Heap de Fibonacci

Igor Andrade Brito

Junho de 2021

1 Introdução

O Heap de Fibonacci foi apresentado inicialmente por Fredman e Tarjan (Fredman and Tarjan 1987) com o objetivo de aprimorar o algoritmo de Dijkstra para o problemas do caminho mais curto de origem única. Essa estrutura realiza um conjunto de operações, sendo que várias delas são executadas em tempos constantes o que torna essa estrutura bem adequada para aplicações que invocam tais operações frequentemente. Para implementar o Heap de Fibonacci é utilizado o conceitos de arvores enraizadas, sendo que essa estrutura é definida como sendo uma coleção de árvores enraizadas ordenadas por um heap minímo, ou seja, o valor de um nó é sempre maior que o valor de seu pai. É importante ressaltar que ela não é uma estrutura eficiente para buscar um item.

2 Operações

As operações que essa estrutura executa são:

MAKE-HEAP(): cria e retorna um novo heap vazio.

INSERT(H, x): insere no heap H o elemento x com uma chave predefinida.

MINIMUM(H): retorna o elemento do heap H cuja chave é mínima. Essa operação não altera H.

EXTRACT-MIN(H): elimina o elemento do heap H cuja chave é mínima, retornando-o.

UNION(H1, H2): cria e retorna um novo heap que contém todos os elementos dos heaps H1 e H2. Os heaps H1 e H2 são "destruídos" por essa operação.

DECREASE-KEY(H, x, k): atribui ao elemento x dentro do heap H o novo valor de chave k, que supomos ser positivo e não ser maior que seu valor de chave atual.

DELETE(H, x): elimina o elemento x do heap H.

3 Estrutura

Na estrutura Heap de Fibonacci, representamos cada elemento por um nó dentro de uma árvore, e cada nó x tem um atributo chave, um ponteiro x.p para seu pai (NULL se não tiver) e um ponteiro x.filho para algum de seus filhos. Os filhos de x estão interligados em uma lista circular, duplamente ligada, que denominamos lista de filhos de x. Cada filho y em uma lista de filhos tem ponteiros y.esquerda e y.direita que apontam para os irmãos à esquerda e à direita de y, respectivamente. Se o nó y é um filho único, então y.esquerda = y.direita = y. Os filhos podem aparecer em qualquer ordem em uma lista de filhos. Cada nó tem dois outros atributos: o número de filhos na lista de filhos do nó x em x.grau e o atributo do valor booleano x.marca indica se o nó x perdeu um filho desde a última vez que x se tornou o filho de um outro nó. Os nós recém-criados não estão marcados, e um nó x se torna desmarcado sempre que passa a ser o filho de outro nó.

Acessamos determinado heap de Fibonacci H por um ponteiro H.min para a raiz de uma árvore que contém uma chave mínima; esse nó é denominado nó mínimo do heap de Fibonacci. Se mais de uma raiz tiver uma chave como o valor mínimo, qualquer raiz pode servir como o nó mínimo. Quando um heap de Fibonacci H está vazio, H.min = NULL. As raízes de todas as árvores em um heap de Fibonacci são interligadas por meio de seus ponteiros esquerda e direita em uma lista circular duplamente ligada denominada lista de raízes do heap de Fibonacci. Assim, o ponteiro H.min aponta para o nó na lista de raízes cuja chave é mínima. As árvores podem aparecer em qualquer ordem dentro de uma lista de raízes. Contamos com um outro atributo para um heap de Fibonacci H: H.n, o número de nós atualmente em H. A figura 1 contém uma representação gráfica do Heap de Fibonacci.

Existem duas vantagens para o uso de listas circulares, duplamente ligadas em Heaps de Fibonacci:

- Inserir um nó em qualquer ponto ou remover um nó de qualquer lugar de uma lista circular duplamente ligada no tempo O(1).
- Concatenar duas listas em uma única lista circular duplamente ligada no tempo O(1).

4 Tempo de execução

Heaps de Fibonacci têm limites de tempo assintótico melhores que os heaps binários para as operações *insert*, *union*, e *descrease-key*, e os mesmos tempos de execução assintóticos para as operações restantes, como mostra a Figura 2. Observe, entretanto, que os tempos de execução de heaps de Fibonacci na Figura são limites de tempo amortizado e não limites de tempo do pior caso por operação. A operação *union* demora somente tempo

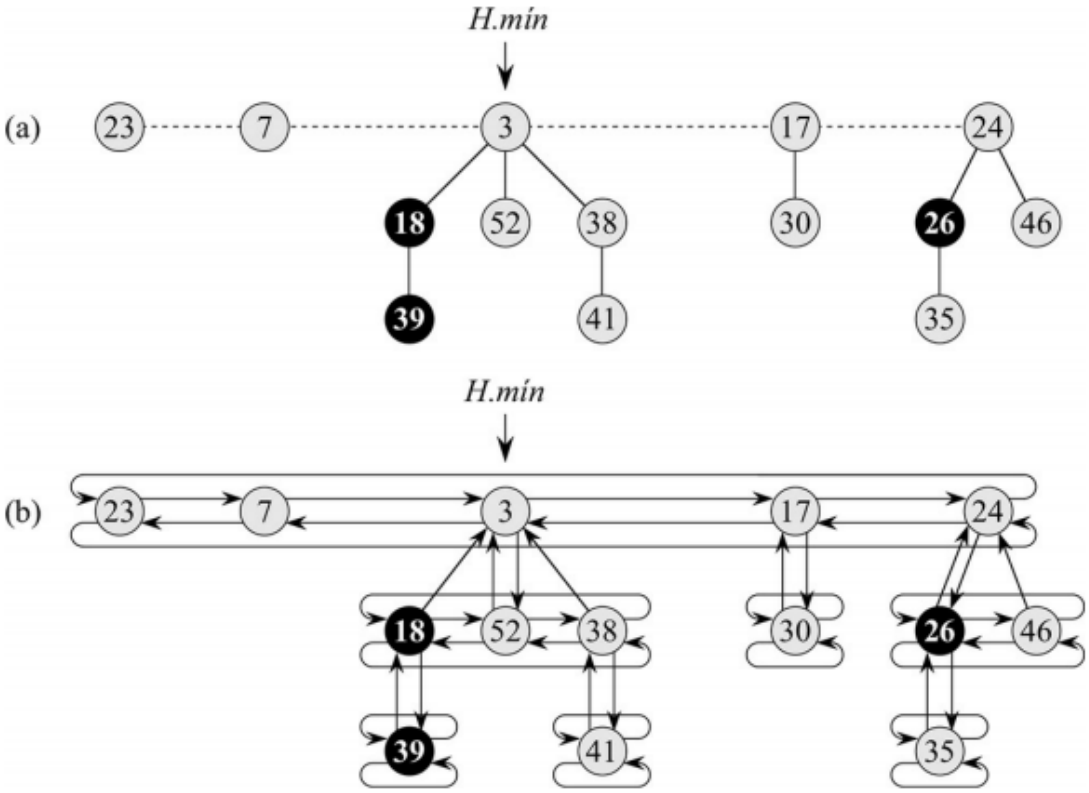

Figure 1: (a): Um heap de Fibonacci com cinco árvores ordenadas por heap mínimo e 14 nós. A linha tracejada indica a lista de raizes. O nó mínimo desse heap contém a chave 3 e os nós pretos são marcados.(b): O mesmo heap porém com os ponteiros para o pai, filhos e irmãos. Retirado de (Cormen 2012)

amortizado constante em um heap de Fibonacci, o que é significativamente melhor que o tempo linear do pior caso exigido em um heap binário.

5 Heaps de Fibonacci na teoria e na prática

De um ponto de vista teórico, os heaps de Fibonacci são especialmente desejáveis quando o número de operações *extract-min* e *delete* é pequeno em relação ao número de outras operações executadas. Essa situação surge em muitas aplicações. Por exemplo, alguns algoritmos para problemas de grafos podem chamar *decrease-key* uma vez por aresta. Para grafos densos, que têm muitas arestas, o tempo amortizado Q(1) de cada chamada de emphdecrease-key significa uma grande melhoria em relação ao tempo do pior caso de heaps binários. Algoritmos rápidos para problemas como cálculo de árvores geradoras mínimas e localização de caminhos mais curtos de origem única tornam essencial o uso de heaps de Fibonacci. Porém, do ponto de vista prático, os fatores

Procedimento	Heap binário (pior caso)	Heap de Fibonacci (amortizado)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

Figure 2: Tempos de excução para as operações com Heap binário e com o Heap de Fibonacci. Retirado de (Cormen 2012)

constantes e a complexidade da programação de heaps de Fibonacci os tornam menos desejáveis que heaps binários (ou k-ários) comuns para a maioria das aplicações. Assim, os heaps de Fibonacci são predominantemente de interesse teórico. Se uma estrutura de dados muito mais simples com os mesmos limites de tempo amortizado que os heaps de Fibonacci fosse desenvolvida, ela também seria de utilidade prática.

Referências

Cormen, Thomas H. (2012). *Algoritmos: teoria e prática*. Campus.

Fredman, Michael L. and Robert Endre Tarjan (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". In: *Journal of the ACM* 34.3, pp. 596–615. DOI: 10.1145/28869.28874.