Trabalho Prático 2

Algoritmos 1

Lucas Paulo Martins Mariz

2018055016

Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brasil

lucaspaulom@hotmail.com

Resumo: *O trabalho a seguir se trata da aplicação de conceitos e algoritmos relacionados ao estudo de paradigmas de programação dinâmica e gulosa bem como suas utilidades no mundo real. Além disso, questões como a discussão de melhores casos, minimizações de custos e maximizações de pontuações, complexidade e performance são tratados com o intuito de concretizar e desenvolver uma capacidade analítica nos alunos para com o tema.*

Abstract: *The following work deals with the application of concepts and algorithms related to the study of programming and greedy paradigms, as well as their real world utilities. In addition, issues such as better case discussion, cost minimization, and maximization of scores, complexity, and performance are described with a view to realizing and developing students' analytical skills for the topic.*

1. Introdução

Foi proposto pela disciplina de Algoritmos 1 o desenvolvimento de abordagens gulosas e dinâmicas com o intuito de responder corretamente problemas de maximização. Mais especificamente, o problema em questão consiste em dado um teto de gastos (N) e um conjunto de ilhas (M), com suas respectivas pontuações e preços, deve-se responder qual é a maior pontuação possível e a quantidade de dias que duraria uma viagem hipotética à essas ilhas, em que escolher ir para uma ilha implica em um dia de viagem.

Sendo assim, o trabalho se dividiu em dois eixos específicos:

- Utilizando um algoritmo guloso, determinar a maior pontuação possível no qual **podem haver repetições de ilhas** (ficar mais de um dia na ilha) e a quantidade de dias que durará a viagem, respeitando o teto de complexidade de *O(NlogN)*.
- Utilizando programação dinâmica, determinar a maior pontuação possível no qual **não podem haver repetições de ilhas** (ficar mais de um dia na ilha) e a quantidade de dias que durará a viagem, respeitando o teto de complexidade de *O(M * N)*.

Além disso, a padronização das entradas constitui um fator de suma importância para a veracidade da saída. Na primeira linha do arquivo de entrada lê-se o teto de gastos N e a quantidade de ilhas M. Nas M linhas seguintes são encontrados o custo de ir para tal ilha e a sua respectiva pontuação, fator que queremos maximizar na resposta final.

2. Instruções de Compilação e Execução

O programa foi desenvolvido na linguagem C, compilado pelo GCC versão 7.4.0 em um sistema operacional Ubuntu 18.04. Todo o código foi executado em meu notebook cujas especificações principais estão descritas abaixo:

- Intel Core i5-7200U CPU 2.5GHZ x 4
- Nvidia G-Force GTX 840M
- 8GB RAM
- SSD Kingston A400 240GB
- Ubuntu 18.04.2 LTS

Portanto, recomenda-se que a execução do código ocorra em um sistema Ubuntu versão 14 ou superior contendo pelo menos a versão 7 do GNU C Compiler (GCC). Foi utilizado um makefile que se encarrega de todas as operações de compilação dos arquivos '.c'.

A execução do projeto aguarda um argumento (argv) indicando o arquivo contendo os dados de entrada e deve seguir o seguinte padrão:

./tp2 path/to/file.txt

3. Implementação

A implementação do trabalho envolveu uma estrutura de dados simples, a qual armazenava as informações referentes as ilhas. Através dessa estrutura, todas as manipulações e cálculos necessários foram feitos, de forma que toda a informação estivesse armazenada em um único ponto. Nas próximas seções serão discutidas as particularidades de cada abordagem bem como os métodos utilizados para a implementação dos algoritmos.

3.1. Algoritmo Guloso

O algoritmo guloso mencionado anteriormente segue uma ideia bastante simples para encontrar a melhor solução. Primeiro, armazenamos na estrutura de dados mais uma variável referente ao custo por pontuação (divisão entre os dados) de cada ilha. Logo após, ordenamos o vetor de ilhas levando em consideração essa nova variável utilizando um *Mergesort*, de forma que o teto de complexidade de *O(MlogM)* seja respeitado. Com o vetor de ilhas ordenado, começamos o processo de apuração de dados. A partir da primeira ilha do vetor, calculamos qual é o máximo de dias que a pessoa pode ficar nela dado o seu orçamento limitado (a solução gulosa permite a repetição de ilhas). Diminuímos o custo multiplicado pela quantidade de dias do orçamento da pessoa, atualizamos a quantidade de dias da viagem e vamos para a próxima ilha, executando esse mesmo processo até o final do vetor.

Algumas ocorrências podem acontecer durante a execução, como por exemplo uma ilha ser tão cara que não poderá ser inserida nenhuma vez na solução, ou uma ilha tão barata que poderá ser inserida diversas vezes.

O algoritmo tem suas limitações e nem sempre terá a resposta ótima esperada justamente por sua abordagem que sempre busca o menor custo por pontuação e maximiza esse custo

dentro do orçamento. Outro fato a ser discutido é que o algoritmo entra em loop caso o custo de ir para uma ilha seja 0, fazendo com que a mesma seja inserida infinitamente no programa de viagem.

3.2. Programação Dinâmica

Para a solução dinâmica foi utilizado uma variação do problema da mochila (*knapsack problem*). O objetivo de maximizar a pontuação dado um orçamento limitado pode ser adaptado em uma mochila que pode carregar N, representando o teto máximo de gastos, na qual queremos analisar a possível inserção de M ilhas de forma a encontrar a solução ótima.

Esse método respeita o fato de não poder haver repetições de ilhas, pois, quando "inserida na mochila", a ilha não é inserida novamente. Como resultado desse processo, teremos uma matriz M * N, na qual a melhor pontuação possível dada as entradas se encontra no elemento da última linha e na última coluna.

Para encontrar a quantidade de dias de duração da viagem é feito um processo de *backtracking* na matriz resultante do *knapsack*. Esse processo constitui em analisar um elemento da matriz e o elemento logo acima dele na mesma coluna. Caso eles sejam diferentes, contamos mais um dia e vamos para a coluna equivalente a subtração da coluna atual com o custo correspondente a ilha da linha atual. O número de execuções é diretamente proporcional a quantidade de ilhas, não afetando o teto de complexidade estabelecido de *O(M * N)*

4. Análise Experimental

Uma massiva quantidade de dados gerados foi considerada na análise que se segue. Foram realizados diversos testes com o intuito de checar a veracidade dos dados. Os dados obtidos com os experimentos realizados estão armazenados no repositório do projeto no *Github* (https://github.com/LucasPMM/Dynamic_Greedy) e na própria pasta do projeto.

Além disso, todo o trabalho foi devidamente testado em diversos ambientes, tanto locais quanto remotos, com o intuito de adquirir os melhores resultados possíveis. Ademais, a utilização do depurador *Valgrind* foi fundamental no processo de correção de erros e prevenção de *memory leaks*

4.1. Testes

Utilizando alguns scripts, foram gerados diversos arquivos de entradas, variando tanto quanto o orçamento disponível (N) quanto o número de ilhas (M). Foram realizados testes para valores de M variando de 4 a 400 de 4 em 4 unidades, sendo que cada teste foi executado 20 vezes com o intuito de obter um valor satisfatório para a média.

Inicialmente, fiz os testes na minha máquina local, no entanto, as constantes trocas de contexto existentes no processador prejudicaram os resultados finais. Ações como navegar na internet ou estar com outros programas abertos na máquina no momento da execução do trabalho implicou em inconstâncias nos resultados e na produção de diversos outliers (pontos fora da curva) na contagem do tempo de execução. Com o intuito de

contornar essa situação, foi simulado um ambiente virtual *Ubuntu* na plataforma *Codenvy* e os mesmos testes foram refeitos.

4.1. Analise de Complexidade

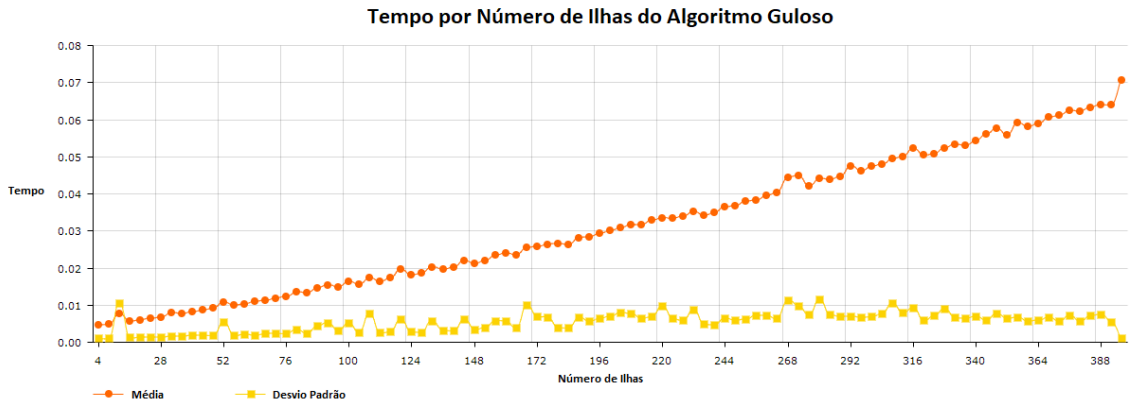

Figura 1: Gráfico Tempo (em milissegundos) por Número de ilhas do Algoritmo Guloso

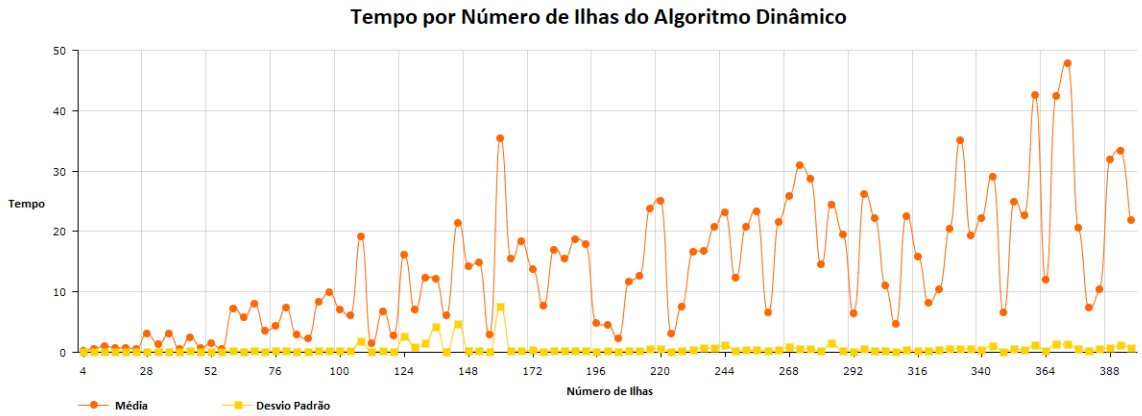

Figura 2: Gráfico Tempo (em milissegundos) por Número de ilhas do Algoritmo Dinâmico

Analisando as saídas de ambos os algoritmos, disponível no diretório do trabalho e no projeto no Github, podemos ter várias conclusões e, por conseguinte, responder algumas perguntas. Em primeiro lugar, percebe-se a grande diferença das respostas. Os resultados obtidos por cada abordagem são diferentes e isso não implica que um algoritmo está certo e o outro errado, apenas significa que métodos diferentes possuem objetivos distintos. Em geral, as pontuações e duração da viagem obtidos pela solução gulosa são maiores que as da programação dinâmica, como podemos ver nos gráficos da figura 3 e 4.

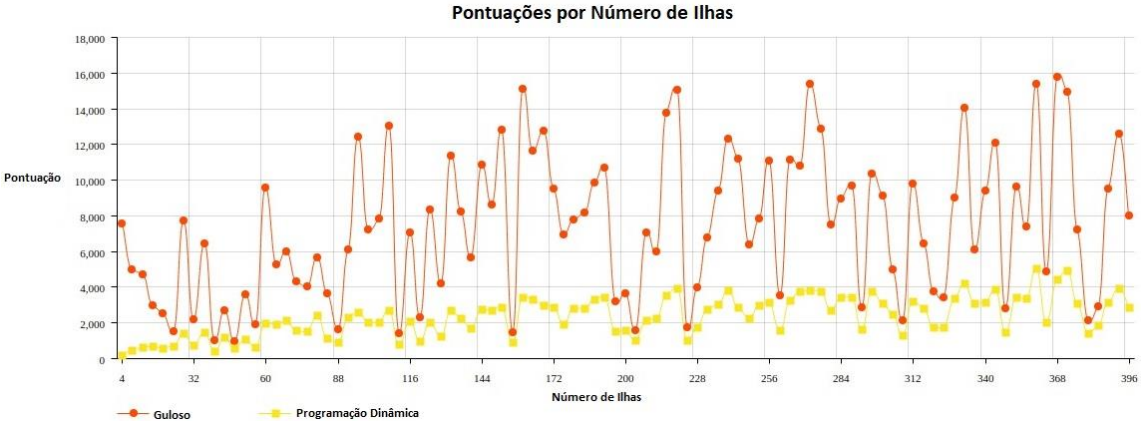

Figura 3: Gráfico Pontuações por Número de Ilhas dos Algoritmos Guloso e Dinâmico.

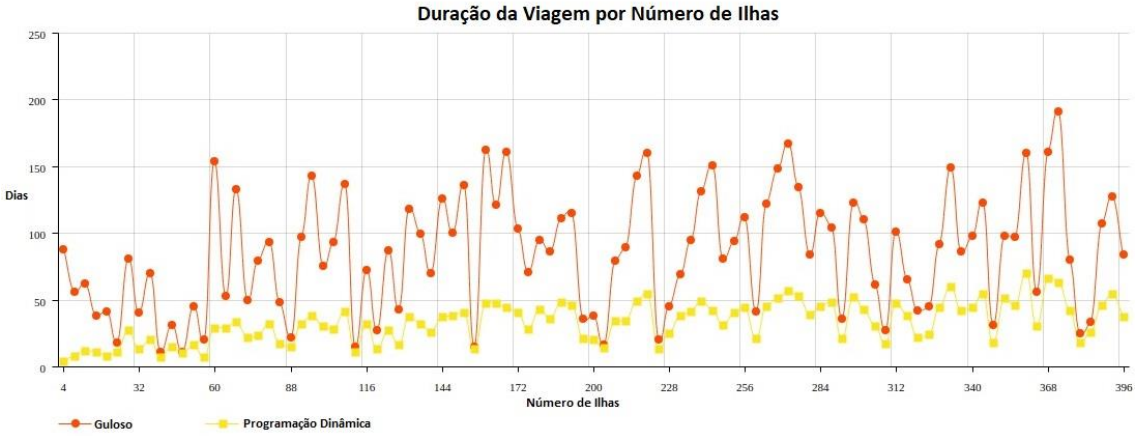

Figura 3: Gráfico Duração da Viagem (em dias) por Número de Ilhas dos Algoritmos Guloso e Dinâmico.

- Caso o objetivo seja aumentar o tempo de estadia em cada ilha ao máximo ou maximizar a pontuação final, deve-se escolher o algoritmo guloso, pois nessa abordagem a repetição de ilhas é permitida.

- Caso o objetivo seja conhecer mais lugares, deve-se escolher o algoritmo de programação dinâmica, pois o mesmo consegue uma combinação que cobre mais ilhas.

Sendo assim, para resolver o problema no qual podem haver repetições de ilhas temos que a solução gulosa atende melhor, maximizando a quantidade de dias que se pode ficar na mesma localização. Por outro lado, a resposta nem sempre é a melhor possível, sendo que o tratamento realizado pela programação dinâmica é capaz de fornecer resultados bastante satisfatórios em casos sem repetição de ilhas.

- **Algoritmo Guloso:**

Como mencionado anteriormente, a solução gulosa ordena o vetor de ilhas pelo custo por pontuação e maximiza a quantidade de dias que se pode ficar em cada ilha dado o orçamento limitado.

Vamos a uma pequena prova do funcionamento da solução gulosa. Suponha que o conjunto solução do nosso problema seja $S = [I_1, I_2, ..., I_k]$. Agora seja $S' = [L_1, L_2, ..., L_k]$ uma solução diferente e melhor que S. Sendo assim, durante a execução do algoritmo, S' teria inserido uma ilha j em seu conjunto solução, a qual possui um custo por pontuação

menor do que àquelas inseridas na solução S. Contudo, caso a existência de tal ilha j fosse verdadeira, o nosso algoritmo guloso a teria incluído no conjunto solução uma vez que ela atende as especificações dele. Chegamos a uma contradição, logo S' não pode existir e S é a nossa solução ótima da abordagem gulosa.

Mesmo com a prova acima, pode-se dizer que algoritmo guloso é valido / contido e funciona dentro do seu escopo e regras, ou seja, ele é ótimo dada a sua abordagem de escolher ou não ilhas levando em consideração os dados de entrada. No entanto, existem casos em que a solução não será a melhor possível, como podemos ver no exemplo abaixo:

6	2
3	5
4	8

O algoritmo guloso encontraria o valor 8 para a pontuação, passando apenas por uma ilha. No entanto, podemos analisar os dados e ver que a solução ótima é a pontuação de 10, ficando dois dias na mesma ilha. Tal fato não acarreta na falsidade do algoritmo, apenas nos mostra que existem casos em que a solução ótima não pode ser encontrada por essa abordagem.

A complexidade de tempo do *Mergesort* é *O(MlogM)* em todos os casos e a complexidade de espaço é *O(M),* devido ao fato de utilizar vetores auxiliares para fazer as interações. Com os dados ordenados, temos um custo de *O(M)* para calcular o tempo de duração da viagem em dias e a pontuação adquirida. Em suma, podemos afirmar que a solução gulosa é dominada superiormente por **O(MlogM) em relação ao tempo** e **O(M) em relação ao espaço.** Esse fato pode ser observado na figura 1, a qual representa uma tendência crescente do tempo em relação a quantidade de ilhas presentes na entrada do programa e um desvio padrão consideravelmente constante.

- **Programação Dinâmica:**

No algoritmo de programação dinâmica foi utilizado como referência o problema da mochila com o intuito de maximizar a pontuação final. Dado que temos como entrada um conjunto de ilhas com suas respectivas pontuações e preços bem como o valor total a ser gasto como a "quantidade de peso" que a mochila é capaz de levar, resolvemos o problema em pequenos casos, calculando a melhor solução para 1 ilha até M. Sendo assim, essa abordagem se torna ótima para o nosso trabalho uma vez que a repetição de ilhas não é permitida.

Seja *OPT(i, w)* o lucro máximo de um subset de itens 1 a i, com um limite de gasto w. Caso *OPT(i, w)* não selecione o item i, então OPT(i, w) selecionará a melhor solução de 1 até i – 1 usando w como teto de gasto. Caso *OPT(i, w)* selecione o item i, pegamos o custo v_i e recalculamos o teto de gastos para $w - w_i$, fazendo o *OPT(i, w)* selecionar a melhor solução de 1 até i – 1 usando o novo teto de gastos. Com isso, chegamos a seguinte equação de Bellman:

$$OPT(i,w) = \begin{cases} 0 & \text{Se i = 0} \\ OPT(i-1,w) & \text{Se } w_i > w \\ \max\{\ OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\ \} & \text{Caso contrário} \end{cases}$$

Podemos observar a figura 2 referente ao gráfico das médias dos tempos pela quantidade de ilhas do algoritmo de programação dinâmica. À primeira vista o gráfico apresenta uma tendência crescente, no entanto, esse fator crescente é bem inconstante. Esse fato é perfeitamente explicado quando levamos em consideração que a complexidade esperada de um algoritmo que usa a ideologia do problema da mochila seja dominado superiormente por *O(M * N)*, sendo M e N gerados aleatoriamente pelo script de testes, como fora pedido na descrição do projeto, fato que distorce o gráfico dependo da grandeza e discrepância dos dados gerados.

O algoritmo de programação dinâmica utilizado no trabalho é dominado por uma **complexidade de tempo de *O(M * N)*** para interpretar todas as entradas e para encontrar o número de dias da viagem e uma **complexidade de espaço também igual a *O(M * N),*** devido ao fato de armazenar os valores ótimos em uma matriz de tamanho M * N.

5. Conclusão

Com a realização dos testes e análise dos dados, é notável que diversos problemas reais podem ser solucionados utilizando os diferentes paradigmas de programação existentes e, dependendo do objetivo buscado, diferentes abordagens devem ser tomadas. Além disso, é necessário destacar que alguns erros de medida podem ter ocorrido devido a aleatoriedade dos testes gerados e pela troca de contexto existente no processador do computador, fazendo com que processos percam ou ganhem prioridade na execução, oscilando o tempo final de execução do trabalho. No entanto, essa situação foi controlada em partes ao executar o trabalho em um sistema remoto livre, em grande parte, das trocas de contexto locais.

Além disso, a distinção dos resultados nas abordagens utilizando o algoritmo guloso e programação dinâmica foram de suma importância para as discussões das seções anteriores e tal fato nos leva a concluir que não se pode afirmar que um algoritmo é melhor que o outro, apenas podemos inferir que existem objetivos diferentes em seus procedimentos e métodos.

Sendo assim, a análise qualitativa de um problema é de fundamental importância tendo em vista a procura pela melhor solução em termos de complexidade e velocidade da execução. Pode-se afirmar que o trabalho teve grande importância no processo pedagógico da disciplina, colocando em pratica e provando conceitos vistos em sala, bem como pode-se dizer que os objetivos foram cumpridos e as dificuldades superadas uma vez que as metas foram alcançadas.

6. Bibliografia

Chaimowicz, L. e Prates, R. “Mergesort”, pdf disponível no Moodle da turma de Estruturas de Dados 2019/1 da UFMG. Acesso em: 25 out 2019.

Ziviani, Nívio. (2011), Projeto de Algoritmos com Implementações em Pascal e C, Cengage Learning, 3° Ed.

Slides da disciplina de Algoritmos 1 da professora Jussara.