Heaps

Consultar pdf: EstrDadosAlgoritmos-ClaudioEsperanca.pdf , 0_Inicio_aula16

1) Heaps são estruturas usadas em Listas de Prioridade e HeapSort. A lista de prioridade pode ser implementada usando Lista, Lista Ordenada, Árvore balanceada e Heap. Faça um quadro comparativo de pior caso para as operações: Seleção, Inserção, Remoção, Alteração (de prioridade) e Construção, para cada uma das implementações citadas.

Listas de Prioridade

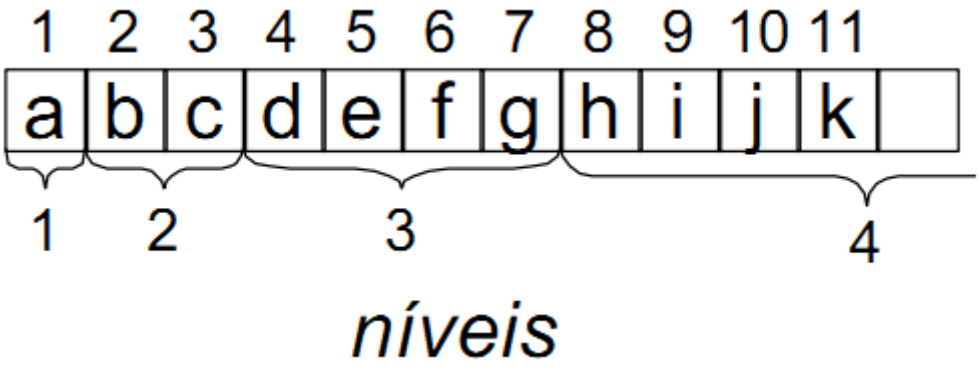
- Implementação

Operação	**Lista**	**Lista Ordenada**	**Árvore Balanceada**	**Heap**
Seleção	$O(n)$	$O(1)$	$O(\log n)$	$O(1)$
Inserção	$O(1)$	$O(n)$	$O(\log n)$	$O(\log n)$
Remoção (do menor)	$O(n)$	$O(1)$	$O(\log n)$	$O(\log n)$
Alteração (de prioridade)	$O(n)$	$O(n)$	$O(\log n)$	$O(\log n)$
Construção	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$

2) Diferentemente de árvores binárias de busca, heaps são implementados usando arrays.

2.1) Para a figura abaixo, onde o vetor começa em 1, determine, mostrando como fez para chegar a solução:

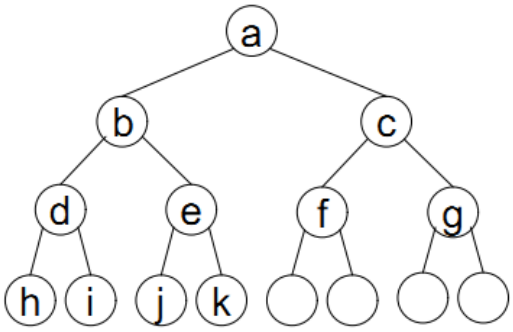
a) Quem é o pai do elemento com índice 5 (e)
b) Quem é o filho esquerdo do elemento com índice 5(e)
c) Quem é o filho direito do elemento com índice 5(e)

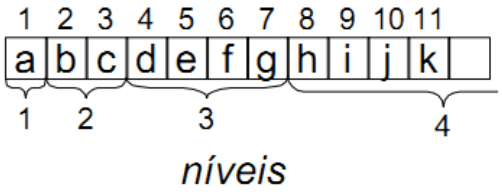


2.2) Agora supondo que este vetor comece em zero, determine, mostrando como fez para chegar a solução:

a) Quem é o pai do elemento d
b) Quem é o filho esquerdo do elemento d
c) Quem é o filho direito do elemento d

- Dado um nó armazenado no índice i, é possível computar o índice
  - do nó filho esquerdo de i : $2\,i$
  - do nó filho direito de i : $2\,i + 1$
  - do nó pai de i : i div 2
- Para armazenar uma árvore de altura h precisamos de um array de $2^h - 1$ (número de nós de uma árvore cheia de altura h)





```c
typdef struct {
    int *v;
    int tam;
} Arv;

// retorna o índice do filho esquerdo
int esq(int i) {
    return 2*i + 1;
}
// retorna o índice do filho direito
int dir(int i) {
    return 2*i + 2;
}
// retorna o índice do pai
int pai(int i) {
    return (i - 1)/2;
}
```

3) Qual a caraterística de um Heap de Máximo e de Mínimo.

5) Se um nó tem seu valor alterado, a manutenção das propriedades do Heap pode requerer que nó migre na árvore:

- para cima (se ele diminuir de valor)
- para baixo (se ele aumentar de valor)

Para cada uma dessas situações utiliza-se um algoritmo de migração:

- subir (i, n, H) migra o nó i para cima no heap H
- descer (i,n,H) migra o nó i para baixo no heap H (sendo n o número total de nós da árvore/heap)

Para inserir, basta colocar o novo valor na posição n + 1 do heap e chamar Subir.

Para remover o menor valor, basta substituir a raiz (H [1]) por H (n) e chamar Descer.

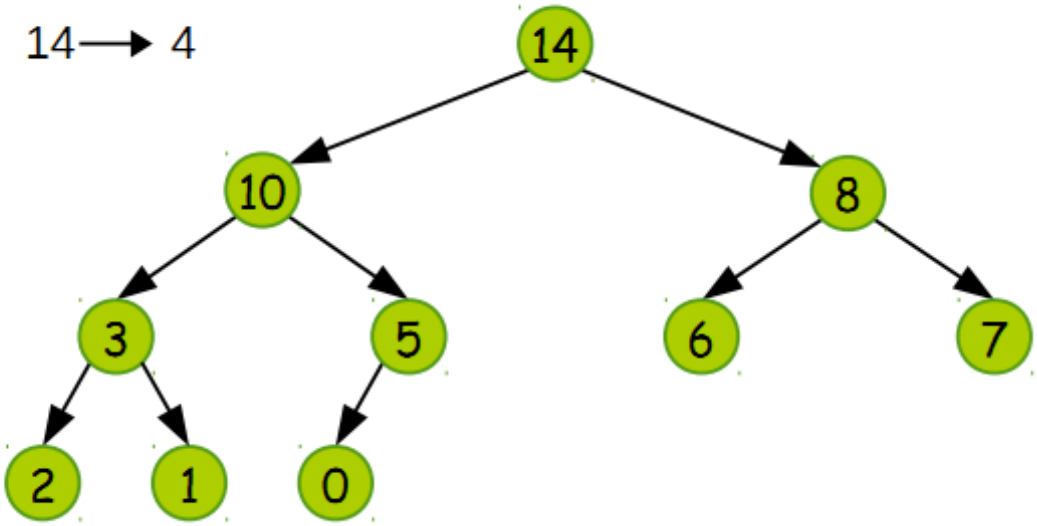
Inserção e Remoção num Heap

- Claramente, *Subir* e *Descer* têm complexidade $O(\log n)$ já que percorrem no máximo um caminho igual à altura da árvore que é completa
- Para inserir, basta colocar o novo valor na posição $n + 1$ do heap e chamar *Subir*
- Para remover o menor valor, basta substituir a raiz ($H[1]$) por H (n) e chamar *Descer*

```
proc Inserir (x, n, H [1 .. n + 1]) {
    n ← n + 1
    H [n] ← x
    Subir (n, n, H)
}
proc RemoverMinimo (n, H [1 .. n]) {
    H [1] ← H [n]
    n ← n – 1
    Descer (1, n, H)
}
```

214

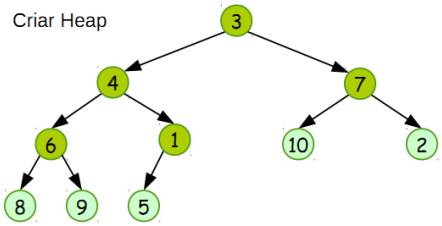
6) Supondo que a prioridade do nó raiz foi diminuída de 14 para 4, faça as operações para restabelecer a propriedade do heap:



9) Dado o vetor abaixo, construir um heap de Maximo, executando passo a passo o algoritmo esperto de inserção. Supor que o vetor começa na posição 1 (3) .

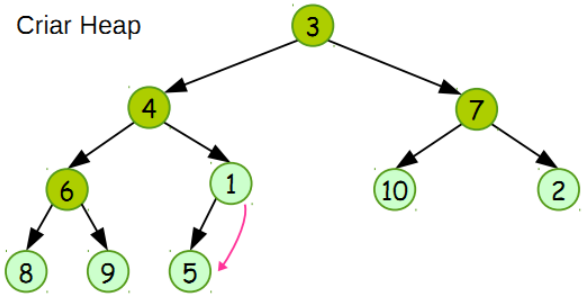
|3|4|7|6|1|10|2|8|9|5|

Criar Fila



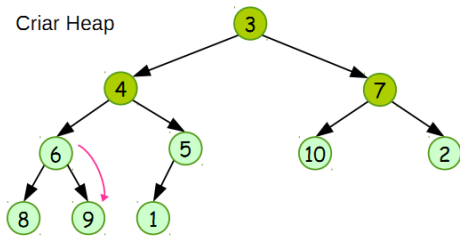
1. as folhas já são filas!

Criar Fila

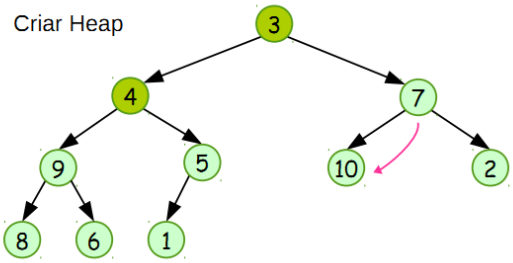


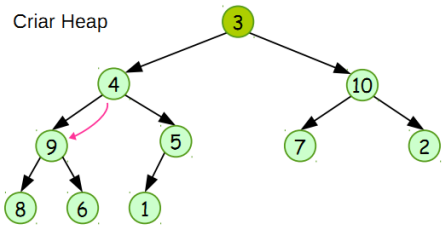
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

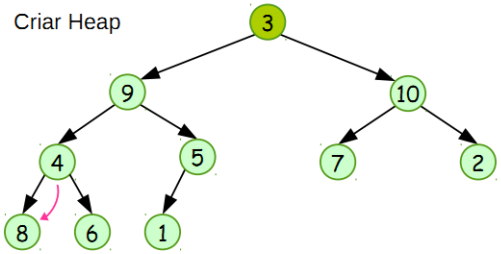
Criar Fila

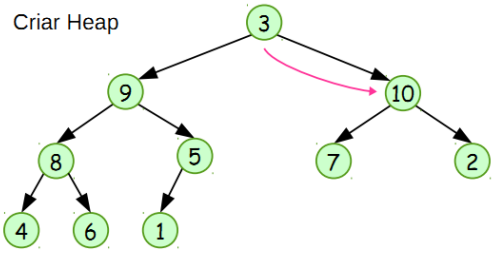


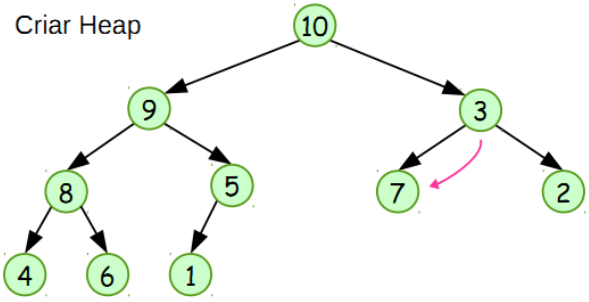
1. as folhas já são filas!
2. inserimos outros elementos na fila (em ordem inversa)

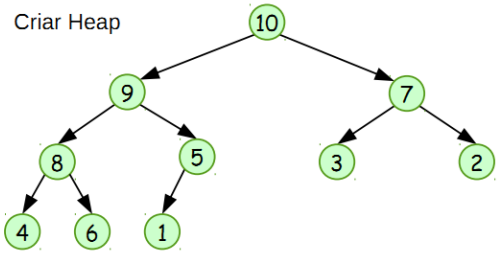

Criar Heap
3
4
7
9
5
10
2
8
6
1


Criar Heap
3
4
10
9
5
7
2
8
6
1

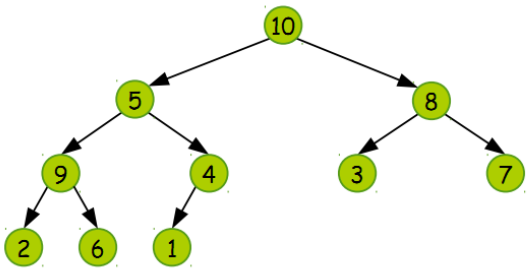

Criar Heap
3
9
10
4
5
7
2
8
6
1


Criar Heap
3
9
10
8
5
7
2
4
6
1

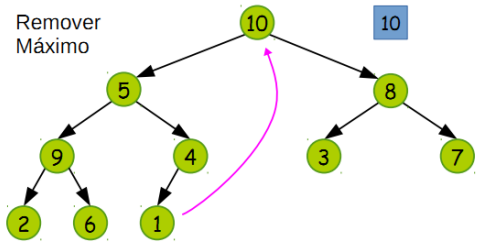

Criar Heap
10
9
3
8
5
7
2
4
6
1



10) Remover a raiz (o elemento de maior prioridade) e rebalancear o heap. Descreva como fará o procedimento e o execute sobre o heap abaixo, passo a passo.



Remover máximo



1. guardamos a raiz com o máximo
2. removemos o último elemento diminuímos a prioridade da raiz

11) Descreva ou implemente um algoritmo para o HeapSort:

HeapSort

- Uma vez que dispomos dos algoritmos para operar sobre um heap é possível usá-los para ordenar um array:
  - Construir o heap usando o método explicado anteriormente
  - Repetidamente remover o menor elemento e movê-lo para o fim do array, acertando o heap:

    $m \leftarrow n$
    enquanto $m > 1$ **fazer** {
    $H[1], H[m] \leftarrow H[m], H[1]$
    Descer $(1, m, H)$
    }
  - Ao final, o array está ordenado decrescentemente
  - Para obter ordem crescente, ou inverte-se a ordem do array ($O(n)$) ou utiliza-se um heap onde a raiz é o maior de todos os elementos

Implementação do Heap-Sort

```
void heap_sort(int v[], int n) {
    Heap fila;
    int i;

    criar_fila(&fila, v, n);

    for (i = n-1; i > 0; i--)
        v[i] = remove_max(fila);
}
```

Qual complexidade? $O(n \log n)$