Projeto de algoritmos: uma visão prática

Aula 03

Rafael Melo
Instituto de Computação
Universidade Federal da Bahia

Programação dinâmica

RELEMBRANDO: Passos da programação dinâmica

- Caracterize a estrutura de uma solução ótima.
- Defina uma recorrência para o cálculo de uma solução ótima considerando a otimalidade dos subproblemas.
- Compute o valor de uma solução ótima de forma *bottom-up*.
  - Definindo uma ordem do cálculo dos subproblemas
- Construa a solução ótima utilizando a informação computada.

Problema da subsequência crescente mais longa

Problema da subsequência crescente mais longa

- *Longest increasing subsequence (LIS)*
- *Dada uma **sequência de n números**, encontre a **subsequência mais longa** cujos números estão em **ordem crescente**.*
- *Exemplo:*
  - *Sequência s = (9, 5, 2, 8, 7, 3, 1, 6, 4)*
  - *Subsequência mais longa tem comprimento 3*
    - $s_1' = (2, 3, 4)$ *ou* $s_2' = (2, 3, 6)$

Problema da subsequência crescente mais longa

- *Se quisermos uma subsequência contígua, o problema seria facilmente resolvido.*
- *Qual o número de soluções contíguas candidatas?*

$$\begin{aligned} T(n) &= n + (n-1) + (n-2) + \ldots + 1 \\ &= \frac{n(n+1)}{2} \Rightarrow T(n) \in \Theta(n^2) \end{aligned}$$

Problema da subsequência crescente mais longa

- *Quantas soluções candidatas possíveis para o problema da subsequência crescente mais longa?*

$$
\begin{aligned}
T(n) &= \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n} \\
&= \sum_{k=0}^{n} \binom{n}{k} \\
&= (1+1)^n \qquad \text{(pelo Binômio de Newton)} \\
&= 2^n \Rightarrow T(n) \in \Theta(2^n)
\end{aligned}
$$

- *Enumeração se torna uma opção inviável na prática*

Problema da subsequência crescente mais longa

- Não é difícil construir um algoritmo com tempo de execução $\Theta(n^2)$ para encontrar a subsequência mais longa de uma cadeia de n símbolos.
- Apesar do número elevado de soluções candidatas, podemos projetar facilmente um algoritmo PD de tempo $\Theta(n^2)$.

Definição dos subproblemas e recorrência

- Suponha que para a sequência $s'=(s_1, s_2, \ldots, s_{i-1})$ conheçamos o comprimento l_k, $k \in \{1, \ldots, i-1\}$, da subsequência mais longa que termina no elemento s_k de s'.
- Como encontrar a subsequência mais longa em $(s_1, \ldots, s_i)$?
- Isto pode ser feito usando a recorrência:

$$l_i = max\{ l_k + 1 : s_k < s_i, k = 1, \ldots, i-1\}$$

- *Qual o valor da solução ótima?*

$$z = max\{ l_i : i = 1, \ldots, n\}$$

Obtenção da solução ótima

- Dada a recorrência para o cálculo de l_i

$$l_i = max\{ l_k + 1 : s_k < s_i , k = 1, \ldots , i-1\}$$

- *Defina como p_i o índice do antecessor de s_i na subsequência crescente mais longa terminando em s_i, definido como:*

$$p_i = argmax\{ l_k + 1 : s_k < s_i , k = 1, \ldots , i-1\}$$

Algoritmo

```python
def PD_subseqcrescente(sequencia):
    n = len(sequencia)

    s = [0] + sequencia
    l = [0 for k in range(n + 1)]
    p = [0 for k in range(n + 1)]

    for i in range(1, n + 1):
        max_ind = 0
        for j in range(1, i):
            if s[j] < s[i] and l[j] > l[max_ind]:
                max_ind = j
        l[i] = l[max_ind] + 1
        p[i] = max_ind

    max_ind = 0
    for j in range(1, n + 1):
        if l[j] > l[max_ind]:
            max_ind = j

    valor_otimo = l[max_ind]

    subseq = []
    i = max_ind
    while i > 0:
        subseq.append(s[i])
        i = p[i]
    subseq.reverse()

    return valor_otimo, subseq
```

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0									
Comprimento l_i	0									
Predecessor p_i	0									

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9								
Comprimento l_i	0	1								
Predecessor p_i	0	0								

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5							
Comprimento l_i	0	1	1							
Predecessor p_i	0	0	0							

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2						
Comprimento l_i	0	1	1	1						
Predecessor p_i	0	0	0	0						

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8					
Comprimento l_i	0	1	1	1	2					
Predecessor p_i	0	0	0	0	3					

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8	7				
Comprimento l_i	0	1	1	1	2	2				
Predecessor p_i	0	0	0	0	3	3				

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8	7	3			
Comprimento l_i	0	1	1	1	2	2	2			
Predecessor p_i	0	0	0	0	3	3	3			

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8	7	3	1		
Comprimento l_i	0	1	1	1	2	2	2	1		
Predecessor p_i	0	0	0	0	3	3	3	0		

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8	7	3	1	6	
Comprimento l_i	0	1	1	1	2	2	2	1	3	
Predecessor p_i	0	0	0	0	3	3	3	0	6	

Exemplo execução

- s = (9, 5, 2, 8, 7, 3, 1, 6, 4)

	0	1	2	3	4	5	6	7	8	9
Seqüência s_i	0	9	5	2	8	7	3	1	6	4
Comprimento l_i	0	1	1	1	2	2	2	1	3	3
Predecessor p_i	0	0	0	0	3	3	3	0	6	6

Tempo de execução

- O tempo de execução pode ser dado pela recorrência
  - $T(n) = T(n-1) + n$
  - implicando que $T(n) \in \Theta(n^2)$.
- Utilizando estruturas de dados como dicionários de uma forma inteligente é possível encontrar a subsequência crescente mais longa em tempo $\Theta(n \log n)$.

Atividades práticas

Contato:
rafael.melo@ufba.br
rafaelmelo.ufba@gmail.com