ACH2002

Aula 18

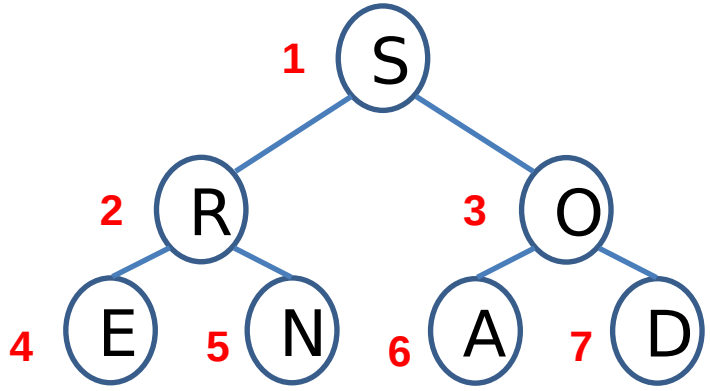
Heaps (cont.)

EACH USP

Lembrando parte da aula 15 (heapsort)

Heap

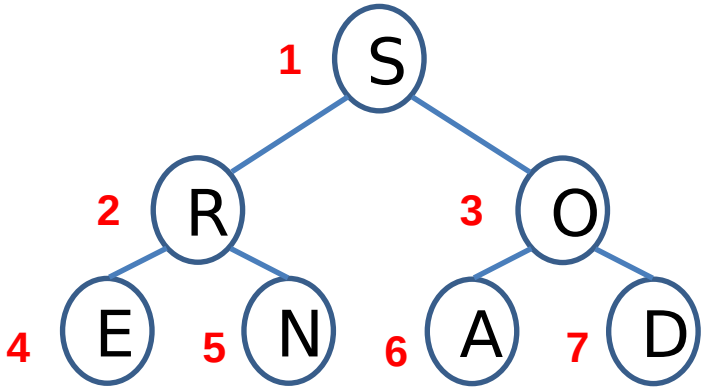
- Estrutura de dados para implementar fila de prioridades
- Definição:
  - um ***heap*** é uma estrutura de dados contendo uma sequência de itens com chaves: *c[1],c[2],..., c[n]* tal que *c[i]* ≥ *c[2i]* e *c[i]* ≥ *c[2i+1]* , para todo *i=1, 2, ..., n/2*.
  - sequência é facilmente visualizada se for desenhada como uma **árvore binária completa**: as linhas que saem de uma chave levam a duas chaves menores de nível inferior.



Heap

- Algoritmos:

1	2	3	4	5	6	7
S	R	O	E	N	A	D

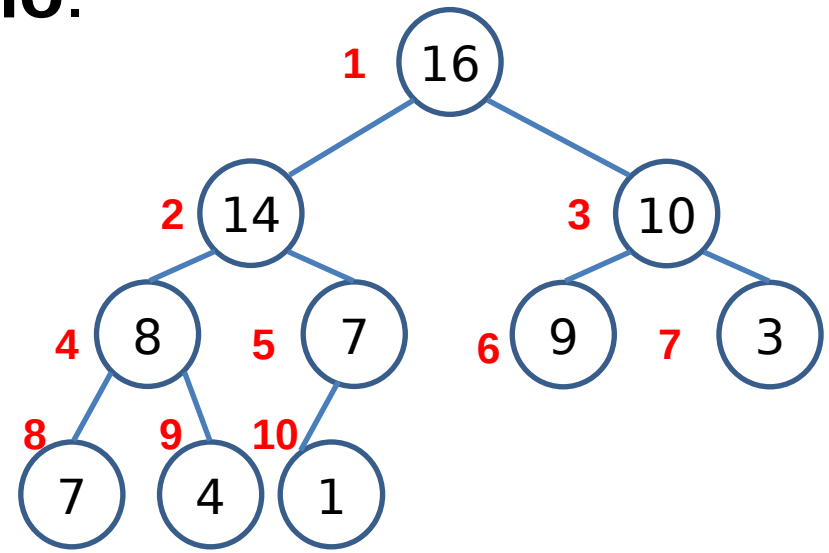



```
pai(i)
   retorna ⌊i/2⌋

filho_esquerdo(i)
   retorna 2i

filho_direito(i)
   retorna 2i+1
```

Complexidades dessas operações:
O(1) !!!

Heap

- Dado um arranjo A que representa um ***heap***, definimos dois tipos de ***heap***:
  - ***heap mínimo***: *A[pai(i)] ≤ A[i]*
  - ***heap máximo***: *A[pai(i)] ≥ A[i]*
- ***HeapSort*** usa ***heap* máximo**.
- Um exemplo com números:

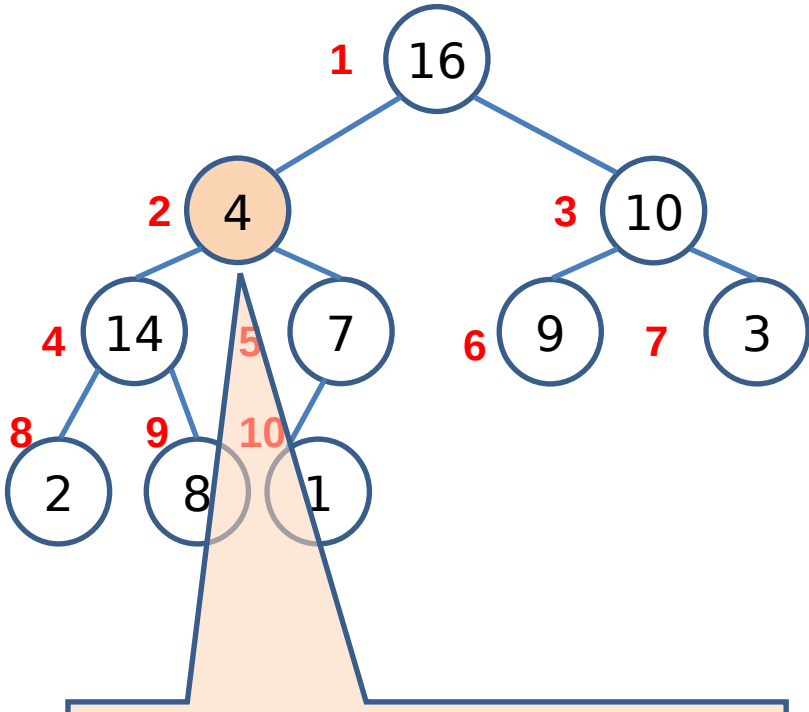


1	2	3	4	5	6	7	8	9	10
16	**14**	**10**	**8**	**7**	**9**	**3**	**7**	**4**	**1**

Refaz Heap Máximo

- Algoritmo:

```
refazHeapMaximo(A[],i)
l ← filho_esquerdo(i)
r ← filho_direito(i)
se l ≤ tamanho-do-heap[A] e A[l] > A [i]
       maior ← l
senão
       maior ← i
fim se
se r ≤ tamanho-do-heap[A] e A[r] > A[maior]
      maior ← r
fim se
se maior ≠ i
       trocar A[i] ↔ A[maior]
       refazHeapMaximo(A,maior)
fim se
```



A[2] violando o *heap* porque é menor que seus filhos!

Solução:

Troca A[2] com o maior dos seus dois filhos!

Refaz Heap Máximo

Corresponde à função MAX-HEAPIFY do livro do Cormen (que também descreve heap máximo)

Tamanho do heap

```
MAX-HEAPIFY(A, i)
1   l = LEFT(i)
2   r = RIGHT(i)
3   if l ≤ A.heap-size and A[l] > A[i]
4       largest = l
5   else largest = i
6   if r ≤ A.heap-size and A[r] > A[largest]
7       largest = r
8   if largest ≠ i
9       exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

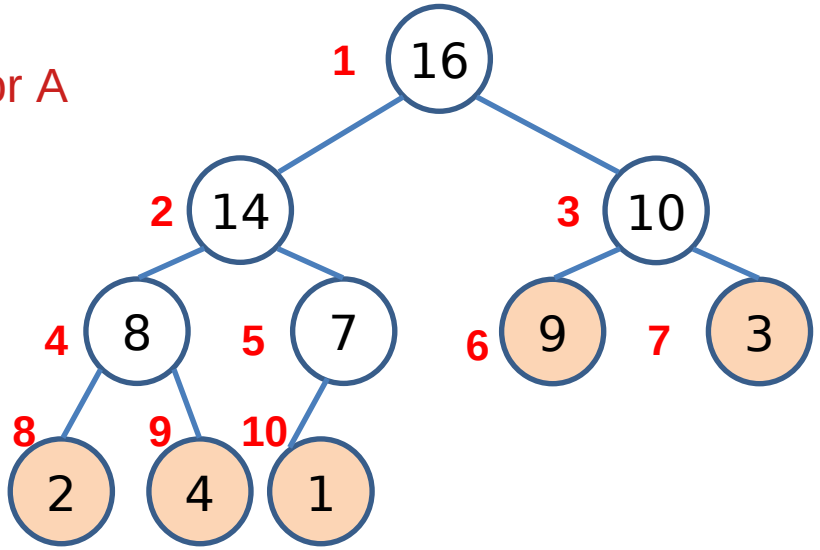
Verifica quem é o maior: i ou um de seus filhos (largest)

Construção do Heap

Corresponde à função BUILD-MAX-HEAP do livro do Cormen (O(n))

Tamanho do vetor A

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

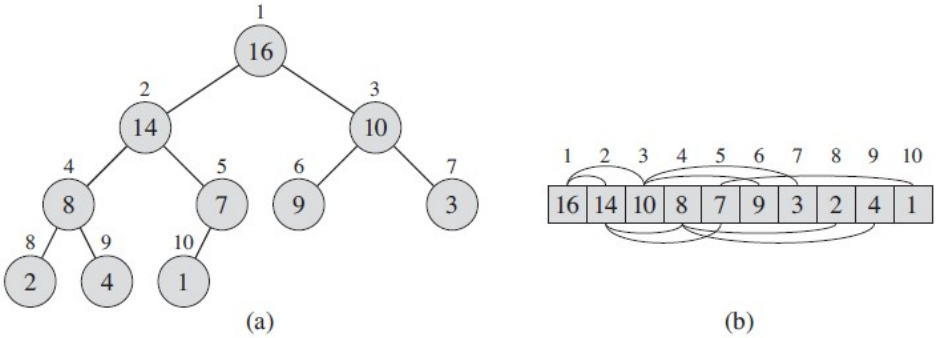



```
constroiHeapMaximo(A[])
tamanhoHeap ← tamanho[A]
para i ← ⌊tamanho[A]/2⌋ até 1, com decremento -1
  faça refazHeapMaximo(A,i)
```

Hoje: extração do heap
e
incremento de chave

Extração do Heap

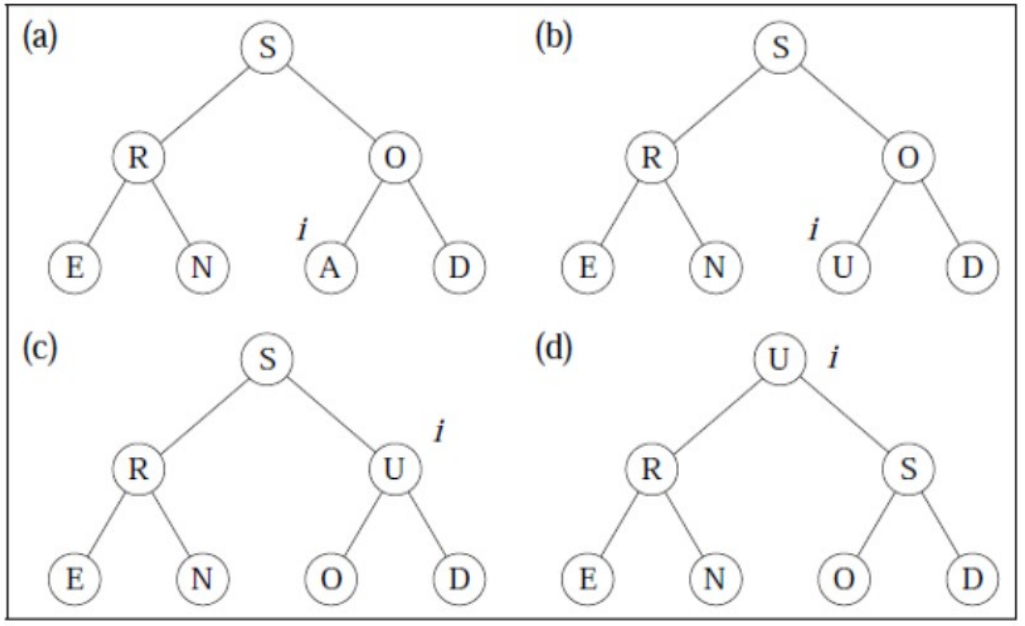
Na aula 15 estava embutido no HeapSort. Aqui vamos implementar à parte.



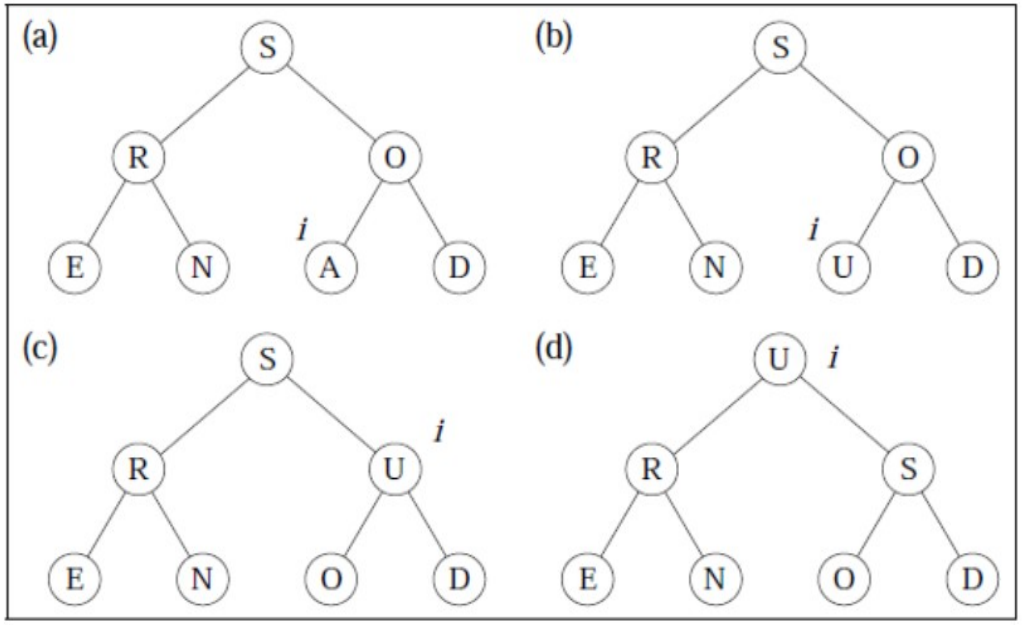

```
HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2        error "heap underflow"
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

Complexidade: O(lg n)

Incremento de chave



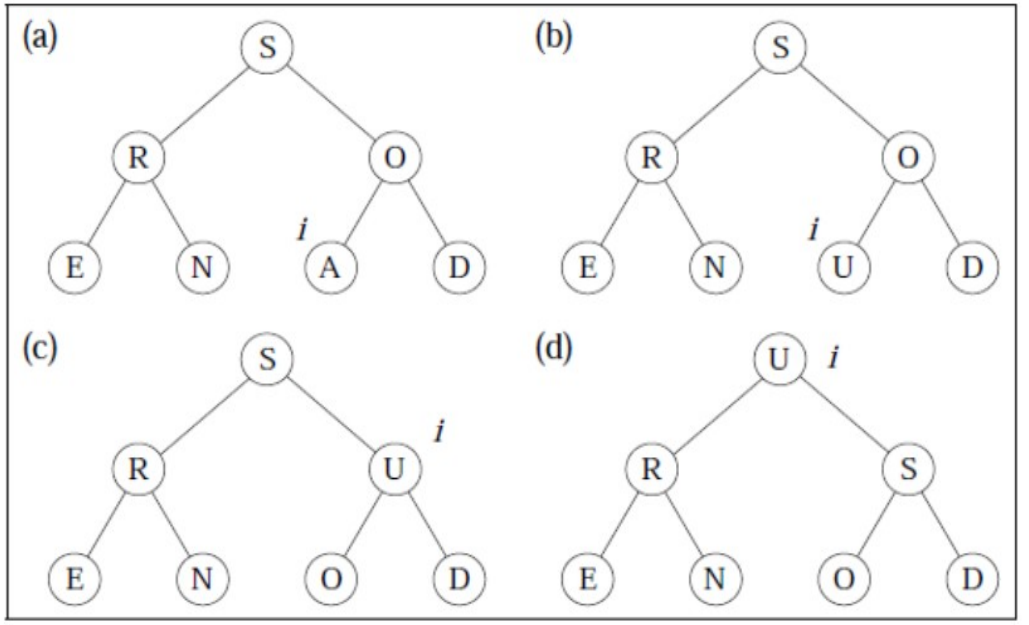
Incremento de chave



HEAP-INCREASE-KEY(A, i, key)

```
1  if key < A[i]
2        error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5        exchange A[i] with A[PARENT(i)]
6        i = PARENT(i)
```

Incremento de chave

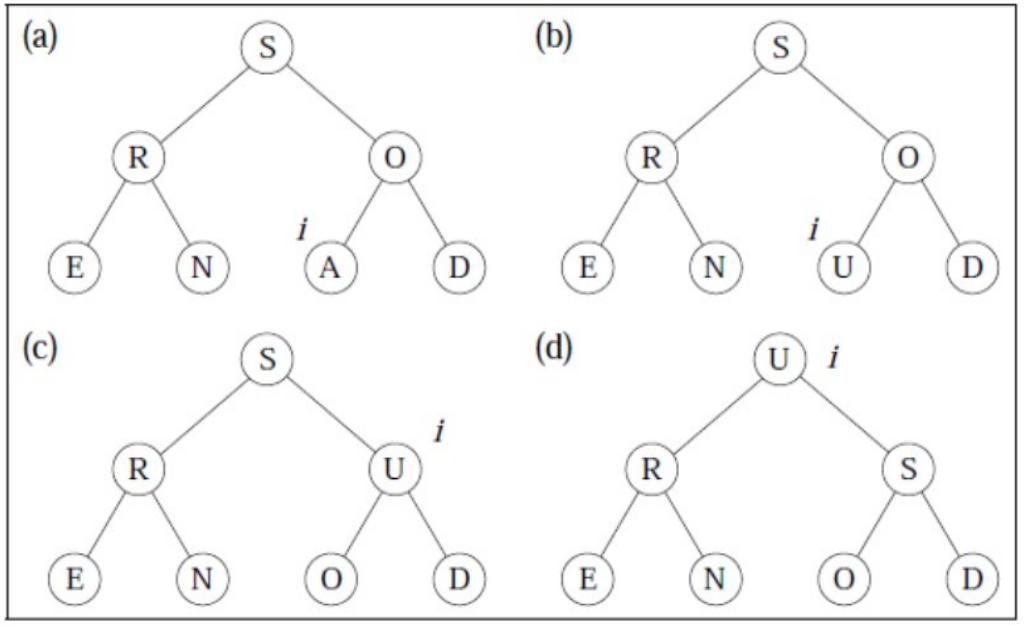


HEAP-INCREASE-KEY(A, i, key)

```
1  if key < A[i]
2        error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5        exchange A[i] with A[PARENT(i)]
6        i = PARENT(i)
```

Complexidade: ?

Incremento de chave



HEAP-INCREASE-KEY(A, i, key)

```
1  if key < A[i]
2        error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5        exchange A[i] with A[PARENT(i)]
6        i = PARENT(i)
```

Complexidade: O(lg n)

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - 3a. ed. Edição Americana. Editora Campus, 2002. Cap 6