Universidade Federal de Minas Gerais

Instituto de Ciências Exatas / Departamento de Ciência da Computação

Disciplina: Algoritmos e Estruturas de Dados III

Professores: Wagner Meira Junior, Marcos Augusto Menezes Vieira

Aluno: Caíque Bruno Fortunato

Documentação do Trabalho Prático 2 – The Force Awakens

1 - Introdução

Este trabalho tem como objetivo fixar os conceitos aprendidos no módulo de Paradgmas de Programação na disciplina Algoritmos e Estruturas de Dados III envolvendo programação dinâmica, algoritmo goloso e força bruta para a resolução de um problema chamado The Force Awakens.

The Force Awakens foi o nome dado ao problema que consiste na reconquista da galáxia após a morte de Darth Vader para ajudar a vitória o lado negro da força e o temível Lord Sith a atingir seu objetivo. Contudo, é necessário economizar recursos escolhendo quais planetas devem-se conquistar primeiro. Contudo, felizmente é possível traçar as rotas entre os planetas, já que é conhecido o número de planetas N e o número de distância entre eles, N + 1. Assim, é possível saber o ponto de saída, chegada, os planetas e a distância entr eles ao longo da jornada, permitindo assim saber quais irei reconquistar.

Como a nave utilizada para a conquista dos planetas, a Estrela da Morte III, está em fase de testes e, consequentemente, deve se deslocar o mínimo possível entre cada planeta, é necessário escolher quais os K planetas que devem ser reconquistados de modo que a maior sub-distância do percurso percorrido seja mínima.

Assim, no final da execução do programa é apresentado a maior sub-distância do percurso percorrido mínimo, resolvendo o trabalho proposto e ajudando o Lord Sith a atingir seu objetivo através do que foi proposto.

O algoritmo é importante para diversas outras aplicações já que conhecer a maior sub-distância de algum percusrso pode auxiliar um motorista a otimizar o uso de gasolina parando em menos postos, por exemplo e também diversas outras otimizações ao longo de um caminho ou algum problema que envolva paradas e distâncias fornecidas.

Finalmente, o estudo de Paradgmas de Programação permitirá o aprofundamento dos conhecimentos adquiridos ao longo da disciplina assim como a prática, importância e melhor entendimento sobre esses algoritmos.

Contudo, infelizmente o trabalho em questão será solucionado apenas por meio do Algoritmo Guloso, devido a dificuldade para a implementação dos outros métodos, o Força Bruta e Programação Dinâmica.

2 - Modelagem do problema do Algoritmo Guloso

O Algoritmo Guloso é uma técnica de algoritmos que ajuda a resolver problemas de otimização, como é o caso do The Force Awakens. O algoritmo sempre realiza uma escolha local que parece ser a melhor no momento na espera que seja a melhor escolha final, levendo a uma solução ótima. Vale lembrar que o Guloso nuca “se arrepende” de sua escolha, uma vez tomada a decisão, ela irá se manter válida até o fim, sem alteração.

Desse modo, a técnica de algoritmo guloso foi aplicada a fim de resolver o problema proposto no trabalho, assim como foi solicitado o emprego do mesmo como uma das três soluções. Uma das vantagens da utilização desse algortimo é pela razão de sua simplicidade e de sua implementação que geralmente não é muito complicada.

A ideia por trás da utilização do AG para resolver o trabalho consiste em:

Conhecendo o valor de N (número de planetas) e de K (número de conquistas) e um vetor com as distâncias entre os planetas de tamanho N + 1 o algoritmo realizará os seguintes passos:

- Encontrar a menor distância entre os planetas
- Somar a menor distância com o vizinho seguinte

Observe que desse modo será considerado que não iremos conquistar o planeta que possui o menor caminho, indo direto para o próximo, razão pelo qual somamos as distâncias.

Além disso, teremos uma posição a menos, já que somamos dois números, agregando a distância.

- O processo acima é realizado até que atingimos o valor de N – K, que justamente a quantidade de planetas que não serão conquistados.
- No final, sobrarão (N – K) + 1 distâncias, basta olhar qual é a maior distância entre elas.

2.1.1 - Exemplo:

Vejamos um exemplo, que está presente no Toy Example:

N = 10 e K = 5
Vetor de distâncias = 1, 8, 9, 8, 1, 7, 1, 1, 1, 1, 2

Teremos então, incialmente uma variável que será menor.

Vetor inicial: 1, 8, 9, 8, 1, 7, 1, 1, 1, 1, 2	Menor = 1, posição 0, agrego com o vizinho
Passo 1: 9, 9, 8, 1, 7, 1, 1, 1, 1, 2	Menor = 1, posição 3, agrego com o vizinho
Passo 2: 9, 9, 8, 8, 1, 1, 1, 1, 2	Menor = 1, posição 4, agrego com o vizinho
Passo 3: 9, 9, 8, 8, 2, 1, 1, 2	Menor = 1, posição 5, agrego com o vizinho
Passo 4: 9, 9, 8, 8, 2, 2, 2	Menor = 2, posição 4, agrego com o vizinho
Passo 5: 9, 9, 8, 8, 4, 2	Maior elemento: 9

2.1.2 - Implementação:

Para a implementação do algoritmo, várias ideias foram pensadas, como exemplo é possível citar a implementação de listas, vetores, entre outros. Contudo, a versão mais simplificada, de simples compreensão e de implementação rápida e fácil foi o uso de uma matriz para modelar o problema. Na matriz todos os passos no exemplo 2.1.1 serão calculados e cada passo estará em uma linha diferente, logo, a última linha fornecerá a resposta procurada.

Uma maneira um pouco diferente de raciocínio seria pensar que temos diferentes vetores, cada um representando um passo diferente, desde o incial até o passo final. Cada passo copia a solução do vetor anterior e calculaa uma nova solução, procurando o menor caminho e agregando com o próximo até que a solução seja a final.

Assim, toda a matriz é iniciada em 0, a primeira linha, que contém o vetor inicial é preenchida automaticamente, sem nenhuma operção, ou seja, há apenas uma cópia do vetor de distâncias para a primeira linha. A partir da próxima linha, os dados da linha anteior serão copiados, a linha será percorrida até achar a menor distância e o valor da coluna + 1 (próxima coluna) passará a ser as duas distâncias agregadas e a anterior será considerada -1. O -1 é sinal de que o espaço na linha está desocupado, ou seja, não iremos fazer mais nenhuma operação naquela posição porque ela simplesmente não será considerada existente.

2.1.3 - Por que o algoritmo funciona?

Uma versão bastante otimizada do força bruta, o algoritmo guloso não faz todas as combinações possíveis a fim de obeter a resposta correta. Pelo contrário, ele faz uma escolha e com ela vai até o final na espera do resultado ser obtido com sucesso. Assim, ele gasta menos tempo fazendo somente as contas necessárias.

A escolha que o algoritmo irá tomar a decisão local de encontrar a menor distância entre dois planetas, na esperança que a decisão global seja o problema que queremos solucinar. Para isso, o objetivo do algoritmo é tantar minimizar distâncias grandes, somando os menores valores até a condição de parada, já discutida no item acima.

Ou seja, com cerca de N – K escolhas o algoritmo será capaz de fornecer a escolha correta e adequada, sem precisar realizar mais cálculos e sendo melhor que o algoritmo de Força Bruta, por exemplo.

3 – Análise de complexidade do Algoritmo Guloso

3.1 - Complexidade de tempo

O algortmo guloso possui apenas uma função que realiza o cálculo desejado para a resolução do problema. Nessa função encontramos alguns laços de repetições que realizam cálculo através de uma matriz.

Analisando os laços mais importantes, podemos analisar o momento de zerar a matriz, que possui complexidade quadrática $O(n^2)$, já que teremos n x n, onde n é o número de linhas e colunas que podem ser iguais no pior caso. Além disso, todas as cópias de linhas realizadas também irá resultar emcomplexidade $O(n^2)$ já que colunas de tamanho n serão transferidas para a linha abaixo, operação que será realizada n vezes.

Quando o preenchimento da matriz pela primeira linha é realizado, a complexidade é O(n), onde n é o número de elementos da coluna. Já nas operações, no loop seguinte, estão sendo realizados vários cálculos, entre eles comparações, por exemplo, que é O(1), cópias de linhas, que como já citado é O(n), assim como a análise de elementos em uma linha. Em todas as operações realizadas na matriz, todas irão seguir esse mesmo raciocínio e etapas (cópia e análise, operações nas linhas, entre outros).

Logo, a complexidade será $O(n^2)$, já que a soma de $O(n) + O(1) + O(n^2)$ irá gerar a complexidade quadrática.

3.2 - Complexidade de espaço

A complexidade espacial do algoritmo guloso é $O(n^2)$, isso porque é necessária a construção de uma matriz n x n, onde n é o número de linhas e colunas, respectivamente. Como necessitamos da alocação da matriz para que as contas sejam realizadas, temos então a complexidade quadrática em questão.

4 – Análise experimental do Algoritmo Guloso

Para a análise experimental foram observadas, antes de tudo, as seguintes restrições:

- N está entre 0 e 500
- K está entre 0 e 250

Por fim, foram realizados diversos testes após a finalização do trabalho com os casos de teste disponibilizados no Moodle e, principalmente, com casos de testes criados através de um programa para a geração de casos de testes gerado para a construção do tópico em questão. Para isso, foram tomados valores de N e K variados.

Os casos de testes foram feitos em uma máquina com 3GB de memória, processador Intel Core 2 Duo x86 com sistema operacional Linux com distribuição Ubuntu 15.10 em seu compilador nativo, GCC.

Pada cada entrada, houveram 10 execuções diferentes e o tempo foi calculado através do comando time ./lista ./[Nome do arquivo de teste] através do uso da variável user, que calcula o tempo que o programa gastou para calcular as entradas disponibilizadas, ignorando o tempo demorado para que o usuário entre com os dados. Os dados das 10 execuções de cada arquivo foram guardadas para que fosse calculada a média e que o resultado final fosse obtido.

Os dados gerados foram os seguintes:

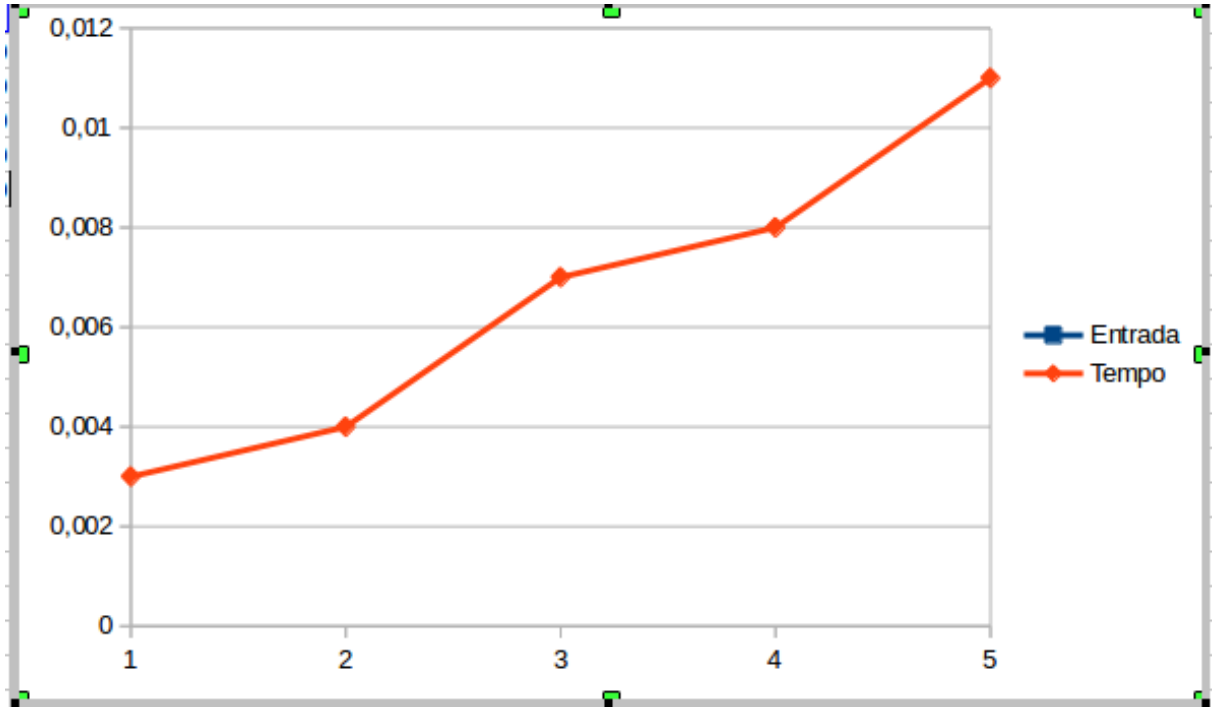
1) Fixando K = 50 e variando N de 100 até 500 com intervalo de 100 unidades, temos:

Entrada (N)	Tempo(s)
100	0,003
200	0,004
300	0,007
400	0,008
500	0,011

2) Fixando N = 250 e variando K de 50 até 250 com intervalo de 50 unidades, temos:

Entrada	Tempo
N = 50	0,003s
N = 100	0,005s
N = 150	0,006s
N = 200	0,006s
N = 250	0,008s

Como os gráficos são relativamentes semelhantes, tomaremos somente o gráfico da primeira tabela para observação, de modo que temos:



O resultado é bastante semelhante com a análise de complexidade, que só vai aumentando a medida que a entrada também aumenta, devido a alocação e operações na matriz.

5 – Conclusão

A implementação do trabalho foi um pouco complicada, com tempo relativamente curto e devido a alguns problemas de compreensão e assimilação do conteúdo de paradigmas de programação infelizmente não foi possível entregar os códigos dos algoritmos de Força Bruta e Porgramação Dinâmica, já que o primeiro a recursão não funcionou corretamente e, consequentemente, ficou ainda mais complicado a construção do PD.

No entanto, a implementação do Algoritmo Guloso, cujo qual foi tema da documentação, ocorreu sem muitas dificuldades. Contudo, o fator que pode ter complicado um pouco a resolução do algoritmo foi justamente o fato de encontrar uma solução que atendesse os requesistos de ser um guloso e que funcionasse corretamente para a maioria dos casos de teste.

Mesmo o trabalho estando incompleto a implementação e entrega do Algoritmo Guloso foi importante para que fosse possível treinar conceitos de paradigmas de programação. Mais do que isso, mesmo sem a entrega dos demais algoritmos todo o tempo analisando como eles seriam, funcionariam e seriam implementados foi importante para o conhecimento e fixação de conceitos importantes para possíveis trabalhos próximos ou para alguma implementação ao longo da carreira de computação ou acadêmica. Esse fato se torna ainda mais importante em uma discplina que contém aulas teóricas, a prática se torna indispnesável para a fixação do conteúdo.

Por fim, problemas como o The Force Awakens serão importante também para o estudo de NP completo, e também para a compreensão de aplicações importantes para encontrar uma otimização em meio a um problema parecido, que pode surgir na vida de qualquer programador ou profissional da área de computação e afins.