Heaps: Uma análise da Coluna 14 de *Programming Pearls*

Manassés Feliciano Paterline
Davi Corradi
Daniel Murad

Março de 2026

Resumo

Este artigo analisa a Coluna 14 da obra *Programming Pearls* de Jon Bentley. Explora-se a estrutura de dados Heap, suas propriedades fundamentais, a eficiência de sua implementação implícita em arrays e suas aplicações práticas em algoritmos de ordenação e filas de prioridade.

Conteúdo

1 Introdução

O livro *Programming Pearls* Bentley 2000, escrito por Jon Bentley, é um clássico da literatura de computação. A obra é dividida em colunas, refletindo sua origem como artigos publicados na revista *Communications of the ACM*. Cada coluna aborda problemas práticos de engenharia de software com soluções elegantes e eficientes. A Coluna 14 foca especificamente em Heaps, demonstrando como uma estrutura simples pode resolver problemas complexos.

2 Fundamentos do Heap

Jon Bentley introduz o Heap como uma estrutura de dados fundamental para equilibrar dois problemas clássicos: a ordenação de dados e a manutenção de filas de prioridade.

2.1 Propriedades de Ordem e Forma

A eficácia do heap deriva de duas propriedades rigorosas que garantem sua performance:

- **Propriedade de Ordem:** Em um *min-heap*, o valor de cada nó é menor ou igual aos valores de seus filhos. Isso implica que o menor elemento está sempre na raiz.
- **Propriedade de Forma:** O heap é uma árvore binária completa, o que significa que todos os níveis estão preenchidos, exceto possivelmente o último, que deve estar preenchido da esquerda para a direita.

Essa configuração garante que a altura da árvore seja sempre $\lfloor \log_2 n \rfloor$, permitindo operações rápidas.

2.2 Implementação Implícita e Eficiência de Memória

Diferente de árvores tradicionais que utilizam ponteiros, o heap utiliza uma representação em array, o que Bentley chama de "implementação implícita". Em um array $x[1 \ldots n]$:

- **Raiz:** $x[1]$.
- **Filho Esquerdo de i:** $2i$.
- **Filho Direito de i:** $2i + 1$.
- **Pai de i:** $\lfloor i/2 \rfloor$.

Esta técnica economiza o espaço que seria gasto com ponteiros e melhora a localidade de cache.

3 Mecânica e Reorganização

Para manter a integridade do heap durante inserções e deleções, são utilizadas duas funções principais que operam em tempo $O(\log n)$.

3.1 A Função `siftup`

Utilizada principalmente durante a inserção. Quando um novo elemento é adicionado ao final do array, a função `siftup` o faz "flutuar"para cima na hierarquia, trocando-o com seu pai até que a propriedade de ordem seja restaurada.

3.2 A Função `siftdown`

Essencial para a extração do elemento mínimo. Após remover a raiz e substituí-la pelo último elemento do array, a função `siftdown` faz esse elemento "afundar", trocando-o com o menor de seus filhos até restabelecer o equilíbrio.

4 Aplicações Práticas

4.1 Filas de Prioridade

O heap é a estrutura ideal para filas de prioridade em sistemas operacionais e simulações. Ele oferece um compromisso excelente: tanto a inserção quanto a extração do mínimo ocorrem em $O(\log n)$, enquanto listas simples exigiriam $O(n)$ para uma dessas operações.

4.2 O Algoritmo Heapsort

Bentley descreve o Heapsort como uma alternativa robusta ao Quicksort. Suas principais vantagens são:

1. **Tempo de Pior Caso:** Garante performance $O(n \log n)$.
2. **Espaço:** É um algoritmo *in-place*, não exigindo memória auxiliar significativa além do array original.

5 Conclusão

A análise da Coluna 14 revela que a sofisticação de uma solução nem sempre reside em sua complexidade, mas em sua capacidade de extrair o máximo de desempenho com o mínimo de recursos. O uso de arrays implícitos para representar árvores binárias é uma lição valiosa de economia de memória e eficiência algorítmica. Bentley nos lembra que, ao entender profundamente as propriedades matemáticas de uma estrutura como o heap, podemos criar sistemas que são, ao mesmo tempo, simples de manter e extremamente poderosos na execução.

Referências

Bentley, Jon (2000). *Programming Pearls*. 2ª ed. Addison-Wesley. ISBN: 9780201657883.