Trabalho Prático 2 - Algoritmos I

Luiz Philippe [1]

[1] Departamento de Ciência da Computação – Universidade Federal de Minas Gerais

luizphilippe@dcc.ufmg.br

Abstract. *It is not always easy to envision an efficient solution to some complex problems. Sometimes, even if such a solution exists, it is necessary to visualize the big picture in a different way, and this is the approach of dynamic programming, to reduce a complex problem for which we do not have immediate solution to a set of smaller problems that are easier to solve. The following statement presents a particular case of a problem that can be solved using such strategies, we will see an applicable solution that makes use of dynamic programming and discuss its advantages and disadvantages.*
Palavras-chave: *dynamic programming, knapsack problem, complexity analysis*

Resumo. *Nem sempre é fácil visualizar uma solução eficiente para alguns problemas complexos. Por vezes, mesmo que tal solução exista, é necessário visualizar o cenário de uma forma diferente, e essa é a proposta da programação dinâmica, reduzir um problema complexo para o qual não temos solução imediata a um conjunto de problemas menores mais fáceis de serem resolvidos. O enunciado a seguir apresenta um caso particular de problema que pode ser atacado utilizando tais estratégias, veremos uma solução aplicável que faz uso de progração dinâmica e discorreremos suas vantagens e desvantagens.*
Palavras-chave: *programação dinâmica, problema da mochila, análise de complexidade*

1. Problema

Adriana é uma ourives muito famosa e conhecida por suas excentricidades. Extremamente metódica, sempre que recebe um malote com diamantes realiza o mesmo procedimento antes da criação de suas joias. Ela faz o chamado Jogo dos Diamantes para cada malote. O jogo dos diamantes assume que para uma coleção de pedras, duas pedras quaisquer podem ser "combinadas". Suponha que tenhamos duas pedras com pesos $p1$ e $p2$, com $p1 \geq p2$, o resultado dessa operação pode ser:

1) Se $p1 = p2$, as duas pedras são completamente destruídas.
2) Se $p1 > p2$, a pedra $p1$ sobra com um novo peso $p1 - p2$, e a pedra $p2$ é completamente destruída.

O jogo termina quando resta no conjunto uma, ou nenhuma pedra e o objetivo é que o peso restante seja o menor possível (caso não reste nenhuma pedra, o peso restante é zero).

2. Implementação

Considere o conjunto de 4 diamantes com os respectivos pesos: 3, 6, 3, 1. Uma boa solução seria:

1) Compare as pedras com peso 3 e 6, retorne 3 ao conjunto. (resultando em 3, 3, 1).
2) Compare as pedras com peso 3 e 3, retorne 0 ao conjunto. (resultando em 1).
3) Passo 4: Retorne 1 como solução ótima do problema.

Vejamos agora qual seria uma solução ruim para o problema:

1) Compare as pedras com peso 3 e 3, retorne 0 ao conjunto. (resultando em 6, 1).
2) Compare as pedras com peso 6 e 1, retorne 5 ao conjunto. (resultando em 5).
3) Passo 4: Retorne 5 como solução do problema.

No primeiro caso, o que fizemos foi a operação:

$$(((6-3)-3)-1) = (6)-(3+3+1) = 1$$

De certa forma, podemos dizer que dividimos nosso conjunto inicial de diamantes $\{3,6,3,1\}$ em dois subconjuntos $\{6\}$ e $\{3,3,1\}$, então, retornamos como solução a diferença entre estes dois conjuntos. De forma geral, podemos dizer que, para resolver o problema "qual é o menor peso restante possível podemos combinando os diamantes?", basta resolver o problema "qual é a diferença mínima possível de se obter dividindo um conjunto de valores em dois subconjuntos?".

2.1. Demonstração

Seja D o conjunto finito de diamantes, $n \in \mathbb{Z}$ o número de diamantes, $d_i \in \mathbb{N}$ o i-ésimo elemento de D, $s \in \mathbb{N}$ a soma dos elementos de D e $p \in \mathbb{N}$ o peso mínimo restante possível após a combinação dos diamantes. Sejam D_1 e D_2 dois subconjuntos disjuntos pertencentes a D de tamanhos n_1 e n_2 respectivamente tais que $p = \Sigma_{i=1}^{n_1} d_1 i - \Sigma_{i=1}^{n_2} d_2 i$ sendo d_{1i} o i-ésimo elemento de D_1 e d_{2i} o i-ésimo elemento de D_2. Assuma que a solução ótima seja obtida com a operação:

$$((((d_1-d_2)-d_3)-d_4)...-d_n) = (d_1)-(d_2+d_3+d_4...+d_n)$$

Nesse caso temos que $D_1 = \{d_1\}$ e $D_2 = \{d_2, d_3, d_4...d_n\}$. Note que mesmo rearranjando os termos da equação ainda é possível fazer essa separação, por exemplo:

$$(d_1-d_{k+1})-(d_{k+2}-d_2)...-(d_k-d_n) = (d_1+d_2...+d_k)-(d_{k+1}+d_{k+2}+d_n)$$

Nesse caso teríamos que $D_1 = \{d_1, d_2..d_k\}$ e $D_2 = \{d_{k+1}, d_{k+2}..d_n\}$.

2.2. Construção da solução

Como o problema consiste apenas em determinar o menor peso possível restante, focaremos apenas em descobrir a diferença entre D_1 e D_2 e não seus elementos.

Para isso, recorremos a uma abordagem com programação dinâmica, dividiremos nosso problema em subproblemas mais simples e armazenaremos os resultados até que a resposta para nossa questão seja trivial. Para isso, teremos uma matriz de memorização que funcionará da seguinte forma.

Organizaremos nossos pesos de diamantes em um vetor de inteiros. Nosso melhor resultado possível é 0, isso acontece quando é possível dividir nosso conjunto de diamantes em dois conjuntos exatamente iguais, ou seja, $\Sigma D_1 = \Sigma D_2 = \frac{s}{2}$. Cada célula m_{ij} da nossa matriz $M_{n+1 \times \frac{s}{2}}$ de memorização responderá a pergunta: existe algum subconjunto dos elementos do subvetor que vai de 0 a i dos nossos diamantes cuja soma dos elementos é j? Se a resposta for sim, armazenamos verdadeiro, caso contrário, armazenamos falso. A primeira coluna, onde $j = 0$ é preenchida completamente com verdadeiro, já que para qualquer subvetor podemos obter um suconjunto cuja soma é 0, para isso basta selecionar um subonjunto vazio desse subvetor. Além disso, podemos ignorar a primeira linha, onde $i = 0$, já que a soma de um subconjunto de um subvetor vazio não pode assumir nenhum valor além de 0, assim temos que todas as células dessa linha com exceção da primera são falsas. Nossa terceira condição é que: caso um elemento de uma coluna da matriz tenha resultado verdadeiro, todos os elementos abaixo dele naquela coluna também seram verdadeiros, já que já encontramos um subconjunto do subvetor com soma igual ao j da coluna. Note que aqui temos subproblemas triviais, uma vez que para responder a pergunta "é possível selecionar um subconjunto do subvetor de 0 até i cuja soma é j? ", basta consultarmos a resposta resultado previamente armazenado para a pergunta "é possível selecionar um subconjunto do subvetor de 0 até $i-1$ cuja soma é $j - v_i$? ". Se aplicarmos essa lógica sucessivamente, eventualmente percorremos a matriz M completamente, a partir daí, obter a resposta do nosso problema inicial se torna trivial.

Como queremos obter um subconjunto dentre todos os nossos diamantes, nossa linha de interesse é onde $i = n$. Assim, podemos consultar a coluna $\frac{s}{2}$ da nossa matriz e fazer a pergunta "é possível obter um subonjunto desse vetor com soma $\frac{s}{2}$? ", caso a célula armazene um valor verdadeiro, o valor é obtível e podemos retornar o resultado 0, já que conseguimos dividir os diamantes em dois subgrupos de peso exatamente iguais, caso a resposta seja negativa, verificamos a coluna anterior e nos fazemos uma nova pergunta: "é possível obter um subonjunto desse vetor com soma $\frac{s}{2} - 1$? ", onde retornaríamos 1 em caso positivo, e em seguida $\frac{s}{2} - 2$, onde retornaríamos 2, $\frac{s}{2} - 3$, onde retornaríamos 3 e assim sucessivamente até $\frac{s}{2} - k$ onde retornaríamos k. Em resultado a essas iterações, teremos o valor maios próximo de $\frac{s}{2}$ possível de um subconjunto dos nosso dados, logo a diferença mínima entre dois subconjuntos, e consequentemente, a resposta ao nosso problema.

3. Instruções de compilação e execução

O makefile preparado para a execução do código tem as seguintes opções:

- make: Compila os arquivos de código fonte e armazena a saída da compilação em uma pasta "./build/".
- make run: Executa o arquivo gerado pela compilação.
- make mem: Executa o arquivo gerado pela compilação em modo de depuração de memória, assim podemos nos certificar que não existe nenhum vazamento acontecendo durante a execução.
- make tests: Executa os script de testes.
- make time: Executa o programa realizando medidas de tempo.
- make clean: Limpa a pasta contendo as saídas da compilação.

4. Análise de complexidade

4.1. Complexidade temporal

A leitura das entradas do problema tem complexidade linear $O(n)$, onde n é o número de diamantes presentes no problema.

Com a abordagem adotada podemos reduzir o custo de cada iteração na nossa matriz de memorização a $O(1)$, assim, teremos apenas o custo temporal de percorrer toda a matriz na etapa de resolução dos subproblemas. Logo, seja m o piso da divisão da soma dos pesos dos diamentes por 2, nossa matriz de memorização terá dimensões $(n+1) \times m$, consequentemente, este é o número de iterações realizadas pra preencher toda matriz. Conclui-se então que esta etapa tem complexidade $O(mn)$.

Finalmente, para extrair a resposta do nosso problema central das respostas dos nossos subproblemas, percorremos toda a linha inferior da nossa matriz de memorização com complexidade linear de $O(m)$.

Portanto, todo o procedimento de solução do problema pode ser descrito pela complexidade de $O(n) + O(nm) + O(m) \in O(nm)$. Veja no gráfico abaixo:

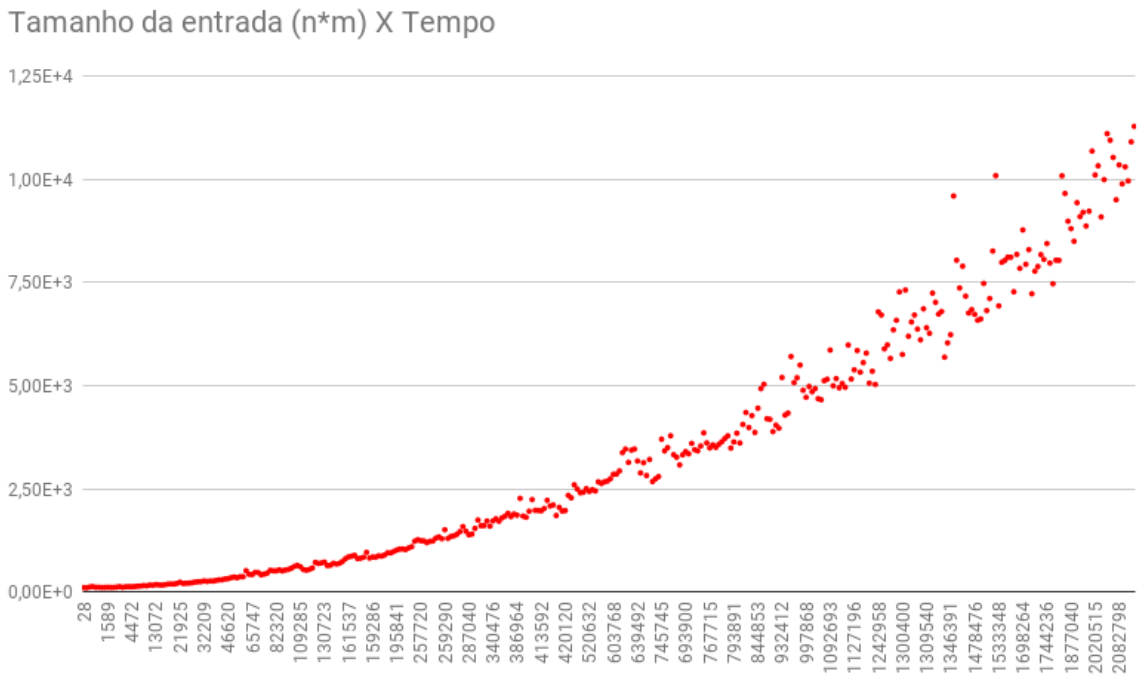

Figura 1. Gráfico de tempo de execução (ms) em função do tamanho da entrada. Mediana de 10 medidas para cada valor de entrada.

4.2. Complexidade espacial

Toda a memória extra utilizada durante a execução é alocada para a matriz de memorização de dimensões $(n+1) \times m$, portanto nossa complexidade temporal é de $O(nm)$.

5. Conclusão

A programação dinâmica e sua estratégia de armazenamento de resultados é uma ferramenta poderosa para resolver problemas complexos reduzindo-os a problemas menos complexos. Embora esta não seja a única solução para o problema é uma das mais performáticas em termos de tempo de execução, muito embora possa apresentar algumas desvantagens em relação a complexidade especial, como é comum em soluções que utilização programação dinâmica devido a necessidade do armazenamento de soluções dos subproblemas.