MAC 5711 - Análise de Algoritmos

Departamento de Ciência da Computação

Primeiro semestre de 2004

Problema da Mochila

O problema da mochila (*knapsack problem*) pode ser enunciado da seguinte maneira.

Problema Mochila (m, n, v, w): *Dados um inteiro* $m \geq 0$, *um inteiro* $n > 0$ *e, para cada* i *em* $\{1, \ldots, n\}$, *um inteiro* $v_i \geq 0$ *e um inteiro* $w_i > 0$, *encontrar um subconjunto* S *de* $\{1, \ldots, n\}$ *que maximize* $v(S)$ *sob a restrição* $w(S) \leq m$.

Nessas notas, $v(S)$ denota a soma $\sum_{i \in S} v_i$ e, analogamente, $w(S)$ denota a soma $\sum_{i \in S} w_i$.

Os números v_i e w_i podem ser interpretados como valor e peso respectivamente de um objeto i. O número m pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta. O objetivo do problema é então encontrar uma coleção de objetos a mais valiosa possível que respeite a capacidade da mochila.

Uma abordagem de programação dinâmica resolve o Mochila: basta construir uma tabela V onde $V(i, j)$ é o valor máximo de um subconjunto de $\{1, \ldots, i\}$ cujo peso é no máximo j:

$$V(i, j) := \max\{v(S) : S \subseteq \{1, \ldots, i\}, w(S) \leq j\}.$$

Aqui i varia de 0 a n e j de 0 a m. O valor $V(i, j)$ pode ser expresso a partir da seguinte recorrência:

$$V(i,j) = \begin{cases} 0 & \text{se } i = 0; \\ V(i-1, j) & \text{se } i > 0 \text{ e } w_i > j; \\ \max\{V(i-1, j), v_i + V(i-1, j - w_i)\} & \text{se } i > 0 \text{ e } w_i \leq j. \end{cases}$$

Os dois casos para $i > 0$ são fáceis de entender: ou o objeto i aparece num subconjunto de $\{1, \ldots, i\}$ cujo peso é no máximo j e cujo valor é máximo ou não. Se ele não aparece, o valor de S é $V(i-1, j)$. Se ele aparece, seu valor é $v_i + V(i-1, j - w_i)$.

Eis o algoritmo de programação dinâmica derivado desta recorrência:

Algoritmo Mochila-PD (m, n, v, w)

para j de 0 a m faça $V[0, j] \leftarrow 0$
para i de 1 a n faça
    $V[i, 0] \leftarrow 0$
    para j de 1 a m faça
        se $w_i > j$
            então $V[i, j] \leftarrow V[i-1, j]$
            senão $V[i, j] \leftarrow \min\{V[i-1, j], v_i + V[i-1, j - w_i]\}$
seja S um subconjunto de $\{1, \ldots, n\}$ com $w(S) \leq m$ e $v(S) = V[n, m]$
devolva S

O algoritmo Mochila-PD consome tempo $\Theta(nm)$. Infelizmente, m pode não ser polinomial no "tamanho da instância" (por tamanho da instância, entenda o tamanho do arquivo texto contendo os dados que descrevem aquela instância). Por exemplo, se $n = 1$ e $m = w_1 = v_1$, o algoritmo consumirá tempo proporcional a v_1 enquanto que o tamanho da entrada é proporcional a $\log v_1$, já que o número de dígitos necessários para representar v_1 é $O(\log v_1)$.

A bem da verdade, o Mochila é um problema NP-difícil, o que significa, informalmente, que é muuuuito improvável que exista um algoritmo polinomial para o Mochila. Veremos um pouco mais sobre isso no final deste curso.

Algoritmos como o Mochila-PD são chamados de *pseudo-polinomiais*: se os números no arquivo de entrada fossem representados não da forma usual mas na base unária, o algoritmo consumiria tempo polinomial no tamanho do arquivo de entrada.

Exercício 1: Como você implementa a linha 8 do algoritmo? O trecho de código que faz isso consome quanto tempo?

Exercício 2: A versão acima do algoritmo consome espaço $\Theta(nm)$. Escreva uma variante do algoritmo que consome espaço $O(m)$ para calcular apenas o valor ótimo, ou seja, o valor de $V(n,m)$.

Exercício 3: Suponha que estamos interessados em instâncias do problema da mochila para as quais a capacidade da mochila é não é muito grande quando comparada ao número de objetos disponíveis. Por exemplo, se $m = O(n^c)$ para alguma constante $c > 0$. Conclua que o algoritmo Mochila-PD é polinomial para este tipo de instância. Como o tempo do algoritmo depende do valor de c? (É independente? Depende linearmente do valor de c? É inversamente proporcional?)

Exercício 4: Considere uma instância (m,n,v,w) do Mochila. Seja $W_{i,j}$ o peso mínimo de um subconjunto de $\{1,\ldots,i\}$ cujo valor é pelo menos j. Escreva uma recorrência para $W_{i,j}$ e use-a para projetar um outro algoritmo de PD para o Mochila. Qual a complexidade do algoritmo resultante?

Exercício 5: Suponha que estamos interessados em instâncias do problema da mochila para as quais os valores dos objetos são pequenos quando comparados à capacidade da mochila. Por exemplo, para alguma constante $c > 0$, considere instâncias em que todo v_i é limitado por $c \log m$. Mostre que um algoritmo polinomial para este tipo de instância. (**Dica:** Use o algoritmo do exercício anterior!) Como o tempo do seu algoritmo depende do valor de c? (É independente? Depende linearmente do valor de c? É inversamente proporcional?)

Exercício 6: Suponha que estamos interessados em instâncias do problema da mochila para as quais os valores dos objetos vêm de um conjunto finito pré-determinado de valores. Por exemplo, o nosso ladrão arrombou um cofre cheio de barras de ouro, prata e outros materiais, cada uma com um valor entre, digamos, cinco possíveis valores diferentes. Descreva um algoritmo polinomial para este tipo de instância do problema da mochila. Seu algoritmo continua polinomial se substituirmos cinco por um outro número fixo mas arbitrário?

Versão fracionária do problema da mochila

Nesse outro contexto do problema, é mais adequado usar a palavra *item* em vez de *objeto*. Imagine que os objetos, ou melhor, itens são, por exemplo, café, farinha, feijão, etc. Uma variante do problema da mochila permite que se escolha não **quais** itens levar na mochila mas **quanto** de cada item levar. Formalmente, a versão **fracionária** do problema da mochila pode ser enunciada da seguinte forma.

> **Problema** Mochila-Frac(m,n,v,w): *Dados um inteiro $m \geq 0$, um inteiro $n > 0$ e, para cada i em $\{1,\ldots,n\}$, um inteiro $v_i \geq 0$ e um inteiro $w_i > 0$, encontrar um vetor x indexado por $\{1,\ldots,n\}$ cujas componentes estão no intervalo* $[0,1]$ *e que maximize $v \cdot x$ sob a restrição $w \cdot x \leq m$.*

Lembre-se que $x \cdot y$ denota o produto interno de dois vetores: $x \cdot y := \sum_i x_i y_i$.

Se x é uma solução do problema, os números x_i podem ser interpretados como quantidades. Por exemplo, $x_i = 0{,}8$ significa levar na mochila 80% do total disponível do item i. O objetivo dessa versão do problema é o mesmo de antes, porém agora podemos levar apenas "uma parte" de um item.

Vimos como programação dinâmica pode ser usada para resolver a versão original do problema da mochila. A versão fracionária será resolvida através de uma outra técnica de projeto de algoritmos: o método guloso. Esse método consiste em resolver o problema iterativamente, fazendo, a cada iteração, a melhor escolha possível. No caso particular do Mochila-Frac, o método guloso consiste em escolher repetidamente o item disponível mais vantajoso: aquele cuja razão entre v_i e w_i é máxima. Eis o algoritmo resultante.

Algoritmo MOCHILA-FRAC-G (m, n, v, w)

```
1   para i de 1 a n faça x_i ← 0
2   enquanto x_i = 0 para algum i e w · x < m faça
3       seja i tal que x_i = 0 e v_i/w_i é máximo
4       se w_i ≤ m − w · x
5           então x_i ← 1
6           senão x_i ← (m − w · x)/w_i
7   devolva x
```

Note que o vetor x produzido pelo algoritmo acima tem no máximo uma componente fracionária. Além disso, claramente $x_i \in [0, 1]$ para todo i.

Vamos mostrar que x é de fato uma solução do MOCHILA-FRAC. Primeiro, note que $x_i \in [0, 1]$ para todo i e $w \cdot x \leq m$ (basta verificar que isso vale no fim de cada iteração do enquanto da linha 2), ou seja, x satisfaz as condições impostas no enunciado do problema. Seja z uma solução do MOCHILA-FRAC que difira de x no menor número possível de componentes. Se z não difere de x em nenhuma componente, então $z = x$ e portanto x é uma solução do MOCHILA-FRAC. Suponha agora que z difere de x em alguma componente. Seja i uma componente em que z e x diferem com v_i/w_i máximo. Pela construção de x, temos que $z_i < x_i$. Se $v_i = 0$ então o vetor z' obtido de z pela substituição de z_i por x_i é uma solução do problema que difere de x em menos componentes que z, uma contradição. Então $v_i > 0$ e, como z tem valor máximo, existe alguma outra componente j de z onde $z_j > x_j$. Note que $v_j/w_j = v_i/w_i$, do contrário, z não seria solução do problema: poderíamos alterar z obtendo um outro vetor que satisfaria as condições do problema e teria valor melhor que z. Mas então podemos obter uma outra solução do problema que difere de x em menos componentes que z, uma contradição.

Este algoritmo executa no máximo n vezes as linhas 2 a 6, já que, a cada vez, x_i recebe um valor positivo. Podemos implementá-lo de forma que ele consuma tempo $O(n \log n)$. (Por exemplo, podemos montar um *heap* com os itens, com o valor de v_i/w_i como chave. O custo de montar tal *heap* é $O(n)$ e, a partir dele, cada execução das linhas 2 a 6 consome tempo $O(\log n)$. Como no máximo as linhas 2 a 6 são executadas n vezes, o custo total do algoritmo é $O(n \log n)$, ou seja, este é um algoritmo polinomial.

Algoritmos gulosos são geralmente simples e muitas vezes eficientes.

Exercício 7: Considere a seguinte instância do problema da mochila: $m = 50$, $n = 3$, $v = (60, 100, 120)$ e $w = (1, 20, 30)$. Exiba uma solução ótima do MOCHILA para esta instância. Esta instância é também uma instância do MOCHILA-FRAC. Exiba uma solução ótima do MOCHILA-FRAC para esta instância.

Exercício 8: Muitas vezes, quando é improvável que exista um algoritmo eficiente para um problema de otimização (mais precisamente, quando o problema é NP-difícil), nos satisfazemos com algoritmos que produzam objetos que são *quase* soluções do problema. Concretamente, para o problema da mochila, uma "quase-solução" é um conjunto de objetos que respeita a capacidade da mochila e tem valor *quase* máximo. Mais precisamente, para um número α no intervalo $[0, 1]$, um algoritmo é uma **α-aproximação** para o problema da mochila se ele produz um subconjunto S de $\{1, \ldots, n\}$ tal que $w(S) \leq m$ e $v(S) \geq \alpha\, v(S^*)$, onde S^* é uma solução do problema. Se, por exemplo, $\alpha = 0{,}9$, tal algoritmo produz um conjunto de objetos cujo valor é 90% do valor de uma solução do problema. Pergunta: a variante natural do algoritmo MOCHILA-FRAC-G para a versão original do problema da mochila, ou seja, o MOCHILA, é uma α-aproximação para algum $\alpha > 0$?