Aula 4
HeapSort
Alteração de prioridades

Revisão

95	60	78	39	28	66	70	33

- É um heap?

Introdução

- Aumento ou diminuição de prioridade
  - Aumento: “subida” do nó na árvore binária
  - Diminuição: “descida” do nó na árvore

Algoritmo de Subida

- Método:
  - Prioridade de **v** é aumentada
  - Se a prioridade do pai de **v** for menor, trocar de posição
  - Repetir essa operação até que o pai seja maior ou que **v** seja a raiz da árvore

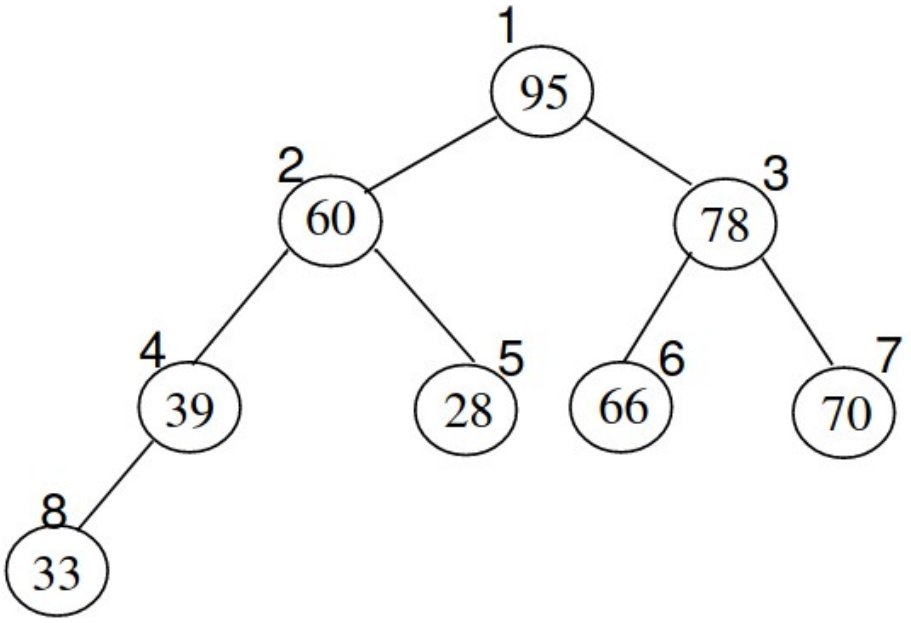
Algoritmo de Subida

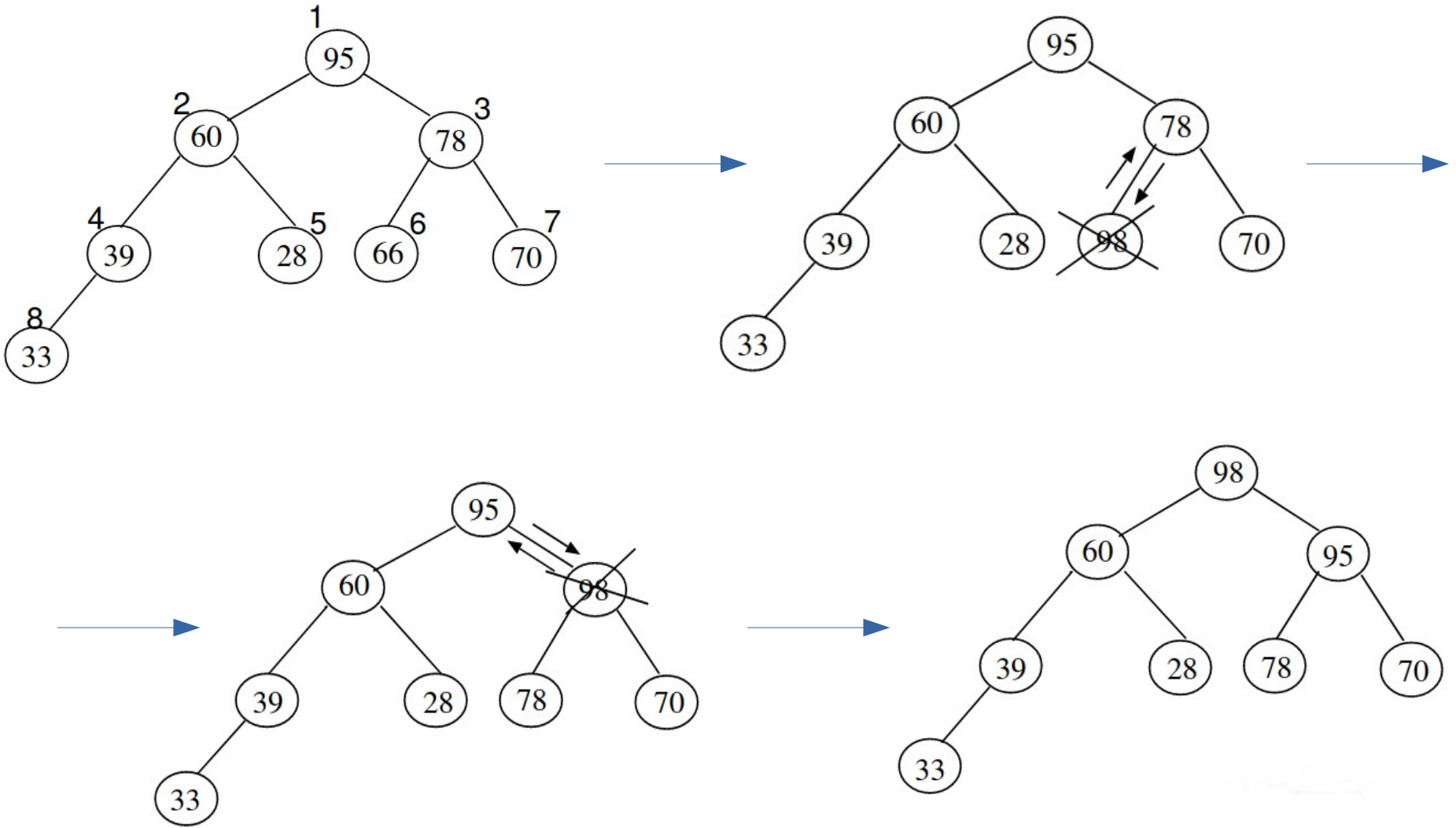
- Algoritmo

```
procedimento subir (i)
    j := ⌊i/2⌋
    se  j≥1 então
        se T[i].chave > T[j].chave então
            T[i] ⇔ T[j]
            subir (j)
```

Exemplo 1

- Aumentar a prioridade do nó 6 de 66 para 98




1
95
2
60
3
78
4
39
5
28
6
66
7
70
8
33
95
60
78
39
28
98
70
33
95
60
98
39
28
78
70
33
98
60
95
39
28
78
70
33

Exercício 1

- Dado o heap a seguir:

92 85 90 47 31 34 20 40 46

  - Aplique o algoritmo de subida para alterar a prioridade do 5º nó de 31 para 93.

Algoritmo de Descida

- Método:
  - Prioridade de **v** é diminuída
  - Se a prioridade do filho de **v** for maior, trocar de posição
  - Repetir essa operação até que os filhos sejam maiores ou que **v** seja uma folha da árvore

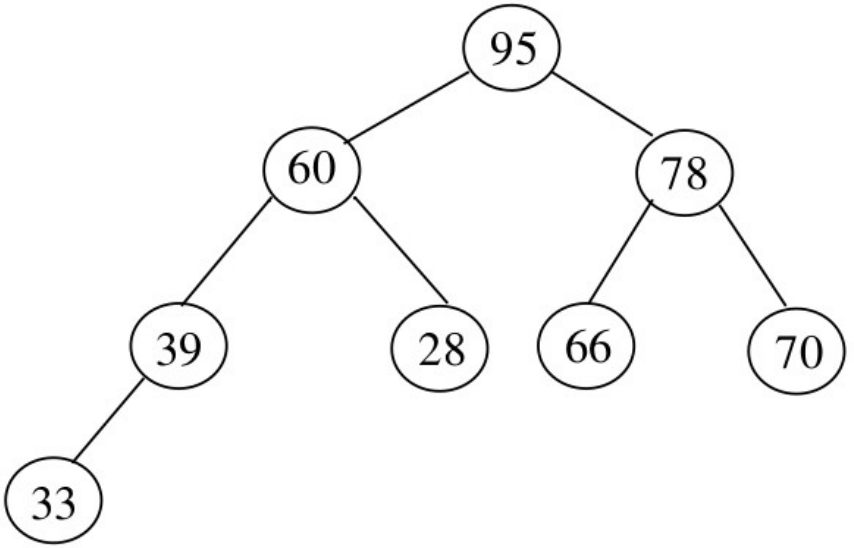
Algoritmo de Descida

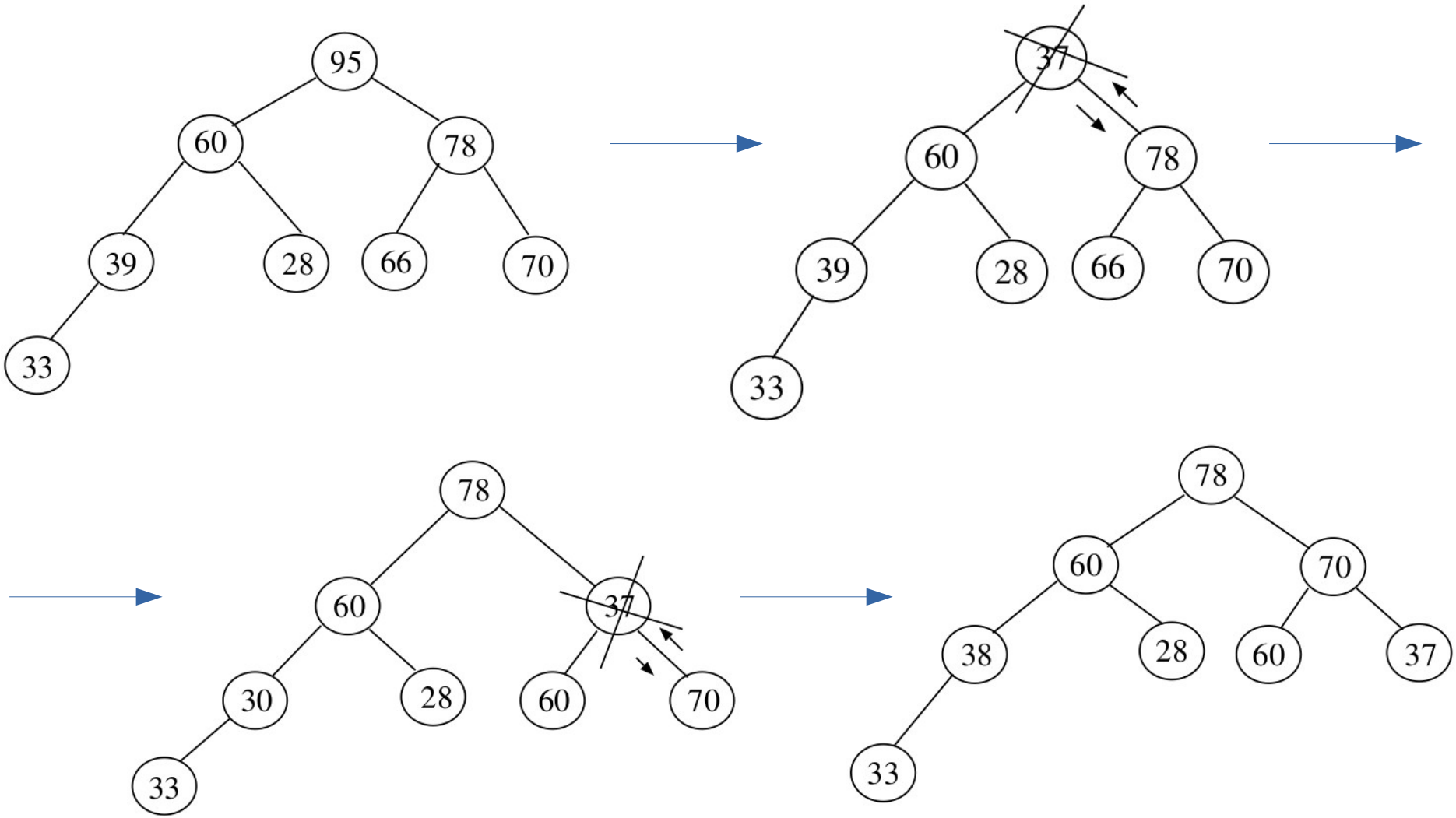
- Algoritmo

```
j : = 2 x 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)
```

Exemplo 2

- Diminuir a prioridade do nó 1 de 95 para 37




95
60
78
39
28
66
70
33
37
60
78
39
28
66
70
33
78
60
37
30
28
60
70
33
78
60
70
38
28
60
37
33

Exercício 2

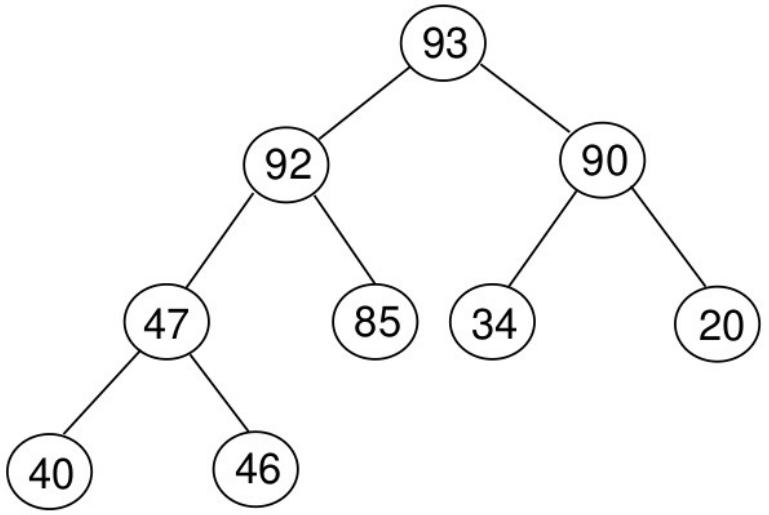
- Dado o heap a seguir:

92 85 90 47 31 34 20 40 46

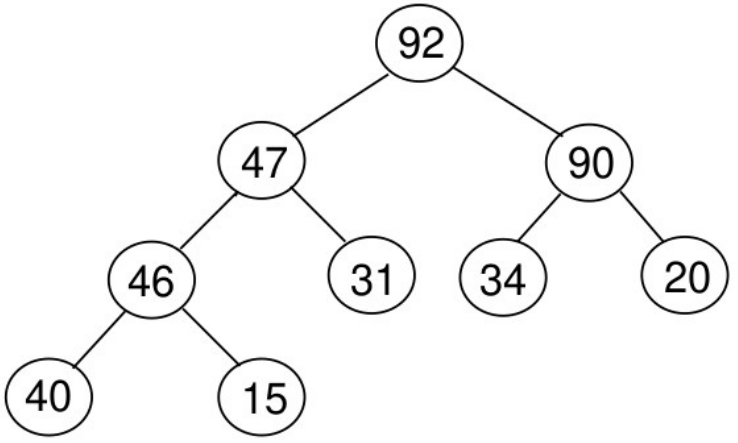
  - Aplique o algoritmo de descida para alterar a prioridade do 2º nó de 85 para 15.

Respostas

- Exercício 1



- Exercício 2



Construção do Heap

- Rearranjar os elementos um vetor de inteiros v[1..m] para que ele se torne um heap

```
static void
constroiHeap (int m, int v[])
{
   for (int k = 1; k < m; ++k) {
      // v[1..k] é um heap
      int f = k+1;
      while (f > 1 && v[f/2] < v[f]) {  // 5
         troca (v[f/2], v[f]);          // 6
         f /= 2;                        // 7
      }
   }
}
```

Construção do Heap

3	8	4	2	1	5	7	6

Construção do Heap

3	8	4	2	1	5	7	6

8	6	7	3	1	4	5	2

Algoritmo da Peneira

- Descida para a raiz do heap

```
static void
peneira (int m, int v[]) {
   int f = 2;
   while (f <= m) {
      if (f < m && v[f] < v[f+1]) ++f;
      // f é o filho mais valioso de f/2
      if (v[f/2] >= v[f]) break;
      troca (v[f/2], v[f]);
      f *= 2;
   }
}
```

Construção do Heap 2

- Possível fazer a construção em O(n)
  - Algoritmo

```
void constroiHeap_2 (int m, int v[]) {
   for (int p = m/2; p >= 1; --p)
      peneira_2 (p, m, v);
}
```

HeapSort

- Duas fases
  - a primeira transforma o vetor em heap
  - a segunda rearranja o heap em ordem crescente

```
void
heapsort (int n, int v[])
{
   constroiHeap (n, v);
   for (int m = n; m >= 2; --m) {
      troca (v[1], v[m]);
      peneira (m-1, v);
   }
}
```

HeapSort

- Animação

https://www.cs.usfca.edu/~galles/visualization/HeapSort.html