Programação Dinâmica: Adicionando uma Variável

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Soma de subconjuntos (Subset Sum)

Soma de subconjuntos

O Problema

Dado um conjunto (multiconjunto) de números naturais $A = \{a_1, a_2, \ldots, a_n\}$ e um valor S, descobrir se existe um subconjunto $A^* \subseteq A$ tal que $\sum_{a \in A^*} a = S$.

Uma instância do problema terá resposta: $true$ ou $false$.

Exemplo

- $A = \{3, 1, 4, 12, 5, 7\}$ e $S = 9 \implies true$.
- $A = \{1, 2, 3\}$ e $S = 9 \implies false$.

Definindo uma solução recursiva para o problema

Notação

Seja $(A = \{a_1, \ldots, a_n\}; s)$ uma instância do problema. Definimos $sol(n, s)$ como sendo a solução ($true$ ou $false$) para esta instância; e $A^* \subseteq A$ como sendo um conjunto tal que $\sum_{a \in A^*} a = S$.

- **Caso base:** se $n = 0$, o problema tem solução $true$ se e somente se $s = 0$.
- Considerando o n-ésimo elemento do conjunto de números (a_n), existem duas possibilidades:
  - $a_n \notin A^*$: A^* é solução da instância $(A' = \{a_1, \ldots, a_{n-1}\}; s)$;
  - $a_n \in A^*$: $A^* - \{a_n\}$ é solução da instância $(A' = \{a_1, \ldots, a_{n-1}\}; s - a_n)$.

Definindo uma solução recursiva para o problema

Recorrência

$$sol(n, s) = \begin{cases} true & \text{se } n = 0 \text{ e } s = 0 \\ false & \text{se } n = 0 \text{ e } s \neq 0 \\ sol(n-1, s) & \text{se } a_n \notin A^* \\ sol(n-1, s-a_n) & \text{se } a_n \in A^*. \end{cases}$$

Algoritmo força bruta recursivo

$\textsc{Subset-Sum}(A, n, s)$

if $n = 0$ **then**
    if $s = 0$ **then** $ans \leftarrow true$
    else $ans \leftarrow false$
else
    $ans \leftarrow \textsc{Subset-Sum}(A, n-1, s)$
    if $ans = false$ *and* $A[n] \leq s$ **then**
        $ans \leftarrow \textsc{Subset-Sum}(A, n-1, s - A[n])$
return ans

Algoritmo força bruta recursivo

```
SUBSET-SUM(A, n, s)
if n = 0 then
    if s = 0 then  ans ← true
    else  ans ← false
else
    ans ← SUBSET-SUM(A, n − 1, s)
    if ans = false and A[n] ≤ s then
        ans ← SUBSET-SUM(A, n − 1, s − A[n])
return ans
```

Complexidade no pior caso?

Suponha $A = \{1, 1, \ldots, 1\}$ e $s = A.length$.

$$T(n, s) = T(n-1, s) + T(n-1, s-1) + \Theta(1) \implies T(n, s) = O(2^n).$$

O algoritmo examina todos os subconjuntos de A.

Sobreposição de problemas

Relembrando

Há sobreposição de problemas se o algoritmo faz chamadas recursivas para subproblemas repetidos.

Exemplo

Considere a instância $(A = \{1, 2, 3, 4, 7\}; s = 8)$.

- Ao considerar 7 e desconsiderar $\{3, 4\}$ em A^*, ficamos com o subproblema $(A = \{1, 2\}; s = 1)$.
- Ao desconsiderar 7 e considerar $\{3, 4\}$ em A^*, ficamos com o subproblema $(A = \{1, 2\}; s = 1)$.

Definindo o memo

- $memo$ deverá armazenar as soluções para todos os subproblemas distintos.
- $memo$ será uma tabela de dimensões $n + 1$ por $s + 1$, sendo inicializada com -1 em cada célula (valor indicando que a solução para aquele subproblema ainda não foi computada);
- cada célula $memo[i][j]$ conterá a solução ($true$, $false$) do subproblema $(\{a_1, \ldots, a_i\}; j)$.

$$memo[i][j] = \begin{cases} memo[i-1][j] & \text{se } a_i > j \\ memo[i-1][j] \text{ ou } memo[i-1][j-a_i] & \text{se } a_i \leq j. \end{cases}$$

Algoritmo memoizado

```
SUBSET-SUM-MEMO(A, n, s)
if memo[n][s] = −1 then
    if n = 0 then
        if s = 0 then  ans ← true
        else  ans ← false
    else
        ans ← SUBSET-SUM-MEMO(A, n − 1, s)
        if ans = false and A[n] ≤ s then
            ans ← SUBSET-SUM-MEMO(A, n − 1, s − A[n])
    memo[n][s] ← ans
return memo[n][s]
```

Análise de complexidade

- # de subproblemas $\times$ tempo / subproblema $=$

Algoritmo memoizado

```
Subset-Sum-Memo(A, n, s)
if memo[n][s] = −1 then
    if n = 0 then
        if s = 0 then  ans ← true
        else  ans ← false
    else
        ans ← Subset-Sum-Memo(A, n − 1, s)
        if ans = false and A[n] ≤ s then
            ans ← Subset-Sum-Memo(A, n − 1, s − A[n])
    memo[n][s] ← ans
return memo[n][s]
```

Análise de complexidade

- # de subproblemas $\times$ tempo / subproblema $=$ $n \cdot s \times \Theta(1) = \Theta(n \cdot s)$.
- Este algoritmo não é polinomial em relação ao tamanho da entrada (pseudo-polinomial).

Ordem de preenchimento do $memo$

- Caso base ocorre quando $n = 0$, e a solução ($true$,$false$) depende do valor de s.
- Consideraremos duas variáveis (i e j), tal que i indica o índice do maior elemento do vetor A (linhas) e j o valor s nos subproblemas (colunas).
- $memo$ será preenchido de cima para baixo, da esquerda para a direita, conforme tabela a seguir:

Consdere $A = \{1, 2, 3, 4, 7\}$ e $s = 8$

	0	1	2	3	4	5	6	7	8
0	T	F	F	F	F	F	F	F	F
1	T	T	F	F	F	F	F	F	F
2	T	T	T	T	F	F	F	F	F
3	T	T	T	T	T	T	T	F	F
4	T	T	T	T	T	T	T	T	T
5	T	T	T	T	T	T	T	T	$\boxed{T}$

Algoritmo iterativo

```
SUBSET-SUM-DP(A, n, s)
for i ← 0 to n do
    for j ← 0 to s do
        if i = 0 then
            if j = 0 then ans ← true
            else ans ← false
        else
            ans ← memo[i − 1][j]
            if ans = false and A[i] ≤ j then
                ans ← memo[i − 1][j − A[i]]
        memo[i][j] ← ans
return memo[n][s]
```

Problema da Mochila Binária (0-1 Knapsack Problem)

Problema da Mochila Binária

Dado um conjunto $S = \{1, 2, \ldots, n\}$ de n itens (cada item i estando associado a um peso w_i e a um valor v_i) e um valor W (representando a capacidade máxima de peso suportada pela mochila), determine um subconjunto de S que maximize o valor total sem exceder a capacidade da mochila, isto é:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{sujeito a:} \quad & \sum_{i=1}^{n} w_i x_i \leq W, \\ & x_i \in \{0, 1\}. \end{aligned}$$

Exemplo

Suponha uma mochida de capacidade $W = 3Kg$, três itens para escolher $S = \{1, 2, 3\}$ (com seus respectivos valores e pesos).

1	2	3
$R\$$ 5,00 $3Kg$	$R\$$ 3,00 $1Kg$	$R\$$ 4,00 $2Kg$

A solução ótima $S^* = \{2, 3\}$ proporciona um total de $R\$$ 7,00.

Estrutura recursiva (subestrutura ótima)

Seja $opt(i, w)$ o valor máximo que pode ser obtido considerando somente os i primeiros itens e com peso máximo w (capacidade restante).

Caso base: $opt(0, w) = 0$ e $opt(i, 0) = 0$ $\forall i \leq n$ e $\forall w \leq W$.

$$opt(i, w) = \begin{cases} 0 & \text{se } i = 0 \text{ ou } w = 0 \\ opt(i-1, w) & \text{se } w_i > w \\ \max\left(opt(i-1, w), v_i + opt(i-1, w-w_i)\right) & \text{se } w_i \leq w \end{cases}$$

Exercícios

1. Escreva um algoritmo recursivo (sem usar PD) para resolver o problema da Mochila Binária.
2. Crie um exemplo que mostra sobreposição de problemas.
3. Apresente um algoritmo memoizado e outro *bottom-up*.
4. Escreva um algoritmo que, dada a informação armazenada no memo, informe quais itens seriam colocados na mochila (qual o conjunto de itens que faz parte da solução ótima).