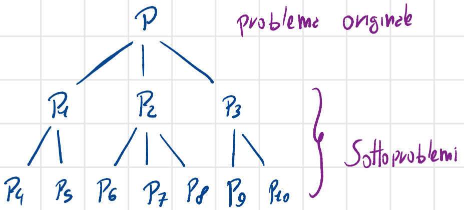
PROGRAMMAZIONE DINAMICA

Programmazione dinamica



Si usa quando la scomposizione del problema è fatta in sotto problemi indipendenti

Operando solo su problemi distinti

risolvendo così un problema e salvarlo in una tabella se in caso si riproponesse

Processo di sviluppo per un algo di P.D

1) Caratterizzare la struttura di una soluzione ottima
2) Definire in modo ricorsivo il valore di una soluzione ottima
3) Calcolare il valore di una soluzione ottima con lo schema bottom up
4) Costruire la soluzione ottima dalle informazioni calcolate

Base per risolvere un problema con la P.D

Numeri di Fibonacci

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

Approccio TOP-DOWN

```
Memo-Fib(n)
    F: nuovo array (0:n)      // dimensione n+1
    for i=0 to n
        F[i] = -1
    return FibRic (n,F)


FibRic (n,F)
    if(n==0 OR n==1) return n;
    if (F[n] ≠ -1) return F[n]      // il ris era già in F()
    else {
        F[n] = FibRic(n-1, F) + FibRic (n-2, F);
        return F[n]
    }
```

Approccio con P.D, bottom-UP

Fib(n)

F: nuovo array: [0 : n]
F[0] = 0
F[1] = 1
For i = 2 to n
    F[i] = F[i-1] + F[i-2];
return F[n]

0	1	2	3	4	5	6	7
0	1	1	2	3	5	8	13

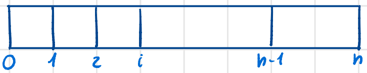
Struttura algoritmo di P.D

1. Sottostruttura Ottima:
   la soluzione ottima del problema deriva dalle soluzioni ottime dei sottoproblemi
2. Sovrapposizione (ripetizione) dei sotto problemi

Taglio delle Corde

Data una corda di lunghezza n cm e una tabella di prezzi, trovare il ricavo massimo r_n ottenibile tagliando la corda e vendendone i pezzi

P_i = prezzo di un pezzo di corda di lunghezza i cm



$\forall$ cm intermedio posso decidere se tagliare o meno $\rightsquigarrow 2^{n-1}$

Ricavo massimo ottenibile: $\underbrace{P(i)}_{\text{Prezzo del pezzo lungo } i} + r_{n-1}$

In generale il ricavo massimo r_n: $r_n = \max\limits_{1 \le i \le n} \left(P[i] + r_{n-1} \right)$ $i = n$, $P[n]$, $r_0 = 0$

Soluzione ricorsiva

```
CUT-ROD(P, n)
    if (n==0)  return 0
    q = -∞

    for i=1 to n
        q = max{q, P[i] + CUT-ROD(P, n-1)}
    return q
```

Inefficente a causa della ripetizione dei sottoproblemi

Soluzione con P.D

1 Definizione dei sottoproblemi

$\Pi_j = r_j$ Corda di lunghezza j cm $\quad 0 \leq j \leq n$

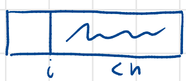
Tabella: array di dim $n+1$

$r[j] = r_j$ = max ricavo ottenibile con corda di j cm

2 Sottoproblemi elementari

$n=0 \quad r_0 = 0 \quad r[0] = 0$

3 Regola ricorsiva per riempire la tabella



$$r_n = \max_{1 \leq i \leq n} (P_i + r_{n-1})$$

4 return $r[n]$

<u>CUT-ROD-PD(P, n)</u>

```
①    r = nuovo array  r[0:n]
②    r[0] = 0
③    for j=1 to n {          // al crescere della lunghezza j della corda
         q = -∞               // ricavo massimo
         for i=1 to j {            // i: posizione primo taglio
             if (q < P[i] + r[j-1])
                 q = P[i] + r[j-i];
         } r[j] = q
     }
④    return r[n];
```

Longest Common Subsequence

Date due sequenze X e Y di m e n caratteri, trovare la sottosequenza comune più lunga (che contiene il maggior numero di caratteri)

Def: Z è una SS di X se si può ottenere da X cancellando alcuni caratteri

$X =$ Spiegare

$Z =$ Spie ✓ $\longrightarrow$ Spie~~gare~~

$Z =$ Spiare ✓ $\longrightarrow$ Spi~~eg~~are

Def: Z è CS di X e Y se è SS di X e Y

$X =$ Spiegare

$Y =$ Ospitare

$Z =$ Spie ✓ $\longrightarrow$ $CS(X, Y)$

$Z =$ Spiare ✓ $\longrightarrow$ $LCS(X, Y)$

Soluzione con forza bruta

$$\left.\begin{array}{l} |X| = m = \#\text{ Caratteri di } X \\ |Y| = n = \#\text{ Caratteri di } Y \end{array}\right\} \quad m \le n$$

Idea: $\underbrace{\forall ss\ z \text{ di } X}_{2^m}$ verifico se $\underbrace{z \text{ è SS di } Y}_{O(n)}$ e tengo traccia delle lunghezze

$\to T(n, m) = O(n 2^m)$

Definizione dei sottoproblemi

$X = x_1, x_2, \dots, x_n$ $\qquad X_i = 0 \le i \le n$ $\quad i$ caratteri di X $\qquad X_0 = \varnothing \{\varepsilon\}$

$Y = y_1, y_2, \dots, y_n$ $\qquad Y_j = 0 \le j \le n$ $\quad j$ caratteri di Y $\qquad Y_0 = \varnothing \{\varepsilon\}$

$\Pi_{ij} = LCS(X_i, Y_j) \qquad 0 \le i \le m \qquad 0 \le j \le n \qquad LCS(X, Y) = \Pi_{mn}$

Soluzione ricorsiva

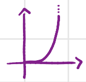
Trovare la lunghezza $|LCS(X,Y)|$

$$|LCS(X,Y)| \begin{cases} |LCS(X_{m-1},Y_{n-1})|+1 & X_m = Y_n \\ \max\{|LCS(X_{m-1},Y)|, |LCS(X,Y_{n-1})|\} & X_m \neq Y_n \end{cases}$$

Probleme di questa soluzione:

I sottoproblemi $LCS(X_{m-1},Y)$ e $LCS(X,Y_{n-1})$ non sono indipendenti poiché condividono il sottoproblema $LCS(X_{m-1},Y_{n-1})$

È una soluzione esponenziale



Soluzione con PD

① Definizione sottoproblemi

Π_{ij} = trovare la lunghezza della $LCS(X_i, Y_j)$ $\quad 0 \le i \le m \quad 0 \le j \le n$

Tabella PD = matrice $(m+1)\cdot(n+1)$

$c[i,j] = |LCS(X_i,Y_j)|$ Matrice

② Memorizzazione in tabella delle soluzioni dei sottoproblemi elementari

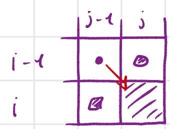
$\forall j,\ 0 \le j \le n \quad LCS(X_0, Y_j) = \emptyset \quad |LCS(X_0,Y_j)| = 0$

$\forall i,\ 0 \le i \le n \quad LCS(X_i, Y_0) = \emptyset \quad |LCS(X_i,Y_0)| = 0$

③ Regola ricorsiva per combinare dei sottoproblemi

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & x_i = y_j \quad i,j>0 \\ \max\{c(i-1,j), c(i,j-1)\} & x_i \neq y_j \quad i,j>0 \end{cases}$$

la tabella si riempie da sx a dx



④ return $c[m,n]$

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
Siano b[1..m, 1..n] e c[0..m, 0..n] due nuove tabelle
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c e b
```

i \ j		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

Edit Distance

Calcola quanto due stringhe sono diverse fra loro, con queste regole

0 - Match = caratteri corrispondenti uguali

+1 - Mismatch = caratteri corrispondenti diversi

+1 - Space = spazio

```
ALBERO          ALBE_RO
LABBRO          _LAB BRO
++0+00 = 3      +0++ + 00 = 4
```

Problema edit distance

trovare un allineamento ottimo, ovvero la distanza minima fra le sequenze.
Che minimizzi i casi di mismatch e space

Regole ricorsive

$$M(i,j) = \begin{cases} 0 & i=0, j=0 \\ i & i\neq 0, j=0 \\ j & i=0, j\neq 0 \\ \min\{M(i-1,j-1)+p,\ M(i-1,j)+1,\ M(i,j-1)+1\} \\ p = \begin{cases} 0 & x_i = y_j \\ 1 & x_i \neq y_j \end{cases} \end{cases}$$

Sottoproblemi elementari

Edit Distance(x,y)

```
M = new matrice (n+1)·(m+1)
For (i=0 to n) {M(i,0) = i}
For (j=0 to m) {M(0,j) = j}
For (i=1 to n) {
    For (j=1 to m) {
        if (x_i == y_j)   p=0;
        else p=1
        M[i,j] = min{M(i-1,j-1)+p, M(i-1,j)+1, M(i,j-1)+1}
    }
} return M
```

$$T(n,m) = \Theta(n \cdot m)$$
$$S(n,m) = \Theta(n \cdot m)$$

Problema dello zaino 0-1

Problema di ottimizzazione per trovare la combinazione di oggetti che massimizza il valore totale degli oggetti presenti nello zaino

Input: n oggetti $A = \{a_1, a_2, \dots, a_n\}$ $\quad A = \boxed{a_1 \mid a_2 \mid \dots \mid a_n}$ $\quad W =$ massimo peso ospitabile

ciascuno con un peso e un valore

$v_1, v_2, \dots, v_n \longrightarrow$ Array V di valori

$w_1, w_2, \dots, w_n \longrightarrow$ Array W di pesi

Output: il sottoinsieme di oggetti che forma il carico più prezioso di peso $\leq W$

Forza bruta

Contiene tutti i possibili sottoinsiemi di elementi, calcolandone peso e valore

$T(n, w) = O(2^n n)$

Tecnica Greedy ①

Ordino per valore e ad ogni passo faccio la scelta ottima in quel momento

$T(n) = \Theta(n \log n)$

Oggetti	Peso (Kg)	Valore (€)
a_1	5	10
a_2	4	5
a_3	4	6

$W = 8$ Kg

Zaino $= \{a_1\} \longrightarrow$ Valore carico $= 10$

Tecnica Greedy ②

Ordino gli oggetti per valore specifico v_i / w_i $\qquad T(n) = \Theta(n \log n)$

Oggetti	Peso (Kg)	Valore (€)	Valore Specifico
a_1	5	10	2
a_2	4	5	1.25
a_3	4	6	1.5

$W = 8$ Kg

Zaino $= \{a_1\}$ $\qquad$ Ottimo $\{a_2, a_3\}$

Soluzione PD

① Sottoproblema generico $0 \le i \le n \quad 0 \le j \le W$

Π_{ij} : trovare il carico più prezioso di peso $\le j$ avendo a disposizione i primi i oggetti

Tabella $(n+1) \cdot (W+1)$ $\quad M[i,j]$ = valore del carico più prezioso di somma $\le j$, composto da un sottoinsieme dei primi i elementi

② Sottoproblemi elementari

$i=0 \quad \forall j,\ 0 \le j \le W, \quad M[0,j] = 0$

$j=0 \quad \forall i,\ 0 \le i \le n, \quad M[i,0] = 0$

③ Regola ricorsiva

- $w_i > j \Rightarrow$ non posso prendere a_i
- $w_i \le j \Rightarrow$ posso prenderlo

```
Zaino-01(n, v, w, W)
    M = new matrice (n+1)·(W+1) inizializzata a 0
Θ(W)  for (j=0 to W)   M[0,j] = 0
Θ(n)  for (i=0 to n)   M[i,0] = 0
      for (i=1 to n) {
          for (j=1 to W) {
              M[i,j] = M[i-1, j];
Θ(nW)         if (w[i] ≤ j) {
                  val = v[i] + M[i-1, j-w[i]];
                  if (val > M[i,j]) {M[i,j] = val}
              }
          }
      }
      return M[n,W];
```

$T(n,W) = \Theta(n \cdot W)$

$S(n,W) = \Theta(n \cdot W)$

Tecnica Greedy

- Sequenza di scelte "localmente ottime" e risolve solo il sottoproblema che deriva da queste scelte
- Non ha sempre soluzione ottima

Elementi di tecnica Greedy

① Scelta Greedy:

Una soluzione ottima può essere ottenuta facendo scelte localmente ottime

② Sottostruttura ottima

Una soluzione ottima contiene soluzioni ottime dei sottoproblemi

Calcolabilità

Classifica i problemi in risolubili e non risolubili, mentre la complessità in "facili" e "difficili"

Problema dell'arresto

Verificare in tempo finito se un algoritmo termina o meno

Arresto: {Instanze} → {0, 1}

è un problema posto in forma decisionale, la sua calcolabilità è chiamata decidibilità

è un algoritmo che indaga sulle proprietà di un altro algoritmo