Corso di laurea in Informatica
Progettazione d'algoritmi

Soluzioni secondo esonero 2024

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO.

Dati due interi non negativi n e k vogliamo sapere in quanti modi è possibile partizionare l'insieme dei numeri da 1 a n in k sottoinsiemi non vuoti.
Ad esempio:

- per $n = 1$ e $k = 2$ la risposta è 0
- per $n = 2$ e $k = 2$ la risposta è 1. L'unica bipartizione possibile è $(\{1\}, \{2\})$
- per $n = 3$ e $k = 2$ e la risposta è 3. Le 3 bipartizioni possibili sono $(\{1, 2\}, \{3\})$, $(\{1\}, \{2, 3\})$ e $(\{1, 3\}, \{2\})$.

Progettare un algoritmo che risolve il problema in tempo $\Theta(n \cdot k)$. **Motivare bene la correttezza e la complessità dell'algoritmo proposto.**

```
per n=10 e k= 1 la risposta è  1
per n=10 e k= 2 la risposta è  511
per n=10 e k= 3 la risposta è  9330
per n=10 e k= 4 la risposta è  34105
per n=10 e k= 5 la risposta è  42525
per n=10 e k= 6 la risposta è  22827
per n=10 e k= 7 la risposta è  5880
per n=10 e k= 8 la risposta è  750
per n=10 e k= 9 la risposta è  45
per n=10 e k= 10 la risposta è  1
```

Uso una tabella bidimensionale $(n+1) \times (k+1)$ dove:
$T[i][j]$ = il numero di modi di partizionare i elementi in j sottoinsiemi non vuoti.
La soluzione al problema sarà il valore $T[n][k]$.

resta ora da definire la formula ricorsiva che permette di calcolare la cella $T[i][j]$ a partire dalle celle precedenti già calcolate:

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

La ricorrenza viene fuori dal seguente ragionamento:

- $T[0][0] = 1$ mentre non c'è modo di partizionare $i > 0$ elementi in 0 parti o 0 elementi in $j > 0$ parti e quindi in questi casi si ha $T[i][0] = T[0][j] = 0$.
- Se $i, j > 0$ posso contare i modi di partizionare considerando cosa accade all'elemento i-esimo:
  - Se scelgo di tenere l'elemento i-esimo da solo allora avrò T[i-1][j-1] modi di farlo
  - Se scelgo di inserire l'i-esimo elemento con almeno un altro elemento allora avrò $jT[i-1][j]$ modi di farlo.

  Posso dunque dire che in questo caso si ha $T[i][j] = T[i-1][j-1] + j \cdot T[i-1][j]$

Implementazione

Complessità $\Theta(n \cdot k)$

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

ESERCIZIO.

Progettare un algoritmo che dato un intero $n \geq 3$ stampa tutte le stringhe ternarie in cui non compaiono 3 elementi adiacenti la cui somma sia un numero pari. Ad esempio:

- per $n = 4$ vanno stampate (non necessariamente nello stesso ordine) le seguenti 21 stringhe:

  0010, 0012, 0100, 0102, 0120, 0122, 0210, 0212, 1001, 1021, 1111

  1201, 1221, 2010, 2012, 2100, 2102, 2120, 2122, 2210, 2212.

L'algoritmo deve avere complessità $O(n \cdot S(n))$ dove $S(n)$ è il numero di stringhe da stampare.
Motivare bene la correttezza e la complessità dell'algoritmo proposto.

Algoritmo

- Utilizziamo un algoritmo di backtracking con l'aggiunta di funzioni di taglio.
- per ogni posizione i consideriamo se inserire o meno l'elemento $x \in \{'0', '1', '2'\}$ nella soluzione sol che andiamo costruendo:
  - $i < 2$ allora l'elemento x può essere aggiunto senza problemi
  - $i \geq 2$ in questo caso consideriamo i tre elementi $x, sol[-1]$ e $sol[-2]$ e l'elemento può essere aggiunto solo se la somma dei tre elementi non produce un numero pari
- grazie alla funzione di taglio descritta la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.

Implementazione:

```
def es(n, sol=[]):
    if len(sol) == n:
        print(''.join([str(x) for x in sol]))
        return
    for k in [0,1,2]:
        if len(sol)<2 or (sol[-1] + sol[-2] + k) %2 == 1:
            sol.append(k)
            es(n, sol)
            sol.pop()
```

```
>>> es(4)
0010
0012
0100
0102
0120
0122
0210
0212
1001
1021
1111
1201
1221
2010
2012
2100
2102
2120
2122
2210
2212
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione di es un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $es(n)$ richiederà tempo

$$O(S(n) \cdot h \cdot f(n) + S(n) \cdot g(n))$$

  dove:

  - $h = n$ è l'altezza dell'albero.
  - $f(n) = O(1)$ è il lavoro di un nodo interno.
  - $g(n) = O(n)$ è il lavoro di una foglia

- Quindi il costo di $es(n)$ è $O(nS(n))$.