Universidade Federal de Ouro Preto
ICEA/DECSI **Disciplina: AEDS I**

Trabalho Prático 1 – Busca e Ordenação

Duração: 3 semanas

1 Objetivos

O objetivo deste trabalho é analisar algoritmos de ordenação e busca.

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 (use a função **strtok**):

- 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 um vetor (*array*) de estruturas que guarda essas informações.

Você também deverá implementar as seguintes funções:

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

2.2 Busca e ordenação

Você deverá implementar dois algoritmos de busca para encontrar informações das escolas no ENEM.

1.) Você deverá implementar o algoritmo de **Busca Sequencial** (sem ordenar) para a operação de busca pelo ID da escola.

2.) Você também deverá implementar o algoritmo **Busca Binária** que recebe um vetor/-lista de informações **ordenadas**. Você é livre para escolher a implementação recursiva ou iterativa. Porém deixe claro na documentação qual foi a sua escolha.

- Para ordenar os dados (antes da busca binária), você deve implementar algoritmos de ordenação distintos (**ao menos o Insertion Sort, Quick Sort e o Heap Sort**).

Vale notar que a execução do algoritmo de ordenação deve ser feita somente uma vez antes da(s) busca(s).

2.3 O que analisar?

A análise deve ser feita sobre o tempo de execução.

- Faça uma estatística do tempo que as funções implementadas consomem a medida que se aumenta o número de operações de busca (por exemplo, 1000, 2000, 3000 operações de busca[1]).
  - Meça o tempo de cada conjunto de buscas utilizando a função `clock()`[2]
- Apresente uma tabela comparativa da complexidade de tempo:
  - Busca Sequencial;
  - Busca Busca Sequencial mais:
    - Insertion Sort
    - Quick Sort
    - Heap Sort
- Compare o tempo da busca com o tempo para realizar a ordenação dos dados. De modo a garantir uma comparação justa, utilize o arquivo **indices.txt** como entrada para as operações de busca. Este arquivo contém um total de 1.000.000 de valores de ID's.
  - Responda: é possível identificar uma quantidade mínima de buscas de modo que compense o custo de ordenar os dados?

[1] Você pode gerar ID de escolas aleatórios para realizar esse experimento

[2] Como medir o tempo de uma função.

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.

2.4 Execução

O seu programa deverá imprimir informações na seguinte ordem:

- O(s) resultado(s) do(s) item(ns) buscado(s) com êxito;
- O método de busca utilizado e, caso seja o Busca Binária, também o nome do algoritmo de ordenação utilizado;
- O tempo de execução.

Um detalhe importante é que o programa deverá ser executado passando-se opções na linha de comando[3]. Por exemplo, os parâmetros do programa podem ser definidos assim:

> buscaenem <algoritmo-busca> [-s <ordenacao>] [<ID> | -b <arquivo>] [-p]

Onde:

- `<algoritmo-busca>` é um parâmetro que indica qual algoritmo de busca será usado (os as siglas `bb` – para busca binária e `bs` – para busca sequencial) será utilizado;
- `-s <ordenacao>` é um parâmetro opcional que indica que será utilizado um algoritmo de ordenação específico será executado antes da busca binária (use `is` – insertion sort, `qs` – quick sort, `hs` – heap sort);
- 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>;
- -p parâmetro opcional; caso presente seu programa deverá imprimir os ID que foram encontrados.

Exemplos desse tipo de chamada seriam:

```
> buscaenem bs 4810 -p
```

Este comando executará a busca sequencial pelo ID 4810 e imprimirá o resultado da busca.

[3] Como ler parâmetros por linha de comando.

```
> buscaenem bb -s qs -b /home/bps/playground/lectures-demo/ids.txt -p
```

Este comando a busca binária depois de ordenar os dados do ENEM usando o quick sort e então realizará a busca por todos os ID's presentes no arquivo ids.txt e, finalmente, imprimirá o resultado das buscas. Observe que o diretório passado por parâmetro é meramente ilustrativo.

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 UM MÁXIMO DE 5 PÁGINAS**. Entre outras coisas, a documentação deve conter:

  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.