TRABALHO PRÁTICO
PARTE 2

PRAZO PARA ENTREGA:16/07/2021

1. Introdução

Spotify é um serviço de streaming de música e podcast mais popular do mundo. O serviço se baseia no pagamento de royalties considerando o número de audições de um artista como proporção total de músicas ouvidas no serviço. A base de dados disponível em https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks reúne informações de aproximadamente 600.000 músicas lançadas entre 1922 até abril de 2021. Além disso, agrupa informações de aproximadamente 1.1 milhões de artistas. Esses dados podem ser encontrados no link supracitado, nos arquivos *tracks.csv* e *artists.csv*. No mesmo link, encontra-se os atributos de cada uma dessas entidades.

2. Objetivos

O objetivo principal deste trabalho é dar ao aluno a oportunidade de analisar e comparar o desempenho de diferentes algoritmos de ordenação aplicados a um conjunto de dados reais. Espera-se que o aluno seja capaz de:

- Manipular adequadamente arquivos em modo texto e binário.
- Compreender e implementar corretamente algoritmos de ordenação;
- Analisar o desempenho dos algoritmos implementados segundo o referencial teórico adquirido na disciplina;
- Apresentar os resultados obtidos com o trabalho de maneira formal.

3. Desenvolvimento

As etapas para o desenvolvimento do trabalho são descritas abaixo.

Etapa 0: Carregamento dos dados

Nesta etapa deve utilizar a implementação feita na entrega anterior para recuperar N registros aleatórios dos arquivos.

Etapa 1: Análise de algoritmos de ordenação

Nesta etapa, você irá comparar o desempenho de diferentes algoritmos de ordenação quando aplicados sobre os dados pré-processados. O seu programa deverá seguir os seguintes passos:

1. importar conjuntos de N registros aleatórios do arquivo binário *artists.bin* gerado pelo pré-processamento
2. realizar a ordenação desses registros, utilizando como chave de ordenação *followers*. Durante a ordenação, deverão ser computados o total de comparações de chaves e o total de movimentações de chaves. Além disso, o tempo de execução do algoritmo deverá ser medido.

Para gerar as estatísticas de desempenho, você deverá executar os passos acima para M diferentes conjuntos de N registros aleatórios. Minimamente, utilize M=3. Ao final, compute as médias de cada uma das métricas (comparações, movimentações e tempo). Salve todos os resultados obtidos em um arquivo saida.txt, contendo tanto os resultados individuais quanto a média final.

Você deverá realizar a simulação descrita acima para os seguintes valores de N:

10.000
50.000
100.000
500.000
800.000

Faça a leitura dessas informações a partir de um arquivo chamado *sort.dat*.

Assim, em resumo, para cada valor de N, você deve importar M conjuntos distintos de N registros aleatórios, ordená-los e computar métricas conforme especificado. Para a ordenação, você deverá utilizar no mínimo 3 algoritmos:

- Quicksort
- Heapsort
- Um outro algoritmo de sua escolha

Etapa 2 - Artistas mais frequentes e suas músicas mais populares

Você deverá implementar um programa que leia N músicas (tracks) aleatórias e conte quantas vezes um mesmo artista se repete dentro dessas N músicas.

Você deverá utilizar uma tabela *hash* para armazenar os artistas. **Lembre-se que não deve haver artistas repetidos.** A função *hash* deve apresentar duas propriedades básicas: seu cálculo deve ser rápido e deve gerar poucas colisões. Além disso, é desejável que ela leve a uma ocupação uniforme da tabela para conjuntos de chaves quaisquer. Escolha um dos métodos estudados para tratar as colisões.

Seu programa deve, para cada música lida, inserir o artista na tabela *hash* de artistas. No final da execução, seu programa deve imprimir os M (parâmetros definidos pelo usuário) artistas mais frequentes. A saída deve ser ordenada de forma decrescente de frequência. Para ordenação, escolha o algoritmo de ordenação com melhor desempenho na Etapa 1. Além disso, para cada artista da lista dos M mais frequentes, o programa deve determinar sua música mais popular.

Os dados N e M devem ser lidos do arquivo *hash.dat*.

Etapa 3 - Programa principal

O programa deve oferecer ao usuário um menu para permitir a escolha de qual etapa será executada:

1. Ordenação
2. Hash
3. Módulo de Teste

O módulo de teste deve permitir a realização de algumas operações para garantir que as funções básicas do programa entregam resultados corretos sob quantidades menores de registros. Esse módulo deve conter funções para escrever as saídas em um arquivo teste.txt das seguintes operações:

- O resultado de cada um dos algoritmos de ordenação.
- O resultado dos artistas mais frequentes e suas músicas mais populares

Para estes testes, considere a ordenação de N=100 artistas aleatórios e N=1000 músicas para a busca dos artistas mais frequentes (use M=10).

4. Relatório

Você deverá confeccionar um relatório detalhado sobre o trabalho desenvolvido. Este relatório deve conter, obrigatoriamente, os seguintes itens:

- Detalhamento das atividades realizadas por cada membro do grupo;
- Explicações dos algoritmos escolhidos, bem como as razões para cada escolha;
- Explicações das estruturas de dados implementadas, bem como as razões para as decisões de implementação;
- Apresentação dos resultados através de tabelas e gráficos que permitam visualizar com clareza o que foi obtido;
- Análise detalhada dos resultados obtidos;
- Toda e qualquer referência utilizada no desenvolvimento do trabalho.

Note que o relatório deve ser formal, bem organizado e bem redigido. A divisão de tarefas do trabalho se aplica somente à implementação. Todo o grupo é responsável pelo relatório (escrita e revisão).

5. Exigências

O trabalho deverá, obrigatoriamente, atender aos seguintes requisitos:

- Implementação em C ou C++
- O projeto deve ser compilável e executável via linha de comando. Não conte com a presença de IDEs como Code::Blocks, Visual Studio ou NetBeans. Caso seu grupo opte por utilizar algum ambiente de desenvolvimento, certifique-se de que o projeto enviado possa também ser facilmente compilado em um sistema operacional Linux sem esses ambientes instalados. Forneça instruções claras e precisas de compilação e execução pela linha de comando. **Recomenda-se a utilização de algum Makefile ou script para a compilação.** Caso o grupo julgue necessário, é possível solicitar que o professor verifique as instruções de compilação **antes** do prazo final de envio.
- **Trabalho entregue após o prazo será penalizado.**
- **O programa desenvolvido deve permitir que o usuário entre com o caminho do diretório que contém os arquivos de dados como um argumento na linha de comando.** Veja o Exemplo:

```
$ ./programa /diretorio/contendo/arquivos/entrada_e_saida
```

Considere que os arquivos .csv e .dat estejam dentro desse diretório indicado. Os arquivos de saída (.bin e txt) também devem ser gerados dentro pasta.

- O programa deve procurar pela existência dos arquivos binários dentro da pasta. Se não existir, deve pré-processá-los (etapa 0) antes de seguir para as etapas 1,2 e 3. Não é permitida a utilização de bibliotecas externas. O processamento do arquivo CSV e os algoritmos de ordenação devem ser implementados pelo grupo. Um dos objetivos do trabalho é que vocês aprendam a trabalhar com arquivos e com os algoritmos escolhidos. Se você tiver dúvida quanto à utilização de alguma função ou biblioteca, entre em contato com o professor.
- Obviamente, todo código deve ser de autoria do grupo. Não é permitida a utilização de códigos de terceiros ou de outros grupos. É permitida a pesquisa por estratégias para a solução dos problemas (e as referências utilizadas nessas pesquisas devem constar do relatório), porém a apropriação de código alheio não será aceita. **Qualquer tentativa de plágio identificada resultará em nota zero. Os códigos fontes serão analisados pelo sistema Moss (http://theory.stanford.edu/~aiken/moss/)**

6. Entrega

O grupo deverá ser formado por **no máximo** 4 alunos, e as responsabilidades de cada aluno devem ser documentadas e registradas. Não é permitido que algum integrante do grupo fique responsável somente pela confecção do relatório. Todos os integrantes devem contribuir com a implementação. A distribuição das responsabilidades deve ser feita de maneira uniforme, de modo que cada membro do grupo se envolva com o trabalho na mesma proporção que os demais.

Todos os itens abaixo devem ser entregues:

1. Código-fonte completo;
    a. Deve ser submetido um link para um repositório git (github) contendo o código do trabalho;
    b. **Não incluir o dataset na submissão via Google Classroom (tanto o .csv quando .bin).**
2. Relatório em Google Doc ou PDF atendendo ao especificado na Seção 4 deste documento.

7. Critérios de avaliação

O grupo será avaliado de acordo com os seguintes critérios:

- Execução correta do código (E); (0-100)
- Atendimento ao que foi solicitado (A) (valor entre 0 e 1);
- Organização do código (O): seu código deve estar bem modularizado e bem documentado; (0-100)
- Qualidade do relatório apresentado (R). (0-100)

Cada membro do grupo será avaliado individualmente, tanto com relação aos detalhes de implementação que ficaram sob sua responsabilidade, quanto ao entendimento em alto nível de abstração do que foi feito pelo grupo como um todo. O entendimento teórico do conteúdo relacionado ao trabalho também será avaliado. A nota individual (M) será um valor de 0 a 1 que irá ponderar a nota da implementação.

A nota final de cada integrante será computada de acordo com a seguinte fórmula:

$$Nota = 0.6 *[(0.7*E*A + 0.3*O) *M + 0.4 * R *A$$

IMPORTANTE:

Caso algum membro do seu grupo tranque ou abandone a disciplina, comunique o professor o quanto antes para que se possa discutir alternativas.

Referências

[1] Yamac Eren Ay. Spotify DataSet 1921-2021 (2021). Disponível em: https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks (acessado em 25 de maio de 2021).