Técnicas de Busca e Ordenação 2022/2

Trabalho Prático T2

3 de novembro de 2022

Leia atentamente **todo** esse documento de especificação. Certifique-se de que você entendeu tudo que está escrito aqui. Havendo dúvidas ou problemas, fale com o professor o quanto antes. As dúvidas podem ser sanadas usando o fórum de dúvidas do AVA.

1 Objetivo

O objetivo deste trabalho é aplicar o algoritmo de Dijkstra em um contexto relacionado a redes de computadores. Para isso, vocês precisarão utilizar e estender a estrutura de dados *heap* vista em aula.

2 O problema de seleção de servidores de conteúdo

Você já parou para pensar como as grandes provedoras de conteúdo da Internet são capazes de entregar um volume gigante de dados para milhões de clientes e com boa qualidade? Bom, nem sempre isso foi possível. Durante algumas décadas, esse cenário era completamente inviável e muitos achavam que até impossível. Então, como isso é possível hoje? Além da evolução tecnológica e fatores de mercado, houve também uma evolução significativa nas técnicas de medição e alocação de recursos. De uma forma **bem simplificada** (e vocês estudarão detalhes em uma disciplina específica de Redes de Computadores), a ideia é a seguinte:

- O mesmo conteúdo (e.g., vídeo) é replicado em diferentes servidores, os quais podem estar espalhados em várias partes de uma região/mundo;
- constantemente, as empresas monitoram o estado do tráfego na Internet. O objetivo desse monitoramento é estimar o tempo que um pacote de dados necessita para sair de um servidor, chegar à rede de acesso de um cliente (ou ao cliente em si) e confirmar sua chegada para o servidor. Esse tempo é denominado RTT (*Round Trip Time*).
- Uma vez que um cliente faz uma requisição por um conteúdo específico, essa requisição é direcionada para o servidor que possuir, por exemplo, o menor RTT com o cliente que originou a requisição.

Infelizmente, a obtenção dos RTTs entre servidores e clientes em tempo real nem sempre é possível, uma vez que o número necessário de medições (de cada servidor para cada cliente) pode ser gigantesco. Uma alternativa é utilizar um conjunto, pequeno, de monitores espalhados pela rede. Nesse contexto, ao invés de medir o RTT entre um servidor e um cliente, mede-se o RTT entre os monitores e o servidor e entre os monitores e o cliente. Daí, aproxima-se o RTT do servidor para o cliente como sendo o menor valor dos RTTs entre o servidor e os monitores mais o RTT entre cada cliente e os monitores.
Se o número de servidores e clientes for muito grande, o número de monitores pequeno e os monitores forem adequadamente instalados, pode-se mostrar que essa abordagem gera uma boa aproximação para os valores reais de RTT e economiza significativamente o número de medições necessárias.
Dadas a configuração de uma rede (i.e., um grafo) e a indicação de quais nós da rede são os monitores, seu trabalho será calcular quão boa a aproximação descrita é.
Observação: o método de aproximação descrito acima também é uma simplificação. Além disso, nem todas empresas seguem essa abordagem. Por fim, vale mencionar que a Internet é um ambiente extremamente complexo e tratá-la como um grafo nem sempre é uma boa decisão...

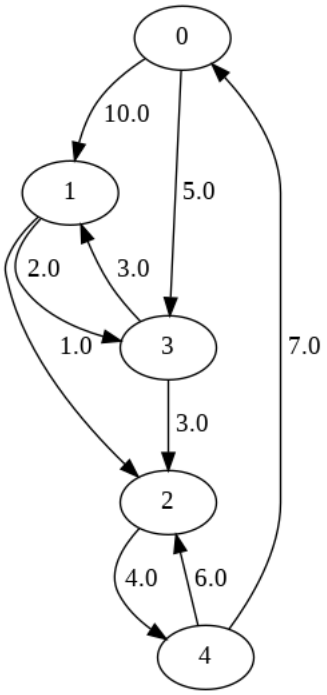

Figura 1: Exemplo de rede. Cada nó representa um computador/dispositivo de rede, uma aresta de a para b significa que o nó a pode mandar informação para o nó b e o peso de cada aresta indica o "tempo" que a informação demora para percorrer o enlace entre os dois nós.

3 Formalização e Exemplo

Em nossa simplificação, vamos assumir que a rede em questão é representada por um grafo $G(V, E, \omega)$. Onde:

- V é o conjunto de nós da rede, representando computadores e dispositivos de rede (você não precisa se preocupar com qual é qual). Os nós estarão numerados de 0 até $|V| - 1$.
- E é o conjunto de arestas direcionadas. Uma aresta $(a, b) \in E$ significa que informação pode fluir na rede do nó a para o nó b. Veja que o contrário não é válido.
- ω é uma função que mapeia o conjunto de arestas nos reais positivos. Mais especificamente, $\omega(a, b)$ indica o "tempo" (ou custo) de mandar uma unidade de informação (pacote) de a para b.

A Figura 1 mostra um exemplo do tipo de grafo que vamos considerar.
Vamos definir $\delta(a, b)$ o custo do menor caminho do nó a até o nó b. Na Figura 1, $\delta(0, 4) = 12$ (seguindo o caminho $0 \to 3 \to 2 \to 4$) e $\delta(4, 0) = 7$ (seguindo o caminho $4 \to 0$).
Vamos então definir o $RTT(a, b) = \delta(a, b) + \delta(b, a)$. Assim, $RTT(0, 4) = 12 + 7 = 19 = RTT(4, 0)$. Neste trabalho, vamos assumir que $RTT(a, b) < \infty$ para quaisquer $a \in V$ e $b \in V$. Ou seja, vamos assumir que sempre existe um caminho direcionado entre quaisquer pares de nós do grafo.
Considere agora os subconjuntos S, C e M de V. Vamos assumir que esse subconjuntos são não vazios e, dois a dois, não possuem interseção. S representa o conjunto de nós servidores, C representa o conjunto de nós clientes e M representa o conjunto dos nós monitores. Vamos definir o RTT estimado entre a e b como sendo:

$$RTT^*(a, b) = \min_{m \in M} (RTT(a, m) + RTT(m, b))$$

Suponha, no exemplo da Figura 1, que $S = \{0\}$, $C = \{4\}$ e $M = \{1, 2\}$. Temos:

$$\begin{aligned} RTT^*(0, 4) &= \min\{RTT(0, 1) + RTT(1, 4), RTT(0, 2) + RTT(2, 4)\} \\ &= \min\{20 + 20, 19 + 10\} \\ &= \min\{40, 29\} \\ &= 29. \end{aligned}$$

Assim, quado calculamos $RTT^*(0,4)$ ao invés de $RTT(0,4)$, inflamos o valor original em aproximadamente 52,6%.

4 Sua tarefa

A sua tarefa será entender a inflação nos RTTs quando a abordagem aproximada, descrita acima, é utilizada. Para isso, você deverá seguir os seguintes passos:

1. Sua entrada será o grafo $G(V, E, \omega)$ e os conjuntos S, C e M;
2. Utilize o algoritmo de Dijkstra para calcular os caminhos mínimos de todos os nós em S para todos os nós em C e de todos os nós em C para todos os nós em S. Com isso, você poderá calcular os RTTs reais entre todos os servidores e clientes.
3. Utilize o algoritmo de Dijkstra para calcular os caminhos mínimos de todos os nós em S para todos os nós em M (e vice-versa) e de todos os nós em M para todos os nós em C (e vice-versa). Com esses valores, você conseguirá calcular os valores de RTT aproximados (i.e., RTT^*) de todos os nós em S para todos os nós em C.
4. Para cada par a, b com $a \in S$ e $b \in C$, calcule a inflação do RTT quando substituímos $RTT(a, b)$ por $RTT^*(a, b)$, a qual é dada por $\frac{RTT^*(a,b)}{RTT(a,b)}$.

5 Entrada e Saída

5.1 Entrada

Todas as informações serão dadas em um arquivo de entrada. Este arquivo vai conter uma descrição de todos os nós, arestas, pesos e conjuntos S, C e M.
A primeira linha do arquivo terá $|V|$ e $|E|$. A segunda linha do arquivo terá os valores de $|S|$, $|C|$ e $|M|$. Após isso:

1. as próximas $|S|$ linhas indicam os nós que são servidores;
2. as próximas $|C|$ linhas indicam os nós que são clientes;
3. as próximas $|M|$ linhas indicam os nós que são monitores;
4. cada uma das próximas $|E|$ linhas terá três valores, x, y e z, indicando que há uma aresta de x para y e que z é o valor de $\omega(x, y)$.

A seguir, o conteúdo do arquivo relativo ao exemplo da Figura 1, assumindo que $S = \{0\}$, $C = \{4\}$ e $M = \{1, 2\}$.

```
5 9
1 1 2
0
4
1
2
0 1 10.0
0 3 5.0
1 3 2.0
3 1 3.0
1 2 1.0
3 2 3.0
2 4 4.0
4 2 6.0
4 0 7.0
```

5.2 Saída

A saída do trabalho deverá ser salva em um arquivo, também de texto, contendo $|S||C|$ linhas e três colunas. As duas primeiras colunas terão todas as combinações de servidores e clientes. Já a terceira coluna, referente ao servidor a e cliente b deverá ter o valor de $\frac{RTT^*(a,b)}{RTT(a,b)}$. As linhas devem estar ordenadas, em ordem crescente, pelo valor da terceira coluna.
Para o exemplo de entrada da seção anterior, a saída deverá ser como abaixo.

```
0 4 1.5263157894736843
```

Observação: utilizem o formato padrão de `double` para a impressão da terceira coluna. Não se preocupem com erros de precisão nas últimas casas decimais, vou desconsiderá-las na correção.

5.3 Execução do trabalho

Para testar seu trabalho, o professor executará comandos seguindo o seguinte padrão.

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab2 <nome_arquivo_entrada> <nome_arquivo_saida>
```

É extremamente importante que vocês sigam esse padrão. Seu programa não deve solicitar a entrada de nenhum valor e também não deve imprimir nada na tela.
Por exemplo, se o nome do arquivo recebido for `2004209608.tar.gz`, os dados de entrada estiverem em `entrada.txt` e o nome do arquivo de saída for `saida.txt`, o professor executará:

```
tar -xzvf 2004209608.tar.gz
make
./trab2 entrada.txt saida.txt
```

6 Sobre o algoritmo de Dijkstra

A parte central deste trabalho é a implementação do algoritmo de Dijkstra utilizando a estrutura *heap* para implementar uma fila de prioridades. Para tal, vocês deverão utilizar e estender o código visto em aula.
Uma boa explicação de como representar grafos em memória (utilizando uma lista de adjacências) e de como o algoritmo de Dijkstra funciona pode ser encontrada facilmente na Web. Em especial, recomenda-se o vídeo do Professor Sedgewick[1]. O algoritmo também está descrito em detalhes na Wikipedia[2].

7 Detalhes de implementação

A seguir, alguns detalhes, comentários e dicas sobre a implementação. Muita atenção aos usuários do Sistema Operacional Windows.

- o trabalho deve ser implementado em C. A versão do C a ser considerada é a presente no Sistema Operacional Ubuntu 20.04.
- o caractere de nova linha será o `\n`.
- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Crie um arquivo `Makefile` que gera como executável para o seu programa um arquivo de nome `trab2`.
- Ao longo do desenvolvimento do trabalho, certifique-se que o seu código não está vazando memória testando-o com o `valgrind`. Não espere terminar o código para usar o `valgrind`, incorpore-o no seu ciclo de desenvolvimento. Ele é uma ferramenta excelente para se detectar erros sutis de acesso à memória que são muito comuns em C. Idealmente o seu programa deve sempre executar sem nenhum erro no `valgrind`.

[1] `https://www.youtube.com/watch?v=uzHJXbToiIU&list=PLRdD1c6QbAqJn0606RlOR6T3yUqFWKwmX&index=75`
[2] `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`

- Não é necessária nenhuma estrutura de dados muito elaborada para o desenvolvimento deste trabalho. Todas as estruturas que você vai precisar foram discutidas em aula ou no laboratório. Veja os códigos disponibilizados pelo professor para ter ideias. Prefira estruturas simples a coisas muito complexas. Pense bem sobre as suas estruturas e algoritmos antes de implementá-los: quanto mais tempo projetando adequadamente, menos tempo depurando o código depois.

8 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 23:59h do dia 02/12/2022. Não serão aceitos trabalhos após essa data.
- **Prazo para tirar dúvidas**: Para evitar atropelos de última hora, você deverá tirar todas as suas dúvidas sobre o trabalho até o dia 30/11/2022. A resposta para dúvidas que surgirem após essa data fica a critério do professor.
- **Grupo**: O trabalho pode ser feito em grupos de até três pessoas. Lembrando que duas pessoas que estiverem no mesmo grupo nesse trabalho não poderão estar no mesmo grupo nos próximos trabalhos. Além disso, pessoas do mesmo grupo no trabalho 1 não poderão estar no mesmo grupo neste trabalho.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código e um `Makefile`, como especificado anteriormente. **Somente uma pessoa do grupo deve enviar o trabalho no AVA. Coloque a matrícula de todos integrantes do grupo (separadas por vírgula) no nome do arquivo do trabalho.**
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

9 Relatório de resultados

Esse trabalho não demandará relatório.

10 Avaliação

- Assim como especificado no plano de ensino, o trabalho vale 10 pontos.
- A parte de implementação será avaliada de acordo com a fração e tipos de casos de teste que seu trabalho for capaz de resolver de forma correta. Casos *pequenos* e *médios* (4 pontos) serão utilizados para aferir se seu trabalho está correto. Casos *grandes* (4 pontos) serão utilizados para testar a eficiência do seu trabalho. Casos *muito grandes* (2 pontos) serão utilizados para testar se seu trabalho foi desenvolvido com muito cuidado e tendo eficiência máxima como objetivo. Todos os casos de teste serão projetados para serem executados em poucos minutos (no máximo 5) em uma máquina com 16GB de RAM.
- Trabalhos com erros de compilação receberão nota zero.
- Trabalhos que gerem *segmentation fault* para algum dos casos de teste disponibilizados no AVA serão severamente penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- O uso de variáveis globais ou da primitiva `goto` implica em nota zero.
- Organização do código e comentários valem nota. Trabalhos confusos e sem explicação sofrerão desconto na nota.
- Caso seja detectada **cópia** (entre alunos ou da Internet), todos os envolvidos receberão nota zero. Caso as pessoas envolvidas em suspeita de cópia discordem da nota, amplo direito de argumentação e defesa será concedido. Neste caso, as regras estabelecidas nas resoluções da UFES serão seguidas.

- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre cópia, acima.)