Estruturas de Dados

Heaps Binários

1. Fila de prioridades (Heap) (Cormen capítulo 6. Seções 6.1, 6.2, 6.3 e 6.5)

Definição: estrutura de dados composta de chaves, que suporta duas operações básicas, a inserção de um novo item e a remoção do item com maior chave. A chave de cada item reflete a prioridade em que se deve tratar aquele item. Muito útil quando precisamos consultar o item de maior prioridade rapidamente.

Aplicações: Ordenação, sistemas operacionais, simulação de eventos, além de diversos algoritmos.

Operações:

- Constrói uma fila de prioridade num vetor de *n* itens;
- Insere um novo item;
- Remove o item com maior prioridade;
- Consulta o item de maior prioridade;

Representações: Lista encadeada ordenada, lista encadeada não ordenada, heaps.

	Constrói	Insere	Retira máximo	Altera prioridade
Lista ordenada	O(N log N)	O(N)	O(1)	O(N)
Lista não ordenada	O(N)	O(1)	O(N)	O(1)
Heaps	O(N)	O(log N)	O(log N)	O(log N)

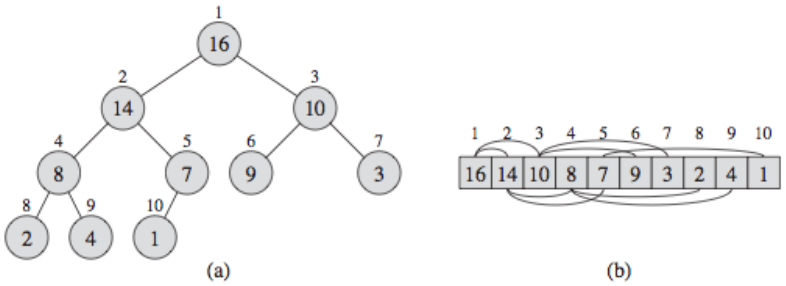
(exemplo de fila de prioridade com listas encadeadas)

Heap binário: é uma estrutura de dados que pode ser representada por um vetor, mas é visualizada como uma árvore binária. Ele deve satisfazer duas propriedades:

- a árvore é uma árvore binária completa. Todos os níveis da árvore, exceto possivelmente o último, são totalmente preenchidos. Os nós do último nível estão sempre preenchidos da esquerda para a direita.
- um nó pai deve ter sempre a chave maior que a de seus filhos.

Representação vetorial: pai na posição *i*, filhos nas posições 2*i* e 2*i* + 1. *A[i]* >= *A[2i]* e *A[i]* >= *A[2i + 1].*

- nós são numerados de 1 a n;
- o primeiro elemento do vetor é a raiz, e ele é sempre o elemento de maior chave;
- o nó k/2 é pai do nó k, 1 < k <= n;
- os nós 2k e 2k+1 são filhos da esquerda e da direita do nó k.
- representação compacta, permite caminhar pelos nós da árvore facilmente.



Consulta do maior elemento:
- Operação trivial. O(1).

Inserção de elementos:
- insira o elemento nível mais baixo da árvore;
- compare o elemento com seu pai. Pare se ele está na ordem correta;
- caso ele não esteja na ordem correta, troque o elemento com seu pai e retorne ao passo anterior.

A complexidade da inserção é O(h). Como a árvore é completa, h = lg n.

Exemplo: insere 15 na árvore anterior.

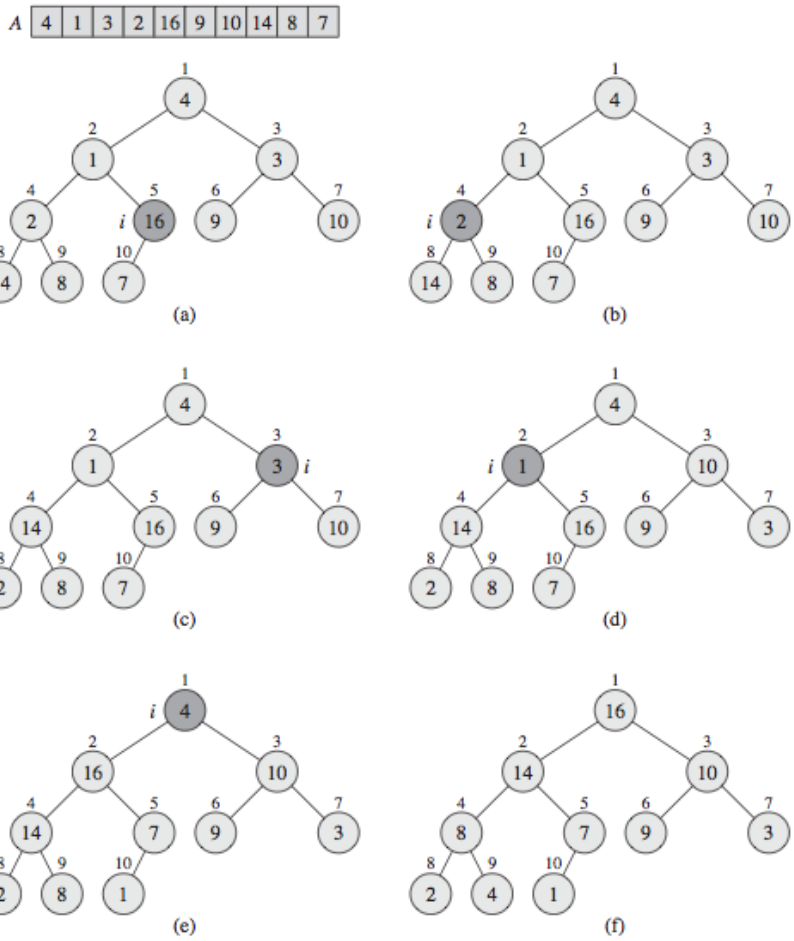
Remoção do maior elemento:
- troque a raiz da árvore pelo último elemento no último nível da árvore;
- compare a nova raiz com seus filhos. Pare se eles estiverem na ordem correta.
- caso eles não estejam na ordem correta, troque o elemento com o filho de maior chave e retorne ao passo anterior.

Esse procedimento é chamado de Max-Heapify ou Min-Heapify, e é um procedimento importante. Ele assume que as subárvores à direita e a esquerda do nó a ser heapificado são heaps, mas que o nó atual pode estar na posição errada (ser menor que seus filhos). O método, então, desce o valor de *i*, até sua posição correta. Sua complexidade é, no pior caso, O(lg n).

Exemplo: remove a raiz da árvore anterior.
Exemplo: max-heapify: troca 14 com 6 na árvore anterior e executa o max-heapify.

Recuperar m elementos mais prioritários: O(m log n). Cada recuperação envolve uma remoção O(log n).

Constrói heap: uma maneira trivial de se construir um heap binário é através de sucesivas inserções. A complexidade do procedimento seria O(n.lgn). No entanto, existe uma forma mais eficiente.
Primeiro, insere-se os nós na árvore de forma aleatória. Em seguida, basta executar a operação Max-Heapify em todos os nós da árvore, começando dos nós mais baixos (ignorando as folhas). Apesar de, aparentemente, essa operação possuir complexidade O(n.lg n), pode-se provar que a complexidade é, na verdade, O(n), uma vez que a operação Max-Heapify na verdade depende da altura do nó que está sendo operado.


A
4 1 3 2 16 9 10 14 8 7
(a)
(b)
(c)
(d)
(e)
(f)