Trabalho Prático de Algoritmos 1

Roberto Gomes Rosmaninho Neto

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

robertogomes@dcc.ufmg.br

1. Introdução

O problema proposta por esse trabalho foi modelar e implementar uma solução que define o melhor roteiro de viagem para Luiza e suas amigas. O problema possui as seguintes características: Dado um conjunto de possíveis destinos cada destino possui um custo fixo e a sua nota de preferência dada pelo grupo de amigas, além disso há um valor máximo de dinheiro que o grupo pode investir com os custos da viagem. Assim, dadas essas características, 2 soluções foram propostas visando solucionar o caso onde a preferência pela maior soma de pontos possível é almejada, podendo ficar mais de um dia em cada ilha, ou seja, poderá haver repetição nas escolhas e o caso em que se deve gastar o máximo possível não podendo ficar mais de um dia em cada ilha, ou seja, não é permitida a repetição de escolhas.

A primeira solução é baseada no paradigma de programação Gulosa, onde a melhor solução local é sempre escolhida na esperança de se encontrar a melhor solução global ao fim da execução do algoritmo.

A segunda solução, por sua vez, é baseada no paradigma de Programação Dinâmica, onde um problema maior é dividido em sub-problemas menores e cada um é solucionado individualmente, mas com cada resultado sendo guardado para evitar cálculos desnecessários.

A primeira solução desse programa possui complexidade $O(mlogm)$ devido ao custo de ordenação do vetor de custo benefício que será detalhado posteriormente, sendo n o número de ilhas. A segunda solução, por sua vez, possui complexidade $O(mn)$, devido ao cálculo de uma matriz de resultados que também sera explicada posteriormente, nesse caso m também representa o número de ilhas e n o valor total disponível para ser gasto na viagem. A complexidade geral do programa é a mesma do segundo caso, devido sua maior complexidade e sem nenhuma outra operação de maior valor assintótico realizada antes ou depois da implementação das soluções.

A entrada do programas deveria ser feita via argumento, pela linha de comando, passando o nome do arquivo de entrada logo após o arquivo executável. A saída do programa por sua vez é dada na saída padrão do usuário. As caraterísticas de relevantes sobre o arquivo de entrada e sobre a saída serão abordadas posteriormente neste documento.

2. Implementação

Este trabalho prático foi implementado na linguagem C++, utilizando as boas práticas de orientação à objetos. Os Ambientes de testes utilizado para executar o código foram um Macbook Air Core i5, dual core de 1.8Ghz, 8GB de RAM e SSD de 128GB e um Dell OptiPlex 7040M com processador i7vPRO. Os compiladores utilizados para

testes foram o Apple LLVM version 10.0.1 (clang-1001.0.46.4) e o GCC 5.4.0 da GNU Compiler Collection.

A implementação desse programa foi realizada utilizando apenas 3 arquivos: main.cpp, lib.h e lib.cpp, sendo todas as operações e funções declaradas no segundo arquivo e implementados no terceiro. Assim, no arquivo main.cpp apenas foram realizadas as operações de leitura de arquivo e escrita na saída padrão, além das chamadas de funções quando necessárias. As principais estruturas de dados utilizadas foram lista encadeada, utilizando Vector, e uma tupla, utilizando Tuple, ambas da biblioteca padrão de c++.

2.1. Funções

Esse programa possui apenas uma classe: Lib. Esta é responsável por armazenar as informações da entrada e da saída, além realizar realizar a implementação das duas soluções para o problema dado.

As funções presentes na classe Lib e suas descrições são as seguintes:

- ***Lib***: Essa função inicializa um objeto com algumas variáveis que são usadas como variáveis globais durante o decorrer do programa, além disso, essa função recebe por parâmetro o números de ilhas a serem consideradas no problema e o valor máximo que as amigas podem gastar nas viagens. Inicializa, também, o número de dias e os pontos totais que serão resultado das operações posteriores, essas variáveis começam com o valor 0.
- ***~Lib***: Destrutor padrão.
- ***set_island***: Essa função recebe um vetor de tuplas por parâmetro e o atribui à variável *_islands* que é usada dentro da classe como a entrada do problema.
- ***set_greedy***: Essa função é responsável por implementar um algoritmo guloso para solucionar o primeiro problema. Primeiramente o valor disponível é atribuído à uma variável temporária e o vetor *_islands* é ordenado utilizando a função *stable_sort* da biblioteca padrão do *C++* que é uma implementação do algoritmo *Merge Sort*. A ordenação foi realizada de forma crescente de acordo com o custo benefício de uma ilha, ou seja, seu valor dividido pela sua pontuação. Então, o vetor é percorrido de tal maneira que dado uma posição avalia-se se o valor da ilha representada por ela é menor que o valor disponível para ser gasto, caso seja verdadeiro esse valor é diminuído da variável temporária, o contator de dias de viagem é incrementado e a condição é avaliada novamente até que seja falsa. Quando a condição for falsa a posição posterior deve ser avaliada até que todas a ilhas tenham sido considerados ou que o dinheiro disponível não seja suficiente para pagar nenhuma outra.
- ***set_dynamic_programming***: Essa função é responsável por implementar um algoritmo baseado em Programação dinâmica para solucionar o segundo problema. Ela pode ser dividida em duas partes, a primeira responsável por calcular a pontuação máxima e a segunda por calcular o número de dias referente a pontuação calculada anteriormente. Na primeira parte, um matriz[M][N] é criada, sendo N o valor disponível para ser gasto e M o número de ilhas, esta deve ser povoada com 0's na primeira linha e primeira coluna. Além disso, matriz é povoada seguindo seguinte critério, cada possibilidade é inclusão ou não de uma ilha na solução ótima é avaliada e armazenada, dessa forma não há a possibilidade que

se calcular mais de uma vez alguma operação. Uma ilha só é adicionada à solução caso a sua pontuação contribua mais para a melhor solução do que a alternativa anteriormente considerada. O cálculo da matriz é realizado por meio de iterações para se evitar custos adicionais de espaço e a possibilidade de se calcular alguma possibilidade mais de uma vez. Assim, ao final deste calculo o resultado da ultima operação indica o valor máximo de pontos somados das ilhas que o grupo de Luiza poderá visitar. A segunda parte da solução é caminha sobre a matriz buscando alterações entre as linhas de uma mesma coluna, pois caso haja a alteração a ilha sendo analisada foi considera melhor que a ilha anterior, logo ela pertence a solução ótima e para caminhar para a próxima análise, o valor da ilha é subtraído do 2 índice e o primeiro é decrementado até que um dos valores seja menor que 1. A cada ilha considerada um dia é incrementado na variável global que será retornada em uma função posterior.

- ***get_days***: Retorna o número de dias calculados pela implementação gulosa para o primeiro problema.
- ***get_total_points***: Retorna a pontuação máxima adquiririda por meio do alguritmo guloso para solucionar o primeiro problema, ou seja, a soma de todas as ilhas visitadas pelo grupo durante a viagem.
- ***get_days_pd***: Retorna o número de dias calculados pela programação dinâmica para a resolução do degundo problema.
- ***get_total_points_pd***: Retorna a pontuação máxima adquiririda por meio do alguritmo guloso para solucionar o primeiro problema, ou seja, a soma de todas as ilhas visitadas pelo grupo durante a viagem.

2.2. Fases de Implementação

A implementação do programa ocorreu em apenas duas fases. A primeira responsável pela implementação das operações necessárias para solução do problema e retorno destas. A segunda responsável pela a leitura de entrada, criação do objeto da classe *Lib* e as chamadas de funções do objeto, além da escrita de saída.

2.3. Implementação das Operações

Essa fase foi basicamente necessária para implementar as soluções do problema. Todas as operações implementadas correspondem as funções descritas acima.

O paradigma de programação gulosa foi escolhido devido a possibilidade de se repetir o número de ilhas escolhidas, ou seja, realizar uma operação mais de uma vez é uma condição possível para a solução. No entanto, esse algoritmo não produz solução ótima, visto que no caso de duas ilhas que proporcionam o mesmo custo benefício a mais cara pode ser escolhida e o objetivo de maximizar o valor gasto pode não ser atingido. Isso ocorre pois as ilhas são ordenadas apenas pelo seu custo benefício, lidar com casos especiais como esse acarretaria no aumento da complexidade da solução e possível adaptação do paradigma utilizado.

Na segunda solução o paradigma da programação dinâmica foi utilizado pois uma das restrições do problema é a proibição de se passar mais de um dia em um ilha, ou seja, operações repetidas não poderiam ser realizadas e o paradigma de programação dinâmica é conhecido por calcular todas as possibilidades de uma solução apenas uma vez e sempre armazenando os resultados obtidos. Essa implementação não é ótima também,

principalmente em espaço, visto que hoje existem implementações que utilizam apenas um vetor e não uma matriz, no entanto ainda é eficiente visto que o escopo deste trabalho é pequeno. Esta implementação aplica a seguinte equação de Bellman:

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

Figura 1. Equação de Bellman para Programação Dinâmica implementada

2.3.1. Leitura, Escrita e Chamadas de Operações na Função Principal

A leitura do arquivo de entrada é realizada utilizando a biblioteca *fstream*. O objeto da classe *Lib* é criado logo após a leitura da primeira linha que contém o número de ilhas e o valor máximo a ser gasto na viagem. Após a criação do objeto, um *vector* do tipo Tupla de um número real e dois inteiros é inicializado e logo após preenchido a medida que cada linha do arquivo de entrada é lido. O primeiro elemento da tupla é o resultado da divisão do segundo elemento pelo terceiro, ou melhor, do primeiro elemento lido, pelo segundo.

Por fim, o vetor é passado pelo objeto por meio da função *set_islands* e as duas funções responsável pelas soluções dos problemas são chamadas. Então é escrito na saída o resultado das operações, estes são obtidos por meio de funções que retornam os valores das varáveis globais do objeto criado.

2.4. Entradas e Saídas

O programa deve receber o nome do arquivo de entrada via linha de comando logo após o executável como mostra o exemplo a seguir:

```
$ ./tp2 input.txt
```

O arquivo de entrada deve conter dois inteiros na primeira linha: N e M. O primeiro representando o valor máximo disponível para ser gasto na viagem e o segundo representando o número de ilhas possíveis. As próximas M linhas possuem dois inteiros D e P, o primeiro representa o custo por dia na ilha e o segundo a pontuação da ilha dada pelo grupo de amigas.

O arquivo de saída deve conter 2 linhas cada uma também com dois inteiros. A primeira linha representa a solução do algoritmo guloso e a segunda da programação dinâmica. O primeiro inteiro de cada linha representa a pontuação máxima e o segundo o número de dias máximo de viagem.

3. Análise Experimental

A principal requisição para a implementação desse programa é a primeira solução possua complexidade assintótica $O(mlogm)$ e para a segunda solução $O(m * n)$. Dessa forma, será mostrado a seguir que todas que as soluções implementadas seguem essas restrições.

3.1. Premissas

Os resultados analisados nesse documento se referem aos testes realizado no OptiPlex 7040M com processador i7vPRO, o programa foi compilado com o GCC 5.4.0 no sistema operacional Ubuntu 16.04.10.

3.2. Metodologia

Os arquivos de entrada foram gerados de modo aleatório por um programa em *Python* seguindo as restrições presentes nesse documento. A quantidade de ilhas variou de 100 em 100, entre 100 e 1000. O valor de cada ilha é aleatório e aumenta de acordo com a ordem de grandeza do número de ilhas.

Cada entrada do programa foi executada 20 vezes para que a média e o desvio padrão dessas entradas pudessem ser analisados, visando realizar essa atividade de forma automática utilizou-se um *script* em *bash*.

O tempo de execução do programa foi contato em Microssegundos e os gráficos representando o Tempo pelo número de ilhas para cada entrada.

3.3. Análise de Complexidade

3.3.1. Tempo Total

A complexidade assintótica total do programa é $O(m * n)$, visto que sua operação mais custosa é realizada na segunda solução que na implementação da programação dinâmica calcula todos os valores de uma matriz que 2 dimensões. O custo de tempo para cada entrada pode ser visualizado a seguir:

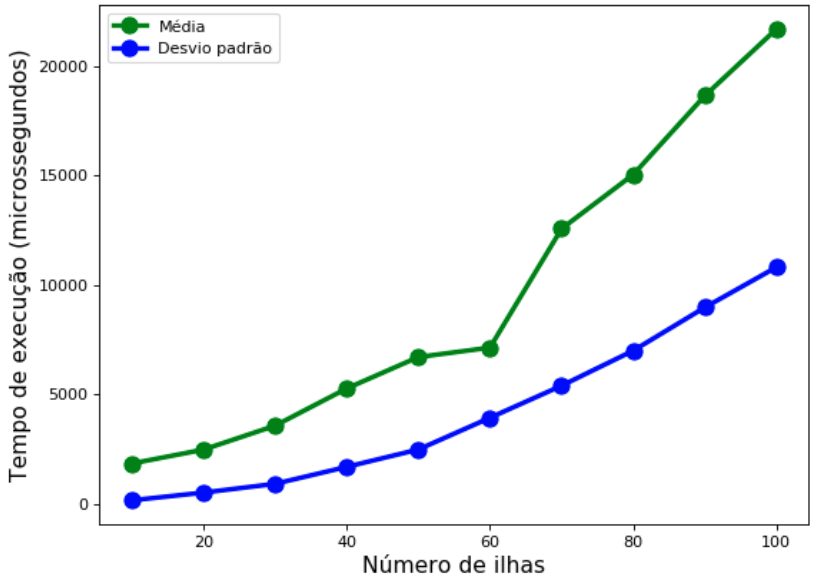

Figura 2. Tempo de Execução do Programa

3.3.2. Tempo do Algoritmo Guloso

A complexidade do algoritmo guloso implementado na primeira solução é majoritariamente resultado da ordenação realizada no vetor de tuplas das ilhas, utilizando o primeiro elemento da tupla como chave de comparação, este elemento é o custo benefício de cada ilha. O algoritmo utilizado para a ordenação foi o Merge Sort implementado na função *std::stable_sort* da biblioteca padrão do C++. É trivial dizer que o Merge Sort possui custo assintótico contante de $\Theta(m * log(m))$, sendo m, nesse caso, o número de ilhas. Após a ordenação há apenas no máximo n operações no vetor para calcular quais ilhas serão visitadas e a soma de seus pontos. A complexidade desta solução pode ser observada no gráfico abaixo:

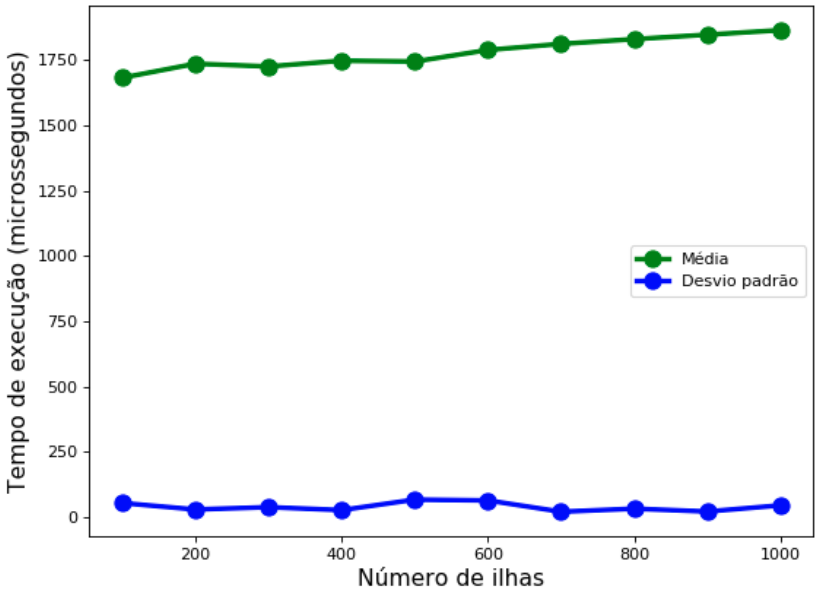

Figura 3. Tempo de Execução do Algoritmo Guloso

3.3.3. Tempo da Programação Dinâmica

A complexidade da implementação da programação dinâmica na segunda solução é majoritariamente resultado do cálculo da matriz de possibilidades de soluções visto que cada célula da matriz deve conter um resultado, dessa forma a complexidade assintótica de tempo dessa solução é $O(m * n)$, sendo m o número de ilhas e n o valor a ser gasto na viagem.

Após as operações na matriz o número de dias deve ser calculado percorrendo a matriz no pior caso de forma linear com complexidade $O(max(m, n))$ conforme pode ser observado no gráfico a seguir:

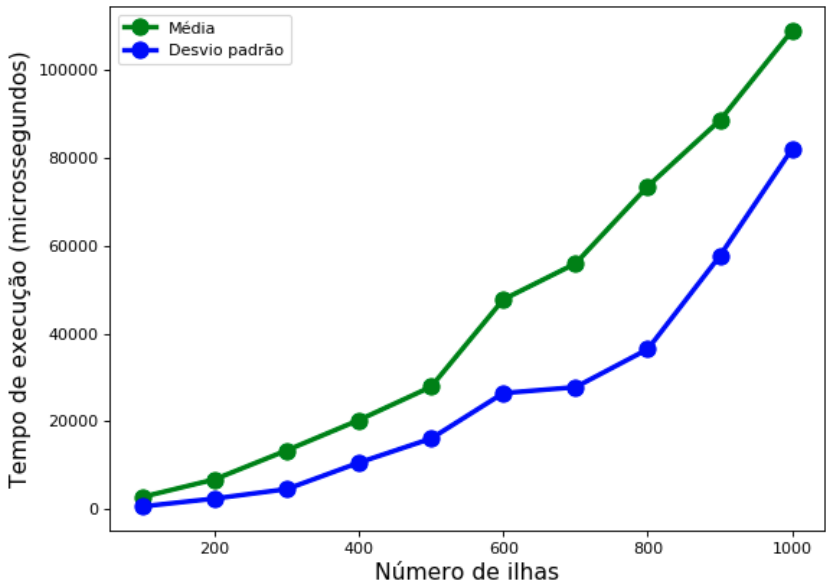

Figura 4. Tempo de Execução da Programação Dinâmica

3.4. Escolha dos Algoritmos

3.5. Caso Guloso

No caso do algoritmo guloso há a possibilidade de Luiza e suas amigas ficarem mais de um dia no mesmo lugar, ou seja, há a possibilidade de uma ilha ser escolhida mais uma vez, assim o algoritmo guloso provê a solução para aumentar o tempo de estadia em uma ilha e por consequência, nos melhores casos, aumenta a eficiência no gasto e garante o melhor custo benefício.

3.6. Caso Programação Dinâmica

No caso da programação dinâmica não há a possibilidade de uma mesma ilha ser escolhida duas vezes, sendo assim há a maior possibilidade do aumento do número de lugares que Luiza e suas amigas poderão conhecer. Nesse caso, o maior número de pontos somados dado a quantia máxima a ser gasta é o objetivo principal e concluído pelo algoritmo. Sendo assim, apesar dele proporcionar a possibilidade de conhecer mais lugares que a solução do algoritmo guloso, seu interesse é reunir os lugares com melhores pontuações e que respeitem o orçamento do grupo.