Técnicas de Busca e Ordenação 2024/2

Trabalho Prático T1

December 1, 2024

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.

1 Objetivo

O objetivo deste trabalho é relacionado ao cálculo de caminhos mínimos utilizando o Algoritmo de Dijkstra. Para isso, será necessária a utilização e extensão de estruturas de dados implementadas como heap (organizados como TAD) vistas em aula.

2 O problema de seleção de caminhos mínimos para transporte/roteamento de informações

Para todo problema de transporte entre pontos específicos de uma malha de postos de parada, o cálculo de caminhos mínimos são essenciais para otimizar o transporte de itens com o menor custo possível. Podemos ilustrar o problema de caĺculo do caminho de custo mínimo como uma ferramente para resolver um problema bastante conhecido do dia dia: o transporte de produtos comprados pela internet. Se imaginarmos um popular *marketplace* como o Mercado Livre, os itens comprados são transportados entre postos da própria loja até atingirem os postos mais próximos do ponto de entrega. Desta forma, é de suma importância garantir o melhor percurso para o transporte dos itens comprados entre os postos do *marketplace*, visando minimizar o custo atrelado ao envio dos produtos.
Portanto, dada a configuração de uma rede de postos de transporte de uma loja (i.e., descrito por um grafo) e a indicação de quais nós desta rede de postos de coleta e transporte estão conectados, seu trabalho será calcular o caminho de custo mínimo de um posto de orgiem para todos os demais postos da rede.

3 Formalização e Exemplo

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

- V é o conjunto de nós da rede, representando postos físicos de coleta e transporte de itens da loja (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 posto a para o posto 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 (item) de a para b.
- chamaremos de d a combinação de uma aresta direcionada E_i e seu custo mapeado pela função ω para levar uma informação de a até b, por exemplo.

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 $4 \to 0$).

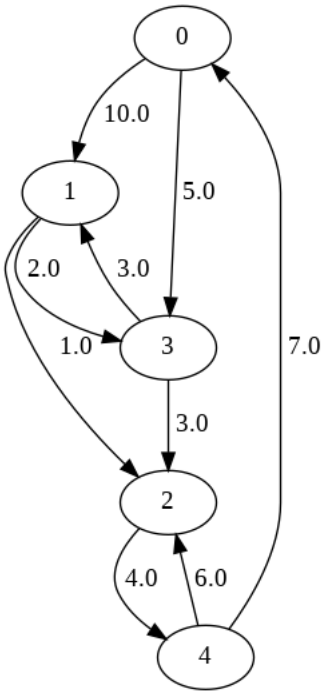

Figure 1: Exemplo de rede de postos de parada do *marketplace*. Cada nó representa um posto de coleta/transporte da rede, uma aresta de a para b significa que o nó a pode mandar informação para o nó b e o peso de cada aresta indica o "tempo" que a informação demora para percorrer o enlace entre os dois nós.

Considerando que $\delta(a, b)$ é o custo de acesso ao nó b partindo do nó a da rede, e que $\delta(a, b)$ pode ser diferente de $\delta(b, a)$ (dependendo do custo de b para a, assumiremos que também existe a possibilidade de $\delta(a, b) = 0$ ou $\delta(b, a) = 0$, caso o caminho da origem para o destino não exista. Ainda, assumiremos que para quaisquer nós a e b da rede de pontos de transporte, $\delta(a, b) < \infty$. Por fim, considera-se que o conjunto V de nós da rede é não vazio.

4 Sua tarefa

A sua tarefa será entender e implementar, de forma modularizada e eficiente, os caminhos de custo mínimo entre todos os pontos de transporte da rede. Para isso, você deverá seguir os seguintes passos:

1. Sua entrada será o nó de origem V_{src} ao qual deseja-se calcular os caminhos de custo mínimo e o grafo $G(V, (E, \omega) = d_{1,\dots,n-1}$ - representado pelas distâncias entre os nós V_i e todos os demais $n - 1$ nós da rede) $\therefore G(V, d_{1,\dots,n-1})$, com V representando o conjunto de vértices da rede de postos de transporte;

2. Utilize o algoritmo de Dijkstra para calcular os caminhos mínimos de todos os nós em V do nó V_{src} para todos os demais nós.

3. Para cada par a, b com a sendo o ponto de partida e b sendo um potencial destino, o caminho de custo mínimo de ser explicitado (mostrado, com todos os nós intermediários do caminho total), bem como o valor de seu custo de transporte.

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 e pesos V, d.

A primeira linha do arquivo terá V_{src}, indicando o nó *source* de onde deve partir-se para calcular as rotas de custo mínimo. Após isso:

1. as próximas $|V|$ linhas indicam os nós da rede;
2. para cada nó V_i da rede, com $i = 0, ..., (|V| - 1)$, d_k distâncias descrevem as arestas e pesos de um nó para os demais da rede, com $k = 0, ..., (|V| - 1)$ e $k \neq i$.

A seguir, o conteúdo do arquivo relativo ao exemplo da Figura 1, assumindo que $V_{src} = 0$, $V = \{5\}$ e $d = \{4\}$.

```
node_0
node_0, 10, 0, 5, 0
node_1, 0, 1, 2, 0
node_2, 0, 0, 0, 4
node_3, 0, 3, 3, 0
node_4, 7, 0, 6, 0
```

5.2 Saída

A saída do trabalho deverá ser salva em um arquivo, também de texto, contendo os caminhos de menor custo do nó V_{src} até cada um dos demais nós da rede, seguido do custo para acessar os respectivos nós. Como pode ser observado, a disposição dos nós está ordenada por seu custo. Atente-se muito bem ao padrão de saída do resultado, uma vez que qualquer resultado diferente será considerado errado.
Para o exemplo de entrada da seção anterior, a saída deverá ser como abaixo.

```
SHORTEST PATH TO node_0: node_0  <- node_0 (Distance: 0.00)
SHORTEST PATH TO node_3: node_3  <- node_0 (Distance: 5.00)
SHORTEST PATH TO node_1: node_1  <- node_3  <- node_0 (Distance: 8.00)
SHORTEST PATH TO node_2: node_2  <- node_3  <- node_0 (Distance: 8.00)
SHORTEST PATH TO node_4: node_4  <- node_2  <- node_3  <- node_0 (Distance: 12.00)
```

Observação: utilizem o formato padrão de `float` para a impressão das distâncias que representam o custo, com precisão decimal de 2 casas.

5.3 Execução do trabalho

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

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab1 <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
./trab1 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].

[1] `https://www.youtube.com/watch?v=uzHJXbToiIU&list=PLRdD1c6QbAqJn0606RlOR6T3yUqFWKwmX&index=75`

[2] `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`

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 nos Computadores do LabGrad (Ubuntu).
- 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 `trab1`.
- 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`.
- 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.
- **Devem ser implementadas DUAS (2) abordagens diferentes de solução para o problema de Dijkstra para o problema de cálculo de caminhos mínimos.** Uma delas deve ser implementada utilizando a estrutura de dados heap e fila de prioriades. A outra abordagem é de livre escolha dos grupos. A reflexão sobre a segunda abordagem é importante, uma vez que escolhas inapropriadas e não documentadas/justificadas no relatório do trabalho implicam em problemas para a execução dos casos de teste e, consequentemente, descontos na nota.

8 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 08:00h do dia 20/03/2025. Não serão aceitos trabalhos após esta data.
- **Grupo**: O trabalho deve ser feito em grupos de até três pessoas. Lembrando que duas pessoas que estiverem no mesmo grupo neste trabalho não poderão estar no mesmo grupo no próximo trabalho.
- **Como entregar**: Pela atividade criada no Classroom. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código, `Makefile`, e um relatório de análise do comportamento dos algoritmos implementados. **Somente uma pessoa do grupo deve enviar o trabalho no Classroom**. Coloque a matrícula de todos integrantes do grupo (**separadas por traço simples "-"**) no nome do arquivo do trabalho. O padrão de nomes de arquivos para a entrega, portanto, é: **"T1-matricula1-matricula2-matricula3-TBO-2024-2.tar.gz"**. É obrigatório respeitar este padrão. Também, coloque as matrículas e os nomes dos integrantes nos cabeçalhos dos códigos fornecidos, bem como na capa do relatório a ser entregue.
- **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

Este trabalho demanda um relatório de análise das duas abordagens implementadas para a solução do problema de caminhos mínimos. Deve ser comparado o desempenho da versão implementada com heap e filas de prioridades com a segunda implementação da solução para Dijkstra. O grupo é livre para definir a abordagem e a estrutura de dados para a segunda solução, porém, é de extrema importância que a análise e comparação realizadas sejam

coesas e reflitam o comportamento do algoritmo implementado. A comparação deve ser realizada em termos de tempo de execução, consumo de memória e casos de teste executados com sucesso. O insucesso para alguns casos de teste na segunda abordagem não reflete em descontos na nota, contanto que a análise apresente quantitativamente as justificativas para tal.
O conteúdo da disciplina dará arcabouço para a análise do comportamento e comparação de desempenho das duas abordagens implementadas. Faça bom uso do conteúdo apresentado nas aulas e disponibilizado no Classroom.

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 (justificados) 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 execução serão penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- Evite o uso de variáveis globais. Isto pode implicar em penalização na nota - use a modularização de forma prudente.
- 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.)
- Cada implementação deve apresentar, **OBRIGATÓRIAMENTE, pelo menos 2 (DOIS) TADs totalmente opacos, sendo que ambas podem compartilhar somente 1 deles. Isto significa que pelo menos 3 (TRÊS) TADs opacos devem ser implementados.** O uso de mais TADs é encorajado. Modele bem o problema para garantir uma modularização eficiente e livre de vazamentos de memória.

BOM TRABALHO!