Programmazione dinamica (III parte)

Progettazione di Algoritmi a.a. 2021-22

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 somma di denaro V, fornire una formula per calcolare il minimo numero di monete richieste per cambiare la somma di denaro V. Assumiamo $v_1=1$ in modo che il problema ammetta sempre una soluzione.

- Ad esempio: Banconota di 6 euro

Valori monete: 1,2,4

- Possiamo cambiare la banconota in 6 modi:
- {1,1,1,1,1,1}, {1,1,1,1,2}, {1,1,2,2},{1,1,4}, ,{2,2,2}, {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 è 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 per cambiare una banconota di valore v quando abbiamo a disposizione monete di valore $v_1,\ldots,v_i$

- 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 $v_i > v$, l'unico caso possibile e` quello in cui la soluzione non include monete di valore v_i → OPT(i,v) = 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_1,…,v_n,V) //v_1<…<v_n

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]}

45

Minimum Coin Change problem

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

46

Coin change problem

- Ad esempio: Banconota di 6 euro

  Valori monete: 1,2,4

- La soluzione che richiede meno monete è {2,4}
- {1,1,1,1,1,1} {1,1,1,1,2}, {1,1,2,2},{1,1,4}, {2,2,2}, {2,4}
- Vediamo la tabella che genererebbe l'algoritmo per questa istanza:

	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6
2	0	1	1	2	2	3	3
3	0	1	1	2	1	2	2

- Mostriamo il calcolo del valore di qualche cella:
- M[2,6]=min{M[1,6] , 1+M[2,4]}=min{6,3}=3
- M[3,3]=M[2,3] =2 in quanto v_3>3
- M[3,4]=min{M[2,4], 1+M[3,0]}=min{2,1}=1

47

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 somma di denaro di valore V, fornire una strategia per trovare in quanti modi possiamo usare le monete in C per cambiare V.
- Ad esempio: Banconota di 6 euro

Valori monete: 1,2,4

- Possiamo cambiare la banconota in 6 modi:
- {1,1,1,1,1,1} {1,1,1,1,2}, {1,1,2,2},{1,1,4}, {2,2,2}, {2,4}

48

Coin change problem

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

- Se i>1 e $v_i \leq v$ 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 i>1 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
  - $N(i,v)=N(i-1, v)$
- Caso base i=1 → N(i,v)=1 per ogni v
- Caso base v=0 → N(i,v)=1 (solo insieme vuoto) per ogni i>0

Coin change problem

- Ad esempio: Banconota di 6 euro

  Valori monete: 1,2,4

- Possiamo cambiare la banconota in 6 modi:
- {1,1,1,1,1,1} {1,1,1,1,2}, {1,1,2,2},{1,1,4}, {2,2,2}, {2,4}
- Vediamo la tabella che genererebbe l'algoritmo per questa istanza:

	0	1	2	3	4	5	6
1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	4
3	1	1	2	2	4	4	6

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. Il numero di sottosequenze è 2^m 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 più 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 $x_i = y_j$
  - 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

Se i=0 o j=0 allora banalmente la sottosequenza comune piu` lunga ha lunghezza 0 perche' almeno una delle due sequenze e` vuota.

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

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=1 to n
9.         If x_i=y_j
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).

55

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

56

L'algoritmo 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)
```