Relatório Knapsack Problem

Denise F. de Rezende

O Problema da Mochila [1] (ou Knapsack Problem) é um problema de combinatória que consiste em, dados a capacidade de uma mochila e um grupo de itens, cada qual com um peso e um valor, determinar qual combinação de itens gera a maior soma de valores respeitando a capacidade dada.

---

1. Introdução

Existem múltiplas versões do Problema da Mochila. Neste relatório, será abordado o Problema da Mochila 0-1 – também conhecido como Problema da Mochila Binária (PMB) – no qual, para preencher a mochila, tem-se apenas duas opções ao considerar cada item: colocá-lo ou não. Em outras variantes do problema podem ser consideradas partes de cada item.

Como extensão do PMB, múltiplas cópias de cada item podem ser consideradas.

O problema proposto abaixo contempla uma disponibilidade infinita de cada item. O objetivo deste é determinar qual quantidade de cada item devemos colocar na mochila para maximizar seu valor, sem ultrapassar a capacidade desta.

Para solucionar o problema proposto, limitamos a disponibilidade infinita de itens considerando apenas a quantidade máxima de cada um deles que cabe na mochila. Decidimos encarar a multiplicidade de elementos como itens distintos para que possamos resolver esse problema da mesma maneira que o PMB.

---

2. Metodologia

Para a resolução desse problema foram seguidas duas abordagens: uma baseada em Busca Exaustiva, também conhecida como Força Bruta (FB), e outra em Programação Dinâmica (PD) [2]. O método baseado em FB consiste em resolver um problema de combinatória examinando todas as possíveis soluções. Já aquele baseado em PD resume-se na resolução de subproblemas com o armazenamento de suas soluções para que, ao se deparar com um mesmo subproblema, se possa apenas consultar sua solução prévia. Dessa forma, com base em soluções ótimas de subproblemas pode-se buscar calcular uma solução ótima do problema maior. Essa técnica pode ser utilizada sempre que se trata de um problema com certas propriedades (por exemplo, subestrutura ótima).

---

3. Implementação

Foram realizadas duas implementações, cada uma baseada em um dos métodos citados acima. Ambas as implementações foram feitas na linguagem Python 3.9.

A implementação baseada em FB foi feita de forma recursiva. Para cada item, consideramos ambas as alternativas: com o item ou sem ele. Essa implementação dá origem a uma árvore binária implícita [Figura 1] pois, para cada item, há duas possibilidades que devem ser consideradas na busca da melhor solução. Para considerar todas as possíveis combinações nessa árvore é preciso considerar, em cada ramo, se a presença de um item traz a melhor solução. Percebemos que a presença de um item é avaliada múltiplas vezes. Por meio dessa implementação recursiva, uma segunda ocorrência de um subproblema requer seu completo recálculo ao invés de uma mera consulta ao valor anteriormente obtido. A relação de recorrência que descreve essa solução é dada por:

$$M(0,C) = 0$$
$$M(n,0) = 0$$
$$M(n,C) = \max\{v(I_n) + M(n-1,C-w(I_n)), M(n-1,C)\},$$

onde n é o número de itens, C é a capacidade total da mochila, $v(I)$ é o valor do item I, e $w(I)$ é o peso do item I.

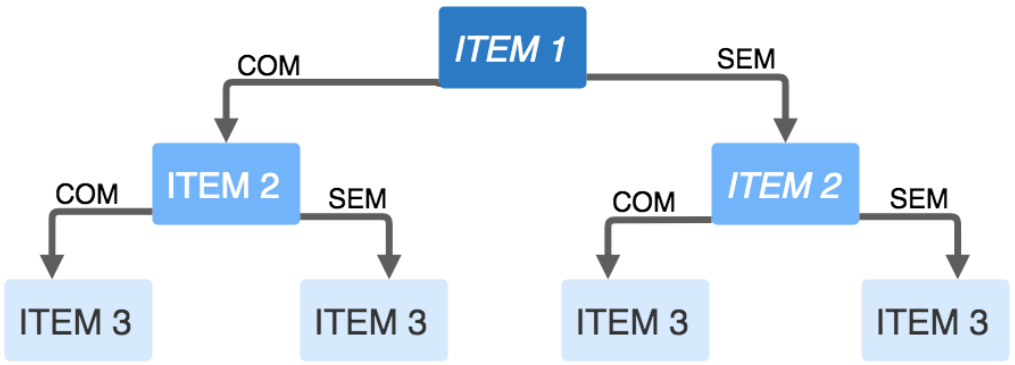

Figura 1: método exaustivo

A implementação baseada em PD foi realizada de forma iterativa com o armazenamento dos resultados de subproblemas em uma tabela T. Cada linha dessa tabela se refere a um item. Logo, se n é a quantidade de itens, temos n linhas. As colunas correspondem às capacidades de $\{0, 1, 2, \ldots, C\}$, onde C é a capacidade total da mochila.

Para completar a tabela, nós a percorremos linha a linha e, para cada célula, consultamos valores já preenchidos na linha anterior. Abaixo veremos como é realizado o preenchimento de cada célula da tabela, que segue a relação de recorrência citada acima:

Primeiramente, preenchermos a linha 1 que corresponde ao primeiro elemento I_1. Se este couber dentro da mochila de capacidade m, sendo m a coluna corrente, o adicionamos. Essa sempre será a melhor solução para a presente célula $T(1,m)$, pois o valor da mochila com um elemento é sempre superior ao valor da mochila vazia.

Para o preenchimento de uma célula $T(k,m)$ de uma outra linha de T, calculamos o valor da mochila com o elemento I_k e o comparamos ao valor da mochila sem I_k. O maior desses valores é preenchido na célula corrente $T(k,m)$.

Para calcular o valor da mochila considerando um elemento I_k, seguiremos os seguintes passos:

1. Adicionamos o item I_k na mochila e subtraímos seu peso $w(I_k)$ da capacidade da mochila m.
2. Para o peso $m–w(I_k)$, consultamos a melhor solução $T(k–1,m–w(I_k))$ previamente calculada.
3. Somamos esse valor $T(k–1,m–w(I_k))$ com o valor do elemento corrente $v(I_k)$.

Para verificar o valor da mochila sem o elemento corrente I_k, consultamos o valor $T(k–1,m)$ da célula imediatamente acima, que representa a solução ótima desse caso.

Assim, iterando linha a linha, completamos a tabela inteira. Ao finalizar, a solução para os n itens dados e uma mochila de capacidade C estará na célula da última coluna e última linha $T(n,C)$.

A seguir apresentamos um exemplo para o caso de 4 itens com multiplicidades considerando uma mochila de capacidade C=6.

	DE 0 A C						
ITENS COM MULTIPLICIDADES	0	1	2	3	4	5	6
Item 1 (PESO=2, V=12)	0	0	12	12	12	12	12
Item 1 (PESO=2, V=12)	0	0	12	12	24	24	24
Item 1 (PESO=2, V=12)	0	0	12	12	24	24	36
Item 2 (PESO=3, V=20)	0	0	12	20	24	32	36
Item 2 (PESO=3, V=20)	0	0	12	20	24	32	40
Item 3 (PESO=4, V=25)	0	0	12	20	25	32	40
Item 4 (PESO=5, V=26)	0	0	12	20	25	32	40

4. Resultados Experimentais

Testes experimentais foram realizados em uma máquina Apple MacBook Pro com 16GB de RAM, processador Intel Core I7 2.6GHz sob MacOS Mojave. Para calcular o tempo de cada execução, foi utilizada a biblioteca `time`, de forma que o tempo medido foi apenas do processo de busca da melhor solução, desconsiderando o tempo de geração de instância.

4.1 Geração das instâncias

Para avaliar o comportamento de ambas as implementações, os casos testes foram separados em duas categorias dependentes da quantidade de itens das instâncias. Em ambas, consideramos instâncias com W distintos pesos gerados aleatoriamente dentro de intervalos especificados mais adiante. Em seguida, computamos a capacidade da mochila para cada instância e, finalmente, obtemos a multiplicidade máxima de cada item. Para determinarmos a capacidade C da mochila, consideramos inicialmente o valor S como a soma dos W pesos dos itens (sem as multiplicidades). Se $S/2$ é maior que o maior dos pesos gerados (P), tomamos $C = S/2$. Caso contrário, tomamos $C = (S+P)/2$.

Para a primeira categoria, tomamos W em {2, 3, 4, …, 11} e para a segunda, consideramos W em {15, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

Cada instância foi construída utilizando-se os seguintes intervalos de pesos:

- 1ª Categoria: os pesos variaram de 10 até $8W$+10.
- 2ª Categoria: os pesos variaram de 10 até $9W$.

Uma vez determinados os W pesos distintos e a capacidade C da mochila, completamos a construção de uma instância, gerando Piso[$C/w(I_k)$] múltiplas cópias de cada item I_k, para k entre 1 e W. A enorme quantidade de itens gerados em razão desse número máximo de múltiplas cópias de cada item teve um grande impacto nos tempos de execução como veremos na seção 4.3.

4.2 Execuções dos testes

Para os testes com a busca exaustiva, foi estabelecido um limite superior de tempo de duas horas para cada instância. Após esse período, o programa abortava a resolução, e devolvia a melhor solução encontrada até aquele momento.

Nos testes realizados com a programação dinâmica, o tempo máximo para todas as instâncias utilizadas nunca excedeu o limite de tempo de duas horas.

Para as instâncias da Categoria 1, foram feitas

- 10 execuções da PD e 10 da FB para $2 \leq W \leq 10$.
- 10 execuções da PD para W=11.
- 3 execuções da FB para W=11.*

*Fizemos apenas três, pois essas excederam o tempo limite e consideramos que isso já indicava o comportamento da FB para essa quantidade de pesos distintos.

Para a Categoria 2 em que temos W entre 15 e 100, foram feitas 10 execuções da PD e nenhuma da FB, pois neste caso já sabíamos que essas instâncias eram inviáveis para a FB dentro do limite de tempo.

4.3 Análise dos resultados

Para apresentar os tempos de execução dos testes realizados, usamos os gráficos abaixo. Esses mostram os tempos médios das implementações para cada número W de pesos distintos.

Na Figura 2, representamos as instâncias da Categoria 1 com o método FB. A seguir, na Figura 3, focamos nas instâncias da Categoria 1 com o método PD.

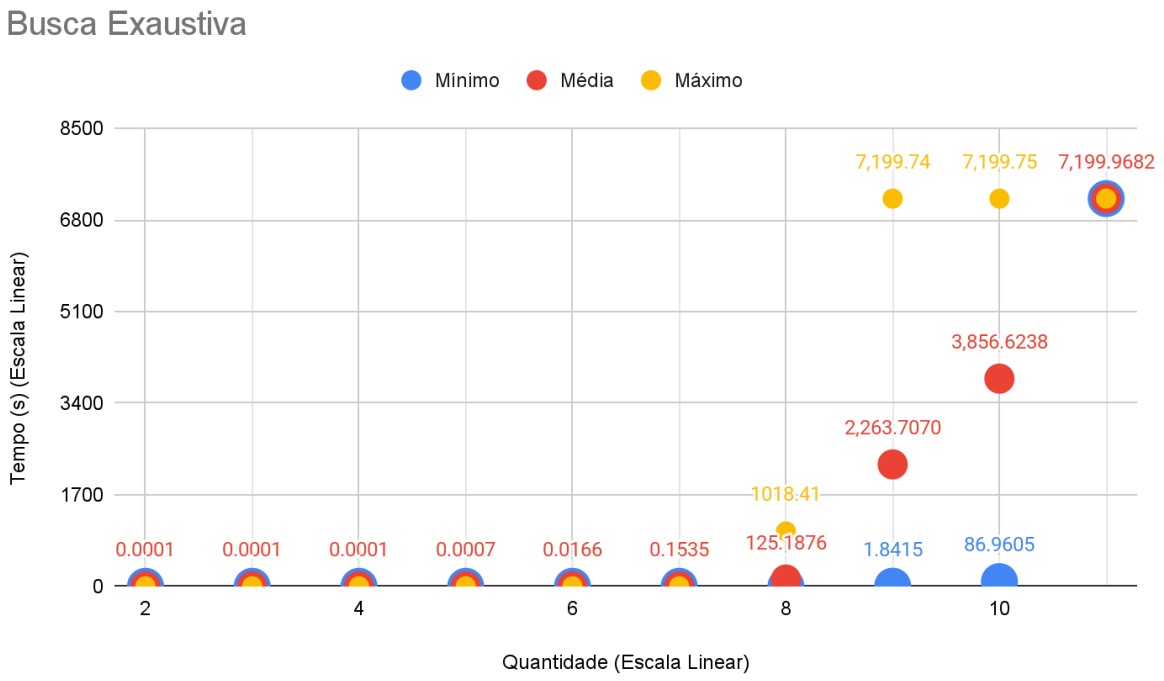

Figura 2: Tempos médios de execução do método FB, por valor de W, para a Categoria 1 de instâncias.

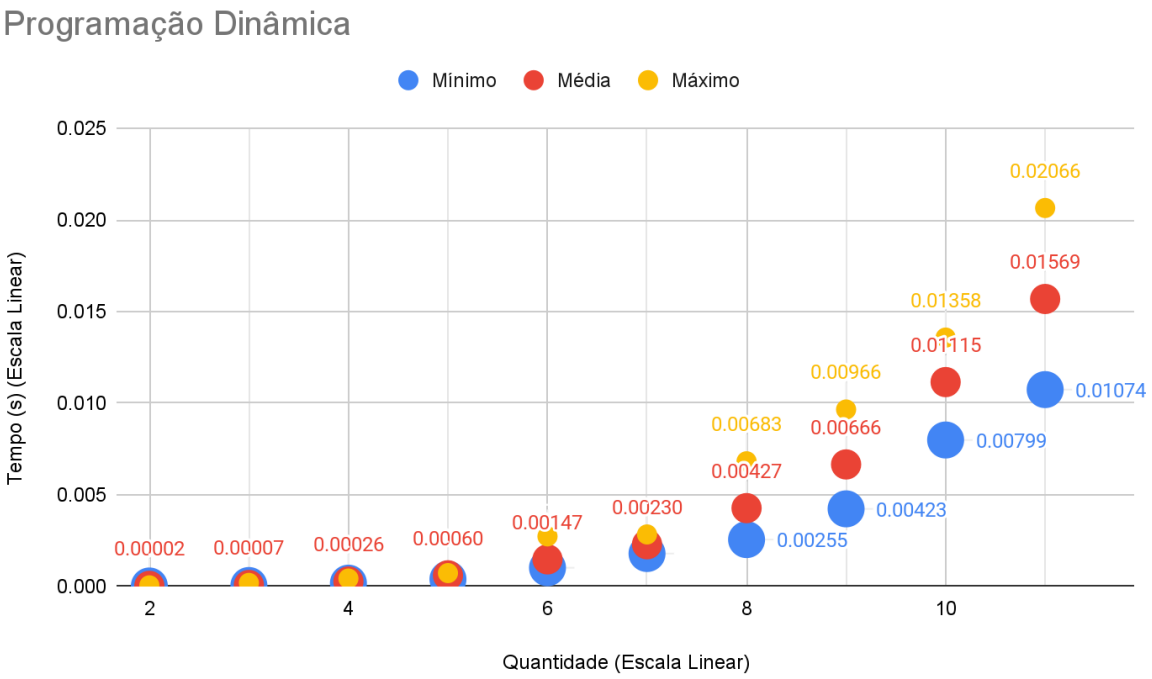

Figura 3: Tempos médios de execução do método PD, por valor de *W*, para a Categoria 1 de instâncias.

Analisando ambos os gráficos acima notamos que o tempo de execução para achar uma solução ótima é extremamente superior na implementação baseada na FB. Essa drástica diferença se deve aos inúmeros recálculos de subproblemas que essa implementação faz para analisar todos os casos. A implementação baseada na FB é eficaz pois ela resolve corretamente o problema, se não for interrompida. Por outro lado, pelos gráficos comparando a implementação baseada na PD com a baseada na FB notamos que ela não é eficiente. Já a implementação baseada na PD é tão eficiente quanto eficaz, pois ela não recalcula os subproblemas, apenas os consulta. Por isso, a implementação feita conforme a PD é efetiva.

Na figura 4, temos a representação gráfica da Categoria 2.

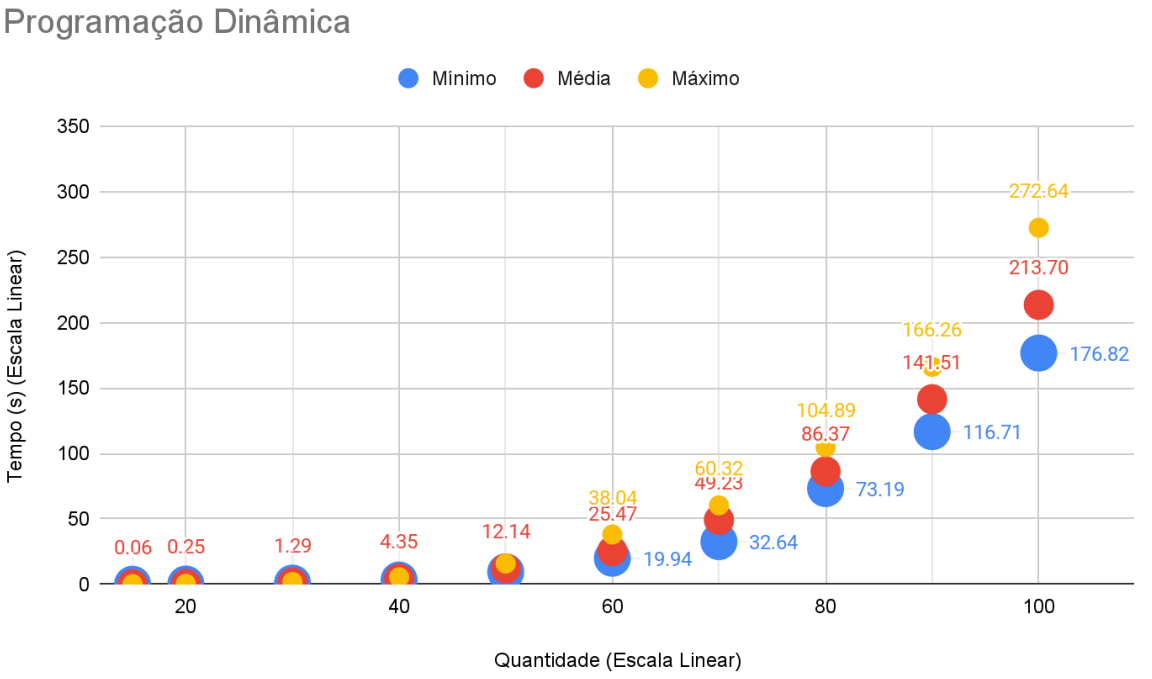

Figura 4: Tempos médios de execução do método PD, por valor de *W*, para a Categoria 2 de instâncias.

O eixo horizontal do gráfico representa apenas a quantidade de pesos distintos W e não a quantidade de itens após a realização da multiplicidade desses. O número de itens n é muito superior a W. Experimentalmente notamos que para W = 100, n variava de 400 a 1500. Na programação dinâmica é necessário montar uma tabela T [$(C+1)$ x n]. Como já fizemos tabelas para $n=1500$ e C=45.000, criamos aproximadamente 67.500.000 células. Por isso, julgamos inviável fazer testes com W superior a 100.

Notamos que quanto maior o W maior era o n e, por isso, mais significativo era o tempo gasto para resolver o Problema da Mochila. Ainda assim, considerando o valor de n, o tempo gasto para achar uma solução ótima era satisfatório.

---

5. Conclusão

Como consequência da análise dos testes, percebemos que, especialmente para maiores quantidades de pesos distintos W, não convém utilizar o método baseado na FB. Já o método baseado na PD, convém ser utilizado para valores de $W \leq 100$, já que, devido às multiplicidades, n pode atingir 1500. Ademais, o método baseado na PD também resolve de forma efetiva outros problemas da Mochila Binária com até 1500 itens.

---

6. Referências

[1] O Problema da Mochila, disponível em: https://pt.wikipedia.org/wiki/Problema_da_mochila. Acessado em 24 de maio de 2021.

[2] Programação Dinâmica, disponível em: https://www.youtube.com/watch?v=xCbYmUPvc2Q. Explicação didática em inglês da PD. Acessado em 14 de maio de 2021.
