Trabalho Prático II - Algoritmos I

Gabriel Victor Carvalho Rocha

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

gabrielcarvalho@dcc.ufmg.br - 2018054907

1. Introdução

O problema abordado foi a implementação de dois diferentes algoritmos para solucionar a decisão de viagem da Luiza e suas amigas para o Arquipélago de San Blas, onde cada ilha possui um custo e uma quantidade de pontos e ela deseja maximizar essa quantidade de pontos de acordo com o dinheiro disponível.

A primeira implementação consiste em um algoritmo utilizando o método Guloso e a segunda utilizando o método Dinâmico, entretanto cada algoritmo resolve o problema com restrições diferentes: A solução gulosa aceita repetições de ilhas, ou seja, Luiza pode permanecer mais de um dia na mesma ilha. Já o Dinâmico, cada ilha pode ser visitada uma única vez.

2. Implementação

Todo o programa foi desenvolvido na linguagem C, utilizando o compilador GCC (GNU Compiler Collection).

2.1. Estruturas de dados

A estrutura de dados utilizada foi um vetor de struct, onde cada índice do vetor armazena o custo, os pontos e o custo benefício (pontos dividido pelo custo) de uma ilha. Um esquema genérico para M ilhas da estrutura aplicada pode ser analisada abaixo:

0	1		M - 3		M - 1
custo	custo		▪ ▪ ▪		custo
pontos	pontos				pontos
p / c	p / c				p / c

2.2. Algoritmos utilizados

As funções mais importantes do código são a solucaoGulosa e solucaoDinamica, que recebem como parâmetro o número total de ilhas, o valor máximo do dinheiro disponível e o vetor de structs.

Utilizamos a solução Gulosa para resolver o primeiro problema, onde repetições de ilhas são possíveis, pois esse problema é NP, então precisamos utilizar o algoritmo Guloso que dará uma solução aproximada do problema. No segundo caso, onde repetições de ilhas

não são permitidas, a solução Dinâmica utilizando uma matriz para armazenar em cada índice a melhor solução deste subproblema nos dará uma solução ótima.

O programa inicia lendo o arquivo e armazenando os valores em variáveis e em um vetor de struct, mas além dos dados fornecidos, no vetor também é colocado o custo benefício de cada ilha, sendo a quantidade de pontos dividido pelo custo. Então é chamada as duas principais funções:

A solucaoGulosa irá ordenar utilizando Mergesort o vetor de struct em ordem decrescente em relação ao custo benefício. Após isso, em um laço for começando da primeira ilha (que terá o maior custo benefício), entrará em um laço while que executará enquanto o preço desta mesma ilha for menor ou igual ao dinheiro disponível, e então é adicionado os pontos desta ilha na pontuação final e incrementado o número de dias totais em 1, e por fim, retirando o seu custo do dinheiro disponível. Caso ainda houver dinheiro e o preço desta ilha for maior, iremos então ir para a ilha que possui o segundo maior custo benefício e assim por diante. Contudo, a solução gerada não será ótima, como veremos nos próximos tópicos.

Já na solucaoDinamica, não precisamos ordenar o vetor, entretanto, precisamos de uma memória adicional: uma matriz de dimensões $número\ de\ ilhas + 1$ por $dinheiro\ total + 1$. Preenchemos a $linha\ 0$ da matriz com valores 0, pois não há ilhas para preencher pontos. Então a matriz é percorrida começando da $linha\ 1$. Seja i a linha que estamos, comparamos o custo da $ilha[i - 1]$ (note que é $i - 1$ pois o vetor começa do índice 0) com o valor da coluna (que varia de 0 à $dinheiro\ total$), caso esse seja maior, colocamos então a melhor solução sem a $ilha[i - 1]$. Senão, será preenchida pelo maior valor entre a melhor solução sem a $ilha[i - 1]$ e a melhor solução com a $ilha[i - 1]$. A pontuação total retornada sem repetição de ilhas estará na última linha e na última coluna da matriz, isto define nossa equação de Bellman:

$$OPT[i][j] = max(OPT[i - 1][j],\ ilha[i].pontos + OPT[i - 1][j - ilha[i].custo])$$

No entanto, para acharmos a quantidade de dias totais, teremos que percorrer a matriz começando do final. Temos que comparar com os pontos da linha acima, se forem iguais, vamos para esta linha. Após acharmos uma linha que possui pontos diferentes, iremos pular para a linha acima e para a coluna subtraída do custo da ilha que se estava, incrementando o número de dias totais em 1.

3. Instruções de compilação, execução

A compilação é feita através de um arquivo Makefile, digitando "*make*" em um terminal Linux é gerado o executável "*tp*2" . Já a execução é realizada da seguinte forma:

*./tp*2 *nomeArquivo.txt*

4. Análise de complexidade

Na complexidade em relação ao tempo, iremos analisar o código principal *main* e assim percorreremos todas as funções na ordem em que aparecerem.

Primeiramente, definindo M como o número de ilhas e N como o dinheiro máximo a ser gasto, o arquivo é lido e armazenamos todas as ilhas em um vetor de struct que possui os

atributos *custo*, *pontos* e *custoBeneficio* (*pontos*/*custo*). Como precisamos apenas da complexidade dos algoritmos principais, iremos omitir o custo destas leituras, então após isso, partimos para cada uma das duas funções:

4.1. solucaoGulosa

Iniciamos alocando um vetor auxiliar para utilização da ordenação, possuindo custo de espaço $O(M)$ e então ordenamos o vetor de struct em ordem decrescente em relação ao custoBeneficio com a função mergesort com custo de tempo $O(M\ log\ M)$ e de espaço $O(1)$. Após isso, é feita a checagem das ilhas decrementando do dinheiro disponível o custo das ilhas que possuem os maiores custoBeneficio (desde que dinheiro disponível seja maior ou igual ao custo da ilha), que considerando o custo mínimo para uma ilha como 1, rodaria no máximo N vezes, tendo complexidade de tempo $O(N)$.

Logo, esta função possui complexidade de tempo $O(M\ log\ M)$ e de espaço $O(M)$.

4.2. solucaoDinamica

Começamos a função alocando memória para uma matriz $M\ x\ N$, tendo custo de espaço $O(M \star N)$, preenchemos a primeira linha com $0's$ com custo $O(N)$ e então começamos a percorrer a matriz inteira colocando as melhores soluções em cada índice da matriz, possuindo complexidade de tempo $O(M \star N)$. E por fim, entramos em um laço que irá checar a quantidade de dias (ou de ilhas incluídas), tendo custo de tempo $O(M)$.

Logo, esta função possui complexidade de tempo $O(M \star N)$ e de espaço $O(M \star N)$.

5. Corretude dos algoritmos

5.1. Guloso

Suponha que o algoritmo Guloso para o problema onde há repetição de ilhas é ótimo, então considere a seguinte entrada:

6 2

3 5

4 8

Ou seja, $N = 6$ e $M = 2$. Considere $pontosFinais$ e $diasTotais$ como os valores da saída que iniciam com 0.

Temos que a $ilha[0]$ possui $custo = 3$, $pontos = 5$ e $custoBeneficio = 1.67$, e que a $ilha[1]$ possui $custo = 4$, $pontos = 8$ e $custoBeneficio = 2$. Pelo algoritmo, iremos pegar a $ilha[1]$, já que possui maior custoBeneficio, então:

$N - 4 = 2$

$pontosFinais + 8 = 8$

$diasTotais + 1 = 1$

8 1

Ou seja, 8 pontos e 1 dia.

Porém, a solução ótima seria se escolhêssemos a $ilha[0]$, pois:

$N - 3 = 3$

$pontosFinais + 5 = 5$

$diasTotais + 1 = 1$

$N - 3 = 0$

$pontosFinais + 5 = 10$

$diasTotais + 1 = 2$

10 2

Ou seja, 10 pontos e 2 dias, a solução ótima. Logo, por contradição, o algoritmo Guloso não é ótimo.

5.2. Dinâmico

Por convenção, vamos determinar que o vetor de *ilhas* comece do índice 1.

Considere a solução onde há 0 *ilhas*, logo, toda a primeira linha será preenchida com $0's$.

Seja $ilha[1]$ a ilha atual e j o valor da coluna, a solução para as colunas $j's$ com valores menores do que o custo da $ilha[1]$ será a solução que não inclui esta *ilha*, ou seja, os valores da linha acima, por outro lado, depois q o j estiver maior ou igual ao custo da $ilha[1]$, podemos pegar a **melhor solução** entre incluir ou não a $ilha[1]$.

Iremos propagar isto por todas as *ilhas* posteriores. Portanto, para a $ilha[N]$, como todas as soluções anteriores a ela são as melhores possíveis, para ter a solução ótima final basta tomar como solução o **maior valor** entre incluir ou não a $ilha[N]$.

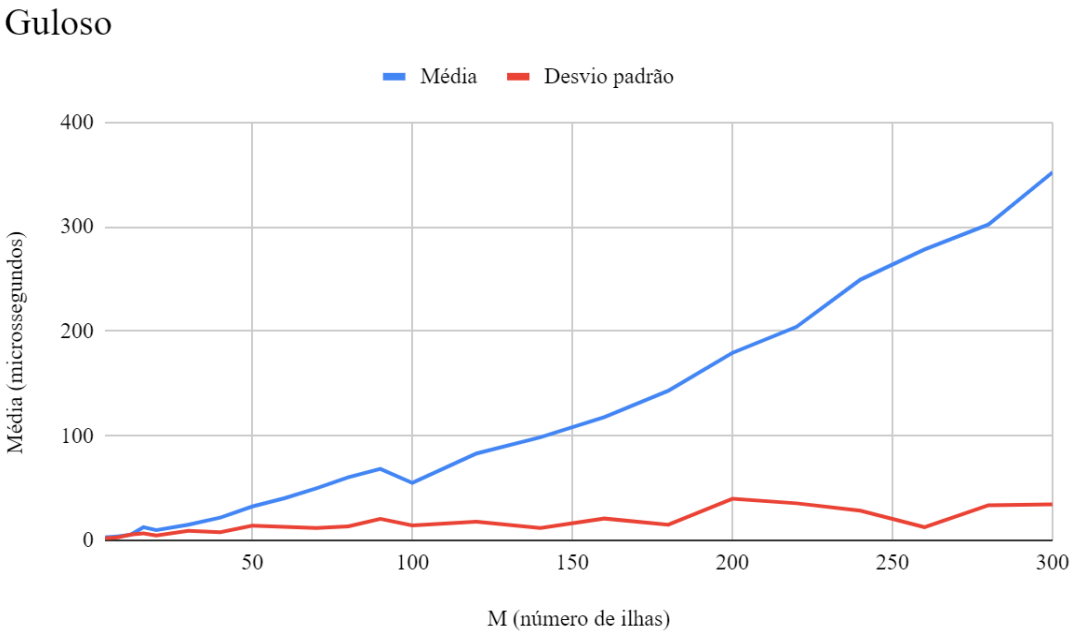
Logo, o algoritmo deve ser ótimo.

6. Análise experimental

Para a realização do experimento, foram feitas 23 entradas diferentes, variando o número M de ilhas de 4 à 300 e variando o dinheiro disponível N começando com 1000. Cada uma das 23 entradas foram testadas 20 vezes, pegando-se a média do tempo e o desvio padrão em microssegundos. Os resultados se encontram nos gráficos abaixos:

6.1. Guloso

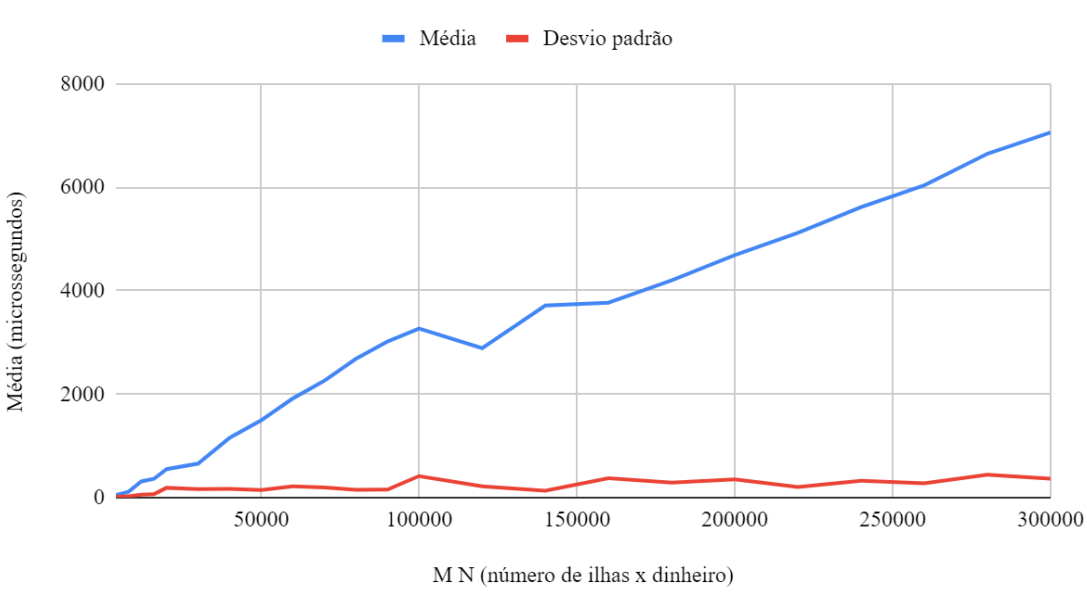
Gráfico do experimento para a solução gulosa:



Como pode-se ser analisado, o gráfico da média dos tempos possui uma inclinação um pouco maior do que a linear em relação ao número M. Podemos então certificar que a complexidade é $O(M \log M)$ como mostrado na Análise de complexidade.

6.2. Dinâmico

Gráfico do experimento para a solução dinâmica:



Analisando o gráfico da média dos tempos, podemos perceber que é linear em relação ao número $M \star N$, ou seja, podemos também certificar que a solução dinâmica é $O(M \star N)$, como mostrado na Análise de complexidade.

6.3. Algumas questões

Cada abordagem pode ser vista como mais vantajosa para cada situação específica.

Como no primeiro problema podemos repetir mais de uma ilha, o método Guloso pode ser vantajoso para permanecer em uma mesma ilha mais de um dia, aumentando o número de dias de estadia em uma ilha, isso ocorre porque após visitar a ilha com maior custo benefício e o dinheiro disponível for maior ou igual ao custo dela, poderá permanecer nesta mesma ilha até que seu dinheiro seja menor do que este custo.

Já no segundo problema, é restrito a visitar uma ilha uma única vez, por esse motivo o método dinâmico pode ser vantajoso para quem deseja visitar mais lugares, pois nesse método após visitar uma ilha e ainda houver dinheiro disponível para alguma ilha, necessariamente será uma nova ilha.

7. Bibliografia

KLEINBERG, Jon.; TARDOS, Éva. Algorithm Design.

ALMEIDA, Jussara.; Paradigmas - Algoritmos Gulosos. PDF disponibilizado via Moodle UFMG.

ALMEIDA, Jussara.; Paradigmas - Programação Dinâmica. PDF disponibilizado via Moodle UFMG.