Trabalho Prático 2 - Algoritmos 1

Victor Hugo Silva Moura - 2018054958

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

1 Introdução

Um grupo de amigas deseja visitar o Arquipélago San Blas, local onde foi gravado o seriado *La casa de papel*, antes que as ilhas do arquipélago desapareçam devido ao derretimento das calotas polares provocado pelo aquecimento global. Para isso, elas reservaram uma certa quantia para a viagem e desejam saber qual a melhor opção de viagem. O grupo estabeleceu uma pontuação para cada ilha de acordo com o quanto querem visitar essa ilha. A pontuação estabelecidas pelas meninas, para cada ilha, equivale a um dia naquela ilha.

Infelizmente elas não podem visitar todas as ilhas, pois o dinheiro que possuem é limitado. Dessa forma, elas criaram dois tipos de roteiro de viagem possíveis. No primeiro caso, dado um orçamento máximo disponível para ser gasto por cada pessoa, qual a maior pontuação possível ao escolher um conjunto de ilhas, podendo ocorrer repetições (elas ficariam na mesma ilha por mais de um dia). A maior pontuação equivale à soma das pontuações de cada visita diária incluída no roteiro. A outra possibilidade de roteiro analisada é dado o orçamento máximo a ser gasto por cada menina, qual a maior pontuação possível, sem repetições de ilhas.

O objetivo do trabalho prático é, dada o orçamento do grupo e um conjunto de ilhas, com suas respectivas pontuações, desenvolver duas soluções:

1. **Utilizando algoritmo guloso determinar:** (a) a maior pontuação possível se as meninas escolherem um roteiro no qual pode haver repetições de ilhas (ficar mais de um dia na mesma ilha) e (b) a quantidade de dias que durará a viagem. O tempo de execução do algoritmo para esse problema não deve ser superior a $O(m \log m)$;

2. **Utilizando programação dinâmica determinar:** (a) a maior pontuação possível se as meninas escolherem um roteiro no qual não pode haver repetições de ilhas e (b) a quantidade de dias que durará a viagem. O tempo de execução do algoritmo para esse problema não deve ser superior $O(n * m)$;

2 Implementação

A solução para o problema foi implementada por meio de um array de ilhas. Cada ilha do array possui três informações: custo por dia, pontuação por dia e custo por pontuação. O último dado será utilizado, e explicado, na implementação gulosa para o problema.

O algoritmo guloso foi utilizado para solucionar a versão do problema onde é possível haver repetições de ilhas durante a viagem. Esse paradigma de programação foi selecionado para esse problema justamente pelo detalhe de poder haver repetições. Como será explicado mais a frente, o algoritmo preza pela melhor solução local e a repete enquanto for possível, ou, no caso desse problema em específico, enquanto essa solução não estourar o orçamento total.

Já a programação dinâmica for utilizada para solucionar a versão do problema onde as ilhas não se repetem na solução. Esse paradigma de programação foi selecionado pois, como será provado mais a frente, sempre retorna a solução ótima, coisa que não ocorre sempre no algoritmo guloso, principalmente quando não podem ocorrer repetições.

2.1 Algoritmo Guloso

O algoritmo guloso para esse problema se baseia em um princípio: selecionar a ilha com o maior custo/benefício atual enquanto ela não estourar o orçamento do grupo de amigas. Assim que essa ilha não foi mais possível, o mesmo processo é feito com a próxima ilha até que tenhamos passado por todas as ilhas.

O custo/benefício é calculado como o custo por dia na ilha sobre a pontuação da ilha. Dessa forma, quanto menor o valor, maior é o custo benefício daquela ilha. Em outras palavras, comparando duas ilhas, a que tem custo por pontuação menor significa que, para um mesmo custo a pontuação ganha por essa ilha é maior que a outra que estamos comparando.

Assim, para escolhermos a cada passo a ilha com o maior custo/benefício, das que ainda são possíveis, é feita uma ordenação das ilhas por esse parâmetro. De forma a garantir que a complexidade requisitada $O(m \log m)$, sendo m o número de ilhas, seja atendida, a ordenação utilizada foi o MergeSort, que garante isso.

Feita a ordenação, o próximo passo é escolher a melhor solução local, que nesse caso é a ilha com o maior custo benefício no momento. Para cada ilha, o custo dela é comparado com o orçamento. Se esse custo for menor que o orçamento, subtraímos o custo vezes o número de vezes que ele cabe no orçamento do orçamento total. Além disso, o número de vezes que o custo cabe no orçamento é adicionado no número de dias da viagem, já que o custo da ilha é por dia.

2.2 Programação Dinâmica

Ao "proibir" a repetição de ilhas, o problema se torna parecido com o problema da mochila. Nesse caso, a "capacidade" da mochila é o orçamento

total das amigas e os itens são as ilhas. Para o itens, o "peso" deles é o custo por dia e o "valor" é a pontuação que foi atribuída pelo grupo.

Dessa forma, a solução com programação dinâmica para esse problema é feita de forma semelhante à solução para o problema da mochila, explorando a divisão do problema principal em subproblemas, com sobreposição, mais simples de resolver. Para facilitar a notação, vamos utilizar a equação de Bellman.

Vamos considerar $OPT(i, w)$ como a pontuação máxima que podemos obter em um sub-conjunto de ilhas $1, ..., i$ e com um orçamento máximo de w. O objetivo que queremos maximizar $OPT(m, W)$, onde m é o número de ilhas e W é o orçamento para a viagem. Note que existem dois casos possíveis para cada ilha i:

- A solução final não inclui a ilha i. Nesse caso, $OPT(i, w)$ é a melhor solução com o sub-conjunto $\{1, ..., i-1\}$
- A solução final inclui a ilha i. Nesse caso, a pontuação da ilha é adicionada na solução e $OPT(i, w) = v_i + OPT(i-1, w-w_i)$, onde v_i é a pontuação da ilha i e w_i é o custo da ilha. Sendo assim, a solução é a soma da pontuação da ilha com a melhor solução que não inclui a ilha, mas com o custo da ilha subtraído do orçamento máximo.

A equação de Bellman para esse problema está apresentada abaixo:

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

A recorrência foi calculada por meio de uma matriz m x W, onde m é o número de ilhas e W é o orçamento total, seguindo as regras descritas acima. O resultado final é a posição $[m][W]$ da matriz. Para diminuir o tamanho da matriz, e consequentemente o tempo de computação da matriz, é calculado um mdc entre os valores das ilhas e o orçamento total. Dessa forma, podemos dar saltos maiores na matriz, de acordo com o mdc, já que valores menores que ele não alterarão o resultado.

3 Análise de Complexidade

3.1 Complexidade de Tempo

Vamos dividir a complexidade de tempo nos dois paradigmas de programação usados para resolver esse problema.

O algoritmo guloso faz duas operações principais: ordenar as ilhas de acordo com o custo/benefício e percorrer as ilhas adicionando a que possui o melhor custo/benefício atual o número de vezes que ela cabe no orçamento restante. Para a primeira operação, foi utilizado o algoritmo MergeSort que possui uma complexidade $O(n \log n)$, sendo n o número de elementos a serem ordenados.

Como para esse problema temos m, ilhas, a complexidade dessa ordenação é $O(m \log m)$. Para a segunda operação, para cada ilha é checado quantas vezes o preço dela cabe no orçamento atual. Vamos chamar esse valor de k. Depois o valor da ilha vezes k é subtraído do orçamento atual e o número de dias é acrescido em k. Sendo assim, temos um número constante de operações para cada ilha e isso é feito para todas as ilhas, gerando um custo linear no número de ilhas $O(m)$. Assim, a complexidade para o algoritmo guloso é:

$$O(m \log m) + O(m) = O(m \log m)$$

A programação dinâmica, desconsiderando o mdc, faz duas operações principais: preenchimento da matriz de OPT e backtracking na matriz para calcular o número de dias da viagem. A primeira operação necessita percorrer toda a matriz para calcular os valores ótimos das subsoluções. Como a matriz é de tamanho m x W, sendo W o orçamento máximo do grupo de amigas, essa operação tem um custo $O(m * W)$. A próxima operação volta nas linhas da matriz procurando por diferenças na linha anterior até chegar na primeira linha e primeira coluna da matriz. Essa operação é linear no número de linhas da matriz, tendo um custo $O(m)$, onde m é o número de ilhas, que também é o número de linhas da matriz. Assim, a complexidade para a programação dinâmica é:

$$O(m * W) + O(m) = O(m * W)$$

3.2 Complexidade de Espaço

Para a complexidade de espaço, vamos dividir novamente a análise nos dois paradigmas de programação de forma a facilitar a análise da mesma. Para o algoritmo guloso, o MergeSort utiliza arrays adicionais para realizar a ordenação. Para cada operação de merge é criado um array adicional de tamanho igual a soma do tamanho das metades sendo ordenadas. A complexidade de espaço dessa ordenação é da ordem de $O(n \log n)$, sendo n o número de elementos da ordenação. Como para esse problema temos m, ilhas, a complexidade dessa ordenação é $O(m \log m)$, mesma complexidade que a complexidade de tempo. Além disso, são criados duas memórias que guardam o ganho total de pontos da viagem e o número de dias gastos, porém o custo delas é constante $O(1)$. Assim, a complexidade de espaço do algoritmo guloso é:

$$O(m \log m)$$

Para a programação dinâmica, a única operação que utiliza espaço adicional é a criação da matriz de OPT. Essa matriz é de tamanho m x W. O custo de criação dessa matriz, portanto, é a da ordem de $O(m * W)$. Além disso, são criadas duas memórias que guardam o ganho total de pontos da viagem e o número de dias gastos, assim como no algoritmo guloso. O custo delas é constante $O(1)$. Assim, a complexidade de espaço do algoritmo guloso é:

$$O(m * W)$$

4 Provas de Corretude

Vamos provar inicialmente para o algoritmo guloso. Esse algoritmo não apresenta sempre uma solução ótima. Sendo assim, será provado, por contra-exemplo, que a solução não é sempre ótima.

Suponha a seguinte entrada para o programa:

1800 2

1000 30

900 20

Calculando o custo/benefício das duas ilhas, temos 33.333... para a primeira ilha e 45 para a segunda, lembrando que o custo/benefício é custo sobre pontuação da ilha, e quanto menor o valor, melhor o custo/benefício. Segundo o algoritmo guloso, nesse caso deveríamos pegar a primeira ilha e o orçamento restante seria de $800. Com esse orçamento, não é possível pegar outra ilha e a quantidade de pontos final é de 30 pontos. Porém a solução ótima seria pegar a ilha dois duas vezes, o que deixaria o orçamento restante zerado, mas uma pontuação final de 40 pontos. Dessa forma conseguimos ver que o algoritmo guloso não é sempre ótimo para todos os casos.

A prova da solução gerada pela programação dinâmica ser ótima se baseia na equação de Bellman descrita durante a implementação da programação dinâmica. Para cada nova ilha adicionada duas possibilidades são verificadas:

- A possibilidade da ilha se adicionada na solução final
- A possibilidade da ilha não ser adicionada na solução final

Dessa forma, "todas" as possibilidades de solução são verificadas e a melhor delas é escolhida. Isso é feito recursivamente para cada subconjunto de ilhas, formando no final a solução ótima para o problema geral. Nem todas as soluções são realmente testadas (o que explica as aspas utilizadas) pois após descobrirmos a melhor solução para um determinado sub-conjunto, não é necessário utilizar as rever as outras opções para esse sub-conjunto enquanto estamos explorando conjuntos maiores. Como "todas" as possibilidade são checadas, podemos dizer que a solução final é sempre ótima.

5 Análise Experimental

Para a análise experimental, foram utilizados orçamentos e custos de ilhas aleatórios (o orçamento é um valor entre 10 e 10000), enquanto que o número de ilhas varia de 0 a 45000, crescendo de 5000 em 5000. Para cada teste, foi computado o tempo médio de execução para 20 execuções. Abaixo temos uma tabela para cada um dos paradigmas de programação mostrando os resultados obtidos:

Execuções - Algoritmo Guloso		
Nº de Ilhas	Média do Tempo de Execução (ms)	Desvio Padrão do Tempo de Execução
0	0.00000	0.00000
5000	0.58345	0.00000
10000	1.22130	0.00001
15000	1.87840	0.00012
20000	2.56050	0.00001
25000	3.26080	0.00025
30000	3.97030	0.00012
35000	4.70010	0.00020
40000	5.42820	0.00031
45000	6.15190	0.00034

Execuções - Programação Dinâmica		
Nº de Ilhas	Média do Tempo de Execução (ms)	Desvio Padrão do Tempo de Execução
0	0.00000	0.00000
5000	157.12400	6.30500
10000	292.89200	45.76500
15000	490.98100	65.77500
20000	690.83000	105.82000
25000	864.48300	213.53400
30000	1005.71400	296.85000
35000	1162.52500	374.86400
40000	1483.29800	420.50000
45000	1096.55100	430.68500

Tabela 1: Número de Ilhas, Média do Tempo de Execução e o Desvio Padrão do Testes Para Cada Paradigma

Foi gerado um gráfico para cada uma das soluções a partir das tabela mostrando a relação entre o tempo de execução e a quantidade de ilhas. Os gráficos gerados estão presentes abaixo:

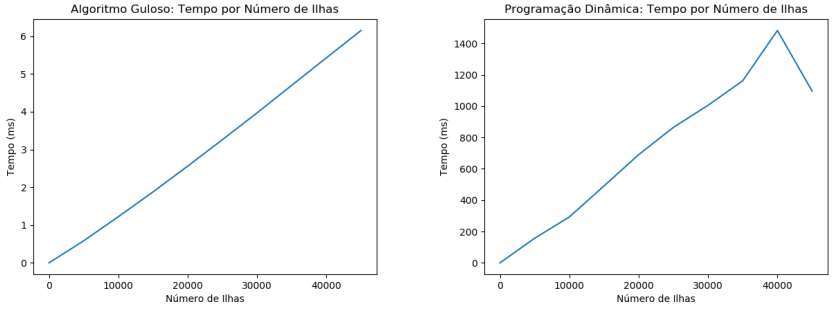

Figura 1: Gráfico de Tempo Para as Duas Soluções

Por meio desses gráficos, podemos ver que para a solução gulosa, o gráfico possui uma leve curvatura devido ao fator logarítmico que influencia na complexidade. Já o gráfico de programação dinâmica segue uma relação quase linear com o número de ilhas, visto que para os testes feitos, o orçamento total possuía um teto máximo, deixando de ser o possível fator dominante da complexidade. A queda que ocorre no final pode ser explicada por um valor de orçamento muito baixo no caso de teste, o que joga a complexidade para baixo, visto que a complexidade é $O(n * m)$, ou $O(m * W)$ caso estejamos considerando o orçamento como o peso da mochila no caso do problema da mochila.

O desvio padrão alto para a programação dinâmica pode ser explicado pela aleatorieade durante a geração das entradas para o programa. A forma como

os casos são feitos pode impactar muito no tempo final de execução.

Referências

[1] GeeksForGeeks. Merge Sort.

[2] B. Kleinberg and É. Thardos. *Algorithm Design*. Pearson, 2005.

[3] N. Ziviani. *Projeto de Algoritmos com Implementações em Pascal e C*. Cengage, 2011.