Universidade Federal de Ouro Preto
ICEA/DECSI
Professor: Bruno Pereira dos Santos

Disciplina: AEDS I

Trabalho Prático 2 – Hashing

Duração: 3 semanas

1 Objetivos

O objetivo deste trabalho é analisar a estrutura de dados Hash com encadeamento separado (*separete chaining*).

2 Descrição

Este trabalho está dividido em três partes descritas abaixo:

2.1 Leitura dos dados

Você deverá escrever um programa para ler um arquivo contendo dados de 30228 escolas no ENEM. Cada linha do arquivo representa uma escola e é composta por **nove** informações, separadas por ponto e vírgulas (você pode usar a função **strtok**[1]):

- id
- estado
- município
- rede
- media_ciencias_natureza
- media_ciencias_humanas
- media_linguagem
- media_matematica
- media_redacao

Os dados de cada escola devem ser armazenados em uma **Tabela Hash** descrita na próxima seção.

Você também deverá implementar a função:

- `void imprime_dado(<tipo> var)`
  - Esta função recebe uma estrutura e imprime as informações.

[1] Exemplo de uso da função strtok

2.2 Hash com encadeamento separado

Dada a implementação da **Tabela Hash** vista em aula, faça adaptações no código de modo a implementar uma **Tabela Hash com Encadeamento Separado (*Separete Chaining*)**[2] para armazenar dados das escolas no ENEM. Use a implementação da lista sequencial também estudada e fornecida.

A inserção e busca por escola na **Tabela Hash** deverá ser feita pelo ID da escola.

- **Inserção**: no caso da posição estar vaga, uma lista sequencial deverá ser criada e armazenada na **Tabela Hash**. Do contrário, os dados deverão ser armazenados ou no início ou o final da **lista** (resolvendo colisões).
  - Observe que não deverá ser possível inserir escolas com o mesmo ID.
- **Busca**: no caso da posição estar vaga, o elemento não existe. Do contrário, deverá ser feita uma **busca linear** na **lista**.

Observe que temos duas **TAD's** definidas em bibliotecas separadas. Deste modo, a **Tabela Hash** poderá apenas usar as funções da **Lista Sequencial**, isto é, nunca terá acesso direto a estrutura da lista.

2.3 O que analisar?

Sabendo que n é o número de chaves (30228), M é o número de *slots* (espaços) na tabela hash e b_j é o número de itens em cada *slot*.

- Avalie a Tabela Hash nos seguintes indicadores:
  - Número de *slots* (espaços) vazios na Tabela Hash;
  - Tamanho médio das listas nos *slots*;
  - Número de colisões;
  - Tempo para inserir todos os dados[3].
- Mostre como os indicadores acima citados variam ao alterar os seguintes parâmetros:
  - Tamanho M da Tabela Hash
    - $M \in \{3022, 15114, 22671, 30228, 60456\}$
  - Função Hash: $f(key) = key \mod < NUM >$
    - $f(key) = key \mod 1$
    - $f(key) = key \mod 10$
    - $f(key) = key \mod 100$
    - $f(key) = key \mod M$
      - Observe que $< NUM >$ não pode ser maior que o tamanho da tabela hash.
- Teste a Função Hash.

[2] O que é uma hash com encadeamento separado.
[3] Como medir o tempo de uma função.

- Ao testar uma função hash, a uniformidade da distribuição dos valores na hash pode ser avaliada através do teste qui-quadrado (χ^2)[4][5][6]. Esse teste é uma medida de qualidade do ajuste. Em outras palavras, é a distribuição real dos itens nos *slots* versus a distribuição esperada (ou uniforme) dos itens.
- A formula do teste qui-quadrado é:

$$test = \frac{\frac{\sum_{j=0}^{M-1}(b_j)(b_j+1)}{2}}{(\frac{n}{2M})(n+2M-1)}$$

- A soma no numerador $\frac{\sum_{j=0}^{M-1}(b_j)(b_j+1)}{2}$ estima o número de *slots* que deveriam ser visitados para encontrar o valor desejado. O denominador $(\frac{n}{2M})(n+2M-1)$ é o numero de *slots* visitados em uma função hash ideal the coloca cada item em um *slot* aleatório. Assim, se a função é ideal, então a formula dará valor 1. Na realidade, uma boa função dará algo entre 0.95 e 1.05. Se o valor é maior, então exite um alto número de colisões (lentidão ao inserir, buscar ou remover). Se o valor é menor, a função dá menos colisões o que não é ruim, mas existe a questão do espaço utilizado.

RESPONDA:

- QUAL É A IMPORTÂNCIA DO TAMANHO DA TABELA HASH, FUNÇÃO HASH E QUANTIDADE DE ELEMENTOS INSERIDOS NO DESEMPENHO (TEMPO E MEMÓRIA) DAS SOLUÇÕES QUE UTILIZAM HASH?
  - Qual é a implicação de M (tamanho da tabela hash) ser muito grande?
  - Qual é a implicação de M (tamanho da tabela hash) ser muito pequena?
- EXISTE UM FATOR (RELAÇÃO) TAMANHO DA TABELA HASH E NÚMERO DE ELEMENTOS INSERIDOS PARA QUE A BUSCA (TEMPO) NA TABELA HASH SEJA EFICIENTE?

Procure organizar inteligentemente os dados coletados em tabelas, e também construa gráficos a partir dos dados (possíveis ferramentas Excel, R, matplotlib, plotly, Google sheets, Gnuplot e outros). Então, disserte sobre os dados nas tabelas e gráficos. Grande parte da avaliação será realizada sobre a análise dos resultados, ou seja, sobre o que você dissertar.

Você também poderá utilizar implementações diversas daquelas apresentadas em sala de aula, além de otimizações, visando completude do trabalho.

Essas informações devem está presentes na seção de estudo de complexidade da documentação.

[4] Hash functions: An empirical comparison.
[5] Saiba um pouco mais sobre hashs.
[6] Saiba mais sobre o teste χ^2

2.4 Execução

O seu programa deverá imprimir informações na seguinte ordem:

- Número de *slots* (espaços) vazios na Tabela Hash;
- Tamanho médio das listas nos *slots*, isto é, soma do tamanho das listas pelo número possível de listas;
- Número de colisões;
- Tempo para inserir todos os dados[7].

Um detalhe importante é que o programa deverá ser executado passando-se opções na linha de comando[8]. Os parâmetros do programa devem ser definidos assim:

> hashenem [argumentos]

Argumentos mandatórios (na sequência):

- Na sequência podemos passar o ID que se deseja buscar ou um conjuntos de ID's armazenados em um arquivo.
  - `<ID>` indica um único ID que se deseja buscar
  - `-b <arquivo>` é um parâmetro que indica que será executada a busca para um conjunto de ID's armazenados no diretório <arquivo>;
- `-d <arquivo>` o endereço de um arquivo contendo informações das escolas;
- -M <tamanho> tamanho da tabela hash.
- -mod <numero> número (positivo ≥ 1) que será usado na função hash.

Argumentos opcionais (na sequência):

- -p parâmetro opcional; caso presente seu programa deverá imprimir os ID que foram encontrados.

Exemplos de chamada seriam:

```
> hashenem 4810 -d ./dados-enem.txt -M 30228 -mod 1 -p
```

Este comando executará a busca pelo ID `4810` em uma tabela hash contendo os dados do arquivo `./dados-enem.txt`. A Tabela Hash terá tamanho `30228` e usará a função $f(key) = key \mod 1$ para inserir e buscar os dados. Este comando pede para imprimir o resultado da busca.

```
> hashenem -b ./ids.txt -d ./dados-enem.txt -M 15114 -mod 15114
```

[7] Como medir o tempo de uma função.
[8] Como ler parâmetros por linha de comando.

Este comando executará a busca pelos IDs presentes no arquivo `./ids.txt` em uma tabela hash contendo os dados do arquivo `./dados-enem.txt`. A Tabela Hash terá tamanho `15114` e usará a função $f(key) = key \mod 15114$ para inserir e buscar os dados. Este comando **não** pede para imprimir o resultado da busca.

Descubra como ler os parâmetros da linha de comando e defina (e documente) a sintaxe a ser utilizada.

3 O que deve ser entregue?

- Código fonte do programa em C (bem identada e comentada). NOVAMENTE, CÓDIGOS FONTE .c e, eventualmente, ARQUIVOS DE CABEÇALHO .h.
- Documentaço do trabalho seguindo o padrão da SBC (Template SBC.). A DOCUMENTAÇÃO DEVERÁ CONTER OBRIGATORIAMENTE **NO MÁXIMO 3 PÁGINAS**. Entre outras coisas, a documentação deve conter:

  (NOTE QUE É UM EXEMPLO DE ESTRUTURA. ITENS PODEM VARIAR A DEPENDER DO QUE É PEDIDO (O QUE SE DEVE ANÁLISAR))

  1.) **Introdução:** descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.

  2.) **Implementação:** descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.

  3.) **Estudo de complexidade:** estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).

  4.) **Listagem de testes executados:** os testes executados devem ser simplesmente apresentados.

  5.) **Conclusão:** comentários gerais sobre o trabalho.

  6.) **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.

  7.) **Em LaTeX:** Caso o trabalho seja elaborado/escrito em LaTeX. Recomendo uso do Overleaf que já conta com o template da SBC.

  8.) **Formato:** **OBRIGATORIAMENTE EM PDF**

4 Dicas

- Consulte as dicas do Prof. Nívio Ziviani de como deve ser feita uma boa implementação e documentação de um trabalho prático (LINK).
- Clique aqui e veja uma ótima referência para estudos de C e algoritmos de ordenação.
- Algoritmos de ordenação em pt-BR.
- Como medir o tempo de uma função em C.
- Como gerar números aleatórios.
- Referência às bibliotecas do C.

5 Como deve ser feita a entrega

A entrega DEVE ser feita pelo Moodle na forma de um único arquivo compactado (zip ou rar), contendo o(s) código(s), os arquivos (de teste se necessário) e a documentação. *Também deve ser entregue a documentação impressa (em frente/verso) na próxima aula após a data de entrega do trabalho.*

6 Comentários gerais:

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- A qualidade da solução/código também faz parte da avaliação;
- Clareza, indentação e comentários no programa também vão valer pontos;
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados (documentação e/ou código fonte) terão nota zero;
- Trabalhos entregue em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.