**Faculdade Antonio Meneghetti- AMF**

Trabalho de Estrutura de Dados:
**Pesquisa e implementação**
Profº. Rhauani Weber Aita Fazul

Integrantes:
Josiel Pereira Paz
Eduarda José Cardoso

Restinga Sêca, 25 de junho de 2024.

# Sumário:

## Introdução:

Filas de prioridade são estruturas de dados essenciais em diversas aplicações, desde sistemas operacionais até algoritmos de grafos. Essas estruturas armazenam elementos associados a uma prioridade, de maneira que os elementos com maior prioridade sejam atendidos antes dos demais. Existem várias maneiras de implementar filas de prioridade, sendo as mais comuns os heaps binários, as árvores de busca binária balanceadas e as listas ordenadas. Neste relatório, focaremos especificamente na implementação de filas de prioridade utilizando heaps binários.

Os heaps binários são estruturas de dados que mantêm uma propriedade específica: em um heap de mínimo, o menor elemento está sempre na raiz, enquanto em um heap de máximo, o maior elemento ocupa a raiz. Esses heaps são representados de maneira eficiente através de árvores binárias completas, onde cada nível da árvore é preenchido completamente antes de se iniciar o preenchimento do próximo nível. Essa característica permite que operações como inserção e remoção de elementos tenham uma complexidade logarítmica, o que as torna muito eficientes.

Além dos heaps binários, também exploraremos os heaps de Fibonacci, que são uma extensão mais avançada dos heaps binários. Os heaps de Fibonacci oferecem tempos de execução amortizados mais eficientes para várias operações, sendo especialmente úteis em algoritmos complexos como o de Dijkstra para encontrar o caminho mais curto em grafos.

Este relatório discutirá detalhadamente os conceitos de heaps de mínimo, heaps de máximo e heaps de Fibonacci. Também abordaremos a importância dessas estruturas de dados, suas principais características, bem como as vantagens e desvantagens de cada tipo de heap.

## Conceitos de Heaps Binários:

Em ciência da computação, heaps (ou montes) são estruturas de dados especializadas que se baseiam em árvores quase completas, sendo amplamente utilizadas para implementar filas de prioridade. Onde cada nó segue a propriedade de heap:

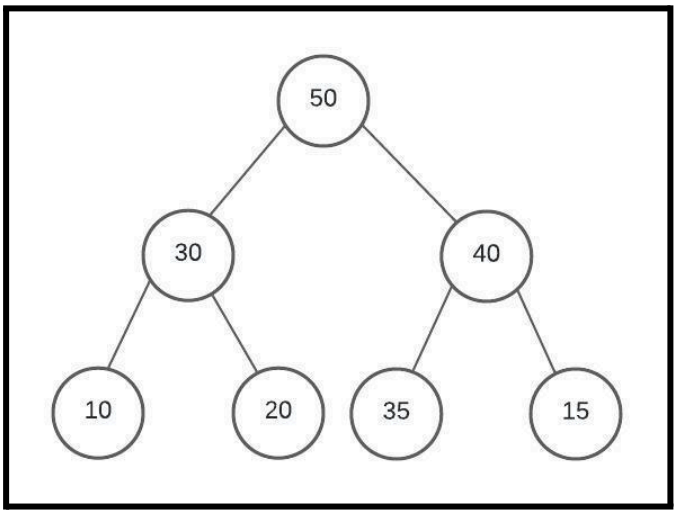
- **Max-Heap:**Um Max-Heap é uma estrutura de dados específica onde cada nó possui uma chave que é maior ou igual às chaves de seus filhos. Em outras palavras, em um Max-Heap, o valor de cada nó pai é sempre maior ou igual ao valor de seus nós filhos, garantindo que o maior elemento de toda a estrutura esteja localizado na raiz, ou seja, no topo da árvore.

  Esta propriedade do Max-Heap é particularmente útil em situações onde é necessário remover o elemento de maior prioridade primeiro. Por exemplo, em um sistema de gerenciamento de tarefas onde as tarefas mais urgentes ou importantes devem ser tratadas antes das outras, um Max-Heap pode ser utilizado para garantir que a tarefa com a prioridade mais alta seja sempre a primeira a ser removida e processada.

  A construção de um Max-Heap envolve a inserção de elementos de maneira a manter a propriedade de heap, o que geralmente é feito utilizando um processo de "heapificação". Quando um novo elemento é adicionado ao heap, ele é inicialmente colocado na última posição da árvore. Em seguida, ele é comparado com seu nó pai e, se for maior, os dois nós trocam de posição. Este processo se repete até que a propriedade de heap seja restaurada.Além disso, a remoção do elemento de maior prioridade (a raiz) envolve a substituição da raiz pelo último elemento da árvore e a subsequente "heapificação" para baixo, onde o novo elemento na raiz é comparado com seus filhos e trocado com o maior deles, se necessário, até que a propriedade de heap seja restabelecida.

  A eficiência das operações de inserção e remoção em um Max-Heap é garantida pela sua estrutura, que permite que ambas as operações sejam realizadas em tempo logarítmico em relação ao número de elementos no heap. Isso torna os Max-Heaps uma escolha excelente para aplicações que requerem a manipulação frequente de elementos com diferentes níveis de prioridade.

Exemplo:




Neste exemplo, 50 é a raiz e o maior elemento.

- **Min-Heap**: Um Min-Heap é uma estrutura de dados específica onde cada nó possui uma chave que é menor ou igual às chaves de seus filhos. Em outras palavras, em um Min-Heap, o valor de cada nó pai é sempre menor ou igual ao valor de seus nós filhos, garantindo que o menor elemento de toda a estrutura esteja localizado na raiz, ou seja, no topo da árvore.

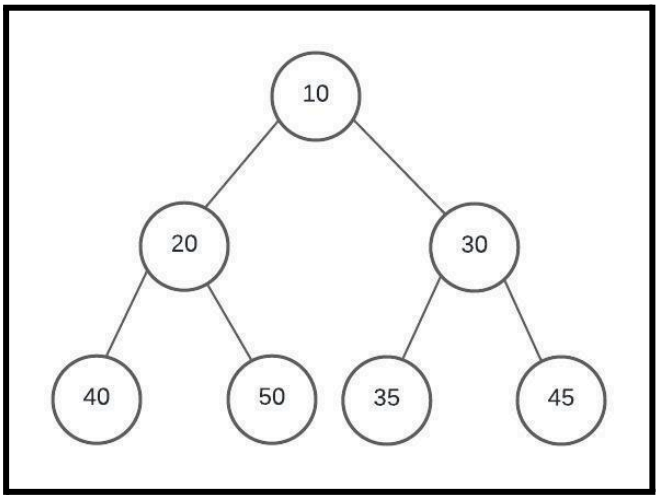
  Essa propriedade do Min-Heap é particularmente útil em situações onde é necessário remover o elemento de menor prioridade primeiro. Por exemplo, em um algoritmo de caminhos mínimos como o de Dijkstra, um Min-Heap pode ser utilizado para garantir que o vértice com a menor distância estimada seja sempre o próximo a ser processado. Isso é essencial para a eficiência e corretude do algoritmo.

  A construção de um Min-Heap envolve a inserção de elementos de maneira a manter a propriedade de heap, o que geralmente é feito utilizando um processo de "heapificação". Quando um novo elemento é adicionado ao heap, ele é inicialmente colocado na última posição da árvore. Em seguida, ele é comparado com seu nó pai e, se for menor, os dois nós trocam de posição. Este processo se repete até que a propriedade de heap seja restaurada.

  Além disso, a remoção do elemento de menor prioridade (a raiz) envolve a substituição da raiz pelo último elemento da árvore e a subsequente "heapificação" para baixo, onde o novo elemento na raiz é comparado com seus filhos e trocado com o menor deles, se necessário, até que a propriedade de heap seja restabelecida.

A eficiência das operações de inserção e remoção em um Min-Heap é garantida pela sua estrutura, que permite que ambas as operações sejam realizadas em tempo logarítmico em relação ao número de elementos no heap. Isso torna os Min-Heaps uma escolha excelente para aplicações que requerem a manipulação frequente de elementos com diferentes níveis de prioridade, especialmente quando a prioridade mais baixa deve ser atendida primeiro.

Exemplo:




Neste exemplo, 10 é a raiz e o menor elemento.

Heaps binários são implementados de maneira eficiente usando arrays, facilitando a localização dos pais e filhos de qualquer nó.

Heap de Fibonacci:

O heap de Fibonacci é uma estrutura de dados mais complexa que oferece operações mais eficientes para determinadas aplicações. Ele é baseado em uma coleção de árvores ordenadas e permite inserções e união de heaps em tempo constante, e remoções em tempo logarítmico amortizado.

**Características:**

- Coleção de árvores ordenadas.
- Operações de união e inserção eficientes.
- Manutenção de múltiplas árvores minimiza a necessidade de reestruturação.

**Vantagens:**

- Excelente desempenho em operações de união.
- Ideal para algoritmos que exigem muitas inserções e poucas remoções.

**Desvantagens:**

- Mais complexo de implementar e gerenciar.
- Pode ser menos eficiente na prática devido à sobrecarga constante.

## Vantagens e Desvantagens dos Heaps Binários:

**Vantagens:**

- Eficiência: Inserção e remoção em tempo \(O(\log n)\).
- Simplicidade: Fácil de implementar usando arrays.
- Uso de Memória: Estrutura quase completa evita desperdício de espaço.

**Desvantagens:**

- Não Ordenado Globalmente: Apenas a relação pai-filho é mantida, sem ordem global entre os elementos.
- Manutenção da Estrutura: Necessidade de ajuste após cada inserção ou remoção, o que pode introduzir sobrecarga.

## Implementação com Heaps Binários:

**Estrutura e Operações:**

Heaps binários são ideais para implementar filas de prioridade devido à eficiência nas operações principais:

- Inserção: Adicionar um novo elemento na primeira posição disponível e ajustar a estrutura para manter a propriedade de heap (heapify-up).
- Remoção do Elemento de Maior Prioridade: Substituir a raiz pelo último elemento e ajustar a estrutura para manter a propriedade de heap (heapify-down).
- Alteração de Prioridade: Modificar a prioridade de um elemento e ajustar a estrutura conforme necessário.

**Relações de Índices em Arrays:**

Heaps binários são frequentemente implementados usando arrays. As relações de índices são:

- Pai de um nó no índice $i$: $\lfloor \frac{i}{2} \rfloor$
- Filho esquerdo de um nó no índice $i$: $2i$
- Filho direito de um nó no índice $i$: $2i + 1$

## Filas de Prioridade:

Filas de prioridade são estruturas de dados que desempenham um papel fundamental em diversos contextos computacionais, permitindo a gestão eficiente de elementos com prioridades associadas. Em contraste com as filas tradicionais, conhecidas como filas FIFO (First In, First Out), onde os elementos são processados na ordem em que são inseridos, as filas de prioridade organizam e processam os elementos com base em suas prioridades.

Quando utilizamos uma fila de prioridade, cada elemento inserido é associado a uma prioridade específica. A fila é projetada de tal maneira que o elemento com a maior (ou menor) prioridade é sempre o primeiro a ser removido. Isso é extremamente útil em situações onde certos elementos devem ser processados antes de outros, independentemente da ordem em que foram inseridos. Por exemplo, em sistemas operacionais, filas de prioridade são usadas para gerenciar processos, garantindo que processos mais críticos sejam atendidos antes dos menos urgentes.

A operação de inserção em uma fila de prioridade envolve a adição de um novo elemento com sua respectiva prioridade. A estrutura da fila deve ser ajustada para manter a ordem de prioridades, o que pode ser realizado de forma eficiente utilizando estruturas como heaps binários ou heaps de Fibonacci. A operação de remoção, por outro lado, sempre retira o elemento com a maior (ou menor) prioridade da fila, garantindo que a próxima operação de remoção atenderá ao próximo elemento mais prioritário.

Em resumo, filas de prioridade são diferenciadas das filas FIFO tradicionais por sua capacidade de ordenar elementos com base em suas prioridades, permitindo um processamento mais flexível e eficiente em diversas aplicações. Elas são essenciais em sistemas que requerem a gestão dinâmica de prioridades, assegurando que os elementos mais importantes sejam processados primeiro, conforme necessário.

## Importância dos Heaps Binários em Filas de Prioridade:

Heaps binários são vitais para a implementação eficiente de filas de prioridade devido às suas propriedades estruturais. Eles garantem operações rápidas de inserção e remoção, essenciais em várias aplicações:

- Algoritmos de Busca: Algoritmos como A*, Dijkstra e Bellman-Ford utilizam filas de prioridade para explorar os nós mais promissores primeiro.
- Sistemas de Atendimento: Em filas de espera de hospitais, bancos ou serviços de suporte, filas de prioridade ajudam a atender clientes ou pacientes conforme a gravidade ou urgência.
- Escalonamento de Processos: Em sistemas operacionais, filas de prioridade decidem qual processo deve ser executado em seguida, considerando prioridade, tempo de espera e recursos disponíveis.
- Codificação de Huffman: Usadas para construir a árvore de Huffman, que codifica dados de forma eficiente.
- Compressão de Dados: Algoritmos como LZ77 utilizam filas de prioridade para localizar padrões de correspondência em strings de dados.

## Conclusão:

Filas de prioridade implementadas com heaps binários são essenciais para diversas aplicações que requerem operações eficientes baseadas em prioridade. Heaps binários, incluindo max-heaps e min-heaps, fornecem uma base sólida para essas implementações, combinando eficiência e simplicidade. O heap de Fibonacci oferece uma alternativa avançada para cenários específicos que exigem muitas inserções e uniões frequentes. Compreender essas estruturas de dados e suas operações é crucial para qualquer estudante de sistemas de informação, dado seu uso extensivo em problemas reais e complexos.

Link com Implementação do código: https://github.com/JosielPaz884/Trabalho-2.git

**Bibliográfia:**


- **Ime:** https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/heap.html

- **Wikipedia:**

  **https://pt.wikipedia.org/wiki/Heap#:~:text=Existem%20dois%20tipos%20de%20heaps,os%20de%20seus%20respectivos%20pais;**

  **https://pt.wikipedia.org/wiki/Fila_de_prioridade;**

  **https://en.wikipedia.org/wiki/Fibonacci_heap;**