Algoritmi e Strutture Dati

Capitolo 10

Tecniche algoritmiche

Programmazione Dinamica: principi generali

1) identificare un numero piccolo di sottoproblemi

es: calcolare l'II di peso massimo di G_j, j=1,…,n

2) descrivere la soluzione di un generico sottoproblema in funzione delle soluzioni di sottoproblemi più "piccoli"

es: OPT[j]=max {OPT[j-1], w_j+OPT[j-2]}

3) le soluzioni dei sottoproblemi sono memorizzate in una tabella

4) avanzare opportunamente sulla tabella, calcolando la soluzione del sottoproblema corrente in funzione delle soluzioni di sottoproblemi già risolti.

Proprietà che devono avere i sottoproblemi

1) essere pochi

2) risolti tutti i sottoproblemi si può calcolare velocemente la soluzione al problema originale

spesso la soluzione cercata è semplicemente quella del sottoproblema più grande

3) ci devono essere sottoproblemi "piccoli"

casi base

4) ci deve essere un ordine in cui risolvere i sottoproblemi

e quindi un modo di avanzare nella tabella e riempirla

ancora sul ruolo dei sottoproblemi

…maledetti, favolosi sottoproblemi!

La chiave di tutto è la definizione dei "giusti" sottoproblemi

La definizione dei "giusti" sottoproblemi è un punto di arrivo

Solo una volta definiti i sottoproblemi si può verificare che l'algoritmo è corretto

Se la definizione dei sottoproblemi è un punto di arrivo, come ci arrivo?

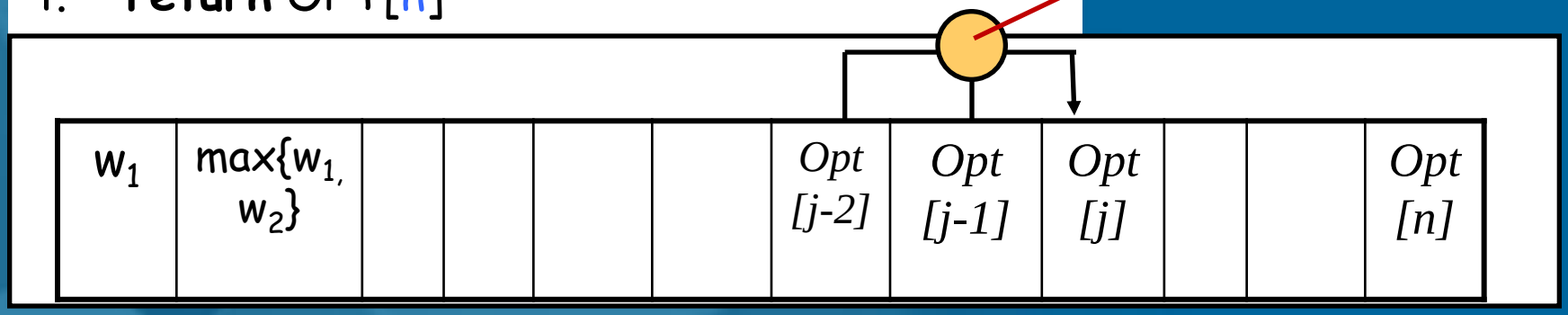
… ragionando sulla struttura della soluzione (ottima) cercata.

La struttura della soluzione può suggerire i sottoproblemi e l'ordine in cui considerarli

1. $OPT[1]=w_1$; $OPT[2]= \max\{w_1, w_2\}$
2. **for** $j=3$ **to** n **do**
3. $OPT[j]= \max\{OPT[j-1], w_j+OPT[j-2]\}$
4. **return** $OPT[n]$

un esempio

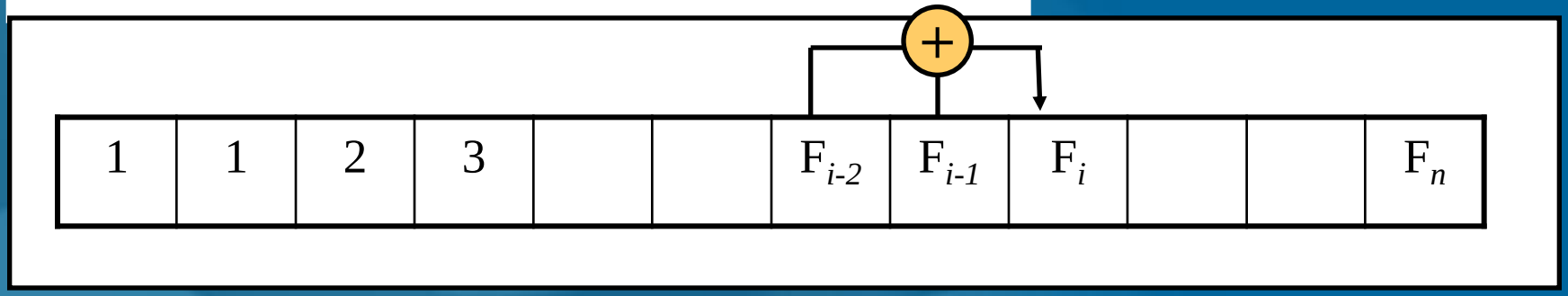
$\max\{\bullet, w_j+\bullet\}$



1. Indentifichiamo i sottoproblemi: il *j*-esimo problema consiste nel calcolo dell'insieme indipendente di peso massimo del sottocammino composto dai primi *j* nodi. Soluzioni memorizzate in una tabella
2. Calcoliamo le soluzioni di alcuni sottoproblemi "facili": OPT[1] e OPT[2]
3. Al *j*-esimo passo, avanziamo sulla tabella e calcoliamo soluzione al *j*-esimo sottoproplema in base alle soluzioni precedentemente calcolate. $OPT[j] \leftarrow \max\{OPT[j-2], w_j+OPT[j-1]\}$
4. Restituiamo il valore memorizzato in un particolare elemento della tabella, ovvero OPT[*n*]

un altro esempio

```
algoritmo fibonacci3(intero n) → intero
    sia Fib un array di n interi
    Fib[1] ← Fib[2] ← 1
    for i = 3 to n do
        Fib[i] ← Fib[i-1] + Fib[i-2]
    return Fib[n]
```



1. Indentifichiamo i sottoproblemi: l'*i*-esimo problema consiste nel calcolo dell'*i*-esimo numero di fibonacci. Soluzioni memorizzate in una tabella
2. Calcoliamo le soluzioni di alcuni sottoproblemi "facili": F[1]=1 e F[2]=1
3. All'*i*-esimo passo, avanziamo sulla tabella e calcoliamo soluzione all'*i*-esimo sottoproplema in base alle soluzioni precedentemente calcolate. F[*i*]←F[*i*-1]+F[*i*-2]
4. Restituiamo il valore memorizzato in un particolare elemento della tabella, ovvero F[*n*]

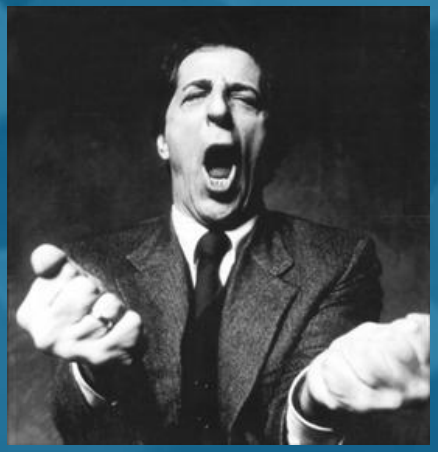
Il calcolo della distanza fra due parole

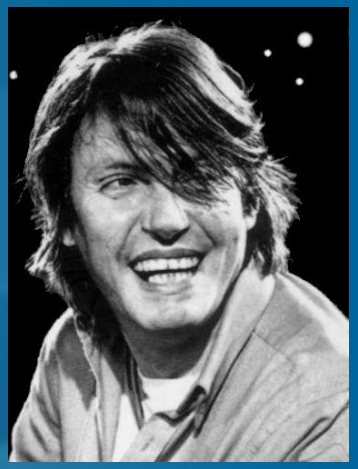
(ancora un algoritmo di programmazione dinamica)

Distanza fra due stringhe

- Come definire la distanza?
- Intuitivamente:

`gaber` e `faber` sono stringhe molto simili (vicine)





`bobdylan` e `gigidalessio` sono molto diverse (lontane)

Distanza fra due stringhe

- Come definire la distanza?

- edit distance:
  - numero **minimo** di “modifiche elementari” per trasformare una stringa in un’altra
- motivazioni:
  - edit distance usata nei correttori ortografici (*spell checker*):
    - se una parola digitata non è nel dizionario sostituiscila con la più vicina presente nel dizionario

Distanza tra stringhe

Siano X e Y due stringhe di lunghezza m ed n:

$$X = x_1 \cdot x_2 \cdot \cdots \cdot x_m \qquad Y = y_1 \cdot y_2 \cdot \cdots \cdot y_n$$

Vogliamo calcolare la "distanza" tra X e Y, ovvero il minimo numero delle seguenti operazioni elementari che permetta di trasformare X in Y

`inserisci`(a):	Inserisci il carattere a nella posizione corrente della stringa.
`cancella`(a):	Cancella il carattere a dalla posizione corrente della stringa.
`sostituisci`(a, b):	Sostituisci il carattere a con il carattere b nella posizione corrente della stringa.

Esempio

- Come posso trasformare la stringa `risotto` in `presto`?
- Con 13 operazioni è semplice:
  - 7 cancellazioni
  - 6 inserimenti
- è 13 la distanza?
- …NO! C'è un numero minore di operazioni che trasforma la prima stringa nella seconda (e viceversa)

Esempio

Azione	Costo	Stringa ottenuta
Inserisco P	1	P|*RISOTTO*
Mantengo R	0	PR|*ISOTTO*
Sostituisco I con E	1	PRE|*SOTTO*
Mantengo S	0	PRES|*OTTO*
Cancello O	1	PRES|*TTO*
Mantengo T	0	PREST|*TO*
Cancello T	1	PREST|*O*
Mantengo O	0	PRESTO|

la distanza è 4

come definiamo i sottoproblemi?

(ragioniamo sulla struttura della soluzione cercata)

…cercando i sottoproblemi giusti

$X = x_1 x_2 \ldots x_m$

$Y = y_1 y_2 \ldots \ldots \ldots y_n$

$\delta(X,Y)$: distanza fra X e Y

Quale è l'ultima operazione eseguita dalla sequenza di trasformazione ottima?

Caso $x_m = y_n$ (facile)

ultima operazione è un mantieni (di costo 0)

➡ $\delta(X,Y) = \delta(x_1 x_2 \ldots x_{m-1}, y_1 \ldots y_{n-1})$

Caso $x_m \neq y_n$

ultima operazione può essere: sostituisci, cancella, inserisci

Caso $x_m \neq y_n$

ultima operazione è un sostituisci

➡ sostituisci (x_m, y_n)

➡ $\delta(X,Y) = 1 + \delta(x_1x_2\ldots x_{m-1}, y_1\ldots y_{n-1})$

$X = x_1x_2\ldots x_m \rightarrow \ldots \ldots \ldots \rightarrow y_1y_2\ldots y_{n-1}\, x_m \rightarrow Y$

sequenza ottima che trasforma

$x_1x_2\ldots x_{m-1}$ in $y_1y_2\ldots y_{n-1}$

Caso $x_m \neq y_n$

ultima operazione è un cancella

➡ cancella (x_m)

➡ $\delta(X,Y) = 1 + \delta(x_1x_2\ldots x_{m-1}, y_1\ldots y_n)$

$X = x_1x_2\ldots x_m \rightarrow \ldots\ \ldots\ \ldots \rightarrow y_1y_2\ldots y_n\ x_m \rightarrow Y$

sequenza ottima che trasforma
$x_1x_2\ldots x_{m-1}$ in $y_1y_2\ldots y_n$

Caso $x_m \neq y_n$

ultima operazione è un inserisci

➡ inserisci (y_n)

➡ $\delta(X,Y) = 1 + \delta(x_1x_2\ldots x_m, y_1\ldots y_{n-1})$

Qualche idea su come definire i sottoproblemi?

$X = x_1x_2\ldots x_m \rightarrow \ldots \ldots \ldots \rightarrow y_1y_2\ldots y_{n-1} \rightarrow Y$

sequenza ottima che trasforma $x_1x_2\ldots x_m$ in $y_1y_2\ldots y_{n-1}$

Approccio

- Denotiamo con $\delta(X,Y)$ la distanza tra X e Y
- Definiamo X_i il prefisso di X fino all'i-esimo carattere, per $0 \leq i \leq m$ (X_0 denota la stringa vuota):

$$X_i = x_1 \cdot x_2 \cdot \cdots \cdot x_i \text{ se } i \geq 1$$

- **sottoproblemi**: risolveremo il problema di calcolare $\delta(X,Y)$ calcolando $\delta(X_i, Y_j)$ per ogni i, j tali che $0 \leq i \leq m$ e $0 \leq j \leq n$
  - $\delta(X,Y) = \delta(X_m, Y_n)$
- Manterremo le informazioni in una tabella D di dimensione $(m+1) \times (n+1)$

Inizializzazione della tabella

- Alcuni sottoproblemi sono molto semplici
- $\delta(X_0, Y_j)=j$ partendo dalla stringa vuota X_0, basta inserire uno ad uno i j caratteri di Y_j
- $\delta(X_i, Y_0)=i$ partendo da X_i, basta rimuovere uno ad uno gli i caratteri per ottenere Y_0
- Queste soluzioni sono memorizzate rispettivamente nella prima riga e nella prima colonna della tabella D

Esempio

		P	R	E	S	T	O
	0	**1**	2	3	4	5	6
R	1						
I	2						
S	3						
O	4						
T	5						
T	6						
O	7						

La tabella D inizializzata dall'algoritmo

Avanzamento nella tabella (1/3)

- Se $x_i = y_j$, il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_{i-1} in Y_{i-1}

$$D[i,j] = D[i-1, j-1]$$

- Se $x_i \neq y_j$, distinguiamo in base all'ultima operazione usata per trasformare X_i in Y_j in una sequenza ottima di operazioni

Avanzamento nella tabella (2/3)

`inserisci`(y_j):

il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_i in Y_{j-1} più 1 per inserire il carattere y_j

$$D[i,j] = 1 + D[i, j-1]$$

`cancella`(x_i):

il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_{i-1} in Y_j più 1 per la cancellazione del carattere x_i

$$D[i,j] = 1 + D[i-1, j]$$

Avanzamento nella tabella (3/3)

$\texttt{sostituisci}(x_i, y_j)$:

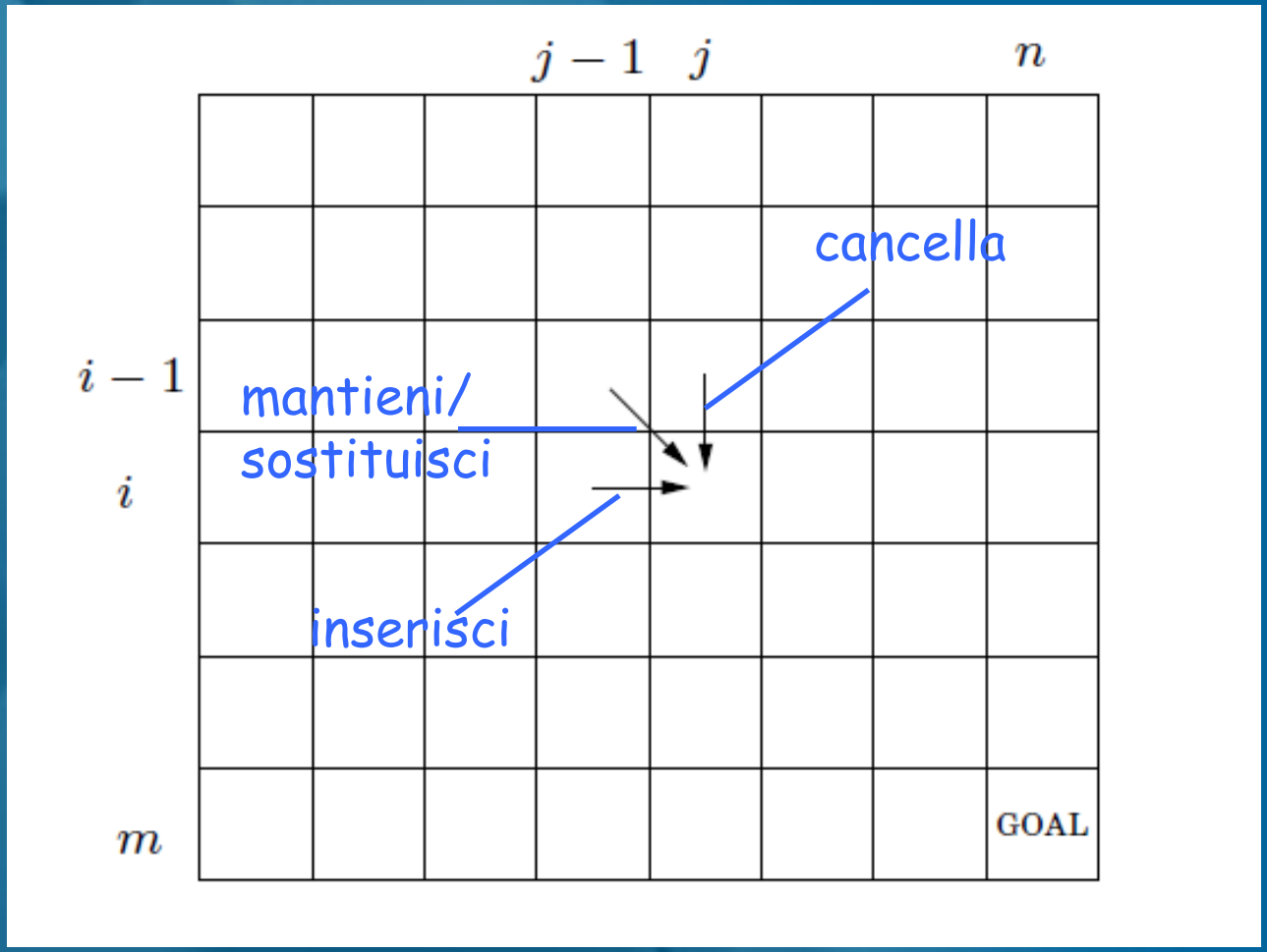
il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_{i-1} in Y_{j-1} più 1 per sostituire il carattere x_i per y_j

$$D[i,j] = 1 + D[i-1, j-1]$$

In conclusione:

$$D[i,j] = \begin{cases} D[i-1, j-1] & \text{se } x_i = y_j \\ 1 + \min\{D[i, j-1], D[i-1, j], D[i-1, j-1]\} & \text{se } x_i \neq y_j \end{cases}$$

…lo schema di avanzamento:



Pseudocodice

algoritmo `distanzaStringhe`$(stringa\ X, stringa\ Y) \rightarrow intero$
    matrice D di $(m+1) \times (n+1)$ interi
    for $i = 0$ **to** m **do** $D[i,0] \leftarrow i$
    for $j = 1$ **to** n **do** $D[0,j] \leftarrow j$
    for $i = 1$ **to** m **do**
        for $j = 1$ **to** n **do**
            if $(x_i \neq y_j)$ **then**
                $D[i,j] \leftarrow 1 + \min\{D[i,j-1], D[i-1,j], D[i-1,j-1]\}$
            else $D[i,j] \leftarrow D[i-1,j-1]$
    return $D[m,n]$

Tempo di esecuzione ed occupazione di memoria: $\Theta(m\ n)$

NOTA: Se tengo traccia della casella della tabella che utilizzo per decidere il valore di D[i,j], ho un metodo costruttivo per ricostruire la sequenza di operazioni di editing ottima.

Esempio

		P	R	E	S	T	O
	0	**1**	2	3	4	5	6
R	1	1	**1**	2	3	4	5
I	2	2	2	**2**	3	4	5
S	3	3	3	3	**2**	3	4
O	4	4	4	4	**3**	3	3
T	5	5	5	5	**4**	**3**	4
T	6	6	6	6	5	**4**	4
O	7	7	7	7	6	5	**4**

La tabella D costruita dall'algoritmo.

In grassetto sono indicate due sequenze di operazioni che permettono di ottenere la distanza tra le stringhe

Esercizio

Modificare l'algoritmo per il calcolo della distanza fra due stringhe in modo da ridurre la complessità spaziale da $\Theta(mn)$ a $\Theta(\min\{m,n\})$, mantenendo la complessità temporale di $\Theta(mn)$.

Esercizio: ricostruire la sequenza di operazioni dalla matrice

Progettare un algoritmo che, data la matrice delle distanze calcolata da `DistanzaStringhe`, calcola una sequenza di operazioni di lunghezza minima che trasforma X in Y. Un algoritmo che, in altre parole, ricostruisce una sequenza di operazioni (inserimento, modifica, cancellazione) che trasforma X in Y. Si analizzi la complessità temporale dell'algoritmo.

Esercizio

Il signor Marche è tornato da un lungo periodo di vacanza fra Roma e Firenze e sta per affrontare un lungo periodo di lavoro che durerà n giorni. Il suo è un lavoro a cottimo e i soldi che riesce a portarsi a casa dipendono da *quanto*, *quando* e *come* lavorerà. Lui sa lavorare in due modi: *forte* e *piano*. Se nel giorno i lavora forte guadagnerà f_i euro, mentre se lavora piano guadagnerà p_i euro. Chiaramente $f_i \geq p_i$. A rendere le cose difficili ci sono dei vincoli di stanchezza. Dopo aver lavorato per un giorno in modo forte, il signor Marche ha bisogno di riposarsi (e quindi di non lavorare) per i successivi due giorni. Inoltre, in ogni caso lui non riesce a lavorare due giorni di seguito. Come potete immaginare i giorni in cui non lavora non guadagna niente.

Progettare un algoritmo di programmazione dinamica che calcoli un piano di lavoro per il signor Marche che gli faccia guadagnare il più possibile.