Trabalho Prático 2 - Algoritmos 1

Breno C. Pimenta
RA: 2017114809

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Brasil

brenocpimenta@gmail.com

1. Introdução

O problema apresentado, “Jogos dos Diamantes” consiste em encontrar o valor minímo após a subtração de todos os pesos de um conjunto de diamante. Repare que há uma questão crucial, que é a otimização do valor, ou seja a ordem da subtração afeta diretamente o resultado. Sendo que existe um limite máximo de 256 diamantes e peso máximo de 128 unidade.

Esse problema possui diferente soluções, nesta documentação será abordada a solução utilizando-se de programação dinâmica e será demonstrada, através de uma comparação com outra abordagem recursiva, seu desempenho, além da análise de complexidade.

2. Implementação

Os valores de entrada são o número de diamantes, seguido pelo peso de cada um. Antes que continuemos com a explicação das estruturas utilizadas, vale ressaltar que a solução do problema pode ser modularizada em duas partes, a primeira que denominaremos números grandes e a segunda que é sem números grandes. Também é importante notar que será utilizado com freqência o valor que é a metade inteira da soma de todos os pesos, o chamaremos de k (Somatório dos pesos dividido por dois).

A primeira parte da solução, ‘números grandes’, será utilizada caso exista um peso P, sendo que P >= k, então a solução do problema é que todos os outros pesos sejam subtraídos desse P e teremos o valor ótimo. A implementação dessa primeira parte se dá com o cálculo acontecendo junto com a verificação, ou seja é inicializada uma variável com o valor zero e uma flag com falso, em seguida todos os pesos são percorridos verificando se P>=k, caso seja, a flag assumirá o valor de verdadeiro e P será somado à variável, caso contrário, todos os pesos que forem menor que k, serão subtraídos da variável. Ao final do laço verificaremos se a flag é verdadeira e caso seja, a variável das somas e subtrações irá conter o valor minimizado da sequência de subtrações. Repare que caso exista um P, tal que P>=k, será necessário para a solução apenas um laço pelos pesos.

Caso não seja verificado um P tal que P>=k, será utilizada outra abordagem, entrará no caso ‘sem números grades’, onde os pesos serão divididos em dois subsets ss1 e ss2. Dentre todos as dibisões posséveis, existe uma divisão, sendo S1 a soma dos pesos em ss1 e S2 que é a soma dos pesos em ss2, onde |S1 – S2| é mínimo e este é o mínimo da nossa solução. Para encontrar esses subsets, temos que buscar S1 ou S2 que mais se aproxime a K, pois caso S1 = k e S2 = k, teremos o mínimo igual a zero, que é o menor possível. E há a necessidade de encontrar apenas um dos dois subsets, ss1 ou ss2, pois

caso encontremos ss1, é possível calcular S1, logo, subtraindo-o da soma total dos pesos tem-se S2.

O algoritmo de força bruta, irá percorrer todas as combinações possíveis deste algoritmo, mantendo as que possuam a soma mais próxima de k, para então retornar o resultado. O algoritmo que será referido como força bruta nesta documentação, a título comparativo, é uma implementação simples recursiva que verifica todas as possíveis combinações.

O algoritmo implementado e analisado nesta documentação utiliza-se de programação dinâmica, ou seja, será implementado uma tabela de somas para que ao percorrer os pesos, se tenha armazenado as possíveis somas dos subsets que foram montados. Esta tabela possui colunas de 0 a k e a mesma quantidade de linhas que diamantes. Após o preenchimento desta tabela, a última linha nos dirá quais os possíveis valores de soma dos subsets até o valor K, ou seja teremos todas as combinações de somas que resultem em valores menores ou iguais a K.

Para inicializar esta tabela, inicialmente coloca-se todos os valores da coluna zero como verdadeiros, pois sempre podemos colocar todos os pesos no outro subset. Iniciaremos a primeira linha com todos os valores falsos, fora zero e o primeiro peso, pois a única combinação que podemos realizar com ele, é sem ele ou somente ele. A título de exemplo, será demonstrado o processo pelo exemplo dos seguintes pesos de diamantes: 2, 3, 2, 4. Onde a soma é 11 e a divisão inteira por 2 é 5, ou seja k = 5. A montagem da tábela fica da seguinte forma:

Pesos\Somas	0	1	2	3	4	5
[2]	T	F	T	F	F	F
[2, 3]	T					
[2, 3, 2]	T					
[2, 3, 2, 4]	T					

Para popular esta tabela será utilizado dois laços, um interno ao outro, o primeiro que percorreremos as linhas, o número de pesos e o segundo onde será percorrido os valores até k. Será realizada duas verificações, a primeira, caso o valor logo a cima na tabela seja verdadeiro o mesmo continua sendo verdadeiro, pois as somas das combinações das linhas anteriores continua sendo válidas:

Pesos\Somas	0	1	2	3	4	5
[2]	T	F	T	F	F	F
[2, 3]	T		↓ T			
[2, 3, 2]	T					
[2, 3, 2, 4]	T					

E a segunda verificação é caso o valor da coluna é maior ou igual ao do peso da linha, o valor da célula será igual ao valor da célula da linha anterior, onde a coluna seja o peso da linha menos a coluna atual, ou seja buscamos a combinação das somas anteriores com o peso da linha atual:

Pesos\Somas	0	1	2	3	4	5
[2]	T	F	T	F	F	F
[2, 3]	T		T	T		
[2, 3, 2]	T					
[2, 3, 2, 4]	T					

Pesos\Somas	0	1	2	3	4	5
[2]	T	F	T	F	F	F
[2, 3]	T		T	T		T
[2, 3, 2]	T					
[2, 3, 2, 4]	T					

Caso nenhuma das verificações sejam atendidas a célula terá o valor falso alocado. Ao terminar os laços a tabela estará populada, segundo o exemplo apresentado estará da seguinte forma:

Pesos\Somas	0	1	2	3	4	5
[2]	T	F	T	F	F	F
[2, 3]	T	F	T	T	F	T
[2, 3, 2]	T	F	T	T	F	T
[2, 3, 2, 4]	T	F	T	T	T	T

A última linha apresenta todas as somas menores ou iguais a k para todas as combinações possíveis de pesos. Para encontrarmos S1 basta percorrer a última linha da última coluna à primeira e selecionar a primeira célula com valor verdadeiro, a coluna desta célula será o S1 do problema, logo para encontrarmos S2, basta subtrair S1 da soma total do pesos, então ao tirar o absoluto da diferença entre S1 e S2 reultará no menor peso possível para o diamante depois de substrações sucessivas com todos os pesos. No caso do exemplo, teremos que S1 = 5, logo S2 = 6, tendo a diferença mínima de 1.

3. Instruções de compilação e execução

Para que se realize os comandos de compilação e execução é impressindível que esteja-se no diretório raiz do projeto, nesse caso no diretório **breno_pimenta**. Caso o arquivo esteja zipado, primeiro realize o unzip e depois entre no diretório. Os comandos e suas respectivas ações são:

- **make**: apenas compila.
- **make run**: compila e executa o programa.
- **make clean**: deleta o arquivo executável.

4. Análise de Complexidade

A metodologia para a análise da complexidade se dará da seguinte forma: será realizado o estudo primeiramente em cada módulo, sendo cada ação dos módulos analisadas separadamente e sem seguida as complexidades serão comparada entre si. Será tomado como referência n o número de pedras e continuará sendo utilizado k.

1. **Módulo 'Números Grandes':** Como há apenas um laço neste módulo a complexidade consiste do número de diamantes, ou seja O(n).
2. **Módulo ' Sem Números Grandes':** Haverá dois laços, o primeiro de 2 a n e o segundo de 1 a k, sendo o segundo interno ao primeiro. Logo haverá (n-1)*k ações, resultando em uma complexidade de O(n*k).

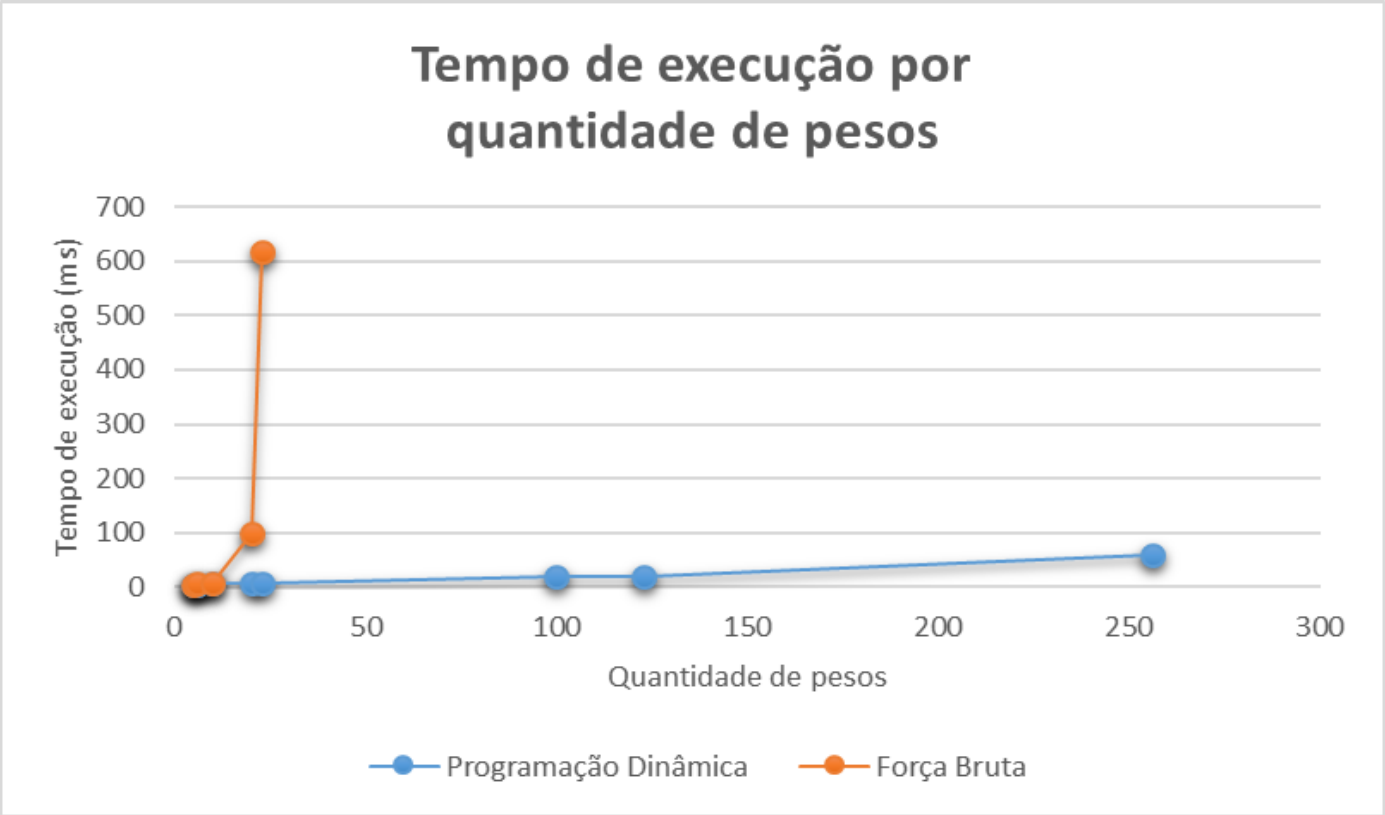
Portanto no pior dos casos para a complexidade temporal teremos O(n) + O(n*k) resultando assim em uma complexidade temporal de O(n*k) para o algoritmo, complexidade pesudo polinomial. E para a complexidade espacial no pior dos casos teremos que ter alocados o vetor com os pesos, no pior dos casos sendo 2k+1 e a tabela para a consulta das somas com n*k células, tendo uma complexidade de O(2k+1) + O(n*k), resultando em O(n*k).

Avaliação experimental: a título de análise será comparado a solução de programação dinâmica O(n*k), com a solução BruteForce recursiva $O(n^2)$. A tabela abaixo mostra uma amostragem de 100 execuções para cada quantidade de peso, para cada algoritmo:

Tabela 1. Tempo de execução médio e desvio padrão em milisegundos.

Qtd. Pesos	Tempo (ms)			
	Program. Dinâmica		Força Bruta	
	Média	Desvio Pad.	Média	Desvio Pad.
5	5,18	1,5399	5,69	2,2948
6	5,57	3,2762	6,69	2,5768
10	8,79	4,6586	7,19	3,4339
20	7,47	3,255	97,98	15,2009
23	7,5	3,2983	616,6	30,2521
100	18,73	6,3989	Não executou	
123	20,8	4,0251		
256	58,61	8,3543		

Gráfico 1. Tempo de execução por quantidade de pesos.



Vemos claramente que a solução de programação dinâmica possui uma complexidade temporal de execução melhor do que a solução de força bruta recursiva. Para os valores apresentados no problema, a solução de força bruta chega a ser inviável, pois diante da inclinação da função de tempo de execução por quantidade de peso, nota-se que com pouco mais de 23 pesos/pedras torna o tempo de execução impraticável.

5. Conclusão

O que mais ressaltou aos olhos durante a criação deste projeto, foi como que com uma entrada relativamente pequena, menor que trinta números, a solução de força bruta já se tornou completamente inviável e a solução de programação dinâmica, apenas com a adição de uma estrutura simples como a de uma tabela, se manteve robusta para entradas com mais de cem valores.

Concluí-se que técnicas de programação dinâmica são ferramentas poderosas para o desenvolvimento de algoritmos.

6. Bibliografia

ZIVIANI, N. Projeto de Algoritmos com implementações em pascal e c. 3ª edição. Cengage Learning.

CORMEN, T. H. Et al. Algoritmos: Teoria e Prática. 3a edição. Elsevier, 2012

C++ reference. Disponível em: < https://en.cppreference.com/w/>. Acesso em: 08 nov. 2019.