Introdução à Programação Paralela — Primeiro Trabalho

2020

1 Grafos

Na área de redes complexas representamos um sistema por elementos e suas relações. Por exemplo, pessoas e suas relações de amizade, ou proteínas e suas relações de interação, ou aeroportos e a existência de um vôo direto entre dois aeroportos.

A representação é conhecida matematicamente como um *grafo*. Um grafo, representado por $G(V, E)$ consiste em um conjunto de *vértices* ou *nós* V e um conjunto de *arestas* ou *ligações* E. Para o nosso propósito, o conjunto V é simplesmente um conjunto de identificadores de nós, com o identificador sendo um número inteiro entre 0 e $N-1$, onde $N = |V|$ é o número de vértices,[1] isto é, $V = \{0, 1, 2, \ldots, N-1\}$. O conjunto E é composto de pares (i, j) onde $i, j \in V$. Aqui assumimos que as ligações não têm direção, isto é, a presença de (i, j) em E indica tanto que i está ligado a j quanto que j está ligado a i. Simplificações adicionais que assumiremos aqui são que não existem ligações entre um vértice e ele mesmo; que uma ligação $(i, j) \in E$ é única, isto é, não existem múltiplas ligações entre o mesmo par de vértices; e que as ligações são binárias: ou elas existem ou não, sem termos ligações mais fortes ou mais fracas.

Grafos densos e esparsos Dizemos que um grafo é *denso* quando uma fração significativa de todas as ligações possíveis está presente; caso contrário, o grafo é *esparso*. Mais precisamente, o grafo é denso se $|E| \propto N^2$ e esparso se $|E| \ll N^2$. Em redes complexas, frequentemente temos $|E| \propto N$.

2 Distâncias

Um *caminho* entre os vértices i e j é uma sequência de vértices $(v_0, v_1, v_2, \ldots, v_{k-1}, v_k)$ tal que $v_0 = i$, $v_k = j$ e $(v_m, v_{m+1}) \in E$ para qualquer $0 \leq m < k$. O número de arestas (v_m, v_{m+1}) percorridas é chamado o *comprimento* do caminho.

Um caminho entre os vértices i e j é denominado *caminho mínimo* se não existe um outro caminho entre esses vértices que tenha um comprimento menor. O comprimento do caminho mínimo entre dois vértices i e j é denominado a *distância* entre esses vértices, representado por d_{ij}. No caso geral, não existem necessariamente caminhos entre todos os pares de vértices do grafo. Quando não existem caminhos entre os vértices i e j em um grafo, convencionamos que $d_{ij} = \infty$.

Definimos a *eficiência* de um vértice como a média do inverso de sua distância a todos os outros vértices, isto é:

$$e_i = \frac{1}{N-1} \sum_{\substack{j=0 \\ j \neq i}}^{N-1} \frac{1}{d_{ij}}, \tag{1}$$

onde e_i indica a eficiência do vértice i. Note que, como $d_{ij} \geq 1$ para $i \neq j$, $1/d_{ij} \leq 1$ em todos os fatores da somatória, e portanto a eficiência é no máximo 1, o que ocorre quando o vértice i está diretamente conectado com todos os outros vértices. Note também que, se não existe caminho entre os vértices i e j então $1/d_{ij} = 0$ e o vértice j pode ser excluido da somatória da eficiência para o vértice i, isto é,

[1]Se C é um conjunto, então $|C|$ é a sua *cardinalidade*, isto é, o seu número de elementos (para conjuntos finitos).

precisamos somar apenas o inverso das distâncias dos vértices que são alcançáveis a partir do vértice i. Quando um vértice i é isolado, $d_{ij} = \infty$ para todos $j \neq i$, e portanto $e_i = 0$, que é o menor valor possível, pois $1/d_{ij} \geq 0$. Portanto, $0 \leq e_i \leq 1$.

3 Representação de grafos

Existem diversas formas de representar um grafo para fazermos cálculos sobre ele. Aqui vamos discutir as três mais comuns. Para isso, vamos usar um exemplo com 5 vértices, sendo que os vértices 1, 2, 3 e 4 estão ligados ao vértice 0 e os vértice 2 e 3 são ligados entre si, com nenhuma outra ligação presente.

3.1 Matriz de adjacência

Podemos representar um grafo através da sua denominada *matriz de adjacência*, $\mathbf{A}$, que é uma matriz tal que o seu elemento na linha i, coluna j, a_{ij} vale 1 se os vértices i e j são ligados, e 0 caso contrário (supondo que as linhas e colunas são numeradas de zero, como os vértices).

Para o nosso grafo de exemplo, a matriz de adjacência será:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3.2 Lista de adjacência

Na representação por lista de adjacência, o grafo é representado através das listas de vizinhos para cada um dos vértices, isto é, para cada vértice i guardamos os identificadores de seus vizinhos.

Para o nosso grafo de exemplo, teríamos as listas:

$$[\{1,2,3,4\}, \{0\}, \{0,3\}, \{0,2\}, \{0\}]$$

3.3 Lista de arestas

Na representação por lista de arestas, simplesmente guardamos uma lista com todas as arestas do grafo.

Para o nosso exemplo, teríamos

$$\{(0,1), (0,2), (0,3), (0,4), (2,3)\}$$

4 Descrição do trabalho

Você deve escrever um programa que, dado um grafo de entrada, escreva um arquivo com o valor das eficiência de todos os seus vértices. Escolha um algoritmo de cálculo de distâncias e a representação do grafo e realize a implementação procurando atingir bom desempenho no código, considerando que as redes utilizadas serão esparsas, com $|E| \propto N$, mas N pode ser grande.

Leituras e escritas em arquivo são demoradas, e o tempo é bastante dependente do desempenho do sistema de arquivos no computador. Por essa razão, ao fazer avaliação de desempenho neste trabalho você deve temporizar apenas a parte do cálculo, deixando de fora a leitura do arquivo de entrada e a escrita dos resultados.

4.1 Entradas

O grafo a ser processado por seu programa será fornecido em um arquivo, com o formato descrito abaixo. O nome do arquivo deve ser lido como primeiro parâmetro da linha de comando (`argv[1]`).

Esta recomendação é importante! Não se deve ler o nome do arquivo com um `cin` ou similar, mas sim da linha de comando. Também, uma vez fornecido o comando com o nome do arquivo a carregar, o programa não deve esperar mais nada nem imprimir qualquer mensagem, mas apenas gerar a saída especificada, a não ser que haja erro na leitura do arquivo.

O arquivo de entrada apresenta a lista de arestas do grafo. Cada linha do arquivo representa uma aresta e consiste em dois valores inteiros, que são os identificadores dos vértices que essa aresta conecta. Os identificadores são de 0 a $N-1$, onde N é o número de vértices do grafo.

Para o nosso exemplo, o arquivo de entrada seria, por exemmplo:

```
0 1
0 2
0 3
0 4
2 3
```

Note que não especificamos uma ordem para as arestas. Por exemplo, o seguinte arquivo descreve o mesmo grafo do exemplo:

```
3 0
3 2
0 1
0 4
2 0
```

No entanto, garantimos que cada aresta irá aparecer apenas uma vez no arquivo.

4.2 Saídas

O programa deve gerar um arquivo de saída com os valores dos e_i calculados para todos os vértices do grafo. Os diversos valores devem ser separados por pelo menos um espaço em branco ou mudança de linha, sendo que o número de espaços em branco ou a escolha de espaço em branco ou mudança de linha fica por sua conta.

Os valores devem ser escritos com pelo menos 6 casas decimais de precisão (esse é o *default* em C e C++).

O nome do arquivo da saída deve ser o do arquivo de entrada, mudando a extensão para `.eff`. Por exemplo, se o arquivo de entrada que define a rede se chamar `test_name.edgelist` então o arquivo de saída deve se chamar `test_name.eff`.

4.3 O que entregar

Você deve entregar o código fonte (C++ ou C) do seu programa. Esse código fonte deve:

1. Incluir comentários descrevendo as decisões tomadas que têm impacto no desempenho (por exemplo, representação do grafo). Se você fez experimentos para escolher a melhor forma, inclua alguns resultados.

2. Ser um código "limpo", isto é, sem resíduos de versões anteriores ou pedaços de código usados apenas para depuração.

3. Ser adequadamente formatado. Diversos ambientes de programação fazem a formatação para você (no VSCode com ALT-SHIFT-F). Caso não use um ambiente que faça a formatação, você pode usar `clang-format` ou então `indent` (procure na Internet).