TRABALHO PRÁTICO
PARTE 2

PRAZO PARA ENTREGA:19/12/2021

1. Introdução

O TikTok é um aplicativo bastante popular que permite aos usuários compartilhar vídeos curtos. O aplicativo para Android pode ser baixado pela Google Play. A base de dados disponível em https://www.kaggle.com/shivamb/35-million-tiktok-mobile-app-reviews [1] reúne informações de aproximadamente 3,5 milhões de *reviews* de usuários do aplicativo na loja da Google. Esses dados podem ser encontrados no link supracitado, no arquivo *tiktop_app_reviews.csv*. Cada registro do arquivo é composto das seguintes informações:

- `review_id`: string identificadora única
- `review_text`: o *review* postado pelo usuário
- `upvotes`: número de votos favoráveis ao *review*
- `app_version`: string indicando a versão do aplicativo a que o review se refere
- `posted_date`: data e hora de postagem

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-se 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 gerado pelo pré-processamento
2. realizar a ordenação desses registros, utilizando como chave de ordenação `upvotes`. 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
1.000.000

Faça a leitura dessas informações a partir de um arquivo chamado *input.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 terceiro algoritmo à sua escolha (não escolher o shell sort ou os outros algoritmos trabalhados na disciplina)

Etapa 2 - Versões do app mais frequentes

Você deverá ler N *reviews* aleatórios e contar quantas vezes uma mesma versão do aplicativo (`app_version`) se repete dentro desses N *reviews*.

Você deverá utilizar uma tabela *hash* para armazenar as versões do aplicativo. **Lembre-se que não deve haver versões repetidas.** 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 *review* lido, inserir a string `app_version` na tabela *hash*. No final da execução, seu programa deve imprimir as M (parâmetro definido pelo usuário) versões do app 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.

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 das versões do app mais frequentes

Para estes testes, considere a ordenação de N=100 reviews aleatórios.

4. Relatório parcial

Você deverá confeccionar um relatório simples de no máximo uma página sobre o trabalho desenvolvido. Este relatório deve conter, obrigatoriamente, os seguintes itens:

- detalhamento das atividades realizadas por cada membro do grupo;
- decisões de implementação;
- toda e qualquer referência utilizada no desenvolvimento do trabalho.

Note que esse relatório, embora simples, deve ser formal 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 ou Visual Studio. 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.
- **Nenhum trabalho será aceito após o prazo.**
- **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
```

- O programa deve procurar pela existência do arquivo binário dentro da pasta (gerado na Parte 1 do trabalho). Se não existir, deve encerrar, emitindo uma mensagem de erro.
- Não é permitida a utilização de bibliotecas externas. Os algoritmos de ordenação devem ser implementados pelo grupo. Caso haja 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 mínimo 3** e **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 nem no github (nem o .csv nem o .bin).**
2. Relatório em Google Docs 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);
- Desempenho do programa (D): evitar o consumo excessivo de memória, e o tempo de execução deve estar dentro de limites aceitáveis para o problema tratado;
- Atendimento ao que foi solicitado (A) (valor entre 0 e 1);
- Organização do código (O): seu código deve estar bem modularizado e bem documentado;
- Qualidade do relatório apresentado (R).

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:

$$\text{Nota} = M*(A*(E + D) + O) + R$$

O critério (A) será considerado de acordo com a composição do grupo. O objetivo é evitar que algum aluno seja prejudicado pela desistência de outros membros do grupo. **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] BANSAL, Shivam. 3.5 Million Tiktok Mobile App Reviews (2021). Disponível em: https://www.kaggle.com/shivamb/35-million-tiktok-mobile-app-reviews
(acessado em 25 de outubro de 2021).