TP2 - ALGORITMOS I

Universidade Federal de Minas Gerais

Luiz Felipe Couto Gontijo - 2018054524

- **Objetivo**

O objetivo do trabalho consiste em realizar a modelagem e implementação de algoritmos e soluções dado o problema de visitação de ilhas proposto. Mais especificamente, realizar duas abordagens diferentes para uma mesma instância do problema, uma utilizando um algoritmo guloso e outra utilizando a técnica da programação dinâmica. Também é pedido que o algoritmo guloso implementado seja da ordem de complexidade O(m*logm) - em que *m* é o número total de ilhas - e o algoritmo de programação dinâmica da ordem O(m*n) - em que *n* é o valor total a ser gasto na visitação.

- **Ambiente de desenvolvimento e Compilação**

O projeto foi desenvolvido em C++.

Para compilar, basta rodar o comando *make* no diretório onde se encontra o arquivo *tp2.cpp.* Após gerado o executável, execute-o passando o arquivo texto de entrada como parâmetro. Segue um exemplo:

```
> make
> ./tp2 teste.txt
```

- **O problema**

Um grupo de meninas deseja fazer uma viagem passando por diversas ilhas do arquipélago de San Blas. Cada ilha possui uma pontuação (que indica o quanto as meninas gostariam de visitar o local) e também um custo referente à esta visitação por dia. Dada o limite de custo total e as ilhas possíveis de visitação, o problema consiste em desenvolver:

Um algoritmo guloso que:

I. Maximize a pontuação total das ilhas.
II. É permitido ficar mais de um dia na mesma ilha.

Um algoritmo utilizando programação dinâmica que:

I. Maximize a pontuação total das ilhas.
II. Não é permitido ficar mais de um dia na mesma ilha.

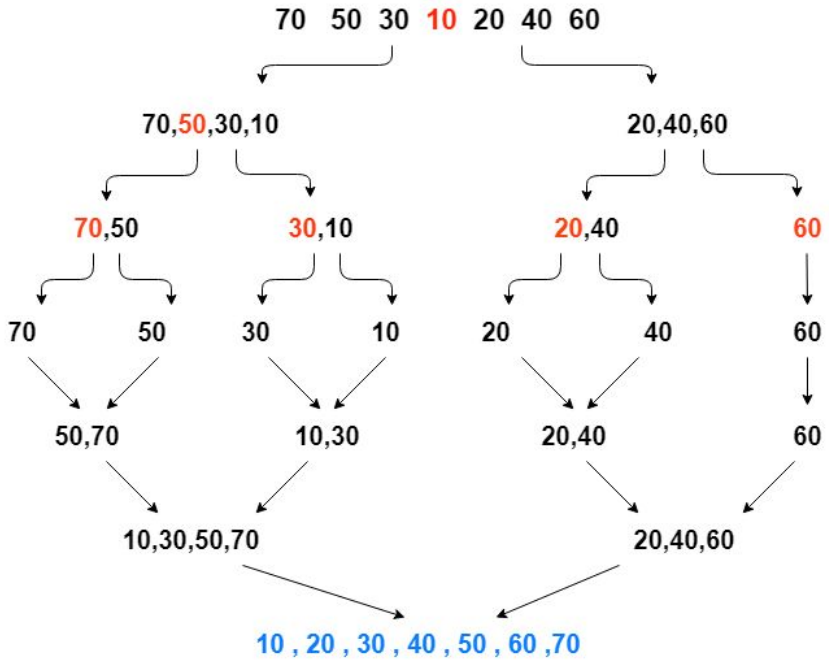
Em ambos os casos, deve-se retornar a maior pontuação possível, assim como o número de dias de duração da viagem.

- **Implementação**

Para a implementação do problema, foi criada uma única classe com a finalidade de receber a entrada do número de ilhas, custo limite total e pontuação respectiva a cada ilha. Nessa mesma classe, foram desenvolvidas as funções tanto do algoritmo guloso como do algoritmo que utiliza programação dinâmica. A seguir, será discutido a lógica e implementação de cada um dos algoritmos separadamente.

- **Algoritmo guloso**

Para o algoritmo guloso, dentro de cada instância de *Ilha,* foi criado, além do custo e pontuação, uma variável adicional que se refere ao *custo benefício.* O *custo benefício* é definido simplesmente pela razão entre a pontuação e o custo da ilha. Dessa forma, sabemos qual ilha é mais vantajosa de se visitar no próximo dia. Feito isso, é necessário ordenar as ilhas recebidas como entrada em ordem decrescente de custo benefício, para pré-definir a ordem de visitação. Para tal ordenação, foi criada uma função de *MergeSort* que ordena o array de *ilhas* em uma complexidade de tempo O(m*logm). O *MergeSort* é um algoritmo de divisão e conquista que divide o array em arrays menores (recursivamente), e faz a ordenação por partes, mas sempre mantendo o resultado da ordenação dos sub-vetores em um vetor temporário. Assim, a idéia básica do MergeSort é criar uma sequência ordenada a partir de duas outras também ordenadas, e depois, juntar as respostas obtidas.



Exemplo MergeSort
(Imagens - Google)

Após feita a o ordenação, o algoritmo é simples: a próxima ilha a ser visitada sempre será aquela ilha que seja a mais vantajosa (com maior custo benefício), mas que não ultrapasse o limite de custo total definido como entrada. Lembrando que pode haver repetição de ilhas, ou seja, é escolhida a primeira ilha do vetor ordenado até que o custo de uma próxima escolha dela ultrapasse o custo limite total. Assim, passamos para a próxima do vetor, e fazemos isso até que o custo total seja atingido ou que não seja mais possível adicionar ilhas. Ao mesmo tempo que fazemos isso, é somada a pontuação de cada ilha visitada, além do número total de dias da viagem. Ao final do algoritmo, são retornados ambos os valores.

A estratégia gulosa, nesse caso, não produz sempre uma solução ótima que maximiza a pontuação das ilhas. Para provar, basta analisar o seguinte contra-exemplo:

- Seja *M=3* e *N=18,* com as ilhas de pontuações [15,11,5] e custos [15,12,6] respectivamente. Assim, o vetor custo benefício seria [15/15, 11/12, 5/6]. O algoritmo guloso selecionaria, então, a primeira ilha do vetor custo benefício. Se isso ocorre, a única ilha visitada seria essa(já que visitar qualquer outra ilha ultrapassaria o custo total), e a pontuação máxima atingida seria 15. Porém, é fácil perceber que visitar as outras duas ilhas resultaria em uma pontuação maior (16), e o custo ainda estaria dentro do estipulado.

Tal comportamento do algoritmo é de certa forma comum para algoritmos gulosos, pois nem sempre a escolha de soluções ótimas locais leva a uma solução ótima do problema como um todo, algo que ocorre com a instância do problema em questão.

- **Algoritmo utilizando programação dinâmica**

Para a programação dinâmica, foi desenvolvido um algoritmo que é uma variação do conhecido *problema da mochila.* A ideia da programação dinâmica é resolver o problema maior dividindo-o em problemas menores, ao mesmo tempo que mantém os resultados dos problemas menores em uma tabela para que não seja necessário recalculá-los. No algoritmo proposto, essa ‘tabela’ será mantida em forma de uma matriz *M(n+1* por *m+1),* em que a linha e coluna extra engloba a solução de valor total zero e nenhuma ilha visitada. As linhas da matriz representam as ilhas, cada uma com sua pontuação e custo, e as colunas representam o limite de custo começando do zero até o valor máximo. A tabela é preenchida da seguinte forma:

M[i][limite_custo] = max(Excluído, Incluído), onde

Excluído = M[i-1][limite_custo]
Incluído = pontuação(i) + M[i-1][limite_custo – custo]

Toda vez que excluímos uma célula, ela é preenchida de vermelho na tabela abaixo, e, quando incluímos, ela é preenchida de verde. A seguir, um exemplo de preenchimento da matriz dada uma certa instância do problema e alguns dos valores explicados.

Limite de custo total = 4
Nº de Ilhas = 3

	0	**1**	**2**	**3**	**4**
Sem Ilhas	0	0	0	0	0
Ilha 1(2)(3)	0	0	3	3	3
Ilha 2(2)(1)	0	0	3	3	4
Ilha 3(1)(3)	0	3	3	6	6

Exemplo de tabela preenchida. O primeiro valor entre parêntesis representa a pontuação e o segundo o custo.

M[1][1]: O caso em que o limite é 1 e apenas temos uma única ilha para visitar (Ilha 1). O custo dela é 2, ultrapassando o custo limite. Assim, não podemos incluí-la na viagem (então colorimos a célula de vermelho), e o valor da pontuação total é zero.

M[1][2]: O custo limite agora é 2, e continuamos tendo apenas a Ilha 1 para incluir. Nesse caso, podemos fazer a inclusão, e portanto, temos que considerar dois casos:

- Se excluído = M [i-1] [limite_custo] = 0
- Se incluído e suficientemente pequeno = valor (i) + M [i-1] [limite_custo - custo] = 3 + 0 = 3

Então, Máx. (0,3) = 3, e, portanto, M [1] [2] = 3 (então colorimos a célula de verde).

M[2][1]:

- Se excluído = M [i-1] [limite_custo] = 3 (usa valores preenchidos anteriormente)

- Se incluído e suficientemente pequeno = valor (i) + M [i-1] [limite_custo - custo] = 1 + 0 = 1
- Máx. (3,1) = 3
- Colorimos de vermelho

Seguindo dessa forma, é possível preencher a tabela verificando os resultados anteriores, economizando tempo de processamento e otimizando o algoritmo.

Após preenchida a matriz, para saber qual a máxima pontuação possível, basta olhar para M[última_linha][última_coluna], ou seja,M[nº_de_ilhas][limite_custo_total]. Assim, a máxima pontuação possível é 6. Já para verificar quantos dias durou a viagem, 'voltamos' a partir do nosso resultado final. Começaremos com a última célula. Isto é M [3] [4]. Para o custo limite total de 4, a pontuação máxima que podemos obter é 6. Como essa é a linha da Ilha 3, e o custo da Ilha 3 é 1, ainda temos 3 para gastar. Então, o algoritmo 'olha' para a coluna de limite de custo 3, e vemos que o máximo também se refere a Ilha 3, porém ela já foi escolhida. Então, o que resta é a Ilha 1, com custo 2 e pontuação 3. Ainda sobra 1 para gastar, porém não há mais ilhas passíveis de visitação. E, portanto, temos que a duração da viagem foi de dois dias. No final, o algoritmo retorna a pontuação máxima obtida e o número de dias da viagem.

Para provar a otimalidade do algoritmo, é necessário mostrar a propriedade do algoritmo de subestrutura ótima, ou seja, a solução ótima do problema contém a solução ótima de seus subproblemas. Portanto,

- Seja O um subconjunto ótimo de *m* ilhas com um custo limite K.
- Queremos mostrar que O contém a solução para todos as sub-instâncias (por indução)
    - Caso 1: Se O **não** contém a ilha *m*, então é claro que há um subconjunto ótimo de m-1 ilhas(pela hipótese de indução).
    - Caso 2: Se O contém a ilha *m,* então O - {m} é uma solução para a instância do problema que inclui as primeiras n-1 ilhas e um limite de custo K - (custo de *m*), que, portanto, é ótimo.

Também é possível provar por contradição:

- Se O - {m} não é uma solução ótima, então deve haver um outro subconjunto Q com uma pontuação maior. Adicionando a *m*-ésima ilha em Q, temos um conjunto de ilhas com maior pontuação que O, o que seria uma contradição.

- **Complexidade**

Algoritmo Guloso

Complexidade de espaço: Em questão de espaço, o algoritmo apenas utiliza um array com M ilhas, e portanto, sua complexidade é O(M).

Complexidade de tempo: Existe um array de M ilhas sendo ordenado por uma algoritmo de MergeSort, portanto, como visto anteriormente, a complexidade de tal ordenação é O(M logM). Após isso, para a escolha das ilhas que serão visitadas, tal array é percorrido, e, no pior caso, a complexidade é O(M). Então, a complexidade de tempo final é O(M logM) + O(M) = O(M logM).

Algoritmo com programação dinâmica

Complexidade de espaço: O algoritmo utiliza apenas uma matriz N * M, e portanto, a complexidade de espaço é O(N*M).

Complexidade de tempo: O algoritmo preenche cada célula da matriz M * N, calculando apenas uma vez cada uma. Quando é necessário buscar um valor calculado anteriormente, o custo é constante, ou seja, O(1). Portanto, o custo se deve apenas ao preenchimento da matriz, que é de O(N*M), assim como a complexidade final.

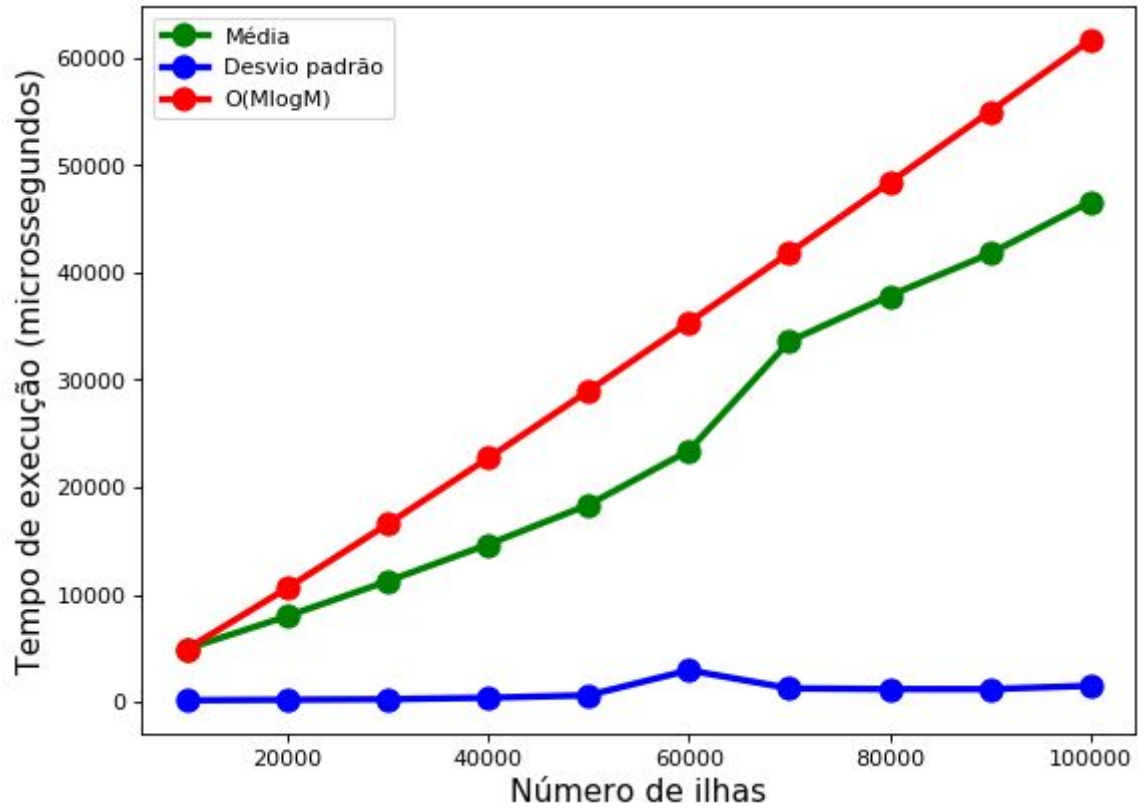
- **Testes**

Para a realização dos testes, foram gerados automaticamente casos de testes que englobam números diferentes de ilhas (M) e de custo total limite(N).

Guloso

No caso do algoritmo guloso, foram feitos 10 testes diferentes com o número M de ilhas variando de 10000 a 100000 (sendo que cada um deles foi repetido 20 vezes, e então retirada a média e o desvio padrão). Com a finalidade de comparação, foi adicionado também o resultado esperado caso a variação de tempo seguisse proporcionalmente *mlogm*. Os resultados são apresentados abaixo:

M	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
O(mlogm)	4934,55	10.612	16.569	22.709	28.984	35.367	41.840	48.389	55.006	61.682
Média	4934,55	7979,8	11248	14669,75	18358,6	23412,85	33594,65	37833,75	41763,1	46610,75
Desvio Padrão	136	188	238	377	606	2.940	1.251	1.182	1.178	1.492

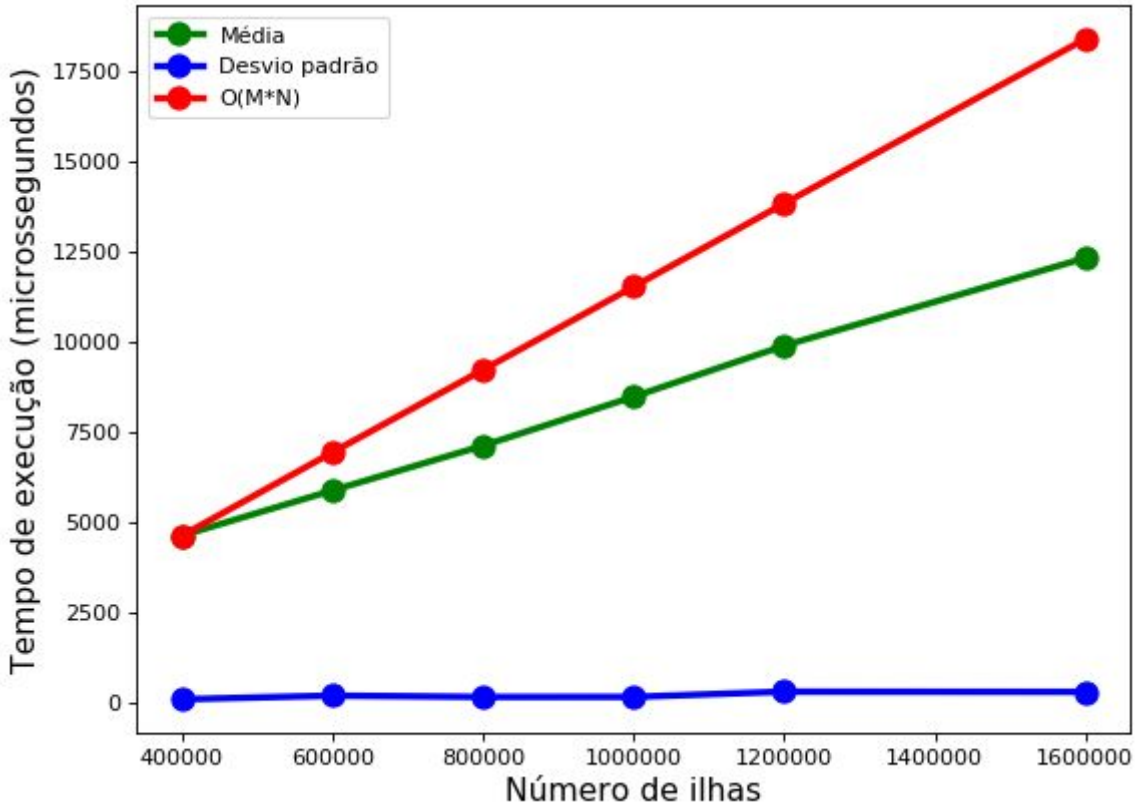


A partir da tabela e do gráfico, é possível perceber que a complexidade de tempo é, de fato, superiormente limitada por O(*mlogm*).

Programação Dinâmica

No caso do algoritmo usando programação dinâmica, foram feitos 6 testes diferentes com o número M*N (número de ilhas **x** limite de custo total) variando de 400000 a 1600000 (sendo que cada um deles foi repetido 20 vezes, e então retirada a média e o desvio padrão). Com a finalidade de comparação, foi adicionado também o resultado esperado caso a variação de tempo seguisse proporcionalmente *m*n*. Os resultados são apresentados abaixo:

N*M	400000	600000	800000	1000000	1200000	1600000
O(M*N)	4623,7	6935,55	9224,28	11530,35	13836,42	18402,43
Média	4623,7	5876,9	7117,75	8476,55	9892,3	12323,5
Desvio Padrão	80,71	182,17	145,54	147,52	292,97	287,16



A partir da tabela e do gráfico, é possível perceber que a complexidade de tempo é, de fato, superiormente limitada por O(*m*n*).

- **Questões**

Qual alternativa de algoritmo seria mais vantajosa para cada abordagem: conhecer mais lugares ou aumentar o tempo de estadia no local.

No caso do algoritmo guloso, vemos que gerar soluções ótimas locais faz com que haja a possibilidade de o número de locais visitados seja menor quando comparado com o da solução ótima. Porém, é fácil perceber que o tempo de estadia em um mesmo local é maior, já que pode haver repetições de ilhas em dias diferentes. Já quando olhamos para o algoritmo utilizando programação dinâmica, é perceptível que maximizamos a pontuação total ao mesmo tempo que o tempo de estadia em um mesmo local é o menor possível. Assim, o tempo de estadia diminui, mas a possibilidade de conhecer lugares novos aumenta. Portanto, o algoritmo guloso é mais vantajoso quando se trata de permanecer em um mesmo local, o que realmente condiz com a característica de algoritmos gulosos, que é a de gerar soluções ótimas locais. Utilizando a programação dinâmica, entretanto, faz com que a solução do problema se torne mais vantajosa globalmente, ou seja, considerando o problema como um todo, gerar a máxima pontuação.

Por que no primeiro problema é utilizado um algoritmo guloso e no segundo programação dinâmica?

Foi utilizado um algoritmo guloso no primeiro problema justamente pela característica de poder repetir ilhas em dias diferentes. Desse modo, o algoritmo guloso teria a função de sempre gerar a solução ótima local com a “esperança” de gerar a solução ótima local. Nesse caso, não é viável utilizar programação dinâmica, já que o número de possibilidades diferentes faria com que a construção da tabela fosse bastante custosa em termos de processamento, execução e, consequentemente, de tempo. Já no segundo problema utilizamos programação dinâmica pois o fato de não poder repetir ilhas faz com que as decisões locais do algoritmo guloso não funcionem, sendo necessário um algoritmo que abrange uma maior gama de possibilidades, sempre olhando para qual seria mais vantajosa. Assim, a programação dinâmica, que divide o problema em subproblemas e utiliza das respostas destes para formar a solução global, seria a melhor opção nesse caso.

- **Referências para a realização do trabalho e Bibliografia**

Algorithm Design, by Jon Kleinberg, Eva Tardos & Iva Tardos.

Cormen , T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009. Versão Traduzida:Algoritmos – Teoria e Prática-3a. Edição, Elsevier, 2012.