Projeto 1 - Estruturas de Dados (ADS)

Título do Projeto: HashTablePy - Implementação de uma Tabela Hash em Python

Equipe: até 5 Alunos

1. Ideia Central

Desenvolver uma classe em Python que implemente a Tabela Hash (também conhecida como Mapa Hash). A classe deverá permitir a inserção, busca e remoção de pares chave-valor. O foco principal será na implementação da função de hash (mapeamento da chave para um índice do array subjacente) e na estratégia de **tratamento de colisões** (quando duas chaves diferentes mapeiam para o mesmo índice). Uma estratégia comum e didática é o **Encadeamento Separado (Separate Chaining)**, usando listas ligadas ou listas Python em cada posição do array.

Estruturas de Dados Chave:

- **Lista (**list**) ou Array:** A estrutura base da tabela hash, um array de tamanho fixo (ou redimensionável) onde cada posição pode apontar para o início de uma lista de elementos que colidiram naquela posição.
- **Listas Ligadas (Implementadas manualmente com Nós) ou Listas Python (**list**):** Usadas em cada "balde" (bucket) do array principal para armazenar os múltiplos pares chave-valor que podem ter colidido no mesmo índice. (Usar listas Python é mais simples, implementar listas ligadas é mais clássico/didático para estruturas de dados).
- **Função de Hash:** Uma função que recebe uma chave e retorna um índice dentro dos limites do array base. Pode começar com algo simples (ex: usando o hash() embutido do Python e o operador módulo %).

2. Aplicação e Relevância

Tabelas Hash são uma das estruturas de dados mais importantes e amplamente utilizadas na computação. Elas são a base para dicionários/mapas em muitas linguagens (incluindo Python), usadas em caches, indexação de bancos de dados, conjuntos (sets), e muito mais. Entender como funcionam, especialmente o crucial tratamento de colisões, é fundamental para a ciência da computação e engenharia de software. Este projeto permite explorar diretamente esses mecanismos.

3. Tecnologia

- **Linguagem de Programação:** Python 3.x
- **Bibliotecas Padrão:** Nenhuma biblioteca externa é estritamente necessária, o foco é na implementação usando estruturas básicas do

Python (listas, classes para Nós se usar lista ligada). hash() embutido pode ser usado para a função de hash inicial.

- **Ambiente:** Uma classe Python (HashTable) e um script simples para demonstrar seu uso (possivelmente via CLI).

4. Cronograma de Execução (5 Semanas)

- **Semana 1: Pesquisa, Planejamento e Estrutura Inicial**
    - **Atividades:** Estudo dos conceitos (Tabela Hash, função de hash, colisões, Encadeamento Separado, fator de carga). Definição da arquitetura (Classe HashTable, classe interna Node se usar lista ligada, métodos _hash, put, get, delete). Decisões de implementação (tamanho inicial do array, como implementar o encadeamento - lista Python ou ligada). Divisão de tarefas. Configuração do Git/GitHub.
    - **Entrega Semanal 1:** Documento de resumo dos conceitos, diagrama de classe, decisões de implementação, plano de tarefas, link do repositório.
- **Semana 2: Implementação da Estrutura Base e Função Hash**
    - **Atividades:** Implementar a classe HashTable com o construtor (inicializando o array/lista base com None ou listas vazias). Implementar a função de hash interna _hash(chave) que retorna um índice. Se usar listas ligadas, implementar a classe Node.
    - **Verificação:** Testes básicos para garantir que a estrutura é criada e a função hash retorna índices válidos.
    - **Entrega Semanal 2:** Classe HashTable inicial com array base, função hash e (se aplicável) classe Node. Relatório de progresso.
- **Semana 3: Implementação da Inserção (**put**) com Tratamento de Colisão**
    - **Atividades:** Implementar o método put(chave, valor): calcular o hash da chave para obter o índice; acessar a lista (ligada ou Python) nesse índice; verificar se a chave já existe na lista (atualizar valor) ou adicionar um novo nó/elemento (chave, valor) à lista.
    - **Verificação:** Testar inserções, incluindo casos que causem colisões (chaves diferentes mapeando para o mesmo índice) e verificar se todos os valores são armazenados corretamente na lista encadeada/Python correspondente.
    - **Entrega Semanal 3:** Método put funcional. Relatório com resultados dos testes de inserção e colisão.

- **Semana 4: Implementação da Busca (**get**) e Remoção (**delete**)**
    - **Atividades:** Implementar o método get(chave): calcular o hash, buscar na lista correspondente pela chave e retornar o valor (ou None/erro se não encontrada). Implementar o método delete(chave): calcular o hash, buscar na lista e remover o nó/elemento correspondente.
    - **Verificação:** Testar buscas por chaves existentes e inexistentes. Testar remoções e verificar se a busca falha após a remoção.
    - **Entrega Semanal 4:** Métodos get e delete funcionais. Script de demonstração CLI básico que usa a classe. Rascunho do README.md.
- **Semana 5: Refinamento, Documentação e Apresentação**
    - **Atividades**: Implementar redimensionamento (rehashing) quando o fator de carga excede um limite. Refinar código e comentários. Finalizar documentação (README com explicação da Tabela Hash, Encadeamento Separado, como usar a classe). Preparar apresentação (problema, solução Tabela Hash, tratamento de colisões, estruturas usadas, demonstração). Garantir código no Git.
    - **Entrega Final:** Código-fonte final (Classe HashTable e demo), documentação, slides.

5. Critérios de Avaliação

- **Funcionalidade:** A Tabela Hash executa corretamente as operações put, get, delete? O tratamento de colisões funciona?
- **Aplicação de Conceitos:** Os conceitos de hashing, tabela hash e tratamento de colisões (Encadeamento Separado) foram implementados corretamente? As estruturas de dados base (array, listas) foram usadas adequadamente?
- **Qualidade do Código:** Código Python claro, modular, comentado.
- **Documentação:** README claro explicando a implementação e o uso.
- **Trabalho em Equipe:** Colaboração via Git, cumprimento do cronograma.
- **Apresentação:** Clareza na explicação da estrutura, do tratamento de colisões e demonstração.