PROGRAMMAZIONE DINAMICA

Prof. Reho Gabriella

Quando si usa P.D.?

- La programmazione dinamica si usa nei casi in cui esista una definizione ricorsiva del problema, ma la trasformazione diretta di tale definizione in un algoritmo genera un programma di complessità esponenziale a causa del calcolo ripetuto sugli stessi sottoinsiemi di dati da parte delle diverse chiamate ricorsive.

Esempi

Analizziamo il calcolo dei numeri di Fibonacci e dei coefficienti binomiali.

Si ha rispettivamente:

$F_0=0$, $F_1=1$ $\qquad$ $F_i = F_{i-1} + F_{i-2}$

(n su k) = (n-1 su k) + (n-1 su k-1)

Entrambe le formule conducono a un numero esponenziale di operazioni se interpretate come espressioni di calcolo ricorsivo (per es. calcolare Fi chiamando ricorsivamente la stessa procedura per calcolare Fi-1 e Fi-2).

- Se invece, partendo da F0=0, F1=1, si calcolano i numeri di Fibonacci in sequenza come F2=F1+F0, F3=..., si arriva a Fi in i-1 passi e l'algoritmo è quindi lineare

Algoritmo ricorsivo per (n su k)

```
int Combin (int n, int k){
   if (n==k || k=0)
      return 1;
   return Combin(n-1,k-1)+ Combin(n-1,k);
}
```

La complessità di questo algoritmo cresce con il numero di chiamate ricorsive che è proprio C(n,k)=n!/(k!(n-k)!). Ciò è dovuto all'elevato numero di sottoproblemi identici che vengono risolti più volte!

Algoritmo polinomiale per il calcolo di (n su k)

- Si può calcolare (n su k) riempiendo le prime n righe del triangolo di Tartaglia a partire dalla riga 0 fino alla riga n-1, calcolando gli elementi della riga i per addizioni sulla riga i-1. Si può quindi calcolare direttamente il valore (n su k)=(n-1 su k)+(n-1 su k-1) con un'addizione perché i due addendi nella riga n-1 sono ora noti. Il numero di passi è quadratico, perché le caselle del triangolo fino alla riga n-1 sono circa $n^2/2$.

```
int C[n+1][n+1];

int  Combin(int n, int k){
  C[0][0]=1;
  for(int i=2;i<n;i++)
   for(int j=1;j<i-1;j++)
    C[i][j]=C[i-1][j-1]+C[i-1][j];

  return  C[n][k];
}
```

- Altre applicazioni classiche della programmazione dinamica si incontrano nel **confronto tra sequenze di caratteri**: problemi nati in relazione agli editori di testo, e oggi importantissimi nelle applicazioni algoritmiche in biologia molecolare (analisi del DNA ecc). Vediamo lo schema di base e come questo possa essere facilmente modificato per risolvere diversi problemi.

Il problema della SOTTOSEQUENZA COMUNE PIU` LUNGA

•Una sottosequenza di una certa sequenza è la sequenza originale con l'eventuale esclusione di alcuni elementi. La lunghezza di una sottosequenza è il numero di elementi che la compongono.
Consideriamo quindi il seguente problema:

- Date due sequenze
  X = (x1, x2, . . . xn) e Y = (y1, y2, . . . , ym),
  trovare la più lunga sottosequenza comune ad X e Y.

- Ad esempio, per le due sequenze di interi
X=(9, 15, 3, 6, 4, 2, 5, 10, 3) e Y=(8, 15, 6, 7, 9, 2, 11, 3,1), la soluzione al problema è la sottosequenza (15, 6, 2, 3).

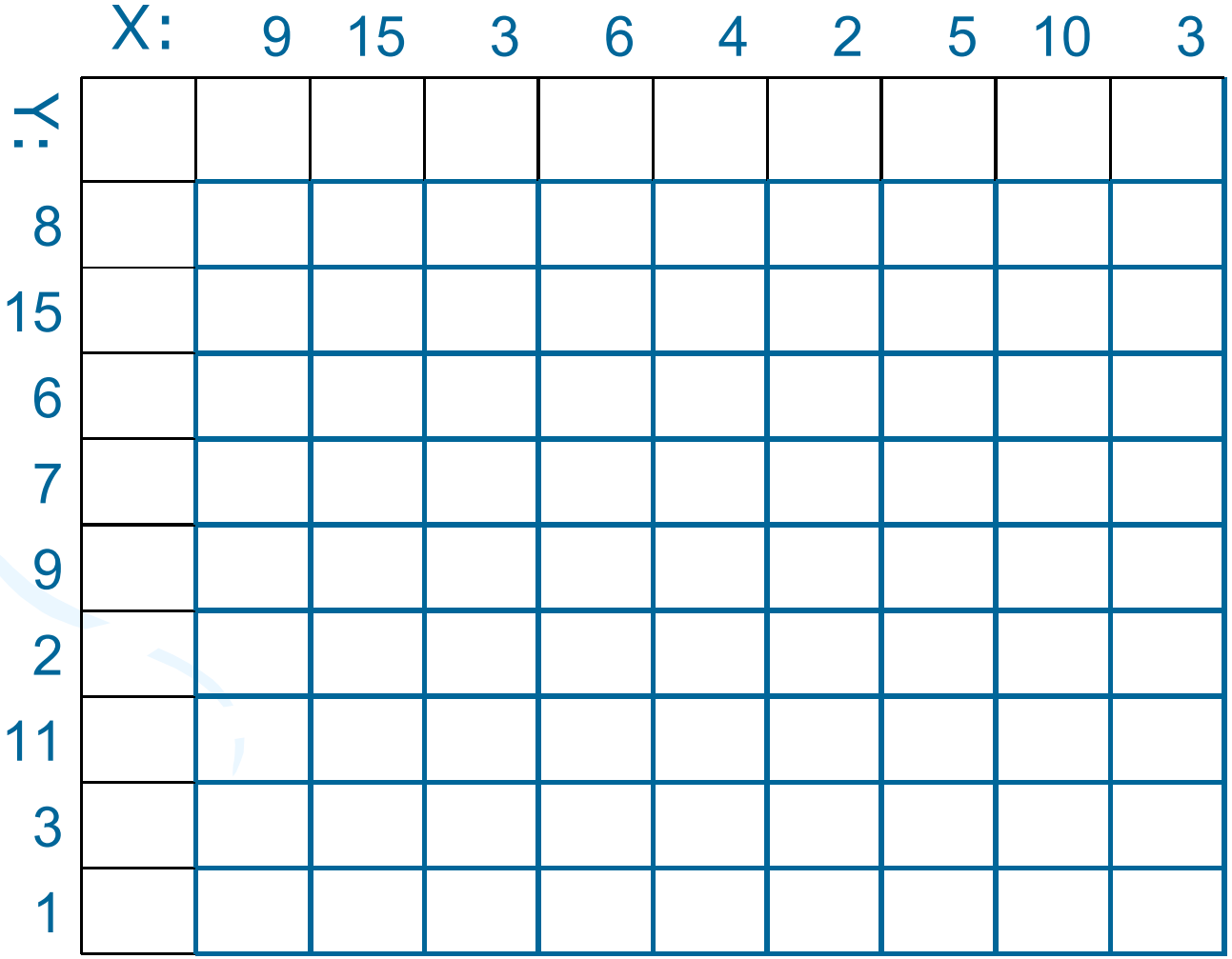
- Per una sequenza di n elementi esistono 2^n distinte sottosequenze.
- Un algoritmo che tenti di risolvere il problema considerando tutte le sottosequenze possibili della prima sequenza per vedere quale di queste sottosequenze è sottosequenza anche della seconda tenendo traccia della sottosequenza comune più lunga via-via trovata, non può che avere una complessità esponenziale.

Soluzione con P.D.

- Per ora considereremo solamente il calcolo della lunghezza della sottosequenza comune più lunga e successivamente vedremo come modificare l'algoritmo per ottenerla.
- Per i<=n e j<=m sia T[i, j], la lunghezza della sottosequenza più lunga comune alle sequenze Xi =(x1, x2, . . . xi) e Yj = (y1, y2, . . . yj).
- La lunghezza della sottosequenza più lunga comune ad X e Y è data quindi da T[n,m].

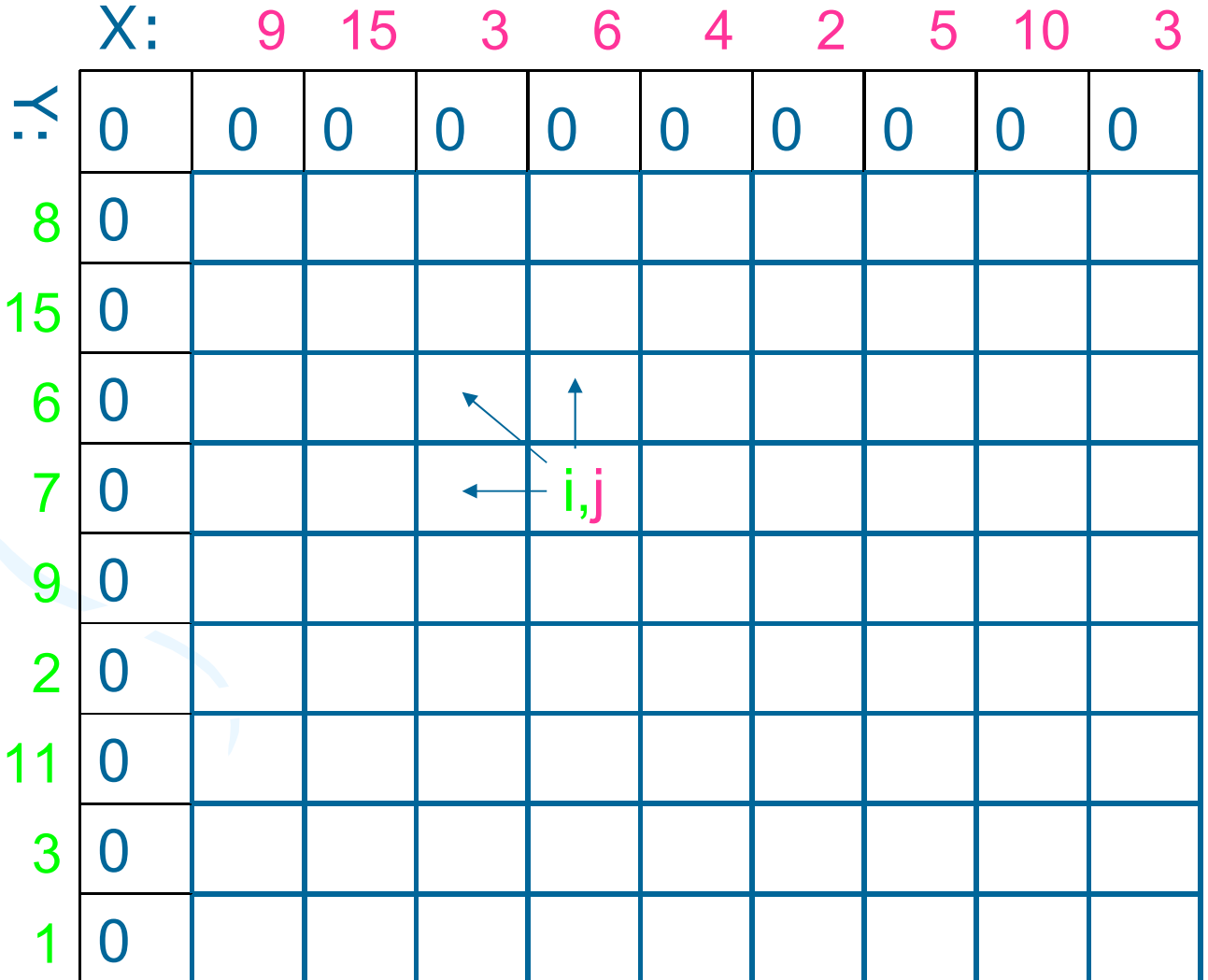
La matrice T[n][m]

- La soluzione è quindi rappresentata da una matrice T di n righe e m colonne



Riempiamo la matrice T[n][m]

E' ovvio che T[i, 0] = 0 per 0<=i<=n
e T[0, j] = 0 per 1<=j<=m.



Riempiamo la matrice T[n][m]

- Per i e j entrambi maggiori di 0 possiamo ragionare come segue distinguendo due casi:
- CASO xi = yj Sia a il simbolo con cui terminano le due sequenze Xi e Yj (cioè xi = yj). In questo caso la sottosequenza comune ad Xi e Yj più lunga termina con il simbolo a (infatti una qualunque sottosequenza comune che non termini con a resta comune anche con l'aggiunta del simbolo a).
- In questo caso quindi si ha T[i, j]=T[i−1, j−1]+1.

CASO $x_i \neq y_j$. In questo caso per la sottosequenza comune S più lunga non possono esserci che le seguenti tre alternative:

- S termina con il simbolo x_i → S avrà lunghezza $T[i, j-1]$
- S termina con il simbolo y_j → S avrà lunghezza $T[i-1, j]$
- S termina con un simbolo a diverso da x_i e y_j → S sarà $T[i-1, j-1]$

Per decidere qual'è la risposta giusta basta quindi porre

$T[i, j] = \max\{T[i, j-1], T[i-1, j], T[i-1, j-1]\} = \max\{T[i, j-1], T[i-1, j]\}$

Dove l'ultima uguaglianza segue banalmente dal fatto che $T[i-1, j-1] <= T[i, j-1]$ e $T[i-1, j-1] <= T[i-1, j]$.

- Da quanto detto si arriva alla seguente formula ricorsiva:

$$T[i, j] = \begin{cases} T[i-1, j-1] + 1 & \text{se } x_i = y_j \\ \max\{T[i, j-1], T[i-1, j]\} & \text{altrimenti.} \end{cases}$$

Riempiamo la matrice T[n][m]

Partiamo dall'elemento T[1,1], considerato che siamo nel caso x1≠y1…

		9	15	3	6	4	2	5	10	3
	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
15	0	0								
6	0									
7	0									
9	0									
2	0									
11	0									
3	0									
1	0									

Riempiamo la matrice T[n][m]

Arriviamo ora all'elemento T[2,2], considerato che siamo nel caso x2=y2…

		9	15	3	6	4	2	5	10	3
	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
15	0	0	1							
6	0									
7	0									
9	0									
2	0									
11	0									
3	0									
1	0									

Riempiamo la matrice T[n][m]

		9	15	3	6	4	2	5	10	3
	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
15	0	0	1	1	1	1	1	1	1	1
6	0	0	1	1	2	2	2	2	2	2
7	0	0	1	1	2	2	2	2	2	2
9	0	1	1	1	2	2	2	2	2	2
2	0	1	1	1	2	2	3	3	3	3
11	0	1	1	1	2	2	3	3	3	3
3	0	1	1	2	2	2	3	3	3	4
1	0	1	1	2	2	2	3	3	3	4

Algoritmo per il calcolo della lunghezza della sottosequenza

```
LUNGHEZZA-SOTTOSEQUENZA:
INPUT le due sequenze x1, . . . , xn e y1, . . . , ym
{
  for (int i=0;i<=n;i++)
     T[i, 0] = 0;
   for (int j=0;j<=n;j++)
     T[0, j]= 0;
 for (int i=0;i<=n;i++)
   for (int j=0;j<=n;j++)
     if (xi==yj) T[i][ j]=T[i − 1][j − 1] + 1;
     else T[i, j]=  max{T[i][j − 1], T[i − 1][ j]}

  return T[n,m]
}
```

Come si ricava la sottosequenza massima?

- Sappiamo come i valori della matrice T sono stati ottenuti quindi per ottenere la sottosequenza comune di lunghezza massima basta ripercorrere ”all’indietro” le varie scelte fatte per ottenere il valore T[n,m].
- Più precisamente la seguente procedura ricorsiva permette di ricostruire la sottosequenza comune

Graficamente…

		9	15	3	6	4	2	5	10	3
	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
15	0	0	1	1	1	1	1	1	1	1
6	0	0	1	1	2	2	2	2	2	2
7	0	0	1	1	2	2	2	2	2	2
9	0	1	1	1	2	2	2	2	2	2
2	0	1	1	1	2	2	3	3	3	3
11	0	1	1	1	2	2	3	3	3	3
3	0	1	1	2	2	2	3	3	3	4
1	0	1	1	2	2	2	3	3	3	4

Algoritmicamente…

```
void STAMPA-SOTTOSEQUENZA-MASSIMA (int i,int  j){
  if (i > 0 && j > 0)
   if (xi == yj){
    STAMPA–SOTTOSEQUENZA–MASSIMA(i – 1, j – 1)
    cout<< xi;
   }
   else
    if (T[i – 1][j]>= T[i][j – 1])
      STAMPA-SOTTOSEQUENZA-MASSIMA(i – 1, j)
    else STAMPA-SOTTOSEQUENZA-MASSIMA(i, j – 1)
}
```

Ridefiniamo il problema precedente in biologia molecolare

- Nelle applicazioni biologiche spesso si confronta il DNA di due, o più, organismi differenti
- La struttura del DNA è formata da una stringa di molecole dette basi: adenina, citosina, guanina, e timina
- La struttura del DNA è rappresentata da una stringa sull'insieme finito {A, C,G,T}
- Il DNA di due organismi
- S1 = ACCGGTCGCGCGGAAGCCGGCCGAA
- S2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

- Uno degli scopi del confronto tra due molecole di DNA è quello di determinare il grado di somiglianza delle due molecole
- Potremmo dire che due molecole si somigliano se, date le stringhe S1 e S2:
  - una è sottostringa di un'altra
  - il numero delle modifiche richieste per trasformare l'una nell'altra è piccolo
  - trovare una terza stringa S3 le cui basi si trovano in ciascuna delle stringhe S1 ed S2: le basi devono presentarsi nello stesso ordine, senza essere necessariamente consecutive

- Ad esempio, date le stringhe:

$$S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$$

$$S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$$

- La stringa S3 sarà:

$$S_3 = GTCGTCGGAAGCCGGCCGAA$$

La dieta di Poldo (coefficiente di difficoltà D=3)

- **Il problema**
- Il dottore ordina a Poldo di seguire una dieta. Ad ogni pasto non può mai mangiare un panino che abbia un peso maggiore o uguale a quello appena mangiato. Quando Poldo passeggia per la via del suo paese da ogni ristorante esce un cameriere proponendo il menù del giorno. Ciascun menù è composto da una serie di panini, che verranno serviti in un ordine ben definito, e dal peso di ciascun panino. Poldo, per non violare la regola della sua dieta, una volta scelto un menù, può decidere di mangiare o rifiutare un panino; se lo rifiuta il cameriere gli servirà il successivo e quello rifiutato non gli sarà più servito.
- Si deve scrivere un programma che permetta a Poldo, leggendo un menù, di capire qual è il numero massimo di panini che può mangiare per quel menù senza violare la regola della sua dieta.
- Riassumendo, Poldo può mangiare un panino se e solo se soddisfa una delle due condizioni:
- 1) il panino è il primo che mangia in un determinato pasto;
- 2) il panino non ha un peso maggiore o uguale all'ultimo panino che ha mangiato in un determinato pasto.
- **Dati in input**
- La prima linea del file input.txt contiene il numero *m* di panini proposti nel menu. Le successive *m* linee contengono un numero intero non negativo che rappresenta il peso del panino che verrà servito. I panini verranno serviti nell'ordine in cui compaiono nell'input.
- **Dati in output**
- Il file output.txt contiene il massimo numero di panini che Poldo può mangiare rispettando la dieta.
- **Assunzioni**
- I pesi di panini sono espressi in grammi, un panino pesa al massimo 10 Kg.
- Un menù contiene al massimo 100 panini.

Esempi di input e output

Esempio 1

File input.txt

```
8
389
207
155
300
299
170
158
65
```

File output.txt

```
6
```

Esempio 2

File input.txt

```
3
22
23
27
```

File output.txt

```
1
```

Esempio 3

File input.txt

```
22
15
14
15
389
201
405
204
130
12
50
13
26
190
305
25
409
3011
43
909
987
1002
900
```

File output.txt

```
6
```