Segundo trabalho prático de Algoritmos I

Universidade Federal de Minas Gerais

2019/2

Ábner de Marcos Neves

2018054605

1. Introdução

A fim de visitar o Arquipélago de San Blas, Luiza e suas amigas juntaram certa quantidade de dinheiro, mas, infelizmente, não o suficiente para visitar todas as ilhas. Por isso, elas pesquisaram o preço de uma diária em cada ilha e pontuaram cada uma delas de acordo com a vontade de visitá-la.

A partir disso, elas querem decidir quais ilhas visitar pela análise de dois tipos de roteiro: um que permite ficar mais de um dia numa mesma ilha e outro que não permite. Ambos precisam respeitar o orçamento disponível e maximizar a soma total das pontuações que elas deram às ilhas.

2. Modelagem e implementação

A generalização de problemas como o proposto é chamada de *Knapsack problem*, ou Problema da mochila, e consiste em: dado um número n de itens, cada um com um valor v_i e um peso w_i, quer-se colocá-los numa mochila que suporta um peso W, de tal forma que a soma dos pesos w_i dos itens escolhidos não ultrapasse W e a soma dos valores v_i desses itens seja a maior possível.

Nesse problema de visitação do arquipélago, as ilhas foram interpretadas como os n itens do problema da mochila, o custo de uma diária na ilha i foi interpretado como seu peso w_i e as pontuações atribuídas pelo grupo de amigas foram interpretadas como os valores v_1, v_2, …, v_n de cada ilha.

O primeiro problema foi resolvido com um algoritmo guloso. A princípio, é calculada a razão de pontuação/custo para cada ilha, sendo todas armazenadas num vetor ordenado. As ilhas que têm as maiores razões são as ilhas mais vantajosas para visita.

Assim sendo, o algoritmo percorre o vetor em ordem decrescente dessa razão e, quando o custo de uma diária não ultrapassa o orçamento disponível, calcula quantos dias é possível ficar nessa ilha mais vantajosa, através de uma divisão inteira simples. As próximas ilhas passam pelo mesmo processo, com o orçamento restante disponível.

O segundo problema foi resolvido com um algoritmo de programação dinâmica. Acontece que a melhor solução para n ilhas e um orçamento w pode ser encontrada pelas melhores soluções de subproblemas da seguinte maneira:

Seja OPT(i, w) a pontuação total da melhor solução possível para um subgrupo {1, 2, …, i} de ilhas e orçamento w, essa solução poderá incluir ou não uma visita à ilha i. Caso não inclua, então:

$$OPT(i, w) = OPT(i-1, w)$$

Caso inclua a ilha i, já não temos mais um orçamento w disponível, mas sim $w\text{-}w_i$. Logo,

$$OPT(i, w) = v_i + OPT(i-1, w\text{-}w_i)$$

A ilha i não pode ser incluída no roteiro se w_i ultrapassar o orçamento total W ou se $OPT(i-1, w)$ for maior que $v_i + OPT(i-1, w\text{-}w_i)$. Isso leva à seguinte equação de Bellman:

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

Como a equação depende de i_s e w_s anteriores, é necessário uma matriz que guarde as subsoluções nessas duas dimensões, para todos os números inteiros entre 0 e n e entre 0 e W.

3. Análise de complexidade

3.1 Algoritmo guloso

O algoritmo guloso calcula a razão pontuação/custo para cada ilha e as armazena num *multimap* ordenado crescentemente. O cálculo dessa razão tem um custo constante e é feito uma vez para cada uma das n ilhas. A ordenação pode ser feita em tempo $O(n \log n)$. Então, para essa parte, temos que:

$$C = n*O(1) + O(n \log n) = O(n \log n)$$

Depois dos cálculos e da ordenação, o algoritmo percorre o vetor uma vez, de trás para a frente, fazendo, para cada elemento, cinco atribuições, seis operações aritméticas e seis acessos aos vetores de custo, pontuações ou ao *multimap*. Como todas essas operações são de custo constante, temos que:

$$C = n*17*O(1) = O(n)$$

No final, basta acessar duas variáveis para saber os resultados finais, o que não altera a complexidade de tempo final do programa, que é:

$$T = O(n \log n) + O(n) = O(n \log n)$$

Durante esse algoritmo, são usados um vetor para os custos das ilhas, um para as pontuações e um outro para a razão pontuação/custo, bem como outras seis variáveis para armazenar os valores finais e intermediários. Assim, a complexidade de espaço final é:

$$E = 3*O(n) + 6*O(1) = O(n)$$

3.2 Programação dinâmica

O algoritmo de programação dinâmica armazena uma matriz *M* de dimensões *n*W*, preenchendo-a sequencialmente, de cima para baixo, da esquerda para a direita, com as somas das pontuações de cada subsolução. Analogamente, é mantida uma outra matriz Q, também de dimensões *n*W*, preenchida da mesma forma, mas com as quantidades de dias de viagem para cada subsolução.

Para preencher as posições M_{ij} e Q_{ij} são feitas duas comparações, duas atribuições, quatro acessos à matriz M, três acessos à matriz Q, três ou cinco acessos aos vetores de custos e pontuações e uma ou três somas, desconsiderando aquelas para manipulação de índices.

Todas essas operações têm custos constantes, então a complexidade para preencher uma posição M_{ij} e uma Q_{ij} é:

$$C(M_{ij}) = C(Q_{ij}) = 19*O(1) = O(1)$$

Tanto a matriz M quanto a Q têm dimensões *n*W*, logo, a complexidade para preenchê-las é:

$$C(M) = C(Q) = n*W*O(1) = O(n*W)$$

Ao findar desse processo, basta acessar a última posição de cada uma das matrizes, o que não altera a complexidade de tempo final do algoritmo, que é:

$$T = O(n*W).$$

Como são utilizadas duas matrizes, dois vetores e duas variáveis, a complexidade total de espaço para esse algoritmo é:

$$E = 2*O(n*W) + 2*O(n) + 2*O(1) = O(n*W)$$

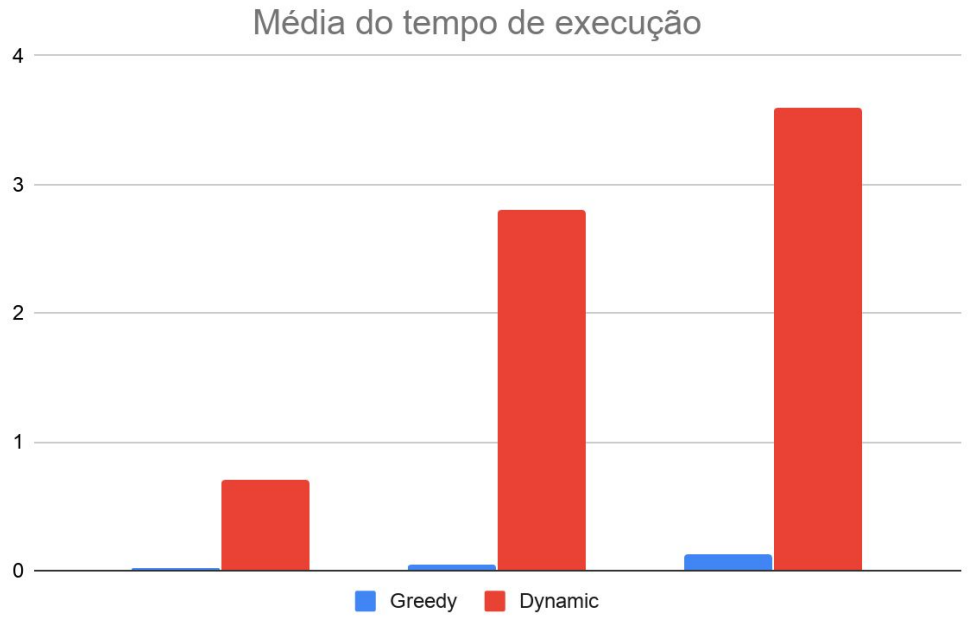
4. Avaliação experimental

Foram feitos testes com 10 entradas diferentes para cada tamanho: 20, 60 e 200 ilhas, totalizando 30 testes. Essas entradas foram geradas com números aleatórios pelo site random.org e o tempo de execução foi medido com auxílio da função *clock()*, da biblioteca *time.h*.

A seguir, um gráfico e uma tabela com as médias do tempo de execução e o desvio padrão de cada um:

Tempo de execução de cada algoritmo

Número n de ilhas	Greedy		Dynamic	
	Média	Desvio padrão	Média	Desvio padrão
20	0,0228	0,00377	0,6989	1,35684
60	0,0437	0,00864	2,8023	3,96263
200	0,1261	0,02466	3,598	4,13988



É notável a diferença de tempo entre os algoritmos guloso e dinâmico. Isso se dá justamente pela complexidade de cada um: enquanto o guloso tem tempo O(n log n), o algoritmo de programação dinâmica tem tempo O(n*W), sendo que W pode ser um número qualquer arbitrariamente grande.

Observando as saídas para cada um dos testes feitos, nota-se que o primeiro problema (em que pode haver repetições de ilhas) sempre alcança pontuação e número de dias maiores que o segundo problema (sem repetições de ilhas). Entretanto, isso não necessariamente quer dizer que o algoritmo guloso é melhor que o de programação dinâmica. Depende da prioridade.

Para maximizar o número de dias de viagem ou pontuação total, o guloso realmente se sobressai, pois inclui o máximo possível das ilhas mais vantajosas (as que têm as maiores razões de pontuação/custo).

Entretanto, querendo-se maximizar o número de ilhas diferentes visitadas, a programação dinâmica tem a melhor abordagem. Isso acontece, pois, quando a ilha *i* faz parte da melhor subsolução, para encontrar a melhor subsolução com o orçamento restante, o algoritmo decrementa também o subgrupo de ilhas possíveis (*i-1*), ao buscar por $OPT(i\text{-}1, w\text{-}w_i)$, não permitindo, portanto, que haja repetições de ilhas.