Análise do Problema da Mochila 0 -1

João C. Coelho Filho, Norma A. M. Tovar Uribe

Departamento de Ciência da Computação – Universidade Federal de Roraima
Boa Vista - RR - Brasil.

jccoelho@me.com, natovar@uniboyaca.edu.co

Abstract. *This article presents solutions to a computational problem known as "Knapsack Problem 0-1", which have a set of elements with weight and value for each one and it looks for a subset of this elements with a higher value without exceeding the weight limit of the knapsack. The first solution uses an explicit list, where all possible possibilities are calculated. The second solution uses dynamic programming. In this work shown with empirical tests that the solution with dynamic programing have better performance of tempo that explicit enumeration. For finish it presents the knapsack problem solution applied in downloads management.*

Resumo. *Este artigo apresenta duas soluções ao problema computacional conhecido como "Problema da Mochila 0-1", no qual dado um conjunto de itens com peso e valor, busca-se encontrar um subconjunto de itens com o maior valor possível sem exceder um dado limite de peso. A primeira solução utiliza a enumeração explícita, na qual todas as possibilidades são calculadas, a segunda utiliza programação dinâmica. É demonstrado com testes empíricos que a solução com programação dinâmica atinge melhor performance de tempo que a enumeração explícita. Apresenta-se ainda uma possível aplicação prática do problema da mochila 0-1, aplicado a um gerenciador de downloads.*

1. Introdução

O problema da mochila 0-1 é a variação NP-Completude do problema da mochila tradicional, como é descrito em (Martello, Pisinger, & Toth, 2000) o problema da mochila é considerado quando se tem N itens de um grupo para posiciona-os em uma mochila de capacidade finita W, cada item tem um valor *v* e um peso *w* e o problema consiste em selecionar a quantidade de itens que ocupam mais espaço na mochila e também que a soma dos seus valores seja o máximo.

Definindo o problema formalmente, dados dois n-grupos de números positivos $\{v_1, v_2, \dots, v_n\}\ e\ \{w_1, w_2, \dots, w_n\}$ y uma capacidade W, se deseja determinar um subconjunto $T \in \mathbb{N}$ de itens tal que maximiza:

$$\sum_{i \in T} v_i \ sujeto\ a \sum_{i \in T} w_i \leq W$$

Existem várias aplicações onde se dá o problema da mochila, uma das aplicações mais importantes é a geração de chaves em um problema de criptografia simétrica, proposto por Merkle-Hellman em 1976, citado por (Tomás, 2013), o

problema é baseado em que, dado um número k, e um conjunto de números C, de deve maximizar o custo em termos computacionais para somar k com e números de C e, assim, assegurar uma comunicação segura.

Entre outras aplicações, estão a seleção de oportunidades de investimento, que tem o orçamento como limitação; desperdício da menor quantidade de tecido, que tem o material como limitante; aproveitar ao máximo o uso de máquinas, que tem o tempo como limitação e, claro, o gerenciamento de arquivos por um gerenciador de downloads, cuja limitação é a largura de banda da rede.

O objetivo deste trabalho é aplicar dois diferentes soluções aproximadas ao problema da mochila 0-1 e analisar seu comportamento frente a diferentes quantidades de entradas. Uma comparação entre os métodos de Força Bruta e Programação Dinâmica será feita em termos do tempo de execução e se comparará sua de complexidade. Finalmente, a melhor solução para a aplicação do gerenciador de downloads será aplicada nele.

2. Algoritmos

2.1 Força Bruta

O algoritmo de força bruta é aquele que compara todas as possibilidades para encontrar a melhor solução entre elas. No caso do problema de mochila, o algoritmo procurará todos os subconjuntos que não excedam a capacidade da mochila e comparará o valor total de cada subconjunto. Para a solução, retornará o subconjunto que obteve o maior valor final.

```
1. Knapsack( w, v, n, W)
2. Se n = 0
3.         Então devolva 0
4. Senão A = Knapsack( w, v, n, W)
5.         Se w[n] > W Então
6.                 Devolva A
7.                 Senão B = w[n] + Knapsack(w, v, n-1, W-p[n])
8.                         Devolva max(A, B)
```

Figura 1. Pseudocódigo de Força Bruta para o Problema da Mochila

No problema da mochila 0-1 um elemento pode estar ou não na mochila, portanto teremos dois estados para cada elemento, e isto quer dizer que teremos 2n combinações possíveis para escolher a melhor, além disso, a função calcula os mesmos subproblemas quantas vezes encontrar eles de novo. A complexidade desta solução é exponencial (2^n)

2.2 Programação Dinâmica

De acordo com a definição de programação dinâmica dada pelo professor (Oliveria, 2018) a programação dinâmica é uma variação da técnica de decomposição, ela é basicamente uma lista de soluções que busca, através da abordagem de divisão e

conquista, minimizar a quantidade computacional para ser usada na solução do problema. Esse paradigma busca resolver os subproblemas apenas uma vez, de modo que, se o mesmo subproblema reaparecer na divisão, o algoritmo simplesmente extrai a solução da memória, caso precise ser resolvida novamente.

A programação dinâmica fornece uma solução exata ao problema da mochila, ela permite resolver a maioria das operações em menor tempo que a versão por fora bruta. A continuação se expõe o pseudocódigo do caso.

```
1.  Knapsack( w, v, n, W)
2.  For w = 0 to W faça
3.          K[0, w] = 0
4.  For i = 0 to n faça
5.          K[i, 0] = 0
6.  For i = 1 to n faça
7.          For w = 1 to W faça
8.          Se w[i] <= w  Então
9.                  Se v[i] + K[i - 1, w - w[i]] > c[i - 1, w] Então
10.                         K[i, w] = v[i] + K[i - 1, w - w[i]]
11.                 Senão K[i, w] = c[i - 1, w]
11.         Senão K[i, w] = K[i - 1, w]
```

Figura 2. Pseudocódigo de Programação Dinâmica para o Problema da Mochila

Seguindo o pseudocódigo ao definir a relação de recorrência que expressa a solução do problema em subproblemas, se tem a seguinte expressão:

$$K_{i,j} = \begin{cases} K_{i-1,j} & ,j < w_i \\ max(K_{i-1,j} p_i + K_{i-1,j-w_i}) & ,j \geq w_i \end{cases}$$

Analisando o pseudocódigo, se tem na linha 6 um laço n, e na linha 4 se tem outro laço aninhado W que incorpora n repetições, logo neste algoritmo a complexidade é de O(n*W), devido a que todos os arranjos de itens são testados em todas as possibilidades das capacidades.

3. Testes

Os testes a seguir foram baseados e executados no seguinte ambiente:

- Hardware: 2,4 GHz Intel Core i7 (Quad-Core), cache 246KB (por núcleo), 8GB DRR3 1600MHz de RAM.
- Sistema Operacional: MacOS 10.13
- Compilador: Clang 900.0.38

A base de dados para os testes consiste em 1 milhão de pares numéricos, que correspondem a pesos (gerados randomicamente entre 0 e 1000) e valores (gerados

randomicamente entre 0 e 100). Todos os testes abaixo foram executados com a mochila com capacidade máxima de 1000.

3.1 Execução do algoritmo por Força Bruta

A versão por força bruta para a resolução do problema apresentou comportamento exponencial, como previsto. Em nossos testes foi inviável calcular a resposta para mais de 200 itens, pois o cálculo passou a levar mais de 60 minutos.

Tempo de Execução

QTD ITENS	TEMPO (SEG)
10	0,00023
50	0,00370
100	1,22560
110	1,48130
120	3,32440
130	4,88530
140	11,05640
150	32,97510
160	136,82430
170	232,79380
180	315,27810
190	1007,23860
200	2923,09820

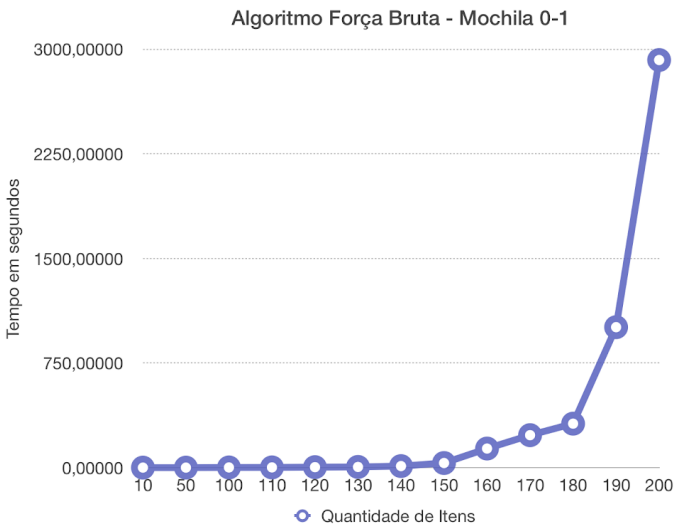

Figura 3. Execução do algoritmo de Força Bruta

3.2 Execução do algoritmo por Programação Dinâmica

Com o mesmo conjunto de dados como entrada, porém em quantidade de itens progressivamente maiores, encontramos os seguintes resultados:

Tempo de Execução-1

QTD ITENS	TEMPO (SEG)
10000	1,05350
20000	2,67370
30000	3,48130
40000	4,87890
50000	5,98520
60000	6,34250
70000	6,96300
80000	7,28720
90000	9,43980
100000	10,89870
110000	11,64600
120000	25,83920
130000	274,98780

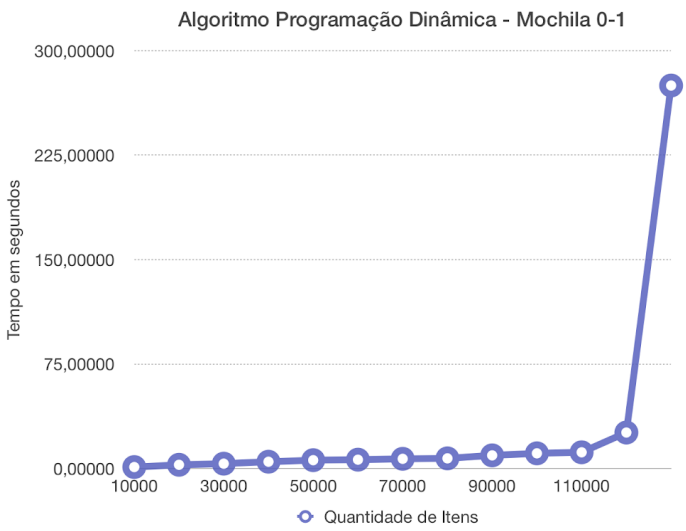

Figura 4. Execução do algoritmo de Programação Dinâmica

Verifica-se que o algoritmo por programação dinâmica apresenta um resultado mais satisfatório que a força bruta, entregando a resposta em tempo razoável para conjuntos de dados com até 120.000 itens, o que pode ser útil para diversas aplicações que necessitem de uma resolução exata, porém mais eficiente que a força bruta.

4. Aplicação: Gerenciador de downloads

Um gerenciador de downloads é um aplicativo que gerencia a maneira como os arquivos são organizados para enviá-los e aproveitar o máximo de espaço possível dentro da largura de banda da rede, levando em conta a prioridade do arquivo. Existem administradores de download que utilizam uma solução aproximada para o problema da mochila para a distribuição de espaço, neste trabalho apresentamos uma amostra de como esta distribuição será dada.

No algoritmo, os tamanhos dos fragmentos dos arquivos são tomados como pesos, a prioridade de cada fragmento representará seu valor e será dada de 1 a 3 do menor para o maior correspondentemente, e o tamanho da largura de banda da rede será a capacidade da mochila, que para este caso será utilizada a largura de banda comum para a Ethernet que é de 10Mbit/s[1]..

As seguintes entradas mostradas são aplicadas em um algoritmo de programação dinâmica que extrai os elementos a serem enviados até que não reste nada no vetor, levando-se em conta que o tamanho de um fragmento nunca será maior que a capacidade da mochila.

Entrada: A entrada e um arquivo de 147 Mbit fragmentado da forma como se mostra no vetor Peso y os respetivos valores de acordo com a importância do fragmento.

$$W = 10 \text{ Mbit}$$
$$\text{Peso (Mbit)} = \{1, 5, 7, 10, 5, 6, 2, 7, 3, 8, 6, 4, 7, 5, 9, 6, 4, 5, 1, 7, 6, 4, 6, 8, 2, 9, 1, 3\}$$
Valor = {1, 2, 1, 2, 3, 2, 3, 1, 2, 1, 3, 2, 1, 3, 1, 2, 2, 1, 3, 3, 3, 2, 1, 1, 2, 2, 3, 1}

[1] Dado fornecido por: https://es.wikipedia.org/wiki/Ancho_de_banda_(informática)

Saída

Tabela 1. Saída do algoritmo do gerenciador de descargas

N	Quantidade de elementos para enviar	Posição do elemento no vector atual[2]	Peso do elemento (Tamanho)	Valor do elemento (Prioridade)	Peso total dos elementos enviados	Valor total do envio (Suma das prioridades)
1	28	26	1	3	10	14
		24	2	2		
		18	1	3		
		8	3	2		
		6	2	3		
		0	1	1		
2	22	10	5	3	10	6
		3	5	3		
3	20	7	4	2	10	5
		6	6	3		
4	18	12	6	3	10	5
		9	4	2		
5	16	11	4	2	9	4
		0	5	2		
6	14	13	3	1	10	4
		9	7	3		
7	12	1	10	2	10	2
8	11	1	6	2	6	2
9	10	5	6	2	6	2
10	9	8	9	2	9	2
11	8	0	7	1	7	1
12	7	0	7	1	7	1
13	6	0	8	1	8	1
14	5	0	7	1	7	1
15	4	0	9	1	9	1
16	3	0	5	1	5	1
17	2	0	6	1	6	1
18	1	0	8	1	8	1

[2] Tenha em conta que cada vez que o algoritmo envia elementos, ele exclui aqueles elementos e deixa o vector com os elementos que falta, por tanto, as posições dos elementos que ficam se acomodam ao espaço, es dizer que as posições mostradas na tabela na são as mesmas posições deles no vector inicial senão as posições deles no vector dos elementos que falam por enviar.

De acordo com a Tabela 1 o algoritmo envia em total o arquivo em 18 pedaços, ele tenta maximizar o valor tendo em conta não ultrapassar o tamanho limite assim, tenta enviar primeiro os elementos que têm mais prioridade. Na posição 5 da Tabela 1 se pode observar que o tamanho total enviado nesta vez é de 9 e que a seguinte é de 10, isto é, devido a que o algoritmo não é uma versão exata para solucionar este tipo de problema e as vezes se podem apresentar situações donde ele não utiliza os valores mais adequados para otimizar o tamanho.

5. Conclusão

Este artigo apresentou dois algoritmos para a resolução do problema da mochila 0-1, o primeiro utilizando a força bruta, com o cálculo explícito de todas as possibilidades e o segundo utilizando programação dinâmica, no qual reutilizam-se soluções já calculadas previamente.

Verificou-se que a utilização da programação dinâmica para a resolução do problema proporciona um aumento significativo do limite de tamanho de entrada, ocasionando mais flexibilidade e eficiência no cálculo de respostas para entradas maiores.

Também foi verificado que a solução por programação dinâmica pode ser utilizada para a instância do problema da mochila aplicada a um gerenciador de downloads que considera largura de banda e prioridade, no qual foram obtidos resultados satisfatórios.

Referências

Martello, S., Pisinger, D., & Toth, P. (2000). New trends in exact algorithms for the 0±1 knapsack problem. *European Journal of Operational Research*, 325-332.

Oliveria, H. (6 de junio de 2018). Programação Dinâmica. *Análise de Algoritmos*. Boa Vista, Roraima, Brasil: Universidade Federal de Roraima.

Tomás, B. (octubre de 2013). *Problema de la mochila*. Obtenido de http://materias.fi.uba.ar/7114/Docs/ProblemaMochila.pdf