

Esercizi di Programmazione Dinamica

- **Il problema del resto**

La ditta che distribuisce le macchinette di caffè presso il Dipartimento di Mat. e Inf. ha deciso di rendere le sue macchinette in grado di restituire il resto nel caso venga inserito un importo superiore a quello necessario, utilizzando il numero minore possibile di monete. A tal fine é stato chiesto agli studenti di Algoritmi 2 del Corso di Studi in Informatica di progettare un algoritmo che risolva il problema di dare un resto C con il minor numero di monete.

Disponendo di un numero illimitato di monete di n diversi valori $v_1, v_2, v_3, \dots, v_n$ si progetti un algoritmo che risolva il problema in $O(nC)$.

suggerimento per la risoluzione

La struttura ricorsiva del problema del resto può essere così definita: *se devo dare un resto C e decido di dare una moneta di valore v_i , il mio problema si riduce a dare un resto $C - v_i$ utilizzando il resto delle mie monete.*

La definizione ricorsiva di una soluzione ottima potrebbe essere la seguente, dove con $r[c, i]$ si indica il numero di monete utilizzate per restituire un resto c utilizzando la moneta v_i .

$$r[c, i] = \begin{cases} \infty & \text{se } v_i > c \\ 1 & \text{se } v_i = c \\ \min_{j=1..n}(r[c - v_i, v_j]) + 1 & \text{se } v_i < c \end{cases}$$

La risoluzione del problema mediante l'utilizzo della programmazione dinamica prevede quindi la costruzione di una tabella di dimensione $C \times n$.

- **Il Piano di Studio**

Alcuni studenti del Corso di Studi in Informatica pensano di essere in grado di laurearsi facendo il minimo sforzo in termini studio.

In particolare sia $A = \{a_1, a_2, \dots, a_n\}$ é l'insieme degli n esami attivati nel Corso di Studi, dove per ogni $i = 1, 2, \dots, n$, l'esame a_i vale c_i crediti formativi, e si supponga di avere per ogni esame a_i un coefficiente d_i che rappresenta il grado di difficoltà dell'esame stesso. Gli studenti possono redigere il loro piano di studio individuale scegliendo nella lista degli esami attivati un insieme di esami tali che la somma dei crediti corrispondenti sia almeno P .

Progettare un algoritmo che redige un piano di studio regolare con difficoltà minima in $O(nP)$.

suggerimento per la risoluzione

Sia $d[p, i]$ la difficoltà minima di un piano di studi con almeno p crediti formativi redatto sull'insieme di corsi $\{a_i, a_2, \dots, a_n\}$. La sottostruttura di

una mia soluzione potrebbe essere descritta nel seguente modo: *dato un numero di crediti P e un insieme di corsi $\{a_i, a_{i+1}, \dots, a_n\}$, per trovare il valore di $d[p, i]$ si deve decidere se inserire il corso a_i , con difficoltà d_i e c.f.u. pari a c_i , e in seguito risolvere i sottoproblemi $d[p, i+1]$ e $d[p - c_i, i+1]$. Dove $d[p, i+1]$ é la soluzione nel caso non scelga il corso a_i , mentre $d[p - c_i, i+1]$ é la soluzione nel caso il mio piano di studi preveda il corso a_i .*

La definizione ricorsiva di una soluzione ottima potrebbe essere cosí definita

$$d[p, i] = \begin{cases} \infty & \text{se } i > n \\ d_i & \text{se } c_i \geq p \\ \min(d[p, i+1], d[p - c_i, i+1] + d_i) & \text{se } c_i < p \end{cases}$$

La risoluzione del problema mediante l'utilizzo della programmazione dinamica prevede quindi la costruzione di una tabella di dimensione $P \times n$.

- **Crea il Palindromo**

Una stringa é palindroma se non cambia leggendo da destra a sinistra e viceversa. Ad esempio *anna* e *osso* sono due palindromi.

Data una stringa $S = a_1a_2\dots a_n$ di n caratteri, calcolare il minimo numero di caratteri che occorre inserire per renderla un palindromo. La complessità dell'algoritmo deve essere $O(n^2)$.

suggerimento per la risoluzione

Sia $n[i, j]$ il numero di caratteri necessari per trasformare la stringa $S[i\dots j]$ in una stringa palindroma. Il caso base consiste in una stringa di 1 o meno caratteri, cioè il caso in cui $i \geq j$. Altrimenti se i caratteri $S[i]$ ed $S[j]$ sono uguali allora posso semplicemente risolvere il sottoproblema $n[i+1, j-1]$. Viceversa dovré inserire almeno un carattere per rendere la stringa palindroma e risolvere ricorsivamente i sottoproblemi $n[i+1, j]$ e $n[i, j+1]$.

La definizione ricorsiva di una soluzione ottima potrebbe essere cosí definita

$$n[i, j] = \begin{cases} 0 & \text{se } i \geq j \\ n[i+1, j-1] & \text{se } i < j \text{ and } S[i] = S[j] \\ \min(n[i+1, j], n[i, j+1]) + 1 & \text{se } i < j \text{ and } S[i] \neq S[j] \end{cases}$$

La risoluzione del problema mediante l'utilizzo della programmazione dinamica prevede quindi la costruzione di una tabella di dimensione $n \times n$.

- **Cerca il Palindromo**

Data una stringa $S = a_1a_2\dots a_n$ di n caratteri, Determinare la lunghezza del piú lungo palindromo sottostringa di S . La complessità dell'algoritmo deve essere $O(n^2)$.

suggerimento per la risoluzione

Sia $l[i, j]$ la lunghezza del piú lungo palindromo contenuto nella stringa $S[i\dots j]$. Il caso base consiste in una stringa di 0 caratteri, cioè il caso in cui $i > j$. Altrimenti se i caratteri $S[i]$ ed $S[j]$ sono uguali allora posso controllare se la stringa $S[i+1\dots j-1]$ é palindroma ed eventualmente aggiungo due caratteri. In tutti gli altri casi devo risolvere ricorsivamente i problemi $l[i+1, j]$ e $l[i, j-1]$

La definizione ricorsiva di una soluzione ottima potrebbe essere così definita

$$l[i, j] = \begin{cases} 0 & \text{se } i > j \\ l[i+1, j-1] + 2 & \text{se } i < j \text{ and } S[i] = S[j] \text{ and } l[i+1, j-1] \geq j-i \\ \max(l[i+1, j], l[i, j-1]) & \text{altrimenti} \end{cases}$$

La risoluzione del problema mediante l'utilizzo della programmazione dinamica prevede quindi la costruzione di una tabella di dimensione $n \times n$.

- **Massima somma di sottovettori**

Dato un vettore V di n interi, si vuole trovare un sottovettore (una sequenza non vuota di elementi consecutivi del vettore) la somma dei cui elementi sia massima. Progettare un algoritmo che risolve il problema in tempo $O(n)$.

suggerimento per la risoluzione

Una soluzione potrebbe consistere nel costruire una matrice s di dimensione $n \times n$ dove $s[i..j]$ indica il valore della somma degli elementi del sottovettore $V[i..j]$. Durante la costruzione é possibile mantenere un massimo temporaneo che ci permetterà di indentificare la casella contenente la somma più grande.

La definizione ricorsiva di una soluzione ottima potrebbe essere così definita

$$s[i, j] = \begin{cases} V[i] & \text{se } i = j \\ s[i, j-1] + V[j] & \text{se } i < j \end{cases}$$

- **Costruzione di ospedali**

Esiste una strada che collega n città. La strada può essere pensata come una retta e la posizione di ciascuna città é pertanto identificata da una singola coordinata. Non ci sono quindi due città nella medesima posizione e la distanza tra due città é il valore assoluto della differenza delle loro coordinate. Disponiamo di risorse sufficienti alla costruzione di k ospedali e, naturalmente, se un ospedale viene costruito in una data città, l'ospedale e la città si troveranno nella stessa posizione. Gli ospedali vanno posizionati in modo che la somma totale di tutte le distanze fra ciascuna città ed il più vicino ospedale risulti minima.

Progettare un algoritmo che prese le n coordinate x_1, x_2, \dots, x_n delle n città ed un intero $k < n$ rappresentante il numero di ospedali che bisogna realizzare, risolva il problema.

- **Somma di sottoinsiemi**

Progettare un algoritmo che, preso un insieme di interi positivi $A = \{a_1, a_2, \dots, a_n\}$ ed un intero k trovi, se esiste, un sottoinsieme di A la somma dei cui elementi sia k . La complessità dell'algoritmo deve essere $O(kn)$