Universidade Federal de Pernambuco
Centro de Informática

IF775 - Tóp. Avc. em Algoritmos: Algoritmos para Streams de Dados 2020.2
Profs. Nivan Ferreira, Paulo Fonseca

Projeto 2 — versão 1 (04/08/2021)

- Este documento contém as regras e diretrizes para o segundo projeto. Leia com atenção todo o conteúdo do documento e tente ater-se às orientações o mais fielmente possível.
- As regras abaixo podem ser modificadas a qualquer tempo pelo professor no melhor interesse acadêmico e didático. As modificações serão comunicadas em tempo hábil através do Google Classroom.
- Eventuais omissões serão tratadas de maneira discricionária pelos professores, levando-se em conta o bom senso, a praxe acadêmica e os interesses didáticos.

1 Objetivo

Neste projeto, os alunos devem implementar algoritmos para data sketches vistos em aula (vide Seção 5), e realizar uma análise experimental com base num dataset fornecido (vide Seção 4). O objetivos são:

1. Consolidar o conhecimento dos algoritmos vistos no curso através de uma implementação com correção e escalabilidade em nível de produção.
2. Exercitar uma análise experimental crítica dos algoritmos implementados, com a documentação adequada dos resultados.

2 Equipes

O projeto deve ser realizado em equipes de 2 integrantes (duplas). Cada integrante deve participar e conhecer em detalhes todas as atividades envolvidas (implementação, documentação e testes).

3 Data de entrega

O trabalho deve ser entregue até **Domingo 22 de Agosto de 2021 às 23h59** (mais detalhes na Seção 8).

4 Dados

Para as análises deste projeto deverão ser usados dois arquivos gerados a partir de um dataset de tráfego de rede[1].

`network_flows.csv` Este CSV contém informação sobre diversos fluxos de rede, cada um descrito pelos seguintes atributos (colunas):

[1] `https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps`

0. **`Flow.ID`** Cada fluxo é descrito por um identificador correspondente a um par de IPs origem e destino. Todas as linhas correspondentes a um tráfego entre o mesmo par origem-destino (em qualquer ordem) são identificadas pelo mesmo ID.
1. **`Source.IP`** Endereço IP da origem.
2. **`Destination.IP`** Endereço IP do destino.
3. **`Protocol`** Inteiro idenfificador do protocolo de transporte (e.g. TCP=6, UDP=17).
4. **`Total.Fwd.Packets`** Quantidade de pacotes no sentido direto (origem → destino).
5. **`Total.Backward.Packets`** Quantidade de pacotes no sentido reverso (destino → origem).
6. **`Total.Length.of.Fwd.Packets`** Tamanho total em bytes dos pacotes no sentido direto (origem → destino).
7. **`Total.Length.of.Bwd.Packets`** Tamanho total em bytes dos pacotes no sentido reverso (destino → origem).
8. **`ProtocolName`** String identificadora da aplicação.

`network_flows_unique.csv` Subconjunto do arquivo anterior contendo apenas um fluxo por ID (par origem-destino).

Os arquivos de dados podem ser obtidos através do Google Classroom.

5 Algoritmos

Devem ser implementados 02 (dois) dos Sketches para dados ordenados da lista a seguir:

1. Q-Digest
2. GK
3. KLL

O projeto tem como objetivo avaliar o desempenho dos sketches implementados individualmente e em comparação um com o outro.

Cada sketch deverá suportar três operações básicas:

update(x) Adiciona o valor x ao sumário

rank(x) Calcula uma estimativa de $rank(x)$ definido como a quantidade de valores estritamente menores do que x representados na coleção.

quantile(q) Retorna um valor x representado no sumário correspondente ao quantil q, com $q \in [0, 1]$, ou seja, queremos retornar x tal que $rank(x) \approx qN$, onde N é o número de elementos armazenados no sketch.

A seguir, especificamos os detalhes específicos de cada algoritmo

5.1 Q-Digest

Deve ser implementado um programa **`qdig`** que cria um Q-Digest a partir dos valores de um campo v do arquivo de dados no formato CSV. O sketch é suposto fornecer ϵW-aproximações dos ranks, sendo W o peso total da stream, conforme explicado no livro-texto.

A sintaxe da chamada do programa deve ser na forma:

```
$ qdig build_options data_file.csv
       [rank | quant] [query_args | --in query_args_file]
```

Os parâmetros de construção (`build_options`) são especificados da seguinte forma:

`--val id_field_no`: Especifica número da coluna do arquivo de entrada a ser usada para alimentar o sketch. **Esta coluna deve conter valores numéricos**. Cada valor deve ser interpretado como tendo peso 1.

`--eps error_bound`: Especifica o valor do parâmetro ϵ do sketch.

`--univ universe_size`: Especifica o valor do parâmetro U do sketch. Ou seja, consideramos os valores contidos no intervalo $[U] = \{0, \ldots, U-1\}$. **Os valores fora desse intervalo devem ser desconsiderados.**

O programa deverá construir o sketch com as opções especificadas a partir do arquivo de entrada e, em seguida, responder as consultas de rank (`rank`) ou quantil (`quant`) para os valores dos argumentos informados (`query_args`). Caso a operação escolhida seja `rank`, esses argumentos devem ser uma lista de inteiros em $[U]$ separados por espaços; caso a operação seja `quant`, os argumentos devem ser números decimais entre 0 e 1 separados por espaços. Alternativamente, os argumentos de entrada para ambas as operações de consulta podem ser fornecidos num arquivo, **sendo um número por linha**, através da opção `--in` seguida do nome do arquivo. O programa deve imprimir várias linhas correspondentes às respostas da consulta para os argumentos de entrada fornecidos.

Exemplo

```
$ qdig --val 4 --eps 0.1 --univ 1000 network_flows.csv rank --in input_vals.dat
```

Cria um *Q-Digest* para os valores da coluna no. 4 (`Total.Fwd.Packets`) entre 0 e 1000 e, em seguida, imprime a estimativa dos ranks dos valores contidos no arquivo `input_vals.dat`.

5.2 GK

Deve ser implementado um programa **`gk`** que cria um sketch GK a partir dos valores de um campo v do arquivo de dados no formato CSV, de forma similar ao Q-Digest explicado acima. O sketch é suposto fornecer ϵW-aproximações dos ranks, sendo W o peso total da stream (quantidade de elementos), conforme explicado no livro-texto.

A sintaxe da chamada do programa deve ser na forma:

```
$ gk build_options data_file.csv
       [rank | quant] [query_args | --in query_args_file]
```

A utilização é quase idêntica a do Q-Digest, com exceção da opção `--univ`, que aqui pode ser omitida, fazendo que todos os valores sejam considerados. Caso ela esteja presente, seu comportamento é igual ao caso do Q-Digest, ou seja, entradas fora dos valores do universo devem ser desconsideradas.

5.3 KLL

Deve ser implementado um programa **`kll`** que cria um sketch KLL a partir dos valores de um campo v do arquivo de dados no formato CSV. O sketch é suposto fornecer (ϵ, δ)-aproximações dos ranks, conforme discutido no livro-texto e em aula.

A sintaxe da chamada do programa deve ser na forma:

```
$ kll build_options data_file.csv
      [rank | quant] [query_args | --in query_args_file]
```

Os parâmetros de construção (`build_options`) são especificados da seguinte forma:

`--val` ***`id_field_no`***`:` Especifica número da coluna do arquivo de entrada a ser usada para a norma. **Esta coluna deve conter valores numéricos**. Cada valor deve ser interpretado como tendo peso 1.

`--eps` ***`error_bound`***`:` Especifica o valor do parâmetro ϵ do sketch.

`--delta` ***`error_prob`***`:` Especifica o valor do parâmetro δ do sketch.

`--mult` ***`multiplier`***`:` Especifica o valor do da constante multiplicadora $0.5 < C < 1$ usada no sketch.

`--univ` ***`universe_size`***`:` Especifica o valor do parâmetro U como nos sketches acima. Este parâmetro também é opcional para o KLL. Caso omitido, todos os valores serão considerados.

O programa deverá construir o sketch com as opções especificadas a partir do arquivo de entrada e, em seguida, responder as consultas de maneira similar aos demais sketches acima.

Exemplo

```
$ kll --val 4 --eps 0.1 --delta 0.05 --mult 0.7 network_flows.csv quant 0.5
```

Cria um KLL para os valores da coluna no. 4 (`Total.Fwd.Packets`) com os parâmetros $\epsilon = 10\%$, $\delta = 5\%$ e $c = 0.7$.

6 Implementação

Conforme descrito acima, cada sketch deve ser implementado num programa separado. Os programas devem ser implementados preferencialmente em C/C++. O objetivo é torná-lo o mais eficiente possível. Os programas serão avaliados na plataforma GNU/Linux. Deve-se tentar minimizar as dependências externas para torná-la facilmente portável entre plataformas.

Podem ser utilizadas APIs externas apenas para o *frontend* dos programas. Os *backends* dos programas devem consistir dos algoritmos vistos em aula, (re-)implementados diretamente pelos alunos. *A detecção de cópia não-relatada de partes substanciais do código desses algoritmos implicará na atribuição da nota 0.0 (zero) ao trabalho como um todo, independente de outras partes.*

IMPORTANTE: As entradas dos arquivos devem ser processadas em modo stream/online, ou seja, uma linha/registro por vez.

7 Testes/Experimentos

Devem ser realizados experimentos para aferir:

1. A correção/qualidade das estimativas fornecidas pelos sketches em função dos parâmetros usados para a construção.
2. O desempenho prático das ferramentas em termos de tempo/espaço.

Os resultados dos experimentos devem organizados em tabelas e gráficos. Para além dos simples dados brutos, deve-se tentar caracterizar um padrão de desempenho dos algoritmos em função dos parâmetros e características das entradas que nos permitam, eventualmente, prever o comportamento em cenários não testados diretamente. Outras ferramentas disponíveis através da literatura e de software de terceiros podem/devem ser utilizados como benchmark para comparação. Também devem ser analisadas as vantagens e desvantagens relativas entre os sketches implementados em condições experimentais comparáveis.

8 Deliverables

Deve ser entregue um arquivo comprimido em formato `.tgz` ou `.zip`. Para facilitar a identificação nomeie o arquivo no formato

login-versão.tgz

onde ***login*** corresponde ao primeiro username em ordem lexicográfica da equipe e ***versão*** corresponde a um número sequencial (1,2,3,...) indicativo da versão submetida[2]. Esse arquivo comprimido deve consistir de um diretório com o seguinte conteúdo *mínimo*.

```
projeto2/
 |
 +-- doc/
 +-- src/
 +-- README.txt
```

O arquivo `README.txt` deve conter uma identificação da ferramenta, dos autores, e as instruções para compilação (vide seção abaixo). O conteúdo de cada diretório será especificado a seguir.

8.1 Código-fonte

Deve ser entegue o código fonte da ferramenta juntamente com um Makefile ou script para compilação no subdiretório `src/`. As instruções para o processo de compilação da ferramenta devem ser dadas no arquivo `README.txt`. Idealmente a compilação deveria consistir apenas na execução de um simples comando como `make` ou similar.

O código deve ser o mais *limpo*[3] possível. Entretanto, os objetivos principais são 1) correção e 2) eficiência. Portanto, deve-se evitar o uso exagerado de modelagem por objetos, padrões de proje-

[2] É comum que sejam submetidas mais de uma versão, devido a correções de última hora. Nesse caso, apenas a última versão é considerada para avaliação

[3] *RC Martin*. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.

tos, etc. que tornem o programa mais lento. Um programa bem estruturado, com os nomes expressivos para os atributos e métodos dos sketches (init, update, query), e com uma separação clara entre interface e backend deve ser suficiente.

Após a compilação, os arquivo executáveis devem estar num diretório `bin`, criado dentro do diretório original, isto é, teremos

```
projeto2/
 |
 +-- bin/   <=== executáveis aqui
 +-- doc/
(...)
```

8.2 Documentação

Uma ajuda com as instruções para a utilização básica de cada ferramenta deve ser obtida através da execução do programa com a opção

`-h, --help`

Além disso, deverá ser entregue um breve relatório dividido em três principais seções:

1. Identificação
   - Identificação da equipe
   - Breve descrição da contribuição de cada membro da equipe ao trabalho
2. Implementação
   - Identificação dos algoritmos implementados
   - Detalhes de implementação relevantes, com impacto significativo (positivo/negativo) para o desempenho da ferramenta, incluindo:
     - Estruturas de dados
     - Funções de hashing
     - etc.
   - Bugs conhecidos e limitações de desempenho notáveis. Se o trabalho não foi integralmente concluído, o que faltou deve ser explicitamente reportado aqui.
   - Identificação explícita de partes eventualmente copiadas de terceiros, caso estritamente necessário, com a devida justificativa.
3. Testes e Resultados
   - Descrição do ambiente de testes
   - Descrição dos experimentos realizados
   - Dados e resultados obtidos (tabelas, gráficos, ...)
   - Discussão dos resultados e conclusão

   IMPORTANTE: No relatório, deve-se buscar expor dados compilados que favoreçam a visualização e interpretação.

Esse relatório deve estar contido no subdiretório `doc/`, num arquivo `.pdf` (*Não* use MSWord ou qualquer formato proprietário).

8.3 Data sets

Os dados utilizados nos testes **NÃO** devem ser submetidos junto com o trabalho em nenhuma hipótese. A inclusão de arquivos de dados será penalizada. Caso seja considerado necessário, deve-se torná-los disponíveis online e indicar o endereço na seção da descrição dos testes do relatório.

9 Avaliação

A avaliação será feita com base nos seguintes critérios:

1. Implementação (peso 6). Inclui a correção, eficiência e qualidade do código-fonte levando-se em conta a quantidade e dificuldade intrínseca dos algoritmos implementados. Será também levada em conta organização e distribuição do código.
2. Testes (peso 4). Inclui a reprodutibilidade dos experimentos, a abrangência dos dados, a organização e apresentação dos resultados, a correção e profundidade das análises e a exposição das conclusões.

O código enviado será compilado e executado localmente, conforme descrito na documentação, e respeitando as instruções deste documento, num ambiente Linux. Certifique-se que sua implementação compila e funciona corretamente nessa plataforma, mesmo que desenvolvida em outra configuração.

9.1 Arguição

A avaliação será feita mediante análise do material submetido e de uma arguição a ser agendada, posteriormente, com cada equipe. Cada integrante deve ter participado de todas as atividades e, portanto, deve conhecer integralmente ser capaz de responder questões sobre qualquer aspecto do projeto.

10 Extras

A equipe pode implementar mais do que os dois sketches obrigatórios, assim como está livre para implementar recursos extras para tornar os programas mais eficientes e/ou flexíveis, o que inclui alguma otimização descrita na literatura. Porém, todos os recursos devem ser compreendidos e devem ser justificados e analisados no relatório. O trabalho-extra poderá receber alguma bonificação.

■ ■ ■