Trabalho Prático 2: The Force Awakens

Entrega: 15/11/2015

26 de outubro de 2015

1 Introdução

Após a vitória da Aliança Rebelde, toda a galáxia comemora 30 anos de paz com o fim do Império Galáctico. O que eles não sabem é que o lado negro da força planeja uma nova revolução e seu objetivo com o novo e temível Lord Sith é reconquistar a galáxia e vingar a morte do seu mestre Darth Vader. Como o lado negro da força ainda está fraco, é preciso economizar recursos e parte do seu plano consiste em escolher quais planetas serão reconquistados primeiro. Fazendo uso de um mapa da galáxia é possível traçar a rota com todos os planetas entre sua localização atual, o ponto de chegada e a distância entre eles. Para sua investida, você e seus aliados construíram uma nova nave de guerra, a Estrela da Morte III, porém os técnicos avisaram que ela ainda está em fase de testes e por isso ela deve se deslocar o mínimo possível entre cada planeta, onde eles terão tempo para realizar reparos. Sabendo desta informação, dada a rota de planetas próximos, você quer escolher os K planetas a serem reconquistados de modo a minimizar a maior sub-distância do percurso percorrido.

1.1 Exemplo

Vamos sempre considerar que a rota entre os planetas é uma linha reta e que o início e fim do caminho não são planetas. Na figura 1 temos um exemplo de rota com 3 planetas e recursos suficientes para a conquista de 2 deles. Também temos as 3 soluções possíveis para o problema. Podemos observar que na última solução, a melhor solução para o problema, escolhemos os 2 últimos planetas, desta forma as distâncias entre o início do caminho, os planetas e o fim são 3, 3 e 1. Vale ressaltar que, embora o problema possa ser representado como um grafo, a solução do mesmo não necessita de algoritmos de grafos.

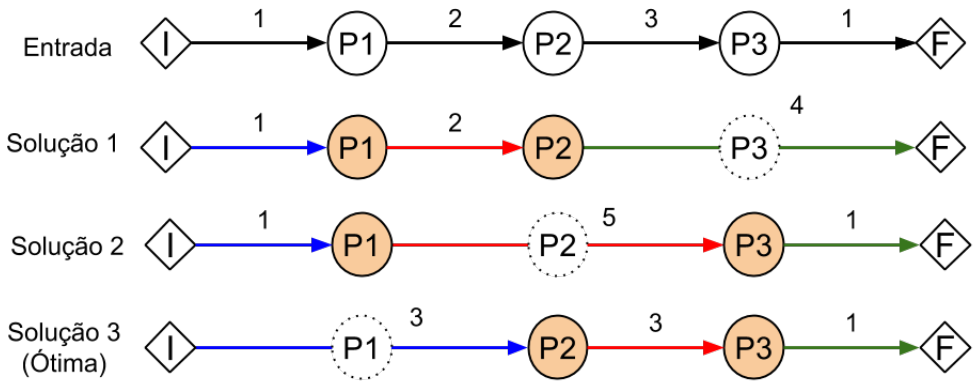

Figura 1: Neste exemplo temos 3 planetas e recursos para conquistar 2 deles, aqui mostramos todas as soluções possíveis para o problema, sendo a melhor a última.

2 Instruções sobre a Implementação

Dada a distância entre N planetas consecutivos de uma rota e o número K de planetas a serem conquistados, sua tarefa é determinar quais planetas serão conquistados de forma a minimizar as sub-distâncias percorridas entre os planetas, o início e o fim do caminho. Lembre-se que:

- Em uma rota não são repetidas arestas. Deseja-se ir de um ponto inicial I até um final F passando uma única vez por cada aresta.
- A rota é uma linha reta. Nunca vai ser possível voltar para um planeta anterior.
- Todas as instâncias testadas terão soluções viáveis.
- $K \leq N$

O problema deverá ser resolvido utilizando três paradigmas de programação diferentes: Força Bruta (FB), Programação Dinâmica (PD) e Algoritmo Guloso (AG). Para a solução com PD discuta as propriedades da subestrutura ótima e sobreposição de subproblemas. Além disso, é importante apresentar a equação de recorrência em que a solução é baseada. Para a solução gulosa, discuta a otimalidade da solução. Se ela sempre fornece a resposta correta, prove que a escolha gulosa leva a solução ótima. Caso contrário, proponha diferentes tipos de teste e discuta a fração de testes para os quais a solução gulosa fornece a resposta correta.

3 Execução

O trabalho deve ser executado da seguinte forma, onde o único argumento 'a' indica o tipo da solução a ser utilizada:

```
./tp2 -a [PD | AG | FB]
```

PD - Programação Dinâmica

AG - Algoritmo Guloso

FB - Força Bruta

4 Formato de Entrada e Saída

Seu programa deverá ler a entrada da entrada padrão ($stdin$) e gravar a saída na saída padrão ($stdout$). Na primeira linha da entrada estará um número inteiro T, $0 < T \leq 100$, que representa o número de instâncias do problema a serem simuladas. A primeira linha de cada instância se inicia com o número N de planetas consecutivos de uma rota e o número K de planetas a serem reconquistados, vamos sempre assumir que $0 < N \leq 500$, enquanto $0 \leq K \leq 250$. Logo após temos as distâncias presentes na rota.

Para o exemplo da figura 1, teríamos a seguinte entrada:

```
1
3 2
1
2
3
1
```

Para cada caso de teste seu programa deve retornar um único número que é o maior sub-distância do caminho:

Para o exemplo da figura 1, teríamos a seguinte saída:

```
3
```

Outro exemplo:

5 O que entregar

Você deve submeter uma documentação de até 10 páginas contendo uma descrição das soluções proposta, detalhes relevantes da implementação, além de uma análise de complexidade de tempo dos algoritmos envolvidos e de complexidade de espaço das estruturas de dados utilizadas. Siga as diretrizes sobre como fazer uma documentação que foram disponibilizadas no portal *minha.ufmg*.

Além da documentação, você deve submeter um arquivo compactado no formato.*tar.gz* 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.

Tabela 1: Toy Example

Entrada	***Saída***
3	
4 3	
7	
2	
6	
4	
5	
4 3	
10	
5	
6	
1	8
2	10
10 5	9
1	
8	
9	
8	
1	
7	
1	
1	
1	
1	
2	

6 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, artifícios como pseudocódigos, 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.

No caso da solução gulosa, qual o subproblema sendo resolvido em cada passo do algoritmo e como a escolha de uma solução para o mesmo continuá ótima nos passos seguintes? Em outras palavras, discorra sobre a sobreposição de problemas na sua solução. No caso da solução de programação dinâmica, explique a sobreposição de problemas e sub-estrutura ótima utilizada na solução. Além disso, é importante apresentar a equação de recorrência em que a solução é baseada.

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. **Qual o custo (assintótico) de execução das soluções de programação dinâmica, gulosa e força bruta? Qual o custo (assintótico) em memória das soluções de programação dinâmica, gulosa e força bruta?** 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 10 páginas. Todo o texto a partir da página 11, 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 preestabelecido. 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 MOODLE 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, um dos testes disponibilizados no MOODLE. 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.

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**.

7 Considerações Finais

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiverem problemas para entender o que está escrito aqui, pergunte aos monitores.
- A penalização por atraso seguirá será de $\frac{2^{d-1}}{0.32}\%$, em que d é o número de dias 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.
- Esteja atento ao tamanho da entrada e às complexidades de seus algoritmos.
- **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.