Overview

• Memoization, sottoproblemi

• Esempi

– Fibonacci
– Shortest Paths

• Guessing

Dynamic Programming (DP)

- Idea semplice ma potente

- Tecnica importante di progettazione di algoritmi
- Un'ampia classe di problemi la cui soluzione sarebbe esponenziale senza DP hanno una soluzione polinomiale solo tramite DP
- Particolarmente importante per problemi di ottimizzazione (min / max)

* DP ≈ "Brute force controllata"

* DP ≈ ricorsone + riuso

Storia - Richard E. Bellman (1920-1984)

Richard Bellman: "Bellman . . . explained that he invented the name 'dynamid programming' to hide the fact that he was doing mathe matical research at RAND under a Secretary of Defense who 'had a pathological fear and hatred of the term, research'. He settled on the term 'dynamic programming' because it would be difficult to give a 'pejorative meaning' and because 'it was something not even a Congressman could object to' " [John Rust 2006]

Numeri di Fibonacci

$$F_1 = F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Obiettivo: calcolare F_n

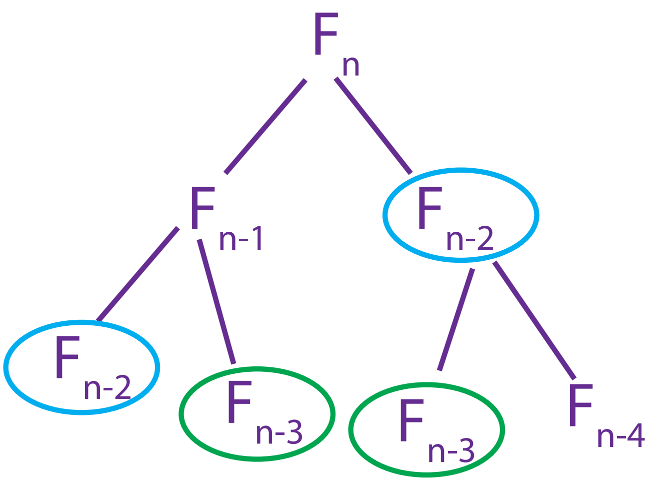
Soluzione "Naive"

Segue dalla definizione ricorsiva del problema

```
fib(n):
    if n ≤ 2: return f = 1
    else: return f = fib(n − 1) + fib(n − 2)
```

$T(n) = T(n-1) + T(n-2) + O(1) \geq F_n \geq 2T(n-2) + O(1) \geq 2^{n/2}$

ESPONENZIALE



Soluzione "Memoized DP"

Memoization => "ricordare"

```
memo = { }
fib(n):
        if n in memo: return memo[n]
        else: if n ≤ 2 : f = 1
              else: f = fib(n−1) + fib(n−2)
              memo[n] = f
              return f
```

- ⟹ fib(k) ricorsione solo la <u>prima volta</u> che viene chiamata, ∀k
- ⟹ solo *n* chiamate "memorized": *k=n, n−1,...,1*
- le chiamate memoized impiegano Θ(1)
- ⟹ Θ(1) tempo per chiamata (ignorando/al netto della ricorsione)

POLINOMIALE

* DP ≈ ricorsione + memorization

- memoize (ricorda) & ri-uso delle soluzioni del sottoproblemi

  – in Fibonacci, i sottoproblemi sono $F_1, F_2, \ldots, F_n$

* ⟹ tempo = #sottoproblemi · tempo/sottoproblema

  - Fibonacci: #sottoproblemi è n,
  - Tempo/sottoproblema è Θ(1) => totale: Θ(n)

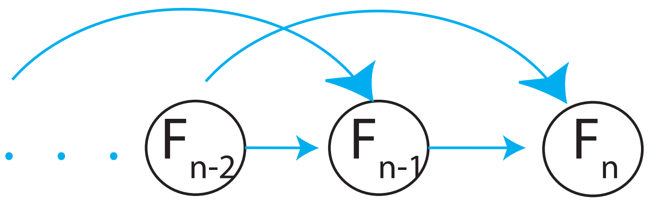
Soluzione DP Bottom-up

```
fib = {}
for k in [1, 2, . . . , n]:
        if k ≤ 2: f = 1
        else: f = fib[k−1] + fib[k−2]
        fib[k] = f
return fib[n]
```

$\Theta(1)$ (corpo del ciclo) $\Theta(n)$ (intero algoritmo)

- Esattamente la stessa computazione della soluzioen memorized (ricorsione “unrolled”)

- L’applicabilità dipende da come i sottoproblemi dipendono l’un l’altro:
- Deve esistere un <u>ordine topologico</u> del grafo diretto aciclico (DAG) delle dipendenze tra sottoproblemi
  - I nodi si definiscono ordinati topologicamente se sono disposti in modo tale che ogni nodo viene prima di tutti i nodi collegati ai suoi archi uscenti
  - Ordinamento parziale
  - Non univoco

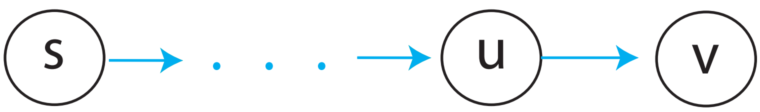


- DP bottom-up spesso è più veloce (non c’è ricorsione)
- Analisi è più semplice
- La tabella può occupare spazio, ma in molti casi si può risparmaire (per esempio, in Fibonacci basta ricrodare solo gli ultimi due valori $\Longrightarrow \Theta(1)$)

Shortest Paths

Guessing

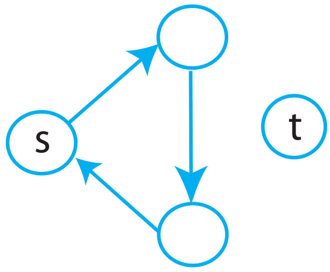
Come progettare la ricorrenza:



- Si vuole il percorso più preve da s a v
- Qual è l’ultimo arco nel percorso?
- <u>GUESS:</u> prova (u,v)
- In tal caso, lo shortest $\underbrace{\text{path da s a v è } \delta(s,u)}_{\text{Sottostruttura ottima}}$ + w(u,v)

- * DP ≈ ricorsione + memoization + guessing

- Formulazione ricorsiva: $\delta(s,v)=\min\{w(u,v) + \delta(s,u) \mid (u,v) \in E\}$



- Soluzione Memoized DP: in presenza di cicli, impiega un tempo infinito
-
- Funzione per i DAG (directed acyclic graphs) in un tempo $O(V + E)$
  - (somma degli "*indegree(v) +1*", per ogni v)

  * Il grafo delle dipendenze dei sottoproblemi dovrebbe essere aciclico

- Per risolvere il problema anche in presenza di cicli, si può trasformare il grafo (aggiungendo sottoproblemi, che rimuovono i cicli di dipendenze). Ho una nuova ricorrenza.

  $\delta_k(s, v)$ = shortest path $s \to v$ usando al più $\leq k$ archi

  - Ricorrenza:
    - $\delta_k(s, u) = \min\{\delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E\}$
    - $\delta_0(s, u) = \infty$ for $s = v$ (caso base)
    - $\delta_k(s, s) = 0$ per ogni k (caso base, se non ci sono cicli negative)

- Obiettivo:
  - $\delta(s, v) = \delta_{|V|-1}(s, v)$ (in assenza di cicli negative)
- ho |V| scelte per k: (k=0…|V|-1)

- memoize
- tempo: #sottoproblemi· tempo/sottoproblema

  $|V|\cdot|V| \quad \cdot \quad O(v) \qquad = O(V^3)$?

- in effetti è il contributo per il nodo v al passo k è: $\Theta(\text{indegree}(v)+1)$ for $\delta_k(s, v)$
- $\Longrightarrow$ tempo = $\Theta(V \sum_{v\in V} (\text{indegree}(V)+1)) = \Theta(V E+V^2)$ =>

  (ma il "+1" in realtà contribuisce solo una volta per vertice (operazione per trattare il caso in cui indegree=0, inizializzazione):

  il contributo è V, non V^2 => $\Theta(V E+V)$ => $\Theta(V E)$
- Algoritmo di BELLMAN-FORD