Trabalho 2 – Estrutura de Dados 2024/01

Trabalho de pesquisa e implementação

## MEMBROS: BRAYAN MARTINS, CARLOS DANIEL MARTINS

## 

## Sumário

1. Introdução
2. Desenvolvimento
3. Filas de Prioridade Implementadas como Heaps Binários
   1. Heaps de Mínimo
   2. Características
   3. Lógica do Algoritmos
4. Filas de Prioridade Implementadas como Heaps Binários
   1. Heaps de Máximo
   2. Características
   3. Lógica do Algoritmos
5. Heaps de Fibonacci
   1. Conceitos
   2. Características
   3. Propriedades
   4. Representação
   5. Lógica do Algoritmos
6. Conclusão
7. Referências Bibliográficas
8. Introdução

Filas de prioridade são estruturas de dados fundamentais que permitem a inserção de elementos com prioridades associadas e a extração do elemento de maior ou menor prioridade. Estas estruturas são amplamente utilizadas em algoritmos de busca, otimização e sistemas operacionais. Neste trabalho, implementamos os algoritmos de Heap Mínimo, Heap Máximo e Heap de Fibonacci, abordando os conceitos de cada um na prática.

A primeira implementação de código foi o Heap Máximo, que teve como objetivo prático estabelecer padrões hierárquicos para a busca de doenças e sintomas em um ambiente na área da saúde. No caso do Heap Mínimo, elaboramos um algoritmo de árvore para simplificar a busca, utilizando o conceito de paginação comum na biologia computacional. O objetivo deste algoritmo é buscar dados e informações de forma mais rápida e eficiente, aprimorando os algoritmos de busca. A implementação do Heap de Fibonacci foi a mais complexa e desafiadora tanto em termos de código quanto de explicação teórica. O objetivo prático deste algoritmo é possibilitar a manipulação de grandes volumes de dados em menor tempo.

Este trabalho inclui exemplos práticos que podem ser utilizados com dados de diferentes escalabilidades, focando em simplificar a busca ou manipulação de dados de forma rápida e eficiente. Este relatório de implementação de heaps, do curso de Sistemas de Informação da Faculdade Antônio Meneghetti, tem como objetivo esclarecer aspectos de implementação de árvores na prática para a pesquisa de dados eficientes. Para isso, a metodologia utilizada foi a aplicação de Heap Mínimo, Heap Máximo e Heap de Fibonacci, de modo a fazer a coleta de dados e demonstrar suas funcionalidades e aplicações práticas.

## 

# Desenvolvimento

O objetivo geral deste relatório é explorar e implementar diferentes tipos de filas de prioridade, especificamente o Heap Mínimo, Heap Máximo e o Heap de Fibonacci, destacando suas aplicações práticas e importância na otimização de algoritmos de busca e manipulação de dados. Este trabalho visa demonstrar como essas estruturas de dados podem ser aplicadas para resolver problemas reais em diversas áreas, como saúde e biologia computacional, proporcionando uma compreensão aprofundada dos conceitos teóricos e práticos envolvidos.

Para alcançar este objetivo, buscamos:

1. Implementar e Analisar Estruturas de Heaps:

Desenvolver algoritmos para Heap Mínimo, Heap Máximo e Heap de Fibonacci, explicando suas bases teóricas e operações fundamentais como inserção, remoção e acesso ao elemento de maior ou menor prioridade.

1. Demonstrar Aplicações Práticas:

Aplicar o Heap Mínima em algoritmos de árvore para simplificar buscas utilizando paginação, comum na biologia computacional, melhorando a eficiência no acesso e processamento de grandes volumes de dados.

Mostrar como o Heap Máximo pode ser utilizado para estabelecer padrões hierárquicos na busca de doenças e sintomas, otimizando processos em ambientes de saúde.

Explorar o Heap de Fibonacci, apesar de sua complexidade, para demonstrar como sua eficiência pode ser crucial na manipulação de grandes conjuntos de dados em menor tempo, aplicável em contextos como redes de computadores e otimização de recursos.

1. Comparar e Avaliar Eficiência:

Avaliar as vantagens e desvantagens de cada tipo de heap em termos de tempo de execução, complexidade de implementação e adequação para diferentes tipos de problemas.

Comparar o desempenho das estruturas implementadas em cenários simulados, fornecendo insights sobre quando e por que usar cada tipo de heap.

3.Heap Mínimo

Objetivo Prático

Simplificar a busca utilizando o conceito de paginação em biologia computacional.

Implementação

Um heap mínimo é uma árvore binária completa onde o valor de cada nó é maior ou igual ao valor de seus filhos. A raiz contém o maior elemento do heap, o que é útil para priorizar e acessar rapidamente os elementos de maior valor ou importância.

Aplicação na Biologia Computacional

As árvores binárias são estruturas de dados que permitem a realização de busca ou pesquisa de forma eficiente [1]. Uma árvore binária pode atingir grandes dimensões, bem como ser utilizada para armazenar dados em memória secundária ou distribuídos pelos nodos de uma rede de computadores. Muitas aplicações em biologia computacional envolvem o processamento de strings utilizando árvores binárias não balanceadas [1].

O Heap mínimo pode ser aplicado para ordenar e acessar rapidamente grandes conjuntos de dados biológicos. Utilizando o conceito de paginação, é possível minimizar o armazenamento em memória e agilizar o processo de busca. Em biologia computacional, onde o volume de dados é frequentemente enorme, a eficiência no acesso e ordenação desses dados é crucial.[1]

Exemplo Prático:

Considere um banco de dados biológico com informações genéticas. Usando um heap mínimo, é possível priorizar as sequências genéticas mais relevantes ou frequentes, permitindo acesso rápido a esses dados para análise e pesquisa.

Lógica do Algoritmo para Heap Mínimo

O conceito de um heap mínimo é baseado em uma árvore binária completa onde o valor de cada nó é menor ou igual ao valor de seus filhos. A raiz contém o menor elemento do heap, assegurando que o elemento de maior prioridade (o menor valor) esteja sempre no topo.

Estrutura da Árvore Binária Completa:

Cada nó tem no máximo dois filhos.

A árvore é preenchida da esquerda para a direita sem lacunas.

Propriedade do Heap Mínimo:

A raiz é o menor elemento.

Cada nó é menor ou igual a seus filhos.

Inserção e Reestruturação:

Ao inserir um novo nó, ele é adicionado no final da árvore.

A árvore é reestruturada (heapify up) para manter a propriedade do heap mínimo.

Remoção da Raiz e Reestruturação:

A raiz (o menor elemento) é removida e substituída pelo último elemento da árvore.

A árvore é reestruturada (heapify down) para manter a propriedade do heap mínimo.

Implementação:

A implementação do heap mínimo utiliza uma estrutura de dados de lista para representar os nós.

As condições de comparação garantem que a propriedade do heap mínimo seja mantida ao inserir ou remover elementos.

## 

1. Heap Máximo

Objetivo Prático

Estabelecer padrões hierárquicos para a busca de doenças e sintomas em um ambiente na área da saúde.

Implementação

Um heap máximo é uma árvore binária completa onde o valor de cada nó é maior ou igual ao valor de seus filhos. A raiz contém o maior elemento do heap, garantindo que a prioridade mais alta (o maior valor) esteja sempre no topo. Isso é crucial para sistemas de triagem médica, onde os casos mais graves, identificados por valores mais altos, precisam ser tratados prioritariamente.

Aplicação na Saúde

O Heap máximo pode ser aplicado em sistemas de triagem médica para priorizar sintomas mais graves. Por exemplo, durante uma pandemia, como a COVID-19 em 2020 e 2021, é vital identificar rapidamente os pacientes com os sintomas mais severos. Um sistema de triagem baseado em heap máximo pode ajudar a ordenar os casos por gravidade, garantindo que os mais críticos sejam atendidos primeiro.

Exemplo Prático:

Imagine um sistema de triagem onde os sintomas são atribuídos com prioridades numéricas (maiores números indicam maior gravidade). Um paciente com dificuldade respiratória severa (prioridade 1) seria atendido antes de um paciente com febre leve (prioridade 5), já que a dificuldade respiratória severa possui a maior prioridade no sistema de triagem baseado em heap máximo.

Lógica do Algoritmo para o heap máximo

O conceito de um heap máximo é baseado em uma árvore binária completa onde o valor de cada nó é maior ou igual ao valor de seus filhos. A raiz contém o maior elemento do heap, garantindo que o elemento de maior prioridade esteja sempre acessível.

Estrutura da Árvore Binária Completa: Cada nó tem no máximo dois filhos. A árvore é preenchida da esquerda para a direita sem lacunas.

Propriedade do Heap Máximo: A raiz é o maior elemento. Cada nó é maior ou igual a seus filhos.

Inserção e Reestruturação: Ao inserir um novo nó, ele é adicionado no final da árvore. A árvore é reestruturada (heapify up) para manter a propriedade do heap máximo.

Remoção da Raiz e Reestruturação: A raiz é removida e substituída pelo último elemento da árvore. A árvore é reestruturada (heapify down) para manter a propriedade do heap máximo. A implementação do heap máximo usa uma estrutura de dados de lista para representar os nós. As condições de comparação garantem que a propriedade do heap máximo seja mantida ao inserir ou remover elementos.

## 

## 

## 

1. Heap de Fibonacci

Objetivo Prático

Manipular grandes quantidades de dados de forma eficiente, minimizando o tempo de execução de operações críticas como inserção, diminuição de chave e extração do mínimo.

Implementação

Um Heap de Fibonacci é uma estrutura de dados que consiste em uma coleção de árvores enraizadas, sendo um dos heaps mais eficientes para várias operações devido às suas propriedades de amortização.

Cormen et al. [2] aﬁrmam que uma Heap de Fibonacci é uma extensão das Heaps Binomiais. Heaps Binomiais são formadas por uma lista ligada de árvores binomiais, diferente de uma Heap Binária, que tem forma de uma única árvore. Heaps de Fibonacci contém uma estrutura mais distendida que permite um limite de tempo assintótico melhor, fazendo uso dessa propriedade principalmente quando o problema apresenta valores altos para execucão, por exemplo extract-min e delete. Em situacões onde operacões de remocão são

O Heap de Fibonacci é particularmente útil em situações onde operações repetidas de união, inserção e diminuição de chave são necessárias. Em contextos onde grandes quantidades de dados precisam ser manipuladas eficientemente, como em redes de computadores, otimização de recursos e grandes bancos de dados, o Heap de Fibonacci oferece uma vantagem significativa devido à sua eficiência amortizada.

Exemplo Prático:

Considere um sistema de roteamento de redes onde as rotas mais curtas precisam ser recalculadas frequentemente à medida que a rede muda. Um Heap de Fibonacci pode manter a estrutura de dados para distâncias mais curtas e permitir atualizações rápidas conforme os nós e as distâncias são alterados.

Lógica do Algoritmo

O Heap de Fibonacci opera com a seguinte lógica:

1. Estrutura de Árvores Enraizadas:
   * A estrutura consiste em uma coleção de árvores enraizadas que formam uma floresta.
   * Cada árvore obedece à propriedade de heap mínimo.
2. Propriedade de Heap Mínimo:
   * A raiz contém o menor elemento, e cada nó é menor ou igual a seus filhos.
3. Operações Eficientes:
   * Inserção: Adiciona um novo nó à lista de raízes.
   * Extração do Mínimo: Remove a raiz mínima e reestrutura a floresta para manter a propriedade do heap.
   * Decremento de Chave: Reduz o valor de uma chave e, se necessário, move o nó para a lista de raízes, cortando-o de seu pai.
   * União: Combina duas heaps de Fibonacci em uma única heap, mantendo a eficiência.
4. Consolidação:
   * Após a extração do mínimo, as árvores da floresta são consolidadas para garantir que não existam duas árvores com o mesmo grau.
5. Conclusão

A conclusão deste relatório sobre Filas de Prioridade e Heaps destaca a importância e as aplicações práticas das estruturas de dados estudadas: Heap Mínimo, Heap Máximo e Heap de Fibonacci. Cada uma dessas estruturas oferece características distintas que as tornam ideais para diferentes contextos e problemas computacionais.

Inicialmente, exploramos o Heap Mínimo, uma estrutura onde o menor elemento está sempre na raiz da árvore binária completa. Esta propriedade é fundamental em aplicações que exigem acesso rápido ao elemento de menor prioridade, como na ordenação e acesso a grandes volumes de dados em biologia computacional. A implementação prática demonstrou como o Heap Mínimo pode ser utilizado eficientemente para otimizar processos de busca e manipulação de dados.

Em seguida, abordamos o Heap Máximo, onde a raiz contém o maior elemento, adequado para sistemas de triagem médica e outras aplicações onde é crucial priorizar operações com os elementos de maior prioridade. A exemplo prático envolvendo triagem de pacientes, ilustramos como o Heap Máximo pode ser implementado para garantir que os casos mais graves sejam tratados com prioridade máxima, otimizando recursos e tempo de resposta.

Finalmente, exploramos o Heap de Fibonacci, uma estrutura avançada que oferece eficiência amortizada para operações como inserção, extração do mínimo e diminuição de chave. Esta estrutura é particularmente útil em cenários onde a manipulação de grandes volumes de dados precisa ser feita de forma rápida e eficiente, como em redes de computadores e otimização de recursos.

Comparativamente, cada tipo de heap apresenta vantagens e desvantagens dependendo do contexto de aplicação. O Heap Mínimo e o Heap Máximo são mais simples de implementar e oferecem bom desempenho para muitos problemas comuns. Por outro lado, o Heap de Fibonacci, apesar de mais complexo de implementar, proporciona vantagens significativas em termos de tempo de execução amortizado para operações críticas.

1. **Referências**

[1] Tavares. R. A. E et al. Um Algoritmo para Paginação de Árvores Binárias Baseado em Empacotamento Unidimensional

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT press, 2009.