Etapa 6 - Heap

Heaps (Árvores de Montículo) - Guia Didático

O que é um Heap?

Um Heap é uma estrutura de dados especial que funciona como uma **árvore binária completa** com uma regra muito específica de organização. Pense em um heap como una fila de prioridades onde alguns elementos são mais "importantes" que outros.

Características Principais

Árvore Binária Completa
- Todos os níveis estão completamente preenchidos, exceto possibly o último
- O último nível é preenchido da esquerda para a direita
- Isso permite representar o heap eficientemente como um array

Propriedade do Heap
Cada nó deve obedecer uma regra em relação aos seus filhos:
- **Max-Heap**: Pai é maior ou igual aos filhos
- **Min-Heap**: Pai é menor ou igual aos filhos

Os Dois Tipos de Heap

Max-Heap (Heap Máximo)
```
       [100]
       /   \
     [85]   [70]
     /  \    /
   [40] [60][65]
```
- **Regra**: Pai ≥ Filhos
- **Raiz**: Sempre o maior elemento
- **Uso**: Quando precisamos acessar rapidamente o maior valor

Min-Heap (Heap Mínimo)
```
       [10]
       /  \
     [20]  [25]
     /  \    /
   [35] [40][30]
```
- **Regra**: Pai ≤ Filhos
- **Raiz**: Sempre o menor elemento
- **Uso**: Quando precisamos acessar rapidamente o menor valor

Como Representar um Heap como Array

Um heap pode ser armazenado em um array simples:

Heap como árvore:
```
       [50]
       /  \
     [40]  [30]
     /  \
   [20] [10]
```

Heap como array:
```
Índice:  0   1   2   3   4
Array: [50, 40, 30, 20, 10]
```

Fórmulas de Navegação
Para um nó no índice **i**:
- **Filho esquerdo**: 2×i + 1
- **Filho direito**: 2×i + 2
- **Pai**: (i-1) ÷ 2 (divisão inteira)

Exemplo: Para o nó no índice 1 (valor 40)
- Filho esquerdo: 2×1 + 1 = 3 (valor 20)
- Filho direito: 2×1 + 2 = 4 (valor 10)
- Pai: (1-1)÷2 = 0 (valor 50)

Operações Principais

Inserção de Elemento
1. Adiciona o novo elemento no final do array
2. Compara com seu pai
3. Se violar a propriedade do heap, troca com o pai
4. Repete até que a propriedade seja restaurada

Complexidade: O(log n) - sobe no máximo a altura da árvore

Remoção da Raiz
1. Remove o elemento da raiz (índice 0)
2. Move o último elemento para a raiz
3. Compara com seus filhos
4. Se violar a propriedade, troca com o maior filho (max-heap) ou menor filho (min-heap)
5. Repete até que a propriedade seja restaurada

Complexidade: O(log n) - desce no máximo a altura da árvore

Heapify - Construindo um Heap
- Processo de transformar um array comum em um heap válido
- Começa pelos nós não-folha e aplica heapify-down
- **Complexidade**: O(n) - mais eficiente que inserir um por um

Aplicações Práticas

Filas de Prioridade
- Sistema de emergências em hospitais (pacientes mais graves são atendidos primeiro)
- Sistema de impressão (documentos prioritários)
- Processamento de tarefas (tarefas urgentes primeiro)

Algoritmo Heap Sort
1. Constrói um max-heap a partir do array
2. Remove repetidamente a raiz (maior elemento) e coloca no final
3. Resultado: array ordenado
- **Vantagem**: Ordenação in-place com O(n log n)

Algoritmos de Grafos
- **Algoritmo de Dijkstra**: Encontra caminhos mais curtos usando min-heap
- **Algoritmo de Prim**: Encontra árvore geradora mínima

Seleção Eficiente
- Encontrar o k-ésimo maior elemento
- Encontrar os k maiores elementos de uma lista

Vantagens e Desvantagens

Vantagens
- **Acesso rápido** ao elemento de maior/menor prioridade: O(1)
- **Inserção e remoção** eficientes: O(log n)
- **Uso eficiente de memória**: Representação com array
- **Versátil**: Usado em vários algoritmos importantes

Desvantagens
- **Busca por elemento arbitrário**: O(n) - tem que procurar em todo o heap
- **Não mantém ordem total**: Só garante a relação pai-filho
- **Não ideal** para operações de busca geral

Exemplo Prático: Sistema de Atendimento Hospitalar

Usando Min-Heap (menor prioridade = mais urgente)

```
       [1 - Emergência]
       /              \
[3 - Estável]       [2 - Urgente]
```

```
      |
[5 - Rotina]
```

- Prioridade 1: Atendimento imediato
- Prioridade 5: Atendimento de rotina

Operações:
- Novo paciente chega: inserção no heap
- Atender paciente: remoção da raiz (mais urgente)
- Sempre mantém o mais urgente no topo

Dicas de Estudo

1. **Visualize** sempre como árvore e como array
2. **Pratique** as fórmulas de navegação entre índices
3. **Desenhe** os passos das operações de inserção e remoção
4. **Entenda** a diferença entre max-heap e min-heap
5. **Relacione** com aplicações reais para fixar o conceito

Os heaps são fundamentais para situações onde precisamos de acesso rápido a elementos de maior ou menor prioridade, sendo uma das estruturas mais práticas e eficientes para esse propósito.

Perguntas de Múltipla Escolha - Heaps

Pergunta 1: Propriedade Fundamental
Qual é a propriedade fundamental de um max-heap?

a) É uma árvore binária balanceada
b) O valor de cada nó é maior ou igual aos valores de seus filhos
c) Todos os níveis da árvore estão completamente preenchidos
d) É uma árvore de busca binária

Resposta: b) O valor de cada nó é maior ou igual aos valores de seus filhos

---

Pergunta 2: Complexidade das Operações
Qual é a complexidade de tempo para inserir um novo elemento em um heap?

a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)

Resposta: b) O(log n)