PROGRAMMAZIONE DINAMICA (2)

Due algoritmi che abbiamo visto sono in effetti algoritmi di programmazione dinamica

- Bellman-Ford
- Floyd-Warshall

- Vediamo questi algoritmi dal punto di vista delle 4 fasi della programmazione dinamica

BELLMAN-FORD: Cammini minimi da una sorgente s a tutti i vertici

CARATTERIZZAZIONE DELLA SOLUZIONE OTTIMA

Sia $\delta^t(s,v)$ cammino minimo di lunghezza $\leq t$

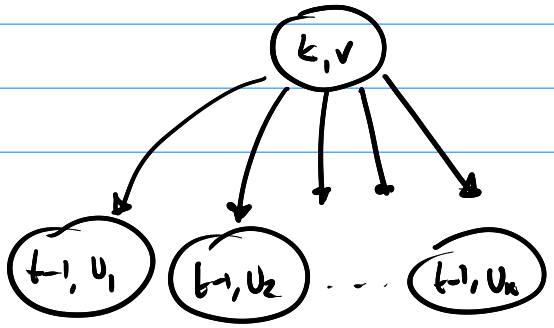
$$\delta^t(s,v) = \begin{cases} \delta^{t-1}(s,v) \\ \delta^{t-1}(s,u) + W[u,v] \quad \forall \{u,v\} \in G \end{cases}$$

DEFINIZIONE RICORSIVA

$$\delta^0(s,v) = \begin{cases} 0 & \text{se } s = v \\ +\infty & \text{se } s \neq v \end{cases}$$

$$\delta^t(s,v) = \min \left(\min_{\{u,v\} \in E} \{ \delta^{t-1}(s,u) + W[u,v] \}, \delta^{t-1}(s,v) \right)$$

I sottoproblemi hanno i parametri (t,v) (almeno sono n^2)



dove (u_i, v) sono gli archi entranti di v

CALCOLO BOTTOM-UP

$D^t[v]$ contiene l'ottimo del sottoproblema (t,v)

$D^0 = [\infty, \dots, \infty]$

for $i = 1 \dots n-1$

$D^t := D^{t-1}$

for $u,v \in E$:

$D^t[v] = \min\{D^t[v], D^{t-1}[u] + W[u,v]\}$

CALCOLO DELLA SOLUZIONE OTTIMA

Quando $D^t[v] \leftarrow D^{t-1}[u] + W[u,v]$

si segna $P^t[v] = u$

OSSERVAZIONI Lo schema precedente è quello che si ottiene applicando la programmazione dinamica, tuttavia

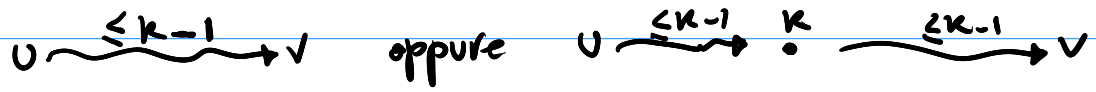
- Lo schema applicato rigidamente produce $n \cdot n^2$ sottoproblemi (t,v) ma il calcolo di D^t ha bisogno solamente di D^{t-1} quindi si possono tenere in memoria solo $\Theta(n)$ valori.
- Oltretutto in questo caso abbiamo visto che un array D è sufficiente, poiché alla fine del passo t

$D[v]$ = costo minimo in un insieme di cammini che contiene tutti i cammini di lunghezza t

FLOYD - WARSHALL : cammini minimi tra due vertici arbitrari.

CARATTERIZZAZIONE DELLA STRUTTURA DELLA SOLUZIONE OTTIMA

- Vertici da 1 a n (al contrario di come abbiamo fatto a lezione)
- Un cammino minimo tra u e v che passa solo tra vertici $\leq k$ ha due possibilità



DEFINIZIONE RICORSIVA

- Abbiamo n^3 sottoproblemi (u,k,v) : $\delta^k(u,v)$ costo del cammino minimo da u a v che passa sol per vertici $\leq k$

- $$\delta^k(u,v) = \min\left(\delta^{k-1}(u,v), \delta^{k-1}(u,k) + \delta^{k-1}(k,v)\right)$$

$$\delta^0(u,v) = \begin{cases} 0 & \text{se } u=v \\ +\infty & \text{se } (u,v) \notin E \\ W[u,v] & \text{se } (u,v) \in E \end{cases}$$

- Abbiamo n^3 sottoproblemi e ogni sottoproblema richiede le soluzioni di 3 sottoproblemi
- Da qui la complessità $O(n^3)$

CALCOLO BOTTOM-UP

```
D⁰ = ...
for k=1...n
  for u ∈ V(G)
    for v ∈ V(G)
      D^k[u,v] = min(D^{k-1}[u,v], D^{k-1}[u,k] + D^{k-1}[k,v])
```

CALCOLO DELLA SOLUZIONE OTTIMA

- Se $D^k[v,r] = D^{k-1}[v,r]$ allora $P^k[v,r] = P^{k-1}[v,r]$
- Se $D^k[v,r] = D^{k-1}[v,k] + D^{k-1}[k,r]$ allora
$$P^k[v,r] = P^{k-1}[k,r]$$

OSSERVAZIONE • Come nel caso di Bellman-Ford, anche se ci sono n^3 sottoproblemi, i sottoproblemi di livello k dipende dal livello $k-1$.

- I precedenti livelli possono essere dimenticati, quindi la memoria utilizzata è $O(n^2)$
- Inoltre, analogamente a Bellman-Ford, possiamo anche utilizzare la stessa matrice D scrivendo

$$D[v,r] = \min\left(D[v,r], D[v,k] + D[k,r]\right)$$

nel passo di update

Memoizzazione

- Una forma di caching delle chiamate a funzione

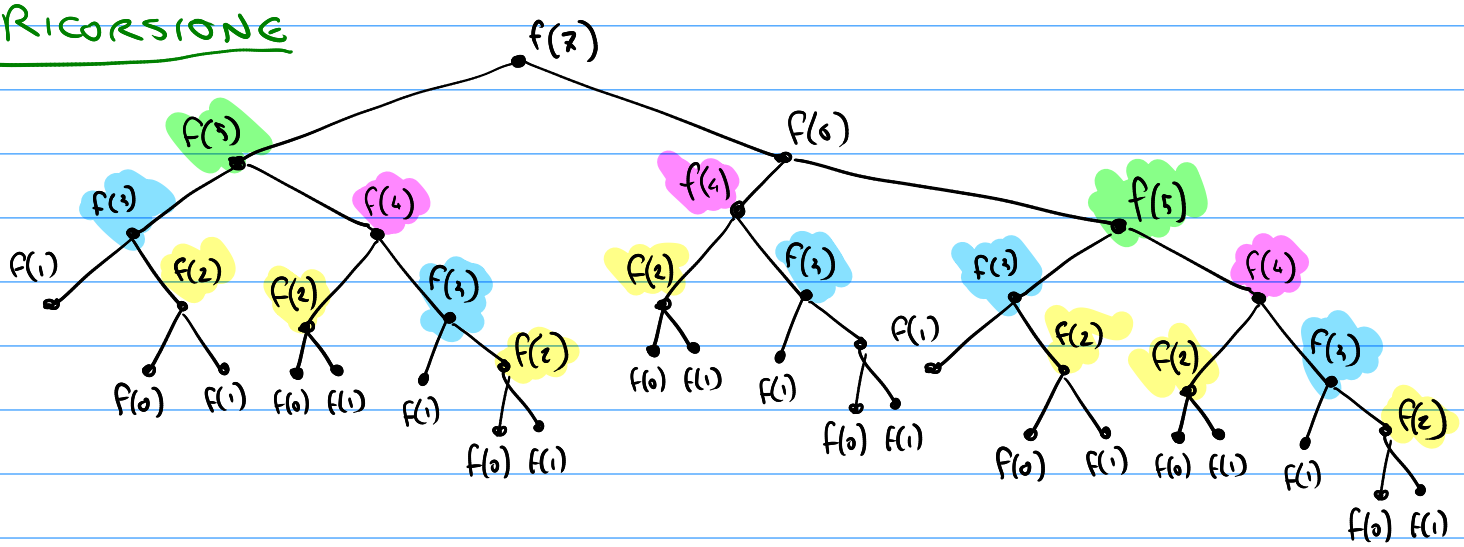
Ritorniamo sulla soluzione ricorsiva di un problema.

Nei passi ① e ② scopriamo che la soluzione ricorsiva di un problema si può ottenere facilmente dalla caratterizzazione delle soluzioni ottime in maniera tale che queste si ottengono facilmente da
SOLUZIONI OTTIME DI SOTTOPROBLEMI

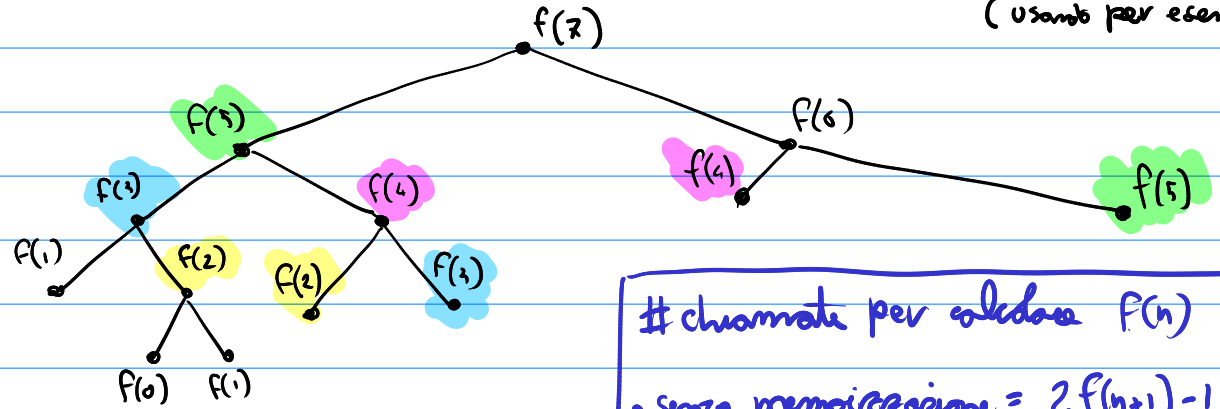
Es. Fibonacci

$$f(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f(n-1)+f(n-2) & n \geq 2 \end{cases}$$

RICORSIONE



La memoizzazione permette di evitare di ricalcolare i valori calcolati in precedenza (usando per esempio un dizionario)



chiamate per calcolare $f(n)$

- senza memoizzazione = $2f(n+1)-1 \approx 2^{\Theta(n)}$
- con memoizzazione =

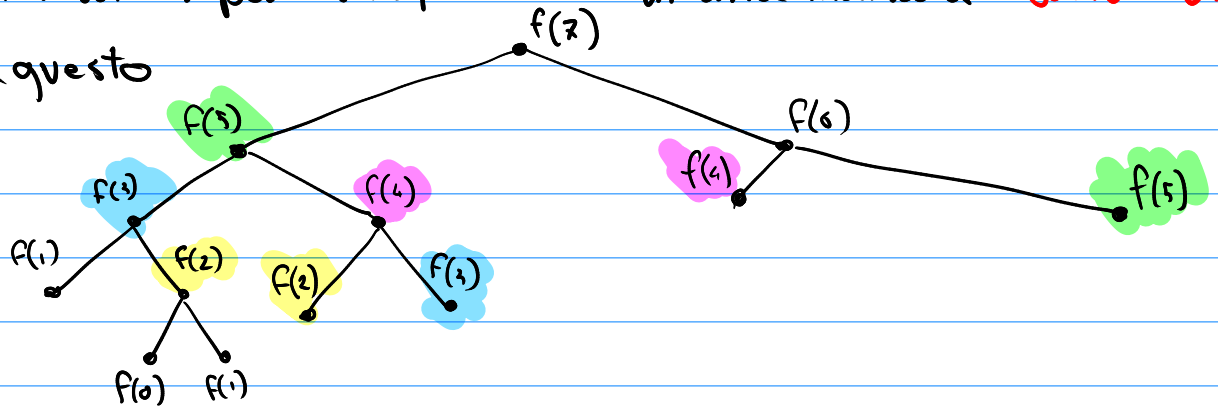
In generale la memoizzazione può essere usata per mantenere il risultato della chiamata di QUALUNQUE FUNZIONE PRIVA DI SIDE EFFECT, chiamata su dati "hashable" (in gergo python) il cui valore dipende SOLO dagli argomenti.

```
def solve(a,b,c):
    if ("solve", a,b,c) in CACHE:
        return CACHE[("solve", a,b,c)]
```

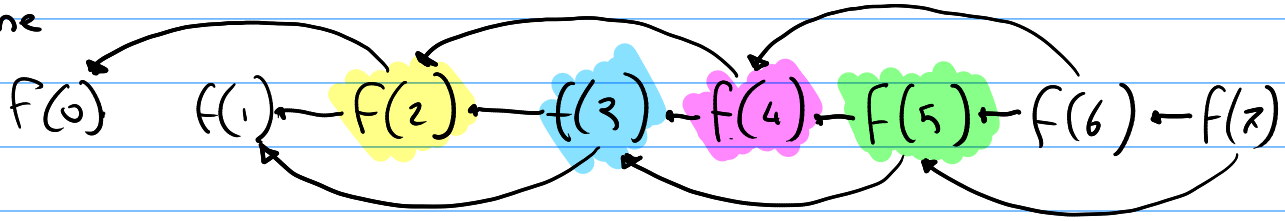
} codice aggiuntivo

Nel caso di questa funzione (e degli esempi di programmazione dinamica visti), è possibile evitare la memoizzazione e calcolare DIRETTAMENTE i risultati per i sottoproblemi in una maniera BOTTOM-UP

Da questo



si ottiene



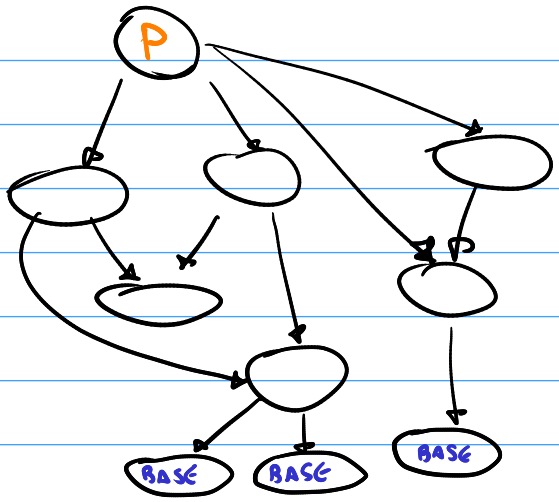
Invece di calcolare in maniera ricorsiva TOP-DOWN

- partendo dal problema principale
- estraendo e calcolando i sottoproblemi

possiamo farlo in maniera BOTTOM-UP

- calcoliamo i sottoproblemi più piccoli (casi base)
- da essi costruiamo le soluzioni a sottoproblemi sempre più complessi
- fino alla soluzione del problema principale

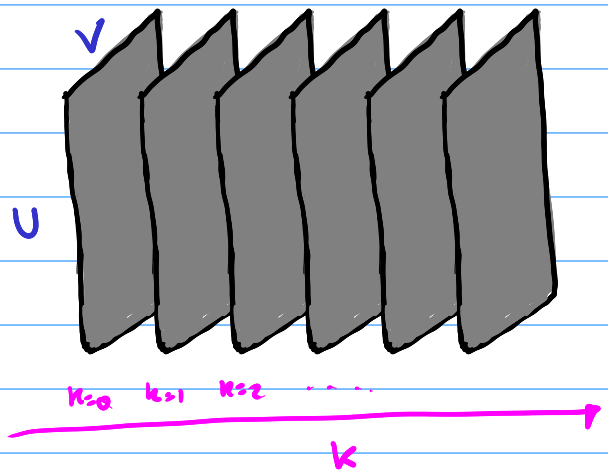
Possiamo quindi considerare il grafo dei sottoproblemi $G = (V, E)$

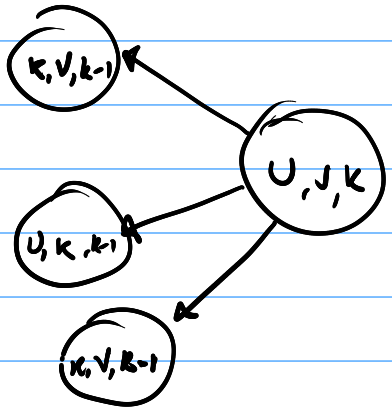


- $V(G)$ = i sottoproblemi
- G è un grafo diretto dove $(u, v) \in E$ se il calcolo della soluzione di u dipende dalla soluzione di v
- P il problema principale è la sorgente
- G è aciclico

- Il calcolo consiste in una visita in ordine topologico inverso di G a partire dai casi base fino al problema principale
- Vengono risolti $|V(G)|$ sottoproblemi
- Vengono effettuate $1+|E(G)|$ letture dei risultati dei sottoproblemi

Es. Floyd-Warshall





In questo caso $G = (V, E)$ con $|V| = \Theta(n^3)$ e $|E| \approx 3 \cdot |V|$

Alcune osservazioni su MEMOIZZAZIONE (TOP-DOWN) contro calcolo BOTTOM-UP

- a parità di operazioni il calcolo bottom-up non utilizza chiamate ricorsive a funzione quindi è più leggero
- il metodo con memorizzazione calcola solo i sottoproblemi veramente necessari per risolvere il problema principale
  (va detto che nei casi visti, tutti i sottoproblemi erano necessari)
- il metodo BOTTOM-UP permette di gestire meglio la memoria

Esempi di gestione della memoria

Fibonacci: a ogni passo ci servono solo 3 valori
- il valore corrente (da calcolare)
- i due precedenti valori

FW: per calcolare uno strato del grafo dei sottoproblemi è sufficiente lo strato precedente.

ESERCIZIO (15.2-4)

- Considerate il problema della parentesizzazione ottima
  Data una sequenza di n matrici,

  descrivere il grafo dei sottoproblemi.