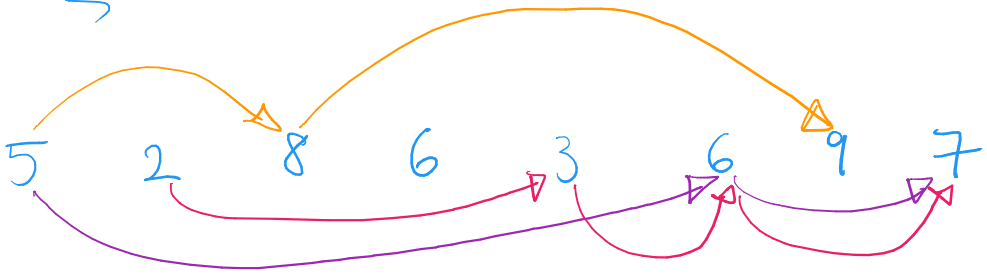
Programação Dinâmica

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

Subsequência Crescente Mais Longa

- Entrada: sequência de números $v[1..n]$
- Solução: maior subsequência $v[i_1], v[i_2], \ldots, v[i_K]$ da entrada, tal que $1 \leq i_1 < i_2 < \ldots < i_K \leq n$ e p/ todo i_ℓ e i_h c/ $\ell < h$ temos $v[i_\ell] < v[i_h]$

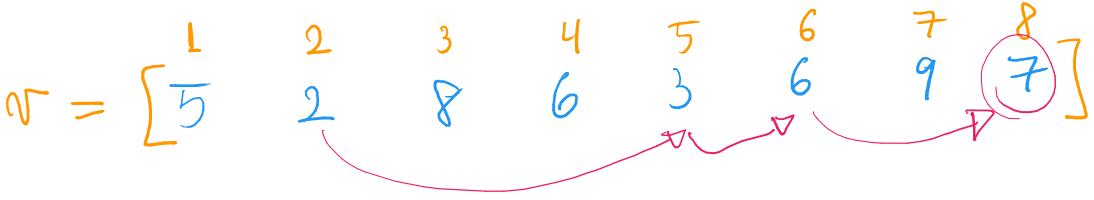
Exemplos



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



$j = 8 \Rightarrow L[8] = 1 + L[6]$ 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 c/ $v[i] < v[j]$

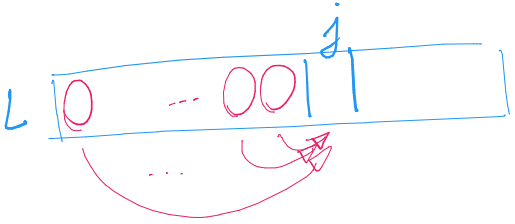
Subestrutura Ótima

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

Dedução Recorrência

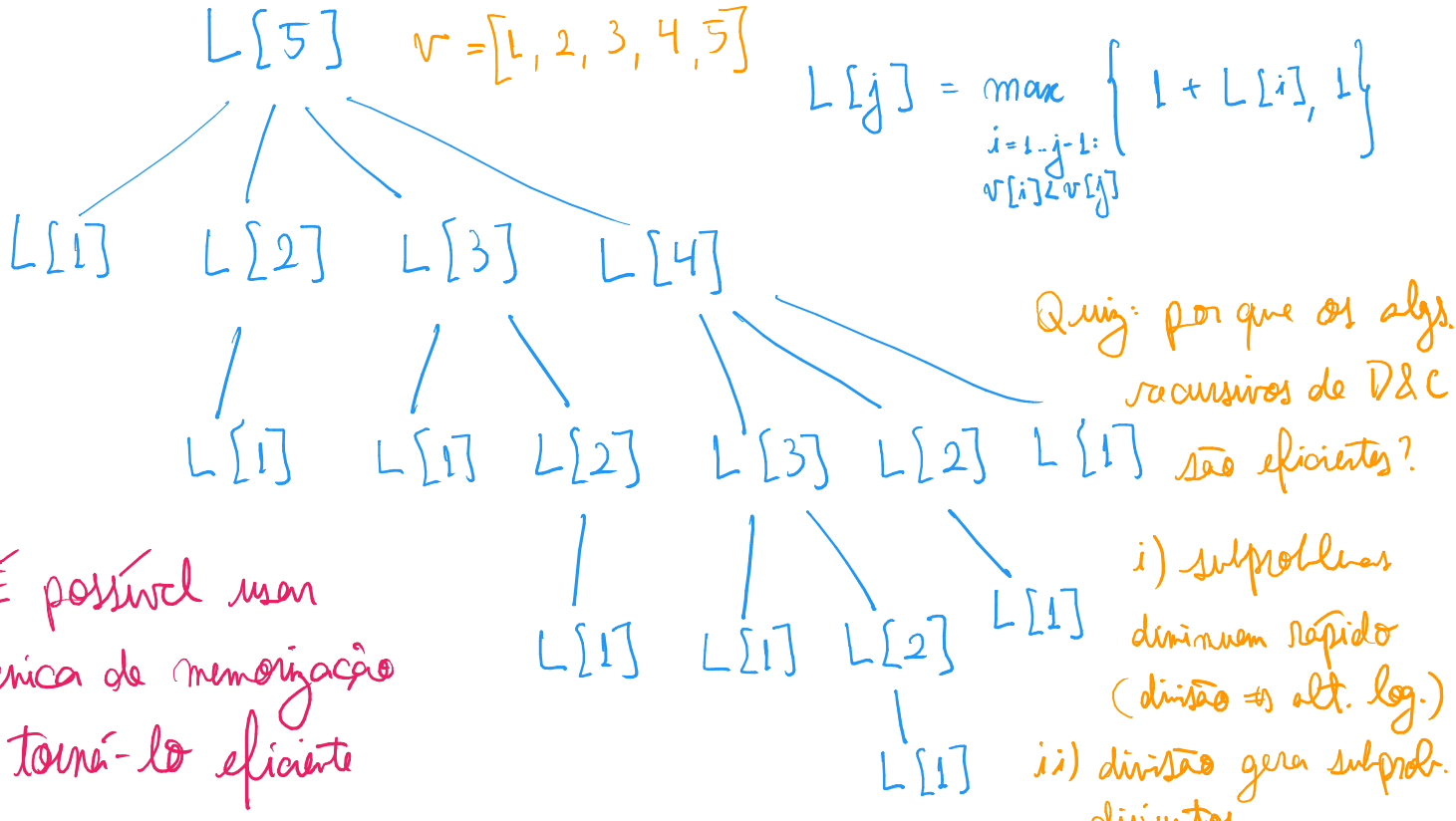
caso base

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



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

Alg. Puramente recursivo é ineficiente



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

i) subproblemas diminuem rápido (divisão ⇒ alt. log.)

ii) divisão gera subprob. disjuntos

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

Alg. iter. c/ tabela

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

```
subSeqCrescMaisLonga(v[1..n]):
    para j = 1 até n:                          ——— O(n) iter.
        L[j] = 1  ⇒ pred[j] = -1
        para i = 1 até j-1:                    ——— O(n) iter.
            se L[i] ≥ L[j] E v[i] < v[j]:
                L[j] = 1 + L[i]  ⇒ pred[j] = i
    devolva max_{j=1..n} {L[j]}
```

Quiz: por que incluí o igual?

Efic. tempo: $O(n^2)$

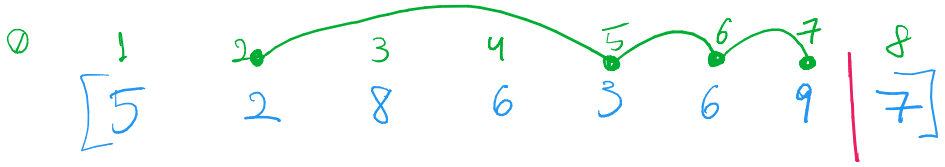
Efic. espaço: $\Theta(n)$

⊛ Para reconstruir a seq, guardar pred[] de cada j

Curiosidade / Extra:

- É possível fazer um alg. mais efic. que mantém a melhor solução de cada tamanho terminados antes de j em uma árvore binária de busca balanceada.
  → ou apenas em um vetor ordenado
- A chave é o último valor da subseq. crescente e o tam. da mesma também é armazenado.
- Assim, a busca pela melhor subseq. que concatenar c/ v[j] leva tempo $O(\log n)$ → busca o antecessor de v[j] e a atualização das seq. armazenadas busca v[j] ou um sucesso.
- 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	4 (7)
último elem. da s.s.	$-\infty$	2	3	6	~~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]