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 1 — versão 1 (02/07/2021)

- Este documento contém as regras e diretrizes para o primeiro 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 por e-mail até **Domingo 25 e Julho 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 em ⟨**PENDING:** endereço de download **??**⟩

5 Algoritmos

Devem ser implementados 02 (dois) dos Sketches a seguir:

1. Weighted Sampling
2. *K* Minimum Values (KMV)
3. HyperLogLog
4. CountMin

A seguir, especificamos a funcionalidade básica a ser implementada para cada um dos sketches.

5.1 Weighted Sampling

Deve ser implementado um programa **`ws`** que cria uma amostra ponderada de tamanho s dos fluxos do arquivo `network_flows_unique.csv`[2], usando um campo w indicado como peso. Uma vez criada a amostra, ela pode ser utilizada para estimação pontual do peso de um subconjunto dos fluxos que respeitam um critério dado. Por simplicidade, esse critério será sempre um valor específico de um dos campos.

A sintaxe da chamada do programa deve ser na forma:

`$ ws [`*`options`*`] `*`input_file.csv`

`--id `*`id_field_no`*`:` Especifica o número (≥ 0) da coluna do arquivo de entrada a ser usado como identificador. Esta coluna não pode conter valores repetidos.

`--weight `*`weight_field_no`*`:` Especifica o número da coluna do arquivo de entrada a ser usado como peso. Esta coluna deve conter inteiros positivos.

[2] O *weighted sampling* processa items com identificadores únicos, *sem repetição*, com pesos positivos.

`--size sample_size`: Especifica o parâmetro s (tamanho da amostra) do Sketch.

`--filter field_no field_value`: Especifica que deve ser estimado o peso do subconjunto dos registros cujo valor do campo de número `field_no` seja igual a `field_value`

O programa deverá imprimir uma linha com o valor da estimativa.

Exemplo

```
$ws --id 0 --weight 6 --filter 8 GOOGLE --size 1024 network_flows_unique.csv
```

Cria um *weighted sample sketch* de tamanho $s = 1024$ para os fluxos descritos no arquivo de entrada `network_flows_unique.csv`, usando o campo no. 6 (`Total.Length.of.Fwd.Packets`) como peso e, em seguida, estima o peso do subconjunto dos registros cujos campo no. 8 (`ProtocolName`) são iguais a `GOOGLE`.

5.2 K Minimum Values (KMV)

Deve ser implementado um programa **`kmv`** que cria KMV sketches para estimar a quantidade de valores distintos de um determinado campo-alvo t, observados na amostra do arquivo de entrada `network_flows.csv`. O programa deve fornecer uma (ϵ, δ)-estimativa de F_0, ou seja, um valor $\hat{F}_0$ tal que

$$\mathbb{P}[|\hat{F}_0 - F_0| \geq \epsilon F_0] \leq \delta,$$

onde F_0 representa a quantidade de valores distintos para o campo-alvo t na entrada. Para tal o programa deverá ajustar automaticamente os parâmetros livres do sketch e determinar a quantidade de instâncias independentes a serem construídas para utilização do truque da mediana, se necessário. O programa também deverá definir uma forma conveniente de mapear os valores dos diferentes campos num domínio numérico $[m]$.

A sintaxe da chamada ao programa deve ser na forma:

```
$ kmv [options] input_file.csv
```

com as seguintes opções:

`--target field_no`: Especifica o número da coluna alvo ($t \geq 0$) do arquivo de entrada cuja quantidade de valores únicos devem estimada.

`--eps error_bound`: Especifica o valor do parâmetro ϵ do estimador (limite do erro relativo desejado).

`--delta error_probability`: Especifica o valor do parâmetro δ do estimador (limite para a probabilidade que o erro relativo da estimação seja superior ao limite especificado).

Exemplo

```
$ kmv --target 7 --eps 0.05 --delta 0.01 network_flows.csv
```

Calcula uma estimativa com erro relativo inferior a 5% com 99% de probabilidade para a quantidade de valores distintos para o campo número 7 (`Total.Length.of.Bwd.Packets`) usando KMV sketches.

O programa deverá imprimir uma linha com o valor da estimativa.

5.3 HyperLogLog

Deve ser implementado um programa **`hll`** que cria HyperLogLog sketches para estimar a quantidade de valores distintos de um determinado campo-alvo t, observados na amostra do arquivo de entrada `network_flows.csv`. O programa deve fornecer uma (ϵ, δ)-estimativa de F_0, ou seja, um valor $\hat{F}_0$ tal que

$$\mathbb{P}[|\hat{F}_0 - F_0| \geq \epsilon F_0] \leq \delta,$$

onde F_0 representa a quantidade de valores distintos para o campo-alvo t na entrada. Para tal o programa deverá ajustar automaticamente os parâmetros livres do sketch e determinar a quantidade de instâncias independentes a serem construídas para utilização do truque da mediana, se necessário. O programa também deverá definir uma forma conveniente de mapear os valores dos diferentes campos num domínio numérico $[m]$.

A sintaxe da chamada ao programa deve ser na forma:

`$ hll [`*`options`*`] `*`input_file.csv`

onde as opções são idênticas às do **`kmv`**, especificadas acima.

O programa também deverá imprimir uma linha com o valor da estimativa.

5.4 CountMin

Deve ser implementado um programa **`cmin`** que cria um CountMin Sketch a partir do arquivo `network_flows.csv` para estimar pontualmente os "pesos totais" de fluxos, usando um campo w indicado como peso. O sketch é suposto fornecer (ϵ, δ)-aproximações $\hat{w}_x$ dos pesos totais w_x de cada fluxo x, isto é,

$$\mathbb{P}[\hat{w}_x - w_x \geq \epsilon W] \leq \delta,$$

onde W representa a soma de todos os pesos dos fluxos da entrada.

A sintaxe da chamada do programa deve ser na forma:

`$ cmin [`*`options`*`] `*`input_file.csv`

com as seguintes opções.

`--id `*`id_field_no`*`:` Especifica número da coluna do arquivo de entrada a ser usado como identificador. Esta coluna pode conter valores repetidos.

`--weight `*`weight_field_no`*`:` Especifica o número da coluna do arquivo de entrada a ser usado como peso. Esta coluna deve conter inteiros positivos.

`--eps `*`error_bound`*`:` Especifica o valor do parâmetro ϵ do estimador (limite do erro relativo desejado).

`--delta `*`error_probability`*`:` Especifica o valor do parâmetro δ do estimador (limite para a probabilidade que o erro relativo da estimação seja superior ao limite especificado).

`--query` *`id_1 id_2 ...`*`:` Especifica os identificadores dos fluxos cujos pesos devem ser estimados.

`--qryfile` *`query_file_name`*`:` Alternativamente, os identificadores dos fluxos consultados podem ser fornecidos num arquivo com um identificador por linha através desta opção.

O programa deverá imprimir várias linhas, cada uma com o valor da estimativa do peso total de cada fluxo indicado na opção `--query`, ou no arquivo indicado pela opção `--qryfile`.

Exemplo

```
$ cmin --id 0 --weight 4 --eps 0.1 --delta 0.05 --qryfile queries.txt
network_flows.csv
```

Cria um *CountMin sketch* para os fluxos, usando a coluna no. 4 como peso (`Total.Fwd.Packets`) e, em seguida, imprime estimativas dos pesos totais dos fluxos cujos ids estão indicados no arquivo `queries.txt`, com erro de até 10% em relação à massa total da stream com pelo menos 95% de probabilidade.

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 o desempenho prático das ferramentas em termos de tempo/espaço, bem como a correção e qualidade das aproximações fornecidas.

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.

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[3]. Esse arquivo comprimido deve consistir de um diretório com o seguinte conteúdo *mínimo*.

```
projeto1/
 |
 +-- 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*[4] possível. Entretanto, os objetivos principais são 1) correção e 2) eficiência. Portanto, deve-se evitar o uso exagerado de modelagem por objetos, padrões de projetos, 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

```
projeto1/
 |
 +-- 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

[3] 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

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

`-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.

■ ■ ■