Árvore Binária e Heapsort

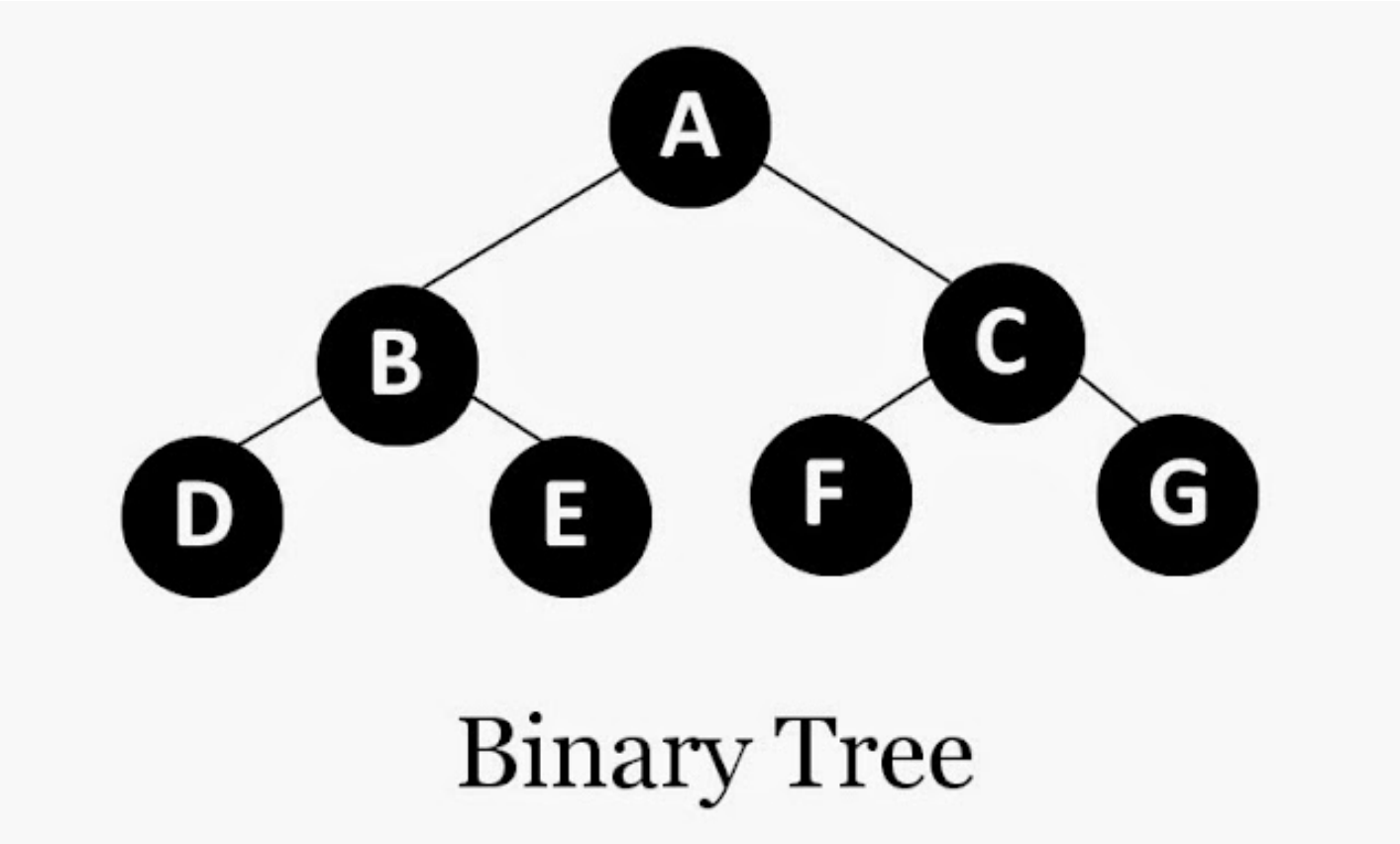
Guilherme Heinrich dos Santos
e Frederico Volkmann

7 de julho de 2025

Sumário

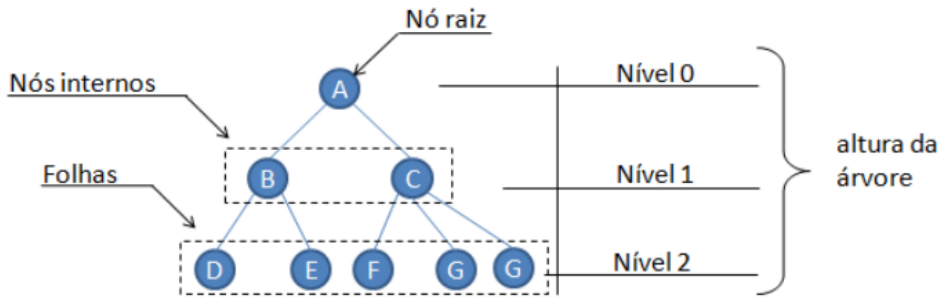
Conceitos Básicos

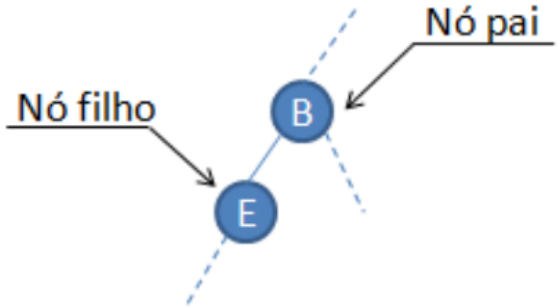
- Estrutura de dados homogênea.
- Organizado Hierarquicamente.



Elementos da Árvore

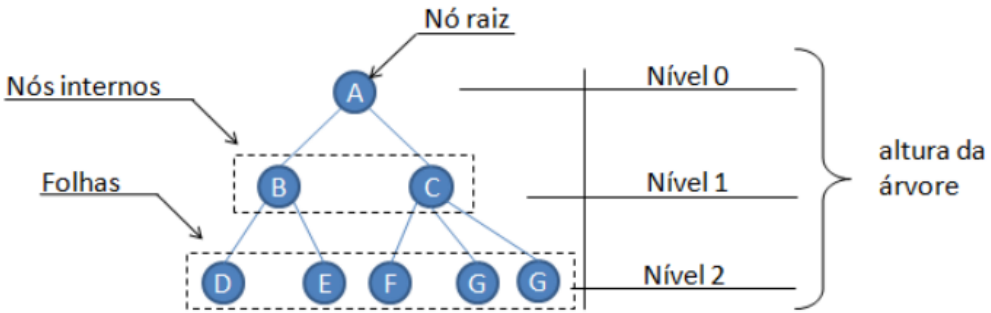
- Nó - um dado que tem um valor e aponta para outros nós
- Raiz - primeiro nó da árvore, só temos uma.
- Folhas ou terminal - últimos nós da árvore.
- Nível é a distância da raiz ao nó.

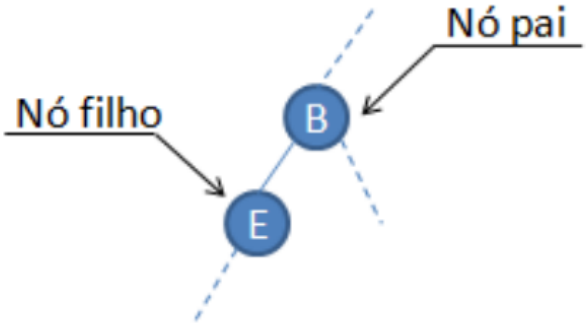




Propriedades da Árvore

- Cada nó, exceto a raiz, tem um antecessor ou pai.
- Cada nó, exceto a folha, tem sucessores ou filhos.
- Todo nó que tem o mesmo pai pode ser chamado de irmão.
- É composta por subárvores.
- Caminho é uma lista de nós distintos e sucessivos.
- Existe sempre um caminho entre a raiz e qualquer nó da árvore.



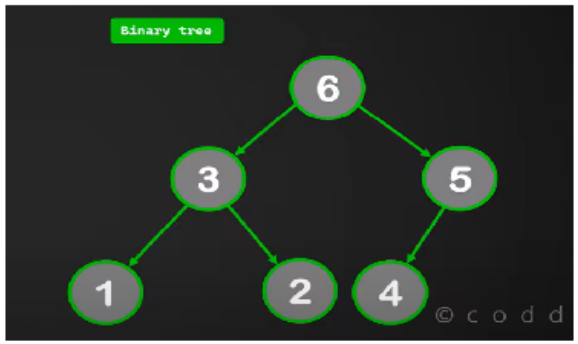


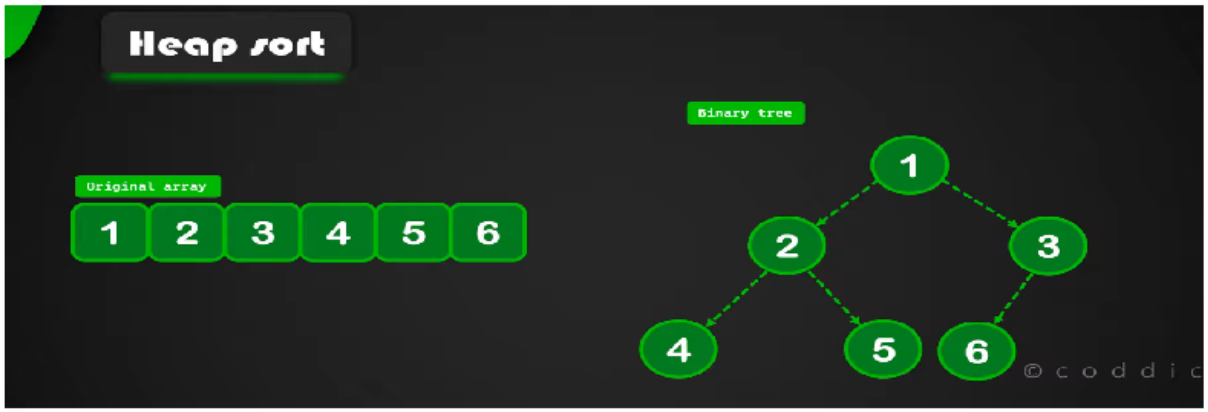
Conceito

Heapsort é um algoritmo de ordenação por comparação que se baseia na estrutura de dados chamada heap Binário. A ideia principal é transformar a lista a ser ordenada em um heap binário máximo (para ordenação crescente) ou heap binário mínimo (para ordenação decrescente), e então extrair os elementos um por um para construir a lista ordenada. O algoritmo tem duas fases: a primeira transforma o vetor em heap (máximo ou mínimo) e a segunda retira elementos do heap em ordem crescente

Array Não-ordenado

Array transformado em árvore máx-heap





Array ordenado

Vetores e árvores binárias

Antes de começar a discutir o Heapsort, precisamos aprender a enxergar a árvore binária que está escondida em qualquer vetor. O conjunto de índices de qualquer vetor `v[1..m]` pode ser encarado como uma árvore binária da seguinte maneira:

- o índice `1` é a *raiz* da árvore;
- o *pai* de qualquer índice `f` é `f/2` (é claro que `1` não tem pai);
- o *filho esquerdo* de um índice `p` é `2p` (esse filho só existe se $2p \le m$);
- o *filho direito* de `p` é `2p+1` (esse filho só existe se $2p+1 \le m$).

1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	999	999	999	999	999	999	999	999	999	999	999	999	999

Heap

O segredo do algoritmo Heapsort é uma estrutura de dados, conhecida como heap, que enxerga o vetor como uma árvore binária. Há dois sabores da estrutura: max-heap e min-heap; trataremos aqui apenas do primeiro, omitindo o prefixo "max-".

Um heap (= monte) é um vetor em que o valor de todo pai é maior ou igual ao valor de cada um de seus dois filhos. Mais exatamente, um vetor `v[1..m]` é um *heap* se

$$v[f/2] \ge v[f]$$

para $f = 2, \ldots, m$. Aqui, como no resto deste capítulo, vamos convencionar que as expressões que figuram como índices de um vetor são sempre calculadas em aritmética inteira. Assim, o valor da expressão `f/2` é $\lfloor f/2 \rfloor$, ou seja, o piso de `f/2`.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

Pontos Fortes

Eficiência de Tempo Consistente (O(nlogn)): Este é o maior trunfo do Heapsort. Diferente de outros algoritmos como o Quicksort (que pode ter desempenho de O(n2) no pior caso), o Heapsort garante um desempenho O(nlogn) em todos os cenários (melhor, médio e pior caso). Isso o torna previsível e confiável para grandes volumes de dados.
Ordenação In-Place: O Heapsort é um algoritmo "in-place", o que significa que ele opera diretamente no array de entrada e requer uma quantidade mínima de memória auxiliar (complexidade de espaço O(1)). Isso é uma grande vantagem em ambientes com restrições de memória.

Pontos Fortes

Não Recursivo: A implementação do Heapsort pode ser totalmente iterativa. Isso evita os riscos de estouro de pilha (stack overflow) que podem ocorrer em algoritmos recursivos com conjuntos de dados muito grandes.
Bom para Sistemas Embarcados e Memória Limitada: Devido à sua natureza in-place e previsibilidade de desempenho, o Heapsort é frequentemente considerado para sistemas onde a memória é um recurso escasso ou onde o desempenho consistente é fundamental.

Pontos Fracos

Não é Estável: Um dos principais pontos fracos do Heapsort é que ele não é um algoritmo de ordenação estável. Isso significa que a ordem relativa de elementos com valores iguais não é preservada. Se você tem uma lista de, por exemplo, [(A, 5), (B, 3), (C, 5)] e a ordena, não há garantia de que (A, 5) aparecerá antes de (C, 5) após a ordenação. Para algumas aplicações, isso pode ser um problema.

Difícil de Paralelizar: A natureza sequencial da construção e extração do heap torna o Heapsort inerentemente difícil de paralelizar de forma eficiente. Isso significa que ele não se beneficia tanto de múltiplas CPUs ou núcleos quanto outros algoritmos de ordenação.

Casos de Uso

Sistemas Embarcados e Dispositivos com Memória Limitada: Um microcontrolador em um aparelho eletrônico, um sistema de navegação GPS antigo ou até mesmo em alguns dispositivos IoT (Internet das Coisas). Esses sistemas frequentemente possuem pouca memória RAM e precisam de algoritmos que operem "in-place", ou seja, sem a necessidade de alocar muita memória extra. O Heapsort é perfeito para isso, pois ele rearranja os dados no próprio local, sem precisar de arrays auxiliares grandes.

Sistemas de Tempo Real (Real-Time Systems): Em sistemas onde o tempo de resposta é crítico, como em equipamentos médicos (monitores cardíacos, máquinas de diálise), controle de tráfego aéreo ou sistemas de automação industrial, a previsibilidade do Heapsort é uma grande vantagem. Essa consistência ajuda a garantir que as operações de ordenação sejam concluídas dentro dos prazos estipulados, sem grandes surpresas.

Casos de Uso

Algoritmos de Seleção (Encontrando o K-ésimo Maior/Menor Elemento): Embora não seja um uso direto para ordenar um array inteiro, o conceito por trás do Heapsort é fundamental para algoritmos de seleção eficientes. Para encontrar o K-ésimo maior ou menor elemento em um conjunto de dados muito grande (sem precisar ordenar todo o conjunto), pode-se usar uma min-heap (para os maiores) ou max-heap (para os menores) de tamanho K. À medida que os elementos são processados, a heap é mantida com os K elementos relevantes, permitindo encontrar a resposta em O(n log k) tempo, que é muito mais eficiente do que ordenar tudo (O(n log n)) se k for muito menor que n.

Perguntas?

Árvore binária: slides do professor.
Heapsort:
https://www.ime.usp.br/ pf/algoritmos/aulas/hpsrt.htmlbuild-heap
https://www.cprogramming.com/tutorial/computersciencetheory/heapsort
Imagens: https://www.youtube.com/watch?v=rkRDRTxQhB4