

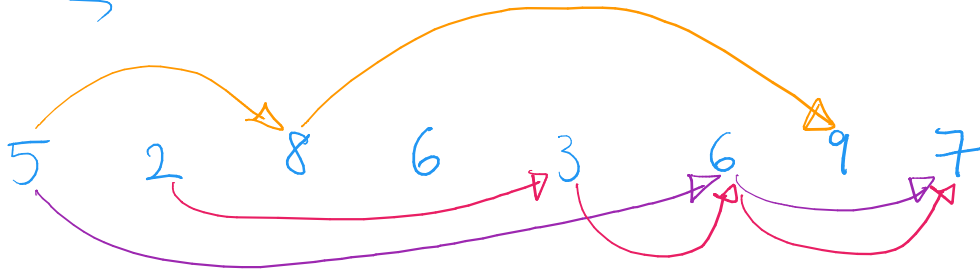
Programação Dinâmica

- Entender problema
- Imaginar sol. ótima
- Mostrar subestrutura ótima
- Deduzir recorrência
- Projetar alg. e/ tabela
- Analisar efic. de tempo e espaço

Subsequência Crescente Mais Longa

- Entrada: sequência de números $v[1..n]$
- Solução: maior subsequência $v[i_1], v[i_2], \dots, v[i_k]$ da entrada, tal que $1 \leq i_1 < i_2 < \dots < i_k \leq n$ e p/ Todo i_e e i_h c/ $e < h$ temos $v[i_e] < v[i_h]$

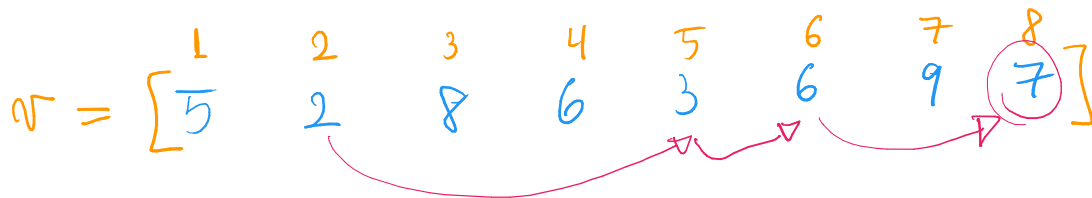
Exemplos



Importante: encontrar uma ordenação p/ os subproblemas e uma forma de resolver um subproblema usando subproblemas menores, i.e., que vem antes na ordenação.

Imaginar solução ótima

- pensando em uma sequência que termina na posição j



$$j=8 \Rightarrow L[8] = 1 + L[6] \text{ neste exemplo}$$

Ou seja, a solução ótima que termina na posição j tem valor $1 +$ o valor da maior subsequência (sol. ótima) que termina em i antes de j e/ $v[i] < v[j]$

Subestrutura Ótima

- provar por contradição, supondo que uma solução ótima p/ $L[j]$ não usa a sol. ótima de seu subproblema $L[i]$

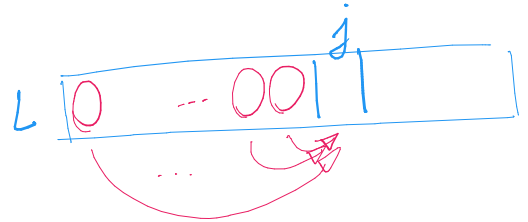
Deduzir Recorrência

case base

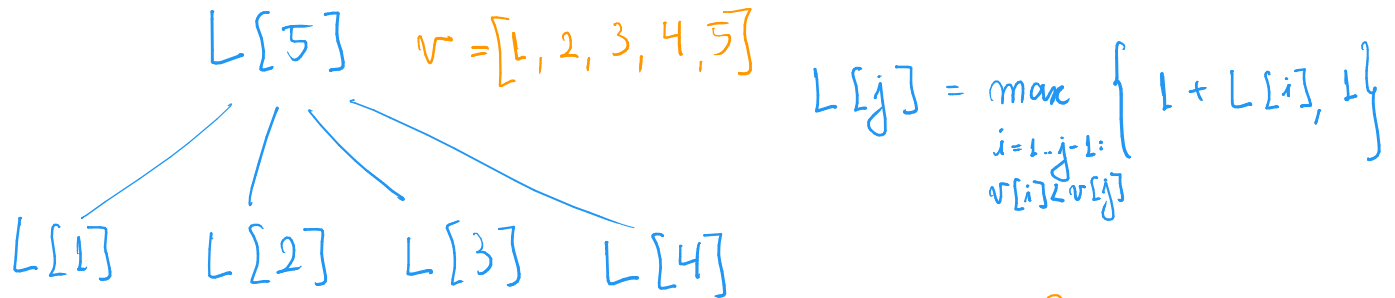
$$L[j] = \max_{\substack{i=1..j-1: \\ v[i] < v[j]}} \left\{ 1 + L[i], 1 \right\}$$

$$\text{alternativa} = 1 + \max_{\substack{i=1..j-1: \\ v[i] < v[j]}} \left\{ L[i] \right\}$$

$$\text{c/ } \max \{ \emptyset \} = 0$$

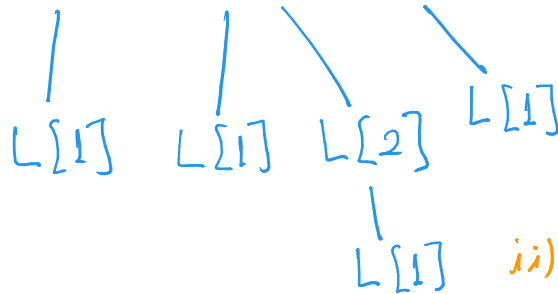


Alg. Paramante recursivo é ineficiente



Quis: por que os algs. recursivos de D&C são eficientes?

⊗ É possível usar técnica de memorização p/ torná-lo eficiente



i) subproblemas diminuem rápido (divisão $\Rightarrow$ alt. log.)
ii) divisão gera subprobl. disjuntos

Alg. iter. e/ tabela

$$L[j] = \max_{\substack{i=1..j-1 \\ v[i] < v[j]}} \{1 + L[i], 1\}$$

sub-Seq cresc Mais Longa ($v[1..n]$):

para $j = 1$ até n : — $O(n)$ iterações.

Efic. tempo:

$L[j] = 1 \rightarrow \text{pred}[j] = -1$

para $i = 1$ até $j-1$:

Quiz: por que inclui o igual?

$O(n^2)$

se $L[i] \geq L[j] \wedge v[i] < v[j]$:

$O(n)$ iterações.

Efic. espaço:

$L[j] = 1 + L[i]$

$\Theta(n)$

devolva $\max_{j=1..n} \{L[j]\}$

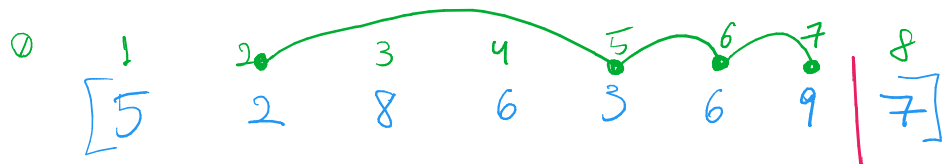
Note que o ótimo pode não estar na última pos.

⊗ Para reconstruir a seq, guardar $\text{pred}[]$ de cada j

Curiosidade / Extra:

- É possível fazer um alg. mais efíc. que mantém a melhor subseq. de cada tamanho terminados antes de j em uma árvore binária de busca balanceada.
- A chave é o último valor da subseq. presente e o tam. da mesma também é armazenado. → ou apenas em um vetor ordenado
- Assim, a busca pela melhor subseq. que concatenar $v[j]$ leva tempo $O(\log n)$ → busca o antecessor de $v[j]$ e a atualização das seq. armazenadas busca $v[j]$ ou seu sucessor.
- * Só mantemos armazenadas as subseq. não dominadas, i.e., que tem final menor ou tam. maior que as demais subsequências

Exemplo



Observe que, para cada tamanho de subsequência, só precisamos guardar aquela que termina no menor valor. *Quiz: por que isso é verdade?*

tam. da subseq.	0	1	2	3	⁷ ↓ 4
último elem. da s.s.	$-\infty$	2	3	6	⁷ ↓ 7
índice do final da s.s.	0	2	5	6	7

Tendo resolvido os subproblemas $L[1]$ até $L[j-1]$, para resolver $L[j]$ basta usar busca binária no vetor

auxiliar p/ encontrar o predecessor de $v[j]$