Fila de Prioridade

Siang Wun Song - Universidade de São Paulo - IME/USP

MAC 5710 - Estruturas de Dados - 2008

Fila de prioridade

Fila de prioridade é uma estrutura de dado que mantém uma coleção de elementos, cada um com uma prioridade associada. Valem as operações seguintes.

- Inserir um elemento novo na fila de prioridade.
- Remover o elemento de maior prioridade da fila de prioridade.
- Uma fila de pacientes esperando transplante de fígado em geral é uma fila de prioridade. Em Sistemas Operacionais, um exemplo é a fila de prioridade de processos aguardando o processador para execução. Os processos mais prioritários são executados antes dos outros.
- Veremos várias formas de implementar uma fila de prioridade. Algumas são eficientes na inserção, outras na remoção. Queremos uma que seja eficiente nas duas operações.

Implementação mantendo a ordem total

Uma maneira de representar uma fila de prioridade é manter uma lista linear ligada ou encadeada em que os elementos estão ordenados por prioridades decrescentes. Assim,

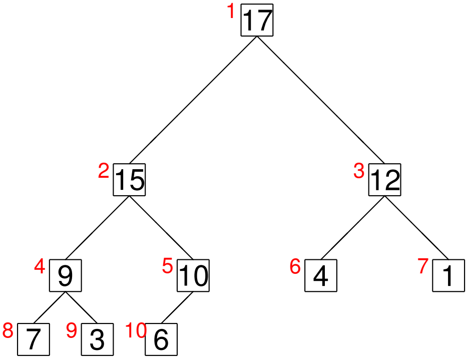
- Para remover um elemento da fila de prioridade: basta remover o primeiro elemento: tempo constante.
- Para inserir um novo elemento: o pior caso é $O(n)$, onde n é o número de elementos na fila de prioridade.

Implementação de forma aleatória sem nenhuma ordenação

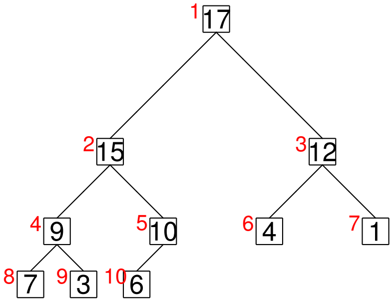
Uma maneira muito simples é armazenar de forma aleatória os elementos em uma lista linear seqüencial, sem nenhuma ordem.

- Para inserir um novo elemento: basta inserir em qualquer lugar, por exemplo no final da lista: Tempo O(1).
- Para remover um elemento da fila de prioridade: é preciso percorrer a lista para obter o elemento com a maior prioridade. Remove-se este elemento, colocando no seu lugar um outro qualquer, por exemplo aquele no final da lista. Tempo O(n), onde n é o número de elementos na fila.

“Heap” é uma estrutura de árvore binária em que cada nó terminal ou não-folha tem uma prioridade maior ou igual à prioridade de seus filhos. Em particular, vamos exigir que apenas o último nível da árvore pode ser incompleto e, nesse nível, se incompleto, os nós devem estar todos “encostados à esquerda”.



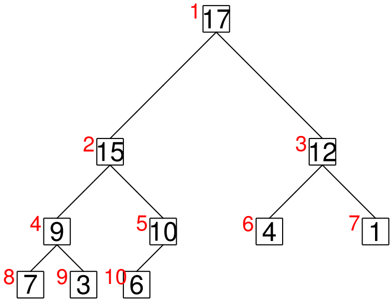
Relação pais e filhos no "heap"



Vamos numerar os nós do "heap" em ordem de níveis crescentes, indo da esquerda para a direita em cada nível (ordem chamada "breadth-first"). A seguinte propriedade útil se verifica.

- O pai do nó *k* é o nó *k* div 2 (isto é, quociente inteiro de *k* / 2).
- Se nó *k* tem um filho esquerdo, este será o nó 2k; se *k* também tem um filho da direita, este será o nó $2k + 1$.

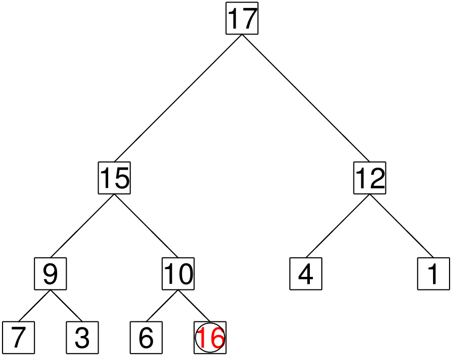
Implementação o “heap” em um vetor seqüencial



Podemos portanto usar um vetor *H* (seqüencial) para representar um “heap”. A estrutura de árvore está implícita nas posições dos nós.

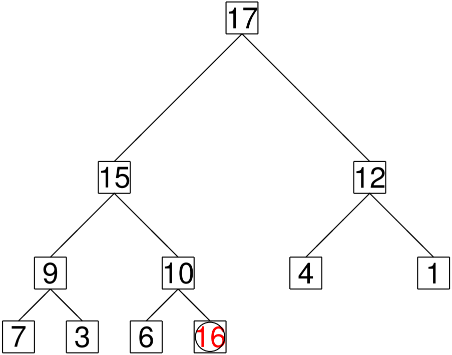
H	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
	17	15	12	9	10	4	1	7	3	6				

Inserir um elemento novo

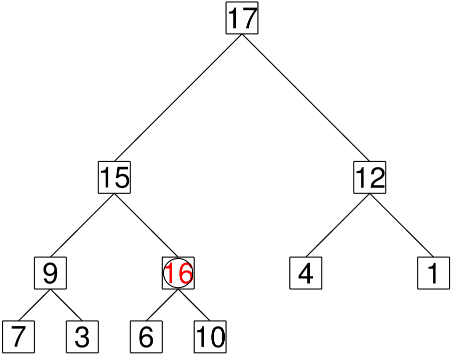


Para inserir um novo elemento com prioridade x, cria-se um novo elemento no fim do array H para receber x. Isso pode perturbar a propriedade do "heap". Para consertar isso: se x for maior que seu pai, então os dois trocam de lugar. Essa operação é repetida até que x encontre o seu lugar correto na árvore. O exemplo insere 16 no heap.

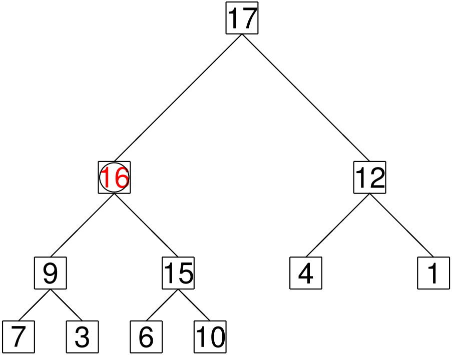
Inserir um elemento novo



Inserir um elemento novo



Inserir um elemento novo



Inserir um elemento no heap

Seja *ult* o índice indicando o último elemento do "heap" (array *H*) antes da inserção. No exemplo da figura, *ult* valia 10. Após a inserção, ele passará a valer 11.

insere(x)

$ult \leftarrow ult + 1$
$k \leftarrow ult$
while $(k$ div $2)$ and $x > H[k$ div $2]$ **do**
    $H[k] \leftarrow H[k$ div $2]$
    $k \leftarrow kdiv2$
end while
$H[k] \leftarrow x$

Note que nesse algoritmo, o novo elemento não é colocado dentro do "heap" até que o lugar apropriado tenho sido obtido. O algoritmo é O(log n), onde log denota logaritmo na base 2.

Remover um elemento do heap

- A remoção em si é muito simples, já que o elemento de maior prioridade é $H[1]$.
- Após a remoção, entretanto, precisamos re-arranjar os elementos do "heap": Colocamos em $H[1]$ o elemento $H[ult]$, liberando assim a última posição. Se o elemento colocado em $H[1]$ for menor que seus filhos, então ele trocado com o maior dos filhos. Isso é repetido até tal elemento ocupar a posição correta.
- Como a inserção, a remoção também é $O(\log n)$.

O algoritmo de remoção

remove(Y)

$Y \leftarrow H[1]$
$x \leftarrow H[ult]$
$ult \leftarrow ult - 1$
$k \leftarrow 1$
while $2k <= ult$ and ($x < H[2k]$ or $x < H[2k+1]$) **do**
  if $H[2k] > H[2k+1]$ **then**
    $H[k] \leftarrow H[2k]$
    $k \leftarrow 2k$
  else
    $H[k] \leftarrow H[2k+1]$
    $k \leftarrow 2k+1$
  end if
end while
$H[k] \leftarrow x$