

Algoritmi e Strutture Dati

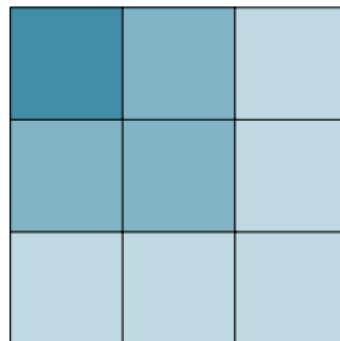
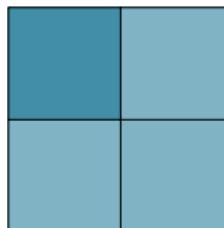
Programmazione Dinamica

Paolo Pasini
pao.lo.pasini@polito.it



Qual è l'idea dietro al paradigma?

- ▶ Identificare e risolvere sotto-problemi
- ▶ Combinare insieme sotto-problemi per risolvere istanze più grandi
- ▶ Occorre rispettare il principio di ottimalità di Bellman!!!





Come affrontare un problema di programmazione dinamica?

- ▶ Visualizzare e ragionare per esempi
- ▶ Identificare sotto-problemi
- ▶ Individuare relazioni tra sotto-problemi
- ▶ Generalizzare le relazioni individuate
- ▶ Risolvere sotto-problemi nel corretto ordine e ricombinare

Longest Increasing Subsequence

Intro



Formulazione del problema

Data una sequenza s_1, s_1, \dots, s_n , identificare la più lunga sottosequenza crescente $s_{i_1}, s_{i_2}, \dots, s_{i_k}$.

$$\begin{aligned} s_{i_1} &< s_{i_2} < \cdots < s_{i_k} \\ i_1 &< i_2 < \cdots < i_k \end{aligned}$$

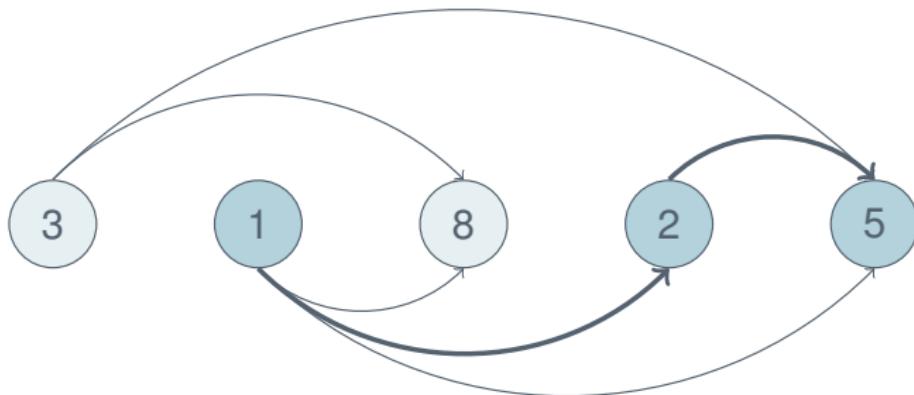
Esempi

$$LIS([3, 1, 8, 2, 5]) \rightarrow 3$$

$$LIS([5, 2, 8, 6, 3, 6, 11, 5]) \rightarrow 4$$

Longest Increasing Subsequence

Visualizzazione



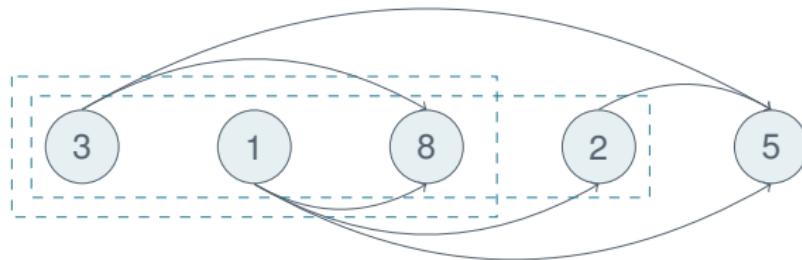
Longest Increasing Subsequence

Individuazione sotto-problemi



Intuizione

- ▶ Tutte le sottosequenze crescenti sono porzioni della sequenza originale
- ▶ Ogni sottosequenza ha un punto di inizio e uno di fine
- ▶ Focus sul punto di fine
- ▶ $LIS[k] \rightarrow$ sotto-problema che finisce all'indice k



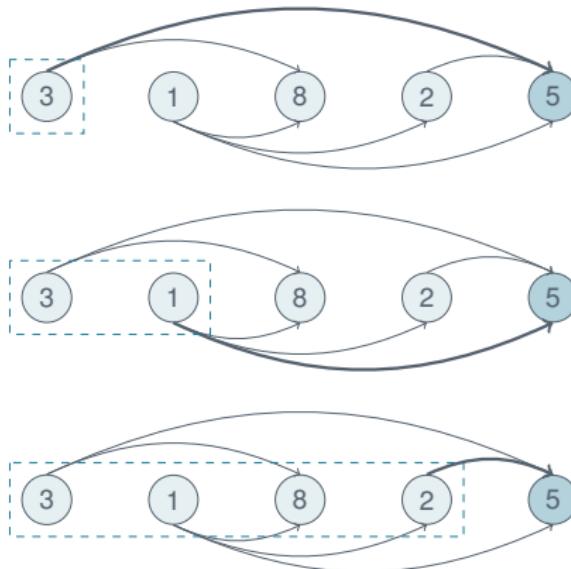
Longest Increasing Subsequence

Relazioni tra sotto-problemi



Ragionamento

- ▶ Da quali sotto-problemi dipende $LIS[4]$?



Longest Increasing Subsequence

Generalizzazione



Caso particolare

- ▶ $LIS[4] = 1 + \max\{LIS[0], LIS[1], LIS[3]\} = 3$

Formalizzazione

- ▶ $LIS[4] = 1 + \max\{LIS[k] \mid k < 4, v[k] < v[4]\}$

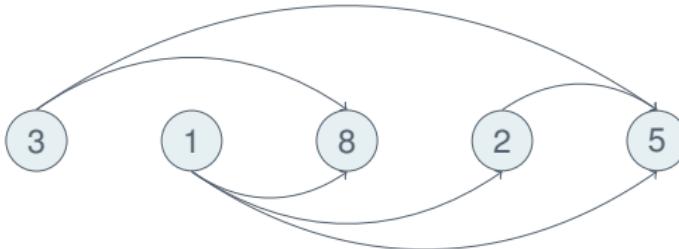
Generalizzazione

- ▶ $LIS[n] = 1 + \max\{LIS[k] \mid k < n, v[k] < v[n]\}$



Algoritmo 1 LIS(v, n)

```
1: L = [1] * n
2: for i = 0 to n do
3:   subproblems = [L[k] for k in [0 . . . i) if (v[k] < v[i])]
4:   L[i] = max(subproblems, default=0)
5: end for
6: return max(L, default=0)
```



Box Stacking

Descrizione

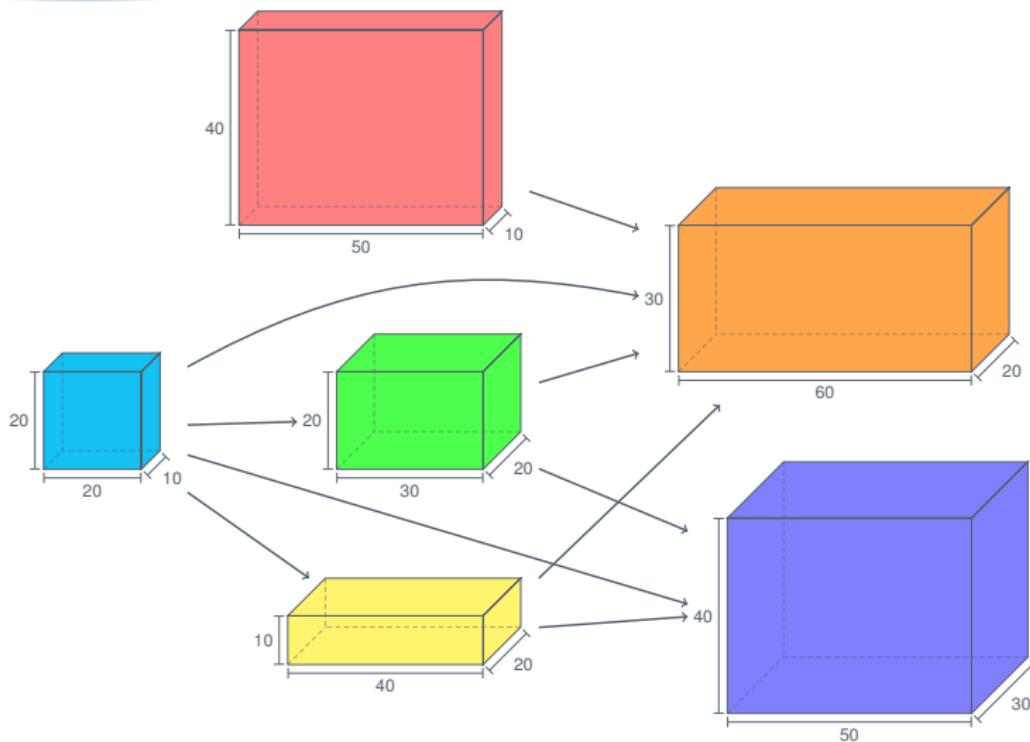


Formulazione del problema

Data un insieme di scatole caratterizzate ognuna da una propria dimensione, espressa come terna *altezza, larghezza e profondità*, identificare la pila di scatole più alta possibile, sapendo che una scatola si può impilare su un'altra solo se la sua larghezza e profondità sono strettamente minori della scatola direttamente sottostante.

Box Stacking

Visualizzazione





Intuizione

- ▶ Un arco (i, j) nel grafo precedente indica che la scatola di indice i può essere impilata sopra la scatola di indice j
- ▶ Ogni pila deve avere una scatola per base
- ▶ $BS[k] \rightarrow$ sotto-problema alla cui base è posizionata la scatola di indice k
- ▶ Per risolvere $BS[k]$ devo conoscere l'ottimo per tutti i sotto-problemi alla cui base ci sia una scatola che possa essere posizionata direttamente sopra la scatola di indice k
- ▶ Occorre garantire un ordinamento adeguato dell'input per risolvere i sotto-problemi nell'ordine corretto. Sia per larghezza o per profondità, in questo caso funziona.