MCTA003-17 - ANÁLISE DE ALGORITMOS
CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO
UNIVERSIDADE FEDERAL DO ABC
PROF. MAYCON SAMBINELLI

Programação Dinâmica

1. No problema da MOCHILA INTEIRA recebemos uma coleção de itens $\mathcal{I}$, onde cada item $i \in \mathcal{I}$ tem um valor v_i e um peso w_i, e a capacidade $W \in \mathbb{N}$ de uma mochila. O objetivo do problema é selecionar um conjunto $S \subseteq \mathcal{I}$ de itens tais que o peso desses itens respeite a capacidade de carga da mochila (i.e., $\sum_{i \in S} w_i \leq W$) e cujo valor $\sum_{i \in S} v_i$ seja máximo. Perceba que o problema não permite pegar frações dos itens, isto é, só é permitido adicionar à mochila itens por inteiro.

Se S é uma solução viável para o problema da MOCHILA INTEIRA, então definimos $v(S) = \sum_{i \in S} v_i$ e $w(S) = \sum_{i \in S} w_i$. Note que uma instância do problema da MOCHILA INTEIRA pode ser definido pelo conjunto de itens $\mathcal{I}$ (junto de todos os valores associados aos itens) e a capacidade da mochila W. Assim, uma instância desse problema pode ser vista como um par $(\mathcal{I}, W)$. Assim, denotamos por $\phi_{\mathcal{I},W}$ o custo de uma solução ótima para uma instância $(\mathcal{I}, W)$.

(a) Suponha que S^* seja uma solução ótima para uma instância $(\mathcal{I}, W)$ do problema da MOCHILA INTEIRA, i.e., $v(S^*) = \phi_{\mathcal{I},W}$. Prove que

$$c(S^* \setminus \{i\}) = \phi_{\mathcal{I}\setminus\{i\},W-w_i}.$$

(b) Suponha que S^* seja uma solução ótima para uma instância $(\mathcal{I}, W)$ do problema da MOCHILA INTEIRA. Prove que, para todo $i \in S^*$, vale que

$$\phi_{\mathcal{I},W} = \phi_{\mathcal{I}\setminus\{i\},W-w_i} + v_i.$$

(c) Prove que

$$\phi_{\mathcal{I},W} = \max_{i \in \mathcal{I}} \{\phi_{\mathcal{I}\setminus\{i\},W-w_i} + v_i\}.$$

(d) Escreva um algoritmo recursivo para o problema da MOCHILA INTEIRA. Além de retornar o valor da solução ótima, o seu algoritmo deve retornar uma solução ótima![1]

(e) É possível que o seu algoritmo resolva o mesmo problema mais de uma vez? Justifique a sua resposta.

(f) Qual o número de subproblemas que o seu algoritmo precisa resolver na pior das hipóteses? Justifique a sua resposta.

(g) É possível armazenar as soluções dos subproblemas em uma tabela? Se sim, quais as dimensões dessa tabela e como você organizaria ela (o que é armazenado em cada entrada)?

(h) Escreva uma PD (doravante Programação Dinâmica) usando a abordagem top-down para o problema da MOCHILA INTEIRA.

(i) Analise o tempo de execução do algoritmo criado no item anterior.

[1] Caso tenha dificuldades com a segunda parte, sugiro que primeiro desenvolva um algoritmo capaz de retornar o valor de uma solução ótima. Em um segundo momento, melhore o algoritmo para retornar a solução também.

(j) Escreva uma PD usando a abordagem Bottom-Up para o problema da MOCHILA INTEIRA.

(k) Analise o tempo de execução do algoritmo criado no item anterior.

(l) O seu algoritmo é polinomial no tamanho da entrada? Justifique a sua resposta.

(m) Escreva um algoritmo capaz de reconstruir uma solução ótima a partir das tabelas e valores retornados pelo seu algoritmo para o problema da MOCHILA INTEIRA.

2. Considere o problema de fazer troco para n centavos usando o menor número total de moedas. Você pode assumir que existe um número infinito de moedas disponíveis para cada valor. Forneça um algoritmo de programação dinâmica que é ótimo para fazer troco tendo disponíveis moedas de valores $M = \{m_1, m_2, \ldots, m_t\}$ centavos, para $t \geq 1$, onde algum $m_k = 1$.

Seja S um multiconjunto (i.e., um conjunto que permite repetição de elementos) de moedas. Denotamos por $v(S) = \sum_{i \in S} i$. Note que se S é uma solução viável para o problema do troco, então $v(S) = n$. Se além de viável S for uma solução ótima, temos que $|S|$ é o menor possível. Vamos denotar $\phi_{M,n}$ o menor número de moedas, cujos valores pertencem a M, necessário para dar um troco no valor de n.

(a) O algoritmo a seguir computa corretamente o valor da solução ótima? Justifique a sua resposta.

```
1: Função TROCO(M[1..t], n)
2:     Ordene M de forma que os valores das moedas fiquem em ordem decrescente
   (note que após a ordenação M[t] = 1)
3:     sol = 0
4:     i = 1
5:     Enquanto n ≥ 0 faça
6:         sol = sol + ⌊n/M[i]⌋
7:         n = n%M[i]
8:         i = i + 1
9:     Devolve sol
```

(b) Seja S^* uma solução ótima para o problema do Troco. Prove que, para todo $i \in S^*$, temos que

$$\phi_{M,n} = \phi_{M,n-i} + 1.$$

(c) Escreva $\phi_{M,n}$ como uma recorrência de subproblemas menores, não necessariamente pertencentes a uma solução ótima (veja a letra (c) do exercício anterior).

(d) Escreva um algoritmo recursivo para o problema do Troco.

(e) Analise o tempo de execução do seu algoritmo e apresente um limitante inferior para o tempo de execução dele. O limitante não precisa ser justo, mas precisa ser minimamente significativo, por exemplo, você não deve fornecer o limitante trivial de que seu algoritmo é $\Omega(1)$.

(f) O seu algoritmo resolve o mesmo problema mais de uma vez? Justifique a sua resposta.

(g) Qual o número máximo de subproblemas distintos que o seu algoritmo precisa resolver? Justifique a sua resposta.

(h) Podemos usar uma tabela para armazenar a solução dos subproblemas? Qual a dimensão dessa tabela? Como fazer isso?

(i) Escreva uma PD usando abordagem Top-Down para o problema do Troco.

(j) Analise o tempo de execução do algoritmo criado no item anterior.

(k) Escreva uma PD usando a abordagem Bottom-Up para o problema do Troco.

(l) Analise o tempo de execução do algoritmo criado no item anterior.

3. Dados um grafo G e dois vértices $u, v \in V(G)$, queremos encontrar a distância entre u e v. Mostre que esse problema apresenta subestrutura ótima.

4. O n-ésimo número de Fibonacci, denotado por $F(n)$, é definido como

$$F(n) = \begin{cases} 1, & \text{se } n \in \{1, 2\} \\ F(n-1) + F(n-2), & \text{se } n \geq 3 \end{cases}$$

(a) Apresente um algoritmo de programação dinâmica Top-Down que compute o n-ésimo número de Fibonnaci (Você não precisa fazer nenhuma análise, apenas apresente o pseudo-código do algoritmo).

(b) Analise o tempo de execução do algoritmo apresentado anteriormente.

(c) Apresente um algoritmo de programação dinâmica Bottom-Up que compute o n-ésimo número de Fibonnaci (Você não precisa fazer nenhuma análise, apenas apresente o pseudo-código do algoritmo).

(d) Analise o tempo de execução do algoritmo apresentado anteriormente.