Trabalho Prático: *The Force Awakens* (versão: 26/8/20)

Disciplina:	Projeto e Análise de Algoritmos
Tema:	Análise de algoritmos, força-bruta, algoritmos gulosos, programação dinâmica
Semestre:	2020/3 (PLE emergencial)
Professor:	Vinícius Dias (viniciusvdias@ufop.edu.br)

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 como 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 (por simplificação, linear) entre todos os planetas entre sua localização atual (I), o ponto final de chegada (F) 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 deve escolher os k planetas a serem reconquistados de modo a minimizar as distâncias do percurso percorrido entre o início do caminho, os planetas escolhidos e o fim.

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. Embora o problema possa ser representado como um grafo a solução do mesmo não necessita de algoritmos de grafos.

Uma dica útil, iniciando de uma solução errada, ou até mesmo uma solução ainda sem a escolha dos planetas, os passos necessários para ir corrigindo essa solução :

1. Nesta solução, você escolhe o número necessário de planetas?
2. Existe uma forma de reduzir a maior sub-distância? Se não houver, você tem o caminho certo!
3. Se sim, no passo anterior, esta redução respeita o passo 1?
4. A redução de um caso aumenta outra sub-distância de forma que ultrapasse o máximo atual? Se sim, não é uma boa redução.

2. Definição formal do problema

Dada a distância entre n planetas consecutivos de uma rota (precisamente, $n+1$ distâncias pois iniciamos de I e terminamos em F, sempre com os planetas entre esses dois pontos – veja figura 1) 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 repetidos trechos. Deseja-se ir de um ponto inicial I até um final F passando uma única vez por cada trecho entre os planetas.
- A rota é uma linha reta. Nunca vai ser possível voltar para um planeta anterior.

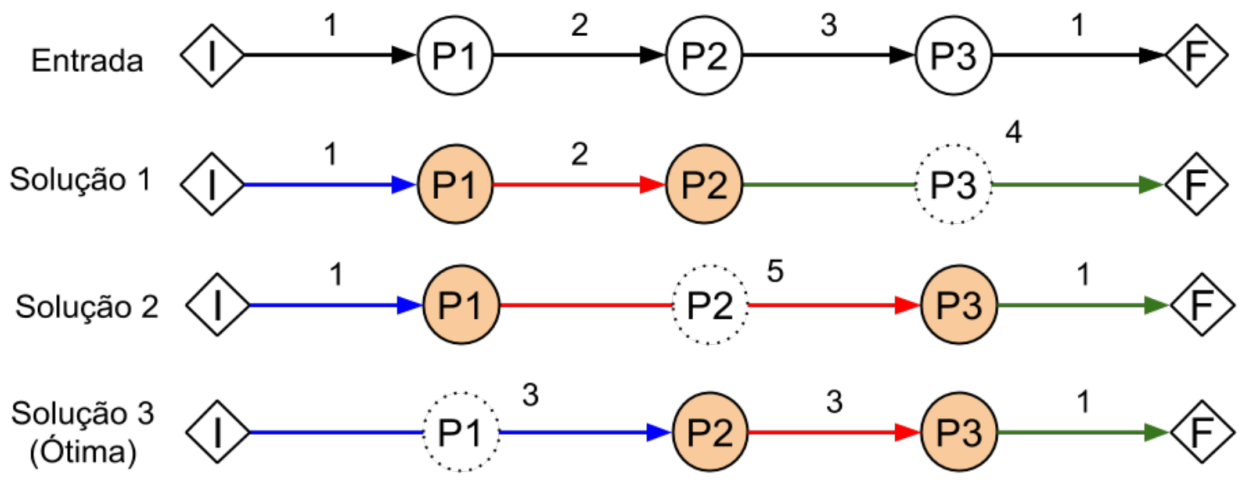

Figura 1. Exemplo de instância do problema ($k = 2$ e $n = 3$) e algumas soluções. Note que a última solução é ótima.

- $k \leq n$

O problema deve 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 seu programa que resolve o problema deve ser executado da seguinte forma:

```
$ ./tp [PD | AG | FB]
```

onde: `PD` indica uma solução usando programação dinâmica, `AG` uma solução usando um algoritmo guloso e `FB` uma solução usando um algoritmo de força-bruta.

4. Entrada e saída

Seu programa deve 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 apos temos as distâncias presentes na rota. A seguinte entrada representa a instância da figura 1:

```
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. Assim, a solução ótima da instância da figura 1 é representada com a seguinte saída:

```
3
```

Outro exemplo:

Entrada	Saída da solução ótima
3 4 3 7 2 6 4 5 4 3 10 5 6 1 2 10 5 1 8 1 7 1 1 1 1 2	8 10 9

5. O que deve ser entregue

1. **Implementação (arquivos `.c` e `.h` apenas):**

   Linguagem. 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.

   Ambiente. 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. Você também pode fazer o download de uma variante de Linux como o Ubuntu[1] e instalá-lo em seu computador ou diretamente ou por meio de uma

[1] <http://www.ubuntu.com>

máquina virtual como o VirtualBox [2]. Há vários tutoriais sobre como instalar Linux disponíveis na web.

Casos de teste. Seu trabalho será executado em conjunto de entradas de teste, não disponibilizadas para vocês: faz parte do processo de implementação aprender a testar a própria solução. A correção será automatizada e por isso, esteja atento ao formato de saída descrito nesta especificação. Por exemplo: se a saída esperada para uma certa entrada for 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 nesta especificação são parte dos casos de teste.

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.

Qualidade do Código. Seu código deve ser bem escrito: (1) dê nomes a variáveis, funções e estruturas que façam sentido; (2) divida a implementação em módulos que tenham um significado bem definido; (3) acrescente comentários sempre que julgar apropriado; (4) não é necessário parafrasear o código, mas é interessante acrescentar descrições de alto nível que ajudem outras pessoas a entender como sua implementação funciona; (5) evite utilizar variáveis globais; (6) 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; (7) lembre-se de indentar o código: escolha uma forma de indentar (tabulações 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. (8) 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.

2. **Implementação (Makefile):** Esse arquivo é obrigatório e usado para compilar o seu trabalho. Uma referência introdutória sobre como usar *makefiles* para compilação pode ser encontrada no rodapé[3].

3. **Documentação (1 arquivo `tp.pdf`):** 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. A seguir, alguns tópicos essenciais esperados em uma boa documentação:

   Cabeçalho. Nomes dos integrantes, números de matrícula, número do grupo, disciplina, código, nome do professor, etc.

   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

[2] <https://www.virtualbox.org>
[3] <https://bit.ly/2PrgIJk>

solução para o mesmo continuá ótima nos passos seguintes? Em outras palavras, discorra sobre a sobreposição de prob- lemas 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 estru- turas 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 com- plexidade 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.

Contribuição de cada integrante do grupo. Para cada integrante do grupo, descreva qual foi sua contribuição específica para este trabalho. Não vale dizer "ajudou igual". Deve ser específico, quem ajudou a escrever e em qual parte, quem ajudou a implementar, quem deu ideias e quais, quem ajudou a fazer a análise e qual a estratégia utilizada, etc.

4. **Apresentação (1 arquivo de vídeo (.mp4 ou .mkv) ou 1 arquivo de audio/podcast (.wav ou .mp3)):** Todos precisam explicar parte do trabalho. O limite de tempo da apresentação é de **30min**. Vocês devem cobrir todo o trabalho, explicando o problema, as estratégias consideradas, as que não deram certo, as que deram certo, alguns detalhes de como implementaram (estruturas de dados usadas, por exemplo), e por fim, uma intuição do porquê das estratégias funcionarem. Qualquer dúvida pergunte.

6. Como deve ser entregue

Apenas uma submissão por grupo. O trabalho deve ser entregue pelo Moodle[4] na forma de um único arquivo zipado (formato `.zip`) contendo documentação, implementação e makefile organizadas em um diretório chamado “tp”:

```
tp/
  |- tp.pdf
  |- <arquivos .c e .h>
  |- ...
  |- makefile
  |- <arquivo de vídeo ou arquivo de audio APENAS SE COM ELE O ARQUIVO ZIP NÃO
  ULTRAPASSAR 50MB>
```

Você deve compactar esse diretório e seu conteúdo em um arquivo único chamado `tp-grupo-<numero-do-seu-grupo>.zip`. Arquivos em outros formatos (RAR, por exemplo) que não `.zip` não serão aceitos. Por exemplo, para o grupo 1: `tp-grupo-1.zip`.

[4] <moodlepresencial.ufop.br>

Atenção: Se o seu arquivo compactado com a apresentação ultrapassar os 50MB limite, você pode subir apenas a apresentação para algum serviço de armazenamento em nuvem (por exemplo, o Google Drive de vocês), adicionar a referência na documentação de vocês e compartilhar comigo. O restante da entrega (implementação + documentação) deve ser entregue pelo moodle.

7. Avaliação

Este trabalho será avaliado de acordo com os seguintes critérios:

- O aluno seguiu a especificação do trabalho: formatos dos arquivos, nome dos arquivos, linguagem de programação, entrada e saída, makefile, etc.;
- Qualidade da documentação: completude, texto bem escrito e formatado, análise de complexidade e experimental bem explicadas, etc.;
- Qualidade da implementação: código indentado, bem comentado, variáveis legíveis, código modularizado, correto, alocação dinâmica correta, etc.

8. Observações gerais

- Leia esta especificação com cuidado;
- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiverem problemas para entender o que está escrito aqui: pergunte!
- Comece o trabalho o quanto antes;
- O trabalho deve ser implementado em linguagem C. Trabalhos em outras linguagens (C++, Java, Python, etc.) não serão aceitos;
- Siga exatamente as especificações de formato de entrega, compilação e execução descritas neste documento;
- Apenas um arquivo deve ser entregue, compactado e no formato `.zip`; Salva a situação em que o arquivo ficar muito grande – veja o texto em vermelho acima;
- Seu trabalho deve ser compatível com ambiente Linux;
- **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 o colegiado será devidamente informado para que providências possam ser tomadas.