SFI5822 — Introdução à Programação Paralela
Trabalho prático 1 — Versão sequencial

2019-09-05

1 Conceitos

Um grafo G é um par $G = (V, E)$ onde $V = \{v_0, v_1, \ldots, v_{n-1}\}$ é um conjunto de n *vértices* e E é um conjunto de m arestas (v_i, v_j) onde v_i e v_j são vértices e a aresta (v_i, v_j) indica que existe uma ligação do vértice v_i para o vértice v_j. Estaremos lidando com *grafos sem direção*, para os quais a presença da aresta (v_i, v_j) implica a presença da aresta (v_j, v_i), e *simples*, isto é, arestas da forma (v_i, v_i) não existem, e existe no máximo uma aresta do tipo (v_i, v_j) para $i \neq j$. Se $(v_i, v_j) \in E$ dizemos que v_i e v_j são *conectados*.

O conjunto de *vizinhos* de um vértice v_i, denominado $\mathcal{N}_i$ é composto pelos vértices v_j para os quais $(v_i, v_j) \in E$,

$$\mathcal{N}_i = \{v_j \mid (v_i, v_j) \in E\}.$$

O *grau* do vértice v_i, indicado por k_i, é o seu número de vizinhos, $k_i = |\mathcal{N}_i|$.

Dizemos que um grafo é *denso* se $m \propto n^2$ e *esparso* se $m \ll n^2$.

2 Trabalho

Você deve implementar, em C++, um código para calcular o número de vizinhos em comum para todos os pares de vértices de um dado grafo. Isto é, determinar, para todos os $i, j = 0 \ldots n-1$

$$n_{ij} = |\mathcal{N}_i \cap \mathcal{N}_j|.$$

Ao escrever seu código, considere que o grafo é esparso e que podem haver diferenças significativas nos graus dos vértices.

Tome cuidado em usar estruturas de dados e algoritmos que permitam o desenvolvimento de um código eficiente. Coloque temporização no seu código, para avaliação do tempo de execução *excluindo a leitura do arquivo de entrada e a escrita do arquivo de saída*. Ao final da execução, escreva o tempo de execução no terminal.

Não é permitido usar bibliotecas adicionais a não ser as presentes no padrão da linguagem C++. O código precisa compilar com o compilador GNU versão 8.

2.1 Entrada

A entrada para o seu programa será um arquivo que descreve o grafo no formato conhecido como *edgelist*. O nome do arquivo de entrada deve ser lido pelo seu programa da linha

de comando. Neste formato, são dados dois a dois os índices de vértices conectados por arestas.

Por exemplo, se o arquivo de entrada contém:

```
0 2 1
4
3 1 2 5 7 8 2 8
6 1 4 7 1 7
7 3
```

Isto indica um grafo com

$$V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

e

$$\begin{aligned} E = \ \{ \ & (v_0, v_2), (v_2, v_0), (v_1, v_4), (v_4, v_1), (v_3, v_1), (v_1, v_3), (v_2, v_5), (v_5, v_2), \\ & (v_7, v_8), (v_8, v_7), (v_2, v_8), (v_8, v_2), (v_6, v_1), (v_1, v_6), (v_4, v_7), (v_7, v_4), \\ & (v_1, v_7), (v_7, v_1), (v_7, v_3), (v_3, v_7) \ \}. \end{aligned}$$

Note como devem ser consideradas ambas as direções das arestas, e como a separação de linhas não é relevante.

Você pode assumir o seguinte sobre o arquivo:

- Os vértices serão numerados de 0 a $n - 1$ no arquivo.
- Cada vértice aparecerá pelo menos uma vez no arquivo. Isto é, o número de vértices do grafo é um a mais do que o maior valor presente no arquivo.
- Se o par i, j aparece no arquivo, então o par j, i não aparecerá.

2.2 Saída

A saída produzida deve ser um arquivo texto (ASCII) no seguinte formato:

- Cada linha corresponde às informações sobre um par distinto de vértices.
- A linha possui três valores inteiros, i, j e k e indica que entre os vértices v_i e v_j existem k vizinhos em comum.
- Apenas i, j, k ou j, i, k devem aparecer no arquivo de saída, nunca os dois.
- Se o número de vizinhos em comum entre dois vértices é zero, o par correspondente **não deve aparecer no arquivo de saída**.

Para a entrada do exemplo acima, a saída deve ser equivalente ao seguinte:

```
0 5 1
0 8 1
1 3 1
1 4 1
1 7 2
```

```
1 8 1
3 4 2
3 6 1
3 7 1
3 8 1
4 6 1
4 7 1
4 8 1
5 8 1
6 7 1
```

O nome do arquivo de saída produzido deve ser o nome do arquivo de entrada, mas com a extensão `.cng` (a extensão anterior, se existente, deve ser removida). Por exemplo, a saída produzida para o arquivo `net102.edgelist` deve ficar no arquivo `net102.cng`.

2.3 Entrega

Você deve entregar o código fonte do seu programa e um texto descrevendo as decisões de implementação utilizadas e sua justificativa. Serão considerados na avaliação do código os seguintes fatores (com diferentes ênfases):

- Se o código compila corretamente, gerando um executável.
- Se o código compila sem emissão de *warnings* no nível mais alto `-Wall -Wextra -Wpedantic`.
- Se o código resolve corretamente o problema pedido. Isto será verificado com um conjunto de redes de teste que serão fornecidas.
- Se o código está corretamente e consistentemente formatado (recomenda-se usar ferramentas como `clang-format` para garantir isso).
- Se o código está bem organizado, com definição de tipos e funções apropriadas, nomes adequados de identificadores e comentários onde necessário, e se não apresenta "resíduos" de versões anteriores que não estão em uso.
- O desempenho do código, que será testado com um conjunto de arquivos de entrada a ser fornecido.