Trabalho Prático 2: Reducing the costs!

Entrega: 08/06/2015

1 Introdução

No Brasil, diversos produtos têm sofrido aumento nos preços, e os combustíveis não ficaram fora disso. Devido a este aumento, você foi contratado por uma empresa de transporte para reduzir seus custos. Como a empresa não quer mandar nenhum funcionário embora, para manter a produtividade e o lucro, ela quer reduzir a quantidade de combustível utilizada por seus caminhões. Desta forma, a sua função é reduzir a quilometragem rodada pelos caminhões na busca e entrega de produtos.

Todo caminhão da empresa faz sempre o seguinte percurso: sai de uma determinada *cidade x*, percorre todas as cidades atendidas pela empresa e retorna à *cidade x* novamente. Em algumas cidades esses caminhões devem pegar os produtos e em outras eles devem entregar os produtos. Desta forma, se um produto que está na *cidade x* deve ser entregue na *cidade y*, o caminhão da empresa deve passar na *cidade x* antes de passar na *cidade y*, pois assim ele não precisa ir em uma mesma cidade mais de uma vez.

Assim, sabendo onde os produtos estão e onde eles deverão ser entregues, você deve determinar a ordem em que as cidades devem ser visitadas para que os caminhões da empresa percorram o menor caminho possível. É importante lembrar que, mesmo que um caminho seja menor, se você visita a *cidade x* antes da *cidade y* e esta última tem produtos para serem entregues na *cidade x*, então esta não é uma ordem válida.

2 Código

Você deve implementar, em linguagem C, um programa que seja capaz de:

- Dado um número *N* de cidades, suas respectivas coordenadas e um conjunto de restrições de precedência, encontrar qual é a menor distância a ser precorrida saindo de uma cidade de origem, percorrendo todas as cidades que devem ser atendidas pela empresa de transporte e retornando a cidade de origem novamente.

- Você deve respeitar as restrições de precedência, ou seja, se na entrada indica que a *cidade x* contém produtos que devem ser entregues na *cidade y*, então a *cidade x* deve ser visitada antes da *cidade y*.

- Cada uma das cidades só podem ser visitadas apenas uma vez, exceto a cidade de origem, que é visitada uma segunda vez quando o caminhão retorna à cidade de origem.
- Caso não seja possível chegar a uma solução por causa das restrições de precedência, deve-se imprimir, ao invés da menor distância, a palavra "Deadlock".
- Uma abordagem de grafos deve ser utilizada para resolver o problema. Qualquer outra abordagem não será considerada e o trabalho receberá nota zero.

3 Entrada e Saída

Seu programa deverá ler a entrada da entrada padrão (`stdin`) e gravar a saída na saída padrão (`stdout`). A entrada de teste é composta por vários casos de teste. Em cada caso de teste, a primeira linha contém um inteiro (*N*) ($1 \leq N \leq 22$) que corresponde ao número de cidades atendidas pela empresa de transporte. Cada uma das *N* linhas seguintes contém dois inteiros separados por espaço, que correspondem, respectivamente, aos valores *x* e *y* ($0 \leq x, y \leq 10^9$) da coordenada daquela cidade. A primeira coordenada da entrada é referente a cidade que tem o identificador 0, a segunda coordenada é referente a cidade com identificador 1, a n-ésima coordenada é referente a cidade com identificador n-1. O ponto de partida do caminhão é a cidade com identificador 0.

Depois de ler as *N* coordenadas das cidades, a entrada da instância contém um inteiro (*R*) ($0 \leq R \leq 462$) que representa o número de restrições de precedência que devem ser consideradas na resolução do problema. Cada uma das *R* linhas seguintes contém dois inteiros, *a* e *b*, que determinam que a *cidade a* deve ser visitada antes da *cidade b*, pois a *cidade a* contém produtos que devem ser entregues na *cidade b*.

Para cada instância do problema, deve ser impresso na saída apenas uma linha com a menor distância necessária para sair da cidade de origem, passar por todas as outras cidades atendidas pela empresa de transporte apenas uma única vez e retornar novamente à cidade de origem, respeitando todas as restrições de precedência listadas na entrada do programa. Esse valor deve ser impresso com precisão de 2 casas decimais.

Um exemplo de entrada e saída para o programa está na Tabela 1.

É importante destacar que uma implementação *naive* do problema pode não rodar dentro do limite de tempo imposto. Dessa forma, você deve fazer uso de técnicas que "podam" parte do espaço de solução do seu algoritmo para certas situações. O TP 0 é um bom exemplo disso, onde você teve que implementar algoritmos de busca em árvores binária. Esses algoritmos fazem uso da propriedade de localização de uma chave nesta estrutura: tudo que é maior está à direita e menor, à esquerda. Dessa forma, ele sempre elimina metade do espaço de busca em cada iteração sendo, no pior caso, O(log n).

Entretanto, uma "poda" pode não ocorrer sempre, ou seja, ela pode depender de algumas situações para ser aplicada. Por exemplo, na PA 4 (estacionamento), uma "poda" simples é, ao procurar uma vaga para o carro no estacionamento, parar imediatamente após encontrá-la, sem precisar percorrer o restante do estacionamento. Note que essa "poda" não é feita quando a vaga não existe e você deve percorrer todo o espaço de solução. Em outras palavras, ela faz o algoritmo executar mais rapidamente em certas situações mas não diminui a ordem de complexidade

ENTRADA	SAÍDA
4	Deadlock
0 0	9.24
0 8	53.74
4 8	7462.06
4 0	
4	
2 1	
2 3	
3 1	
1 3	
6	
1 1	
2 1	
2 2	
3 2	
2 4	
1 4	
2	
2 1	
5 4	
20	
1 1	
2 2	
3 3	
4 4	
5 5	
6 6	
7 7	
8 8	
9 9	
10 10	
11 11	
12 12	
13 13	
14 14	
15 15	
16 16	
17 17	
18 18	
19 19	
20 20	
19	
0 1	
1 2	
2 3	
3 4	
4 5	
5 6	
6 7	
7 8	
8 9	
9 10	
10 11	
11 12	
12 13	
13 14	
14 15	
15 16	
16 17	
17 18	
18 19	
12	
1800 1000	
900 200	
1300 1500	
2200 600	
1300 1700	
200 700	
600 1100	
1100 400	
100 1500	
1300 1400	
2500 800	
2500 140	
0	

Table 1: Toy example: teste o qual o TP deve gerar a saída correta para ser elencado para uma entrevista. Esse teste está disponível em um arquivo no moodle.

no pior caso. Para o trabalho, você deve ser capaz de procurar situações as quais você pode efetuar ”podas”.

Sem fazer ”podas” (branch and bound), é improvável que seu programa passe pelo toy example no PRÁTICO. Logo, se seu toy foi aceito, é um bom indicativo que suas ”podas” estão sendo eficazes e é provável que seu algoritmo execute dentro do limite de tempo para boa parte dos testes. Entretanto, para uma pequena parcela, é necessário ir além e realizar diversas podas no algoritmo, de forma que ele execute mais rapidamente.

4 O que entregar

Você deve submeter uma documentação de até 12 páginas contendo uma descrição de sua solução para o problema, além de uma análise de complexidade de tempo dos algoritmos envolvidos e uma análise da memória gasta pelo seu programa. Siga as diretrizes sobre como fazer uma documentação que foram disponibilizadas no portal *minha.ufmg*.

Além disso, você deve cobrir os seguintes itens em sua documentação:

- **A formalização matemática do problema.**
- **A prova de que o problema é NP-completo.**
- **Mostrar que existe uma instância fácil para o problema.**
- **Descrição das ”podas” realizadas com sua eficácia, ou seja, verificar se ela se aplica a qualquer situação. Caso não se aplique, mostrar a configuração ideal da ”poda” (a qual ela tem o máximo de sua eficácia) e o pior caso (quando ela não é aplicada).**

Estes itens são essenciais para seu trabalho e serão bastante considerados na correção do mesmo. Dessa forma, é altamente recomendável responder essas questões em sua documentação.

Além da documentação, você deve submeter um arquivo compactado no formato *.zip* contendo todos os arquivos de código (*.c* e *.h*) que foram implementados. Além dos arquivos de código, esse arquivo compactado deve incluir um *makefile*. Consulte os tutoriais disponibilizados no *minha.ufmg* para descobrir como fazer um. Finalmente, lembre-se de não incluir nenhuma pasta no arquivo compactado.

5 Avaliação

Seu trabalho será avaliado quanto a documentação escrita e à implementação. Eis uma lista **não exaustiva** de critérios de avaliação utilizados.

Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa. Para tal, artificios como pseudo-códigos, exemplos ou figuras podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é preciso incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando os mesmos influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo dos principais algoritmos implementados e uma análise de complexidade de espaço das principais estruturas de dados de seu programa. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de items a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

Limite de Tamanho Sua documentação deve ter no máximo 12 páginas. Todo o texto a partir da página 13, se existir, será desconsiderado.

Guia Há um guia e exemplos de documentação disponíveis no moodle.

Implementação

Linguagem e Ambiente

- Implemente tudo na linguagem **C**. Você pode utilizar qualquer função da biblioteca padrão da linguagem em sua implementação, mas **não** deve utilizar outras bibliotecas. **Trabalhos em outras linguagens de programação serão zerados. Trabalhos que utilizem outras bibliotecas também.**
- Os testes serão executados em **Linux**. Portanto, garanta que seu código compila e roda corretamente nesse sistema operacional. A melhor forma de garantir que seu trabalho rode em Linux é escrever e testar o código nele. Há dezenas de máquinas com Linux nos laboratórios do DCC que podem ser utilizadas. Você também pode fazer o download de uma variante de Linux como o **Ubuntu** (`http://www.ubuntu.com`) e instalá-lo em seu computador ou diretamente ou por meio de uma máquina virtual como o **VirtualBox** (`https://www.virtualbox.org`). Há vários tutoriais sobre como instalar Linux disponíveis na web.

Casos de teste Seu trabalho será executado em um conjunto de entradas de teste.

- Essas entradas *não* serão disponibilizadas para os alunos até que a correção tenha terminado. Faz parte do processo de implementação testar seu próprio código.

- Você perderá pontos se o seu trabalho produzir saídas incorretas para algumas das entradas ou não terminar de executar dentro de um tempo limite pré-estabelecido. Esse tempo limite é escolhido com alguma folga. Garanta que seu código roda a entrada de pior caso em no máximo alguns minutos e você não terá problemas.
- A correção será **automatizada**. Esteja atento ao formato de saída descrito nessa especificação e o siga precisamente. Por exemplo: se a saída esperada para uma certa entrada o número 10 seguido de uma quebra de linha, você deve imprimir apenas isso. Imprimir algo como "A resposta e: 10" contará como uma saída **errada**.
- Os exemplos mostrados nessa especificação são parte dos casos de teste.
- Os testes disponíveis no PRATICO são apenas preliminares. Os monitores executarão os trabalhos nos demais casos de teste.
- **Você deve entregar algum código e esse código deve compilar e executar corretamente para, pelo menos, o *toy example* da tabela 1, que é o primeiro teste do PRATICO. Se isso não ocorrer, a nota do trabalho prático será zerada.**

Alocação Dinâmica Você deverá fazer uso das funções `malloc()` ou `calloc()` da biblioteca padrão C, bem como liberar *tudo* o que for alocado utilizando `free()`, para gerenciar o uso da memória que está disponível para você.

Makefile Inclua um makefile na submissão que permita compilar o trabalho.

Qualidade do Código Seu código deve ser bem escrito:

- Dê nomes a variáveis, funções e estruturas que façam sentido.
- Divida a implementação em módulos que tenham um significado bem definido.
- Acrescente comentários sempre que julgar apropriado. Não é necessário parafrasear o código, mas é interessante acrescentar descrições de alto nível que ajudem outras pessoas a entender como sua implementação funciona.
- Evite utilizar variáveis globais.
- Mantenha as funções concisas. Seres humanos não são muito bons em manter uma grande quantidade de informações na memória ao mesmo tempo. Funções muito grandes, portanto, são mais difíceis de entender.
- Lembre-se de indentar o código. Escolha uma forma de indentar (tabs ou espaços) e mantenha-se fiel a ela. Misturar duas formas de indentação pode fazer com que o código fique ilegível quando você abri-lo em um editor de texto diferente do que foi utilizado originalmente.
- Evite linhas de código muito longas. Nem todo mundo tem um monitor tão grande quanto o seu. Uma convenção comum adotada em vários projetos é não passar de 80 caracteres de largura.
- **Tenha bom senso**.

6 Considerações Finais

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiver problemas para entender o que está escrito aqui, pergunte a nós, monitores.
- A penalização por atraso seguirá será de $\frac{2^{d-1}}{0.32}\%$, em que d é o número de dias **úteis** de atraso.
- Você **não** precisa de utilizar uma IDE como Netbeans, Eclipse, Code::Blocks ou QtCreator para implementar esse trabalho prático. No entanto, se o fizer, **não** inclua os arquivos que foram gerados por essa IDE em sua submissão.
- **Seja honesto**. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que tomem as devidas providências.