UniSENAI

Trilha 05

Tabelas Hash

Instruções para a melhor prática de Estudo

1. **Leia atentamente todo o conteúdo:** Antes de iniciar qualquer atividade, faça uma leitura detalhada do material fornecido na trilha, compreendendo os conceitos e os exemplos apresentados.

2. **Não se limite ao material da trilha:** Utilize o material da trilha como base, mas busque outros materiais de apoio, como livros, artigos acadêmicos, vídeos, e blogs especializados. Isso enriquecerá o entendimento sobre o tema.

3. **Explore a literatura:** Consulte livros e publicações reconhecidas na área, buscando expandir seu conhecimento além do que foi apresentado. A literatura acadêmica oferece uma base sólida para a compreensão de temas complexos.

4. **Realize todas as atividades propostas:** Conclua cada uma das atividades práticas e teóricas, garantindo que você esteja aplicando o conhecimento adquirido de maneira ativa.

5. **Evite o uso de Inteligência Artificial para resolução de atividades:** Utilize suas próprias habilidades e conhecimentos para resolver os exercícios. O aprendizado vem do esforço e da prática.

6. **Participe de debates:** Discuta os conteúdos estudados com professores, colegas e profissionais da área. O debate enriquece o entendimento e permite a troca de diferentes pontos de vista.

7. **Pratique regularmente:** Não deixe as atividades para a última hora. Pratique diariamente e revise o conteúdo com frequência para consolidar o aprendizado.

8. **Peça feedback:** Solicite o retorno dos professores sobre suas atividades e participe de discussões sobre os erros e acertos, utilizando o feedback para aprimorar suas habilidades.

Essas instruções são fundamentais para garantir um aprendizado profundo e eficaz ao longo das trilhas.

Tabelas Hash

---

1. Conceito de Hashing e Funções de Hash

Hashing é uma técnica usada para mapear dados de tamanho variável em valores de tamanho fixo, chamados de **hash codes** ou **hash values**. Esse processo é feito por meio de uma **função de hash**, que recebe uma entrada (por exemplo, uma chave) e retorna um valor (o código hash), que é utilizado como o índice para armazenar dados em uma tabela hash.

- **Função de Hash**: Uma função de hash é um algoritmo que transforma uma chave de tamanho arbitrário em um índice para uma tabela de tamanho fixo.
    - **Exemplo**: Uma função simples pode ser h(k)=kmod mh(k) = k \mod mh(k)=kmodm, onde kkk é a chave e mmm é o tamanho da tabela.

---

2. Colisões e Técnicas de Tratamento

Uma **colisão** ocorre quando duas chaves diferentes são mapeadas para o mesmo índice da tabela hash. Para resolver colisões, existem duas técnicas principais:

1. Encadeamento (Chaining):

- Cada posição da tabela hash armazena uma lista (ou cadeia) de todas as chaves que mapeiam para o mesmo índice.
- Quando uma colisão ocorre, a nova chave é adicionada à lista associada àquele índice.

Exemplo em Pseudocódigo (Encadeamento):

```
function insert(key, value):
    index = hash(key)  // Calcula o índice
    if table[index] is empty:
        table[index] = new list()
    table[index].append((key, value))  // Insere na lista encadeada

function search(key):
    index = hash(key)
    for each (k, v) in table[index]:
        if k == key:
            return v
    return None  // Não encontrado
```

2. Endereçamento Aberto (Open Addressing):

- Em vez de usar listas encadeadas, o **endereçamento aberto** resolve colisões procurando outra posição na tabela (linearmente, quadraticamente, ou por duplo hashing).
- **Probing linear**: Ao ocorrer uma colisão, busca-se a próxima posição disponível (incrementando sequencialmente o índice).
- **Probing quadrático**: A próxima posição disponível é determinada por i2i^2i2, onde iii é o número de tentativas.

Exemplo em Pseudocódigo (Probing Linear):

```
function insert(key, value):
    index = hash(key)
    while table[index] is not empty:
        index = (index + 1) mod table_size  // Probing linear
    table[index] = (key, value)

function search(key):
    index = hash(key)
    while table[index] is not empty:
        if table[index].key == key:
            return table[index].value
        index = (index + 1) mod table_size
    return None  // Não encontrado
```

---

3. Aplicações das Tabelas Hash

Tabelas hash têm diversas aplicações em sistemas computacionais que requerem acesso rápido a dados, tais como:

- **Dicionários**: Tabelas hash são usadas para implementar estruturas de dados de dicionários, como no Python ou JavaScript.
- **Caches**: Um cache pode usar uma tabela hash para mapear endereços de memória a blocos de dados armazenados.
- **Sistemas de Banco de Dados**: As tabelas hash são usadas para acelerar a busca por registros em grandes bases de dados.
- **Verificação de Integridade**: As funções de hash são usadas em algoritmos de verificação de integridade de dados, como checksums ou assinaturas digitais.

---

4. Análise de Desempenho

O desempenho de uma tabela hash depende de sua função de hash e da técnica de resolução de colisão. O tempo esperado para operações de inserção, busca e remoção é **O(1)** (tempo constante), desde que a função de hash distribua as chaves uniformemente. No pior caso, o desempenho pode ser **O(n)**, caso todas as chaves colidam no mesmo índice.

Fatores que Afetam o Desempenho:

- **Função de Hash**: Deve ser bem projetada para evitar muitas colisões. Funções que distribuem chaves uniformemente pela tabela são ideais.
- **Tamanho da Tabela**: Se a tabela é muito pequena em comparação ao número de chaves, haverá mais colisões.
- **Carga da Tabela (Load Factor)**: É a proporção entre o número de elementos e o tamanho da tabela. Se o load factor for muito alto, o número de colisões aumentará.

Comparação entre Encadeamento e Endereçamento Aberto:

- **Encadeamento**: Garante tempo de busca eficiente, mesmo com muitos elementos, desde que as listas encadeadas não fiquem muito grandes.
- **Endereçamento Aberto**: Pode ser mais eficiente em termos de uso de memória, mas pode degradar o desempenho à medida que o load factor cresce.

Lista de Exercícios de Fixação

1. **Implementação de Funções de Hash**:
    - Implemente uma função de hash simples que recebe uma chave inteira e retorna um índice em uma tabela de tamanho 10.
    - Modifique a função para que funcione com strings, somando os valores ASCII dos caracteres e utilizando o operador módulo.
2. **Encadeamento**:
    - Implemente uma tabela hash com encadeamento, onde cada índice da tabela armazena uma lista encadeada de pares (chave, valor).
    - Crie as funções para inserir, buscar e remover elementos da tabela.
3. **Endereçamento Aberto (Probing Linear)**:
    - Implemente uma tabela hash utilizando probing linear para resolver colisões.
    - Verifique o comportamento da tabela à medida que você insere mais elementos, e analise o que acontece quando a tabela se aproxima de sua capacidade máxima.
4. **Comparação de Técnicas de Tratamento de Colisões**:
    - Implemente tanto o encadeamento quanto o endereçamento aberto e compare o desempenho de ambas as técnicas em um conjunto de 1000 inserções e buscas.
    - Utilize uma função de hash simples e um load factor de 0.75. Qual das abordagens apresenta melhor desempenho?

5. **Aplicação Prática de Tabela Hash**:
    - Implemente um sistema de dicionário utilizando tabelas hash, onde o usuário pode armazenar e recuperar palavras com seus significados. Use encadeamento para resolver colisões.
    - Adicione a funcionalidade para lidar com remoção de palavras e buscar palavras que não estão no dicionário.
6. **Análise de Desempenho**:
    - Crie uma tabela hash e insira 500 elementos utilizando uma função de hash eficiente. A tabela deve ser ajustada para diferentes tamanhos (50, 100, 250).
    - Meça o tempo médio de busca e remoção de elementos e discuta como o tamanho da tabela afeta o desempenho.
7. **Função de Hash Personalizada**:
    - Crie uma função de hash para strings que distribua os valores uniformemente para uma tabela de tamanho 100. Teste a função com diferentes conjuntos de dados e observe a distribuição dos índices gerados.
    - Qual a proporção de colisões que você observa? Como você ajustaria a função para melhorar a distribuição?

Nota

As tabelas hash são fundamentais para a implementação de estruturas de dados que requerem eficiência no armazenamento e recuperação de informações. Embora sua operação seja geralmente constante, a escolha de uma boa função de hash e uma técnica eficiente para resolver colisões é crucial para manter o desempenho esperado.