Programmazione dinamica (IV parte)

Progettazione di Algoritmi a.a. 2019-20

Matricole congrue a 1

Docente: Annalisa De Bonis

Minimum Coin Change Problem

- Dato un insieme infinito di monete con valori $v_1<v_2<v_3<...<v_n$ e una banconota di valore V, fornire una formula per calcolare il minimo numero di monete richieste per cambiare la banconota. Assumiamo $v_1=1$ in modo che il problema ammetta una soluzione.

- Ad esempio: Banconota di 6 euro

Valori monete: 1,2,4

- Possiamo cambiare la banconota in 5 modi:
- {1,1,1,1,1,1}, {1,1,1,1,2}, {1,1,2,2},{1,1,4}, {2,4}
- La soluzione che include meno monete e` quindi {2,4}

Minimum Coin Change Problem

- Greedy non sempre funziona
- Strategia Greedy: esamina i valori delle monete in ordine decrescente e per ciascun valore esamninato utilizza quante piu` monete di quel valore
- Sistema di monete canonico: sistema per il quale la strategia greedy fornisce la soluzione ottima
- Esempio di sistema canonico: sistema USA include monete con questi valori 1, 5, 10, 25 cent.
  - Voglio cambiare 8 cent. La strategia greedy produce la soluzione {5,1,1,1} che e` la soluzione ottima.
- Esempio di sistema non canonico: 1, 4, 5 cent.
- Voglio cambiare 8 cent. La strategia greedy produce la soluzione {5,1,1,1} mentre la soluzione ottima e` {4,4}

Minimum Coin Change Problem

OPT(i,v)= minimo numero di monete di valore $v_1,\ldots,v_i$ per cambiare una banconota di valore v

- Se $v_i \leq v$ bisogna considerare sia il caso in cui la soluzione include monete di valore v_i sia il caso in cui non le contiene:
  - Numero monete nella soluzione ottima tra quelle che contengono una moneta di valore v_i e` = 1+ numero monete nella soluzione ottima per l'importo $v-v_i$ quando si possono utilizzare monete di valore $v_1,\ldots,v_i$
  - Numero monete nella soluzione ottima tra quelle che non contengono una moneta di valore v_i e` = numero monete nella soluzione ottima per l'importo v quando si possono utilizzare monete di valore $v_1,\ldots,v_{i-1}$ (v_1=1)

    → $OPT(i,v)= \min\{OPT(i,v-v_i)+1, OPT(i-1,v)\}$
  - Se v=0 allora OPT(i,v)=0 per ogni i
  - Se i=1 allora OPT(i,v)=v per ogni v

Minimum Coin Change problem

```
MinCoinChange(n, V)
For i = 1 to n
  M[i, 0] ← 0 //importo da cambiare = 0 → soluzione contiene 0 monete
For v = 1 to V
  M[1, v] ← v //si possono usare solo monete da un euro
For i = 1 to n
   For v= 1 to V
    if v < v_i
     then M[i, v] ← M[i-1,v]
    else
     M[i, v] ← min{1+M[i,v-v_i],M[i-1,v]}
```


81

Minimum Coin Change problem

Esercizio: scrivere l'algoritmo che costruisce la soluzione ottima del minimum change coin problem.

82

Coin change problem

- Dato un insieme infinito di monete C di n diversi valori $v_1<v_2<v_3<...<v_n$ ed una certa banconota di valore V, fornire una strategia per trovare in quanti modi possiamo usare le monete in C per cambiare la banconota.
- Ad esempio: Banconota di 6 euro

Valori monete: 1,2,4

- Possiamo cambiare la banconota in 5 modi:
- {1,1,1,1,1,1} {1,1,1,1,2}, {1,1,2,2},{1,1,4}, {2,2,2}, {2,4}

83

Coin change problem

N(i,v)=numero di modi in cui possiamo cambiare v con monete di valore $v_1,...,v_i$

- Se $v_i \leq v$ allora la soluzione puo` includere o meno una moneta di valore v_i
  - Dobbiamo sommare il numero di soluzioni che includono monete di valore v_i al numero di soluzioni che non includono monete di valore v_i
  - $N(i,v)= N(i,v-v_i)+ N(i-1, v)$
- Se il valore v_i e` maggiore dell'importo da coprire allora
  - Le soluzioni possibili sono solo quelle che non includono monete di valore v_i
  - $N(i,v)=N(i-1, v)$

84

Sottosequenza comune piu` lunga

Def. Dati una sequenza di caratteri $x=x_1,x_2,\ldots,x_m$ ed un insieme di indici $\{k_1,k_2,\ldots,k_t\}$ tali che $1 \le k_1<k_2<\ldots<k_t \le m$, la sequenza formata dai caratteri di x in posizione $k_1,k_2,\ldots,k_t$ viene detta sottosequenza di x.
N.B. I caratteri della sottosequenza non devono essere necessariamente consecutivi in x.

Problema: Date due sequenze $x=x_1,x_2,\ldots,x_m$ e $y=y_1,\ldots,y_n$, vogliamo trovare la sottosequenza piu` lunga comune ad entrambe le sequenze.

Esempio: x=BACBDAB e y=BDCABA,
BCAB e` una sottosequenza comune a x e y di lunghezza massima.
I caratteri della sequenza BCAB appaiono nelle posizioni 1, 3, 6, 7 in x e nelle posizioni 1, 3, 4, 5 in y.
BDAB e` un'altra una sottosequenza comune a x e y di lunghezza massima.
anche BABA e` un'altra sottosequenza comune a x e y di lunghezza massima.

Sottosequenza comune piu` lunga

Approccio brute force: Per ogni sottosequenza di x controlla se la sottosequenza compare in y.
Ci sono 2^m sottosequenze di x per cui l'algoritmo sarebbe esponenziale.

Perche ci sono 2^m sottosequenze di $x=x_1,x_2,\ldots,x_m$?

Risposta: ogni sottosequenza corrisponde ad una sequenza di m bit dove il k-esimo bit e` 1 se x_k fa parte della sottosequenza e 0 se x_k non fa parte della sottosequenza.

Sottosequenza comune piu` lunga

Input: $x=x_1,x_2,\ldots,x_m$ e $y=y_1,\ldots,y_n$

Sia OPT(i,j) la lunghezza della sottosequenza piu` lunga comune a $x_1,\ldots,x_i$ e $y_1,\ldots,y_j$.

Per calcolare OPT(i,j) consideriamo i 3 seguenti casi:

- Se $x_i = y_j$ allora la sottosequenza comune piu` lunga termina con x_i
  - In questo caso la soluzione ottima e` formata dalla sottosequenza piu` lunga comune a $x_1,\ldots,x_{i-1}$ e $y_1,\ldots,y_{j-1}$ seguita dal carattere $x_i = y_j$
- Se $x_i \neq y_j$ e la sottosequenza comune piu` lunga termina con un simbolo diverso da $x_i$ allora la soluzione ottima e` data dalla soluzione ottima per $x_1,\ldots,x_{i-1}$ e $y_1,\ldots,y_j$
- Se $x_i \neq y_j$ e la sottosequenza comune piu` lunga termina con un simbolo diverso da $y_j$ allora la soluzione ottima e` data dalla soluzione ottima per $x_1,\ldots,x_i$ e $y_1,\ldots,y_{j-1}$

Sottosequenza comune piu` lunga

Quindi si ha che per i>0 e j>0

$$OPT(i,j) = \begin{cases} OPT(i,j)=0 & \text{se } i=0 \text{ o } j=0 \\ OPT(i-1,j-1)+1 & \text{se } i>0,\ j>0 \text{ e } x_i = y_j \\ \max\{OPT(i-1,j),OPT(i,j-1)\} & \text{altrimenti} \end{cases}$$

Sottosequenza comune piu` lunga: algoritmo

```
ComputaLunghezzaLCS(X,Y)
1. m← lunghezza di X
2. n ← lunghezza di Y
3. For i=1 to m
4.      M[i,0]← 0
5.  For j=0 to n
6.      M[0,j]← 0
7. For i=1 to m
8.     For j=0 to n
9.         If x_i=y
10.          Then M[i,j]← 1+M[i-1,j-1]
11.               b[i,j]="\"
12.        Else if  M[i-1,j]>M[i,j-1]
13.          Then M[i,j]← M[i-1,j]
14.               b[i,j]="↑"
15.        Else  M[i,j]← M[i,j-1]
16.               b[i,j]="← "
```

L'algoritmo oltre a computare i valori M[i,j]=OPT(i,j), memorizza nelle entrate b[i,j] della matrice b delle frecce in modo che successivamente la sottosequenza comune piu` lunga possa essere ricostruita agevolmente (si veda algoritmo nella slide successiva).

89

Sottosequenza comune piu` lunga: algoritmo

x=BACBDAB e y=BDCABA

		Ø	B	D	C	A	B	A
		0	**1**	**2**	**3**	**4**	**5**	**6**
Ø	**0**	0	0	0	0	0	0	0
B	**1**	0	1	1	1	1	1	1
A	2	0	1	1	1	2	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

Seguendo le frecce viene stampata BABA

90

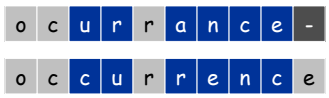
L'algortimo che stampa la sottosequenza comune piu` lunga

```
Stampa-LCS(b,X,i,j)
1. If i=0 or j=0
2.    Then return
3. If b[i,j]="\"
4.   Then Stampa-LCS(b,X,i-1,j-1)
5.           print(x_i )
6. Else if b[i,j]="↑"
7.    Then Stampa-LCS(b,X,i-1,j)
8. Else Stampa-LCS(b,X,i,j-1)
```

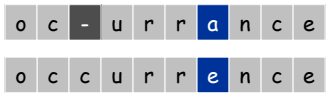
91

Allineamento di sequenze

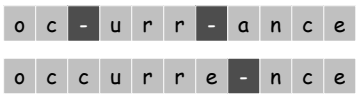
- Quanto sono simili le due stringhe seguenti ?
  - **ocurrance**
  - **occurrence**
  - Allineamo i caratteri
  - Ci sono diversi modi per fare questo allineamento
  - Qual e` migliore?



6 mismatch, 1 gap



1 mismatch, 1 gap



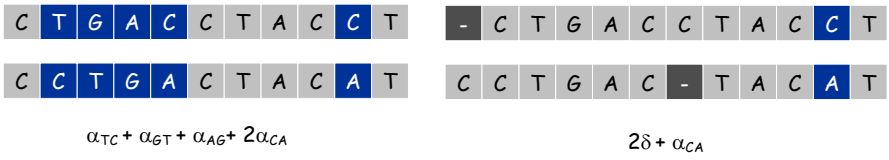
0 mismatch, 3 gap

92

Edit Distance

- Applicazioni.
  - Base per il comando Unix diff.
  - Riconoscimento del linguaggio.
  - Biologia computazionale.

- Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty δ;
  - Mismatch penalty α_{pq}. Si assume $\alpha_{pp}=0$



$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$

$2\delta + \alpha_{CA}$

93

Applicazione del problema dell'allineamento di sequenze

- I problemi su stringhe sorgono naturalmente in biologia: il genoma di un organismo e` suddiviso in molecole di DNA chiamate cromosomi, ciascuno dei quali serve come dispositivo di immagazzinamento chimico.

- Di fatto, si puo` pensare ad esso come ad un enorme nastro contenente una stringa sull'alfabeto {*A*,*C*,*G*,*T*}. La stringa di simboli codifica le istruzioni per costruire molecole di proteine: usando un meccanismo chimico per leggere porzioni di cromosomi, una cellula puo` costruire proteine che controllano il suo metabolismo.

Continua nella prossima slide

94

Applicazione del problema dell'allineamento di sequenze

- Perche` le somiglianze tra stringhe sono rilevanti in questo scenario?
- Le sequenze di simboli nel genoma di un organismo determinano le proprieta` dell'organismo.
- Esempio. Supponiamo di avere due ceppi di batteri X e Y che sono strettamente connessi dal punto di vista evolutivo.
- Supponiamo di aver determinato che una certa sottostringa nel DNA di X sia la codifica di una certa tossina.
- Se scopriamo una sottostringa molto simile nel DNA di Y, possiamo ipotizzare che questa porzione del DNA di Y codifichi un tipo di tossina molto simile a quella codificata nel DNA di X.
- Esperimenti possono quindi essere effettuati per convalidare questa ipotesi.
- Questo e` un tipico esempio di come la computazione venga usata in biologia computazionale per prendere decisioni circa gli esperimenti biologici.

Allineamento di sequenze

- Abbiamo bisogno di un modo per allineare i caratteri di due stringhe
  - Formiamo un insieme di coppie di caratteri, dove ciascuna coppia e` formata da un carattere della prima stringa e uno della seconda stringa.
- Def. insieme di coppie e` un matching se ogni elemento appartiene ad una sola coppia
- Def. Le coppie (x_i,y_j) e $(x_{i'},y_{j'})$ si incrociano se $i < i'$ ma $j > j'$.

	x_1	x_2	x_3	x_4
	-	T	A	C	C
	C	T	-	-	T
	y_1	y_2		y_3

Def. Un allineamento M e` un insieme di coppie (x_i,y_j) tali che
- M e` un matching
- M non contiene coppie che si incrociano

Esempio: CTACCG **vs.** TACATG

Soluzione:

M = (x_2,y_1), (x_3,y_2), (x_4,y_3), (x_5,y_4), (x_6,y_6).

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G

-	T	A	C	A	T	G
y_1	y_2	y_3	y_4	y_5	y_6	

Allineamento di sequenze

- Obiettivo: Date due stringhe $X = x_1 x_2 \ldots x_m$ e $Y = y_1 y_2 \ldots y_n$ trova l'allineamento di minimo costo.

$$\text{cost}(M) = \underbrace{\sum_{(x_i,y_j)\in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmatched}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gap}}$$

- Affermazione.
- Dato un allineamento M di due stringhe $X = x_1 x_2 \ldots x_m$ e $Y = y_1 y_2 \ldots y_n$, se in M non c'e` la coppia $(x_m,y_n)$ allora o $x_m$ non e` accoppiato in M o y_n non e` accoppiato in M.
- Dim. Supponiamo che x_m e y_n sono entrambi accoppiati ma non tra di loro. Supponiamo che x_m sia accoppiato con y_j e y_n sia accoppiato con x_i. In altre parole M contiene le coppie (x_m,y_j) e (x_i,y_n). Siccome i<m ma n>j allora si ha un incrocio e cio` contraddice il fatto che M e` allineamento.

Allineamento di sequenze: struttura del problema

- Def. OPT(i, j) = costo dell'allineamento ottimo OPT per le due stringhe $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.
  - Caso 1: in OPT x_i e y_j sono accoppiati.
    - OPT(i, j) = Costo dell'eventuale mismatch tra x_i e y_j + costo dell'allineamento ottimo di $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$
  - Caso 2a: in OPT x_i non e` accoppiato.
    - OPT(i, j) = Costo del gap x_i + costo dell'allineamento ottimo di $x_1 x_2 \ldots x_{i-1}$ e $y_1 y_2 \ldots y_j$
  - Case 2b: in OPT y_j non e` accoppiato.
    - OPT(i, j) = Costo del gap y_j + costo dell'allineamento ottimo di $x_1 x_2 \ldots x_i$ e $y_1 y_2 \ldots y_{j-1}$

$$OPT(i,j) = \begin{cases} j\delta & \text{se } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{altrimenti} \\ i\delta & \text{se } j = 0 \end{cases}$$

Allineamento di sequenze: algoritmo

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
   for i = 0 to m
      M[i, 0] = iδ
   for j = 0 to n
      M[0, j] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

- Analisi. Tempo e spazio $\Theta(mn)$.
- Parole inglesi: m, n $\leq$ 10.
- Applicazioni di biologia computazionale: m = n = 100,000.
- Quindi mxn=10 miliardi . OK per il tempo ma non per lo spazio (10GB)