MAC0438 – Programação concorrente – 1s2012
EP2
Data de entrega: 10/05/2012

Prof. Daniel Macêdo Batista

1 Problema

A busca do menor caminho entre dois pontos de uma rede é equivalente à busca do menor caminho em um grafo, e esse problema é resolvido utilizando o conhecido algoritmo do caminho mais curto de Dijkstra [1].

Apesar do algoritmo de Dijkstra devolver a melhor solução para o problema do menor caminho, nem sempre este menor caminho é utilizado para todos os pacotes. Isso acontece porque se todos os pacotes sempre seguirem a sugestão do algoritmo de Dijkstra, aquele caminho acaba se tornando um gargalo na rede por conta do congestionamento. Por causa disso, é comum que certos protocolos de roteamento não utilizem apenas o menor caminho entre dois pontos, mas sim os n menores caminhos e haja um sorteio a respeito de qual desses n serão usados por vez, diminuindo assim a probabilidade de que um único caminho torne-se um gargalo na rede.

A sua tarefa neste EP é implementar um programa concorrente que encontre os n menores caminhos entre o vértice 0 e todos os outros vértices de um grafo. Ou seja, a saída final do seu programa deverá apresentar no máximo $n \times (m - 1)$ caminhos, onde m é a quantidade de vértices do grafo.

2 Requisitos

2.1 Threads e barreira de sincronização

O seu EP deverá criar uma quantidade de threads igual à quantidade de núcleos do computador e essas threads deverão se sincronizar sempre que todas elas terminarem de encontrar um caminho (Cada thread terá o papel de encontrar exatamente 1 caminho). Se o computador onde o código for executado tiver apenas 1 unidade de processamento, a quantidade de threads deve ser 2. Você deve implementar algum algoritmo para barreira de sincronização simétrica e essa implementação deve estar explícita no seu código. EPs que implementem outro tipo de barreira ou EPs que utilizem alguma biblioteca que já traga a barreira implementada terão nota ZERO. Você deverá informar no LEIAME do seu código, e no relatório, em qual arquivo e em quais linhas a barreira está implementada.

Quando todas as threads chegarem na barreira de sincronização, a lista dos n menores caminhos para cada destino deve ser atualizada e esse procedimento deve ser repetido até que os n menores caminhos do vértice 0 para cada destino sejam encontrados.

As threads só poderão partir para a próxima iteração de busca depois que a lista for atualizada. EPs que permitam a saída de threads da barreira antes da hora terão nota ZERO.

[1] `https://en.wikipedia.org/wiki/Dijkstra_algorithm`

2.2 Linguagem

Seu programa deve ser escrito em C, C++ ou java. Programas escritos em outra linguagem terão nota ZERO.

2.3 Entrada e saída

O seu programa receberá como entrada o valor de n e o nome de um arquivo texto que contém a topologia do grafo como uma matriz de adjacências. Em todos os grafos passados como entrada, a existência de uma aresta de um vértice `A` para um vértice `B` implicará na existência de uma aresta do vértice `B` para o vértice `A`. Para o cálculo do menor caminho deve ser considerado que o peso de cada aresta vale 1. Todos os grafos passados como entrada serão grafos conexos.

O seu programa deverá aceitar uma opção `-debug` (Tem que ser assim, com o "`-`"na frente) e quando essa opção for passada na linha de comando, as seguintes informações devem ser apresentadas cada vez que as threads se encontrarem em uma barreira:

- Número da iteração, que vai ser um contador que armazena quantas vezes as threads se encontraram na barreira;
- Ordem com que as threads chegaram na barreira. Cada thread precisa de um identificador;
- $m - 1$ listas atualizadas, cada uma com os n menores caminhos. Inicialmente pode haver menos do que n já que a lista ainda está sendo construída.

Quando o programa for executado com ou sem a opção `-debug`, ele precisará informar ao término da execução:

- Número de iterações: a quantidade de vezes que as threads se encontraram na barreira;
- $m - 1$ listas, cada uma com os n menores caminhos, ou um valor menor que n caso não hajam tantos caminhos assim entre os vértices.

Cada caminho deve ser apresentado da seguinte forma:

```
(x) 0 - b - c - d - e - f - g
```

Onde `(x)` é a quantidade de arestas do caminho (no exemplo acima, 6) e `b` a `g` são os identificadores dos roteadores do caminho entre 0 e `g`. A identificação dos vértices deve ser feita considerando a ordem com que eles são apresentados na matriz de adjacências. Não mude esse padrão para evitar que os resultados saiam diferentes do esperado. Abaixo é apresentado um exemplo de saída do programa:

```
Para o vertice 1:
(2) 0 - 3 - 1
(3) 0 - 2 - 3 - 1

Para o vertice 2:
(2) 0 - 1 - 2
(2) 0 - 3 - 2

Para o vertice 3:
(3) 0 - 2 - 1 - 3
(3) 0 - 1 - 2 - 3
```

3 Relatório

Você deve entregar um relatório apresentando a saída do seu programa para o grafo abaixo, um grafo que representa a topologia da NSFNet[2]:

```
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0
0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1
0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

A saída deve ser decorrente da execução do programa com n igual a 2 e com a opção `-debug`. Apesar do relatório apresentar o resultado para o grafo da NSFNet, isso não significa que o EP será avaliado apenas com esse cenário.

O relatório também deve apresentar informações sobre a implementação da barreira (qual barreira foi implementada e onde ela está implementada – arquivo e número de linhas). Outras informações que você julgar necessárias para explicar o seu código podem ser apresentadas no relatório.

4 Sobre a entrega

Você deve entregar um arquivo .tar.gz contendo os seguintes itens:

- fonte, Makefile (ou similar), arquivo LEIAME;
- relatório em .pdf.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-membros_da_equipe. Por exemplo: ep2-joao-maria.

A entrega do .tar.gz deve ser feita através do PACA.

O EP pode ser feito individualmente ou em dupla.

[2] http://www.nsfnet-legacy.org/about.php

5 Bônus para o programa mais eficiente

Os EPs que receberem nota 10,0 passarão por uma avaliação extra para verificar qual é o mais eficiente. Os códigos serão executados em diversos cenários definidos pelo professor e aquele que rodar, em média, mais rápido será considerado o mais eficiente. Os autores deste EP ganharão 1,0 ponto extra na média final.