Corso di laurea in Informatica
Progettazione d'algoritmi

Tecnica Programmazione Dinamica 2a

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO 1

Dato un intero n vogliamo contare le stringhe binarie lunghe n in cui non compaiono 2 zeri consecutivi.

Ad esempio:

per $n = 1$ sono 2: 0, 1
per $n = 3$ sono 5: 010, 011, 101, 110, 111.

L'algoritmo proposto deve avere complessità $O(n)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

- Utilizzeremo una tabella monodimensionale di dimensioni $n+1$ e definiamo il contenuto delle celle come segue:
  $T[i]$ = numero di stringhe binarie lunghe i dove non compaiono 2 zeri consecutivi.

- Una volta riempita la tabella la soluzione al nostro problema la troveremo nella locazione $T[n]$.

- Resta da definire la regola ricorsiva con cui calcolare i valori $T[i]$ nella tabella.

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 2 & \text{se } i = 1 \\ T[i-1] \ + \ T[i-2] & \text{altrimenti} \end{cases}$$

.

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 2 & \text{se } i = 1 \\ T[i-1] \ + \ T[i-2] & \text{altrimenti} \end{cases}$$

la ricorrenza viene fuori dal seguente ragionamento:

- il conteggio può essere visto come la somma delle stringhe da contare lunghe i che terminano con 1 e le stringhe da contare lunghe i che terminano con 0:

  1. le stringhe da contare lunghe i che terminano con 1 si ottengono accodando la cifra 1 a quelle da contare lunghe $i-1$. Queste stringhe sono $T[i-1]$
  2. le stringhe da contare lunghe i che terminano con 0 devono avere al penultimo posto la cifra 1, si ottengono dunque accodando 10 alle stringhe da contare lunghe $i-2$. Queste stringhe sono dunque $T[i-2]$

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 2 & \text{se } i = 1 \\ T[i-1] \ + \ T[i-2] & \text{altrimenti} \end{cases}$$

Implementazione:

```
def es(n):
    T = [0] * (n+1)
    T[0], T[1] = 1, 2
    for i in range(2, n + 1):
        T[i] = T[i-1] + T[i-2]
    return T[n]

>>> es(3)
>>> 5
```

Complessità $\Theta(n)$

ESERCIZIO 2

Dato un intero n vogliamo contare quanti modi diversi ci sono di sistemare n persone in un albergo che dispone di camere singole e camere doppie.

Ad esempio:

- per $n = 2$ sono 2: $\{[1],[2]\}$ $\{[1,2]\}$

- per $n = 4$ sono 10: $\{[1], [2], [3], [4]\}$, $\{[1,2], [3], [4]\}$, $\{[1,3],[2],[4]\}$, $\{[1,4], [2], [3]\}$, $\{[2,3], [1], [4]\}$, $\{[2,4], [1], [3]\}$, $\{[3,4], [1], [2]\}$, $\{[1,2], [3,4]\}$, $\{[1,3], [2,4]\}$, $\{[1,4], [2,3]\}$

L'algoritmo proposto deve avere complessità $\Theta(n)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

- Utilizzeremo una tabella monodimensionale di dimensioni $n+1$ e definiamo il contenuto delle celle come segue:
  $T[i]$ = numero di modi in cui è possibile sistemare i persone nell'albergo.

- Una volta riempita la tabella la soluzione al nostro problema la troveremo nella locazione $T[n]$.

- Resta da definire la regola ricorsiva con cui calcolare il valore $T[i]$ calcolati i valori $T[j]$ che precedono nella tabella.

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 1 & \text{se } i = 1 \\ T[i-1] \ + \ (i-1) \cdot T[i-2] & \text{altrimenti} \end{cases}$$

.

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 1 & \text{se } i = 1 \\ T[i-1] \ + \ (i-1) \cdot T[i-2] & \text{altrimenti} \end{cases}$$

la ricorrenza viene fuori dal seguente ragionamento:

- possiamo vedere le disposizioni delle i persone come la somma di due diverse tipologie: quelle in cui la persona i finisce in camera da sola e quelle in cui la perona i finisce in camera con una delle altre $i-1$ persone:.

  1. le differenti disposizioni in cui i finisce da solo sono $T[i-1]$, sono infatti tutti i possibili modi di disporre le altre $i-1$ persone.
  2. le differenti disposizioni in cui i finisce in camera con una delle altre $i-1$ persone sono $(i-1) \cdot T[i-2]$ infatti ci sono $i-1$ modi di scegliere il compagno di stanza e una volta scelto il compagno ci sono $T[i-2]$ modi di disporre tutte le altre $i-2$ persone nelle varie camere.

$$T[i] = \begin{cases} 1 & \text{se } i = 0 \\ 1 & \text{se } i = 1 \\ T[i-1] \ + \ (i-1) \cdot T[i-2] & \text{altrimenti} \end{cases}$$

Implementazione:

```
def es3(n):
    T = [0 for _ in range(n + 1)]
    T[0]=T[1]=1
    for i in range(2, n + 1) :
        T [i] = T[i-1] + (i-1)* T[i-2]
    return T[n]

>>> es3(4)
10
>>> es3(5)
26
```

La complessità è $\Theta(n)$

ESERCIZIO 3.

Dato un array A di n interi positivi vogliamo calcolare la somma massima per una sottosequenza di elementi di A non consecutivi.

Ad esempio:
per $A = [5, 4, 2, 10, 6, 8]$ la risposta è 23 infatti per A la sottosequenza di interi non consecutivi di valore massimo è quella di seguito evidenziata in rosso
5, 4, 2, **10**, 6, **8**

L'algoritmo proposto deve avere complessità $O(n)$

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

- Utilizzeremo una tabella monodimensionale di dimensioni $n-1$ e definiamo il contenuto delle celle come segue:

$T[i] =$ la somma massima per una sottosequenza di elementi non consecutivi in $A[:i+1]$

- una volta riempita la tabella la soluzione al problema sarà data da $T[n-1]$

Resta da definire la regola ricorsiva con cui calcolare i valori $T[i]$ della tabella

$$T[i] = \begin{cases} A[0] & \text{se } i = 0 \\ \max(A[0], A[1]) & \text{se } i = 1 \\ \max(T[i-1], T[i-2] + A[i]) & \text{altrimenti} \end{cases}$$

Resta da definire la regola ricorsiva con cui calcolare i valori $T[i]$ della tabella

$$T[i] = \begin{cases} A[0] & \text{se } i = 0 \\ \max(A[0], A[1]) & \text{se } i = 1 \\ \max(T[i-1], T[i-2] + A[i]) & \text{altrimenti} \end{cases}$$

La ricorrenza viene fuori dal seguente ragionamento:

1. **Il vettore contiene un solo elemento** allora meglio prenderlo perché positivo $T[0] = A[0]$.
2. **Il vettore contiene due soli elementi** allora meglio prendere il più grande dei due $T[1] = \max(A[0], A[1])$.
3. Alla soluzione di A[:i+1] appartiene l'ultimo elemento A[i]. In questo caso l'elemento A[i-1] non può appartenere alla soluzione e quindi il massimo deve essere $T[i-2] + A[i]$. Se al contrario A[i] non appartiene alla soluzione allora aper la soluzione deve aversi $T[i] = T[i-1]$.

   In generale possiamo quindi dire che $T[i] = \max(T[i-2] + A[i], T[i-1])$

Implementazione:

```
def es(A):
    n = len(A)
    T[0] = A[0]
    T[1] = max(A[0], A[1])
    for i in range(2, n):
        T[i] = max(T[i-1], T[i-2] + A[i])
    return T[n-1]

>>> A=[5, 4, 2, 10, 6, 8]
>>> es(A)
23
```

Complessità $\Theta(n)$

Nota che per calcolare un nuovo valore della tabella bastano i due valori precedenti quindi il programma si può migliorare passando dalla complessità di spazio $\Theta(n)$ alla complessità di spazio $\Theta(1)$

Corso di laurea in Informatica
Introduzione agli Algoritmi

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO

Dato un intero n vogliamo contare quante sono le stringhe binarie lunghe n in cui non compaiono 3 zeri consecutivi.

Ad esempio
- per $n = 1$ sono 2: 0, 1
- per $n = 4$ sono 13: 0100, 0010, 0011,0101, 0110, 1010, 1001, 1100, 0111, 1011, 1101, 1110, 1111

L'algoritmo proposto deve avere complessità $O(n)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

```
per n =  1 la risposta è  2
per n =  2 la risposta è  4
per n =  3 la risposta è  7
per n =  4 la risposta è  13
per n =  5 la risposta è  24
per n =  6 la risposta è  44
per n =  7 la risposta è  81
per n =  8 la risposta è  149
per n =  9 la risposta è  274
```

ESERCIZIO

Abbiamo un gruppo di n amici che debbono pernottare presso un hotel. L'hotel dispone di stanze da 1, 2 e 3 posti letto. Progettare un algoritmo che dato l'intero n calcola quante diverse allocazioni delle n persone sono possibili.

- Ad esempio per $n = 4$ la risposta deve essere 14. Ecco di seguito le 14 possibilità:

{[1], [2], [3], [4]}	{[1], [2,3,4]}	{[1], [2,3], [4]}	
{[1], [2,4], [3]}	{[1], [3,4], [2]}	{[1,2], [3,4]}	{[1,2], [3], [4]}
{[1,3], [2,4]}	{[1,3], [2], [4]}	{[1,4], [2,3]}	{[1,4], [2], [3]}
{[1,2,3], [4]}	{[1,2,4], [3]}	{[1,3,4], [2]}	

L'algoritmo proposto deve avere complessità $O(n)$

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

```
per n =  1 la risposta è  1
per n =  2 la risposta è  2
per n =  3 la risposta è  5
per n =  4 la risposta è  14
per n =  5 la risposta è  46
per n =  6 la risposta è  166
per n =  7 la risposta è  652
per n =  8 la risposta è  2780
per n =  9 la risposta è  12644
```

ESERCIZIO.

Dato un array A di $n \geq 2$ interi positivi vogliamo calcolare la somma massima per una sottosequenza di elementi di A con elementi che distano almeno 2 in A.

Ad esempio:
per $A = [1, 10, 3, 2, 4, 1, 7]$ la risposta è 17 infatti per A la sottosequenza di interi a distanza almeno 2 di valore massimo è quella di seguito evidenziata in rosso 1, **10**, 3, 2, 4, 1, **7**.

L'algoritmo proposto deve avere complessità $O(n)$

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.