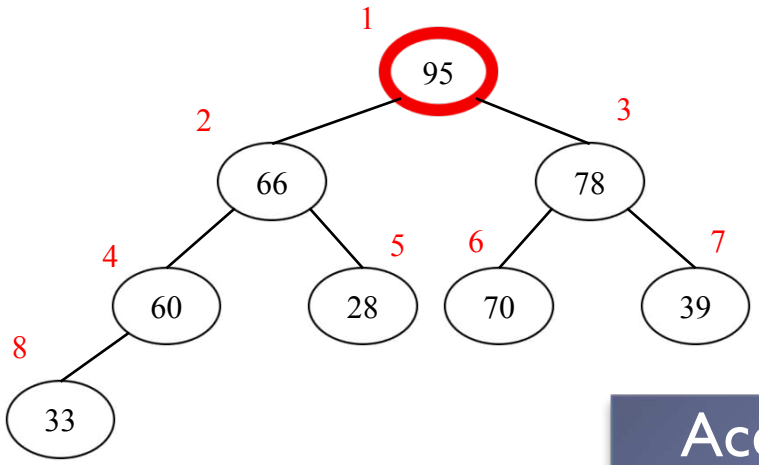


Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exemplo
Construção de Heap

- Construir uma Heap a partir da lista 28, 33, 39, 60, 66, 70, 78, 95

95	66	78	60	28	70	39	33



Acertar prioridade do elemento n/2 até o elemento 1, nessa ordem

Exercícios

1. Verificar se essas sequências correspondem ou não a um heap

33 32 28 31 26 29 25 30 27

36 32 28 31 29 26 25 30 27

33 32 28 30 29 26 25 31 27

35 31 28 33 29 26 25 30 27

Exercícios

2. Seja o heap especificado a seguir: 92 85 90 47 71 34 20 40 46. Sobre esse heap, realizar as seguintes operações:

(a) Inserir os elementos 98, 75, 43

(b) Remover o elemento de maior prioridade (sobre o heap original)

(c) Remover o elemento de maior prioridade (sobre o heap resultante do exercício (b)

(d) Alterar a prioridade do 5°. nó de 71 para 93 (sobre o heap original)

(e) Alterar a prioridade do 5°. nó de 71 para 19 (sobre o heap original)

Exercícios

3. Construa um heap baseado na seguinte lista:

18 25 41 34 14 10 52 50 48

Usar o método de construção que assume que as folhas estão corretas ao invés de inserir elemento a elemento