Algoritmi e Strutture Dati 2023-24
(M. Benerecetti)

Contents

1 Lezione 02 - 15/09/2023 **2**

1.1 Algoritmo di Conteggio 2

1.1.1 Algoritmo di conteggio v2 3

1.1.2 Algoritmo di conteggio v3 4

1.2 Algoritmo della massima sottosequenza contigua 4

1.2.1 Algoritmo v2 5

1 Lezione 02 - 15/09/2023

1.1 Algoritmo di Conteggio

Descrivere un algoritmo che accetta come input un intero $N \geq 1$ e produce in output il numero di coppie ordinate $i, j \in \mathbb{N} \quad (i, j) : 1 \leq i \leq j \leq \mathbb{N}$
Esempio:

- Input:N=4
- Output:10 {(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,3),(3,4),(4,4)}

```
1 Conta(N):
2   ris = 0; //Assegnamento costante (1 operazione elementare)
3   for i=1 to N do //Assegnamento/Incremento + confronto (2 op. elementari)
4     for j=1 to N do //idem for di sopra
5       if i<=j then //2 letture + confronto (3 op. elementari)
6         ris = ris+1 //lettura+scrittura+assegnamento (3 op. elementari)
7   return ris //1 op. elementare
```

Ogni riga ha un costo che corrisponde alle operazioni elementari effettuate, mediamente ogni op. elementari ha un costo di 1 unità di tempo, prendiamo come esempio il for al primo giro: Assegnamento + Confronto (2 op. elementari), invece i successivi giri: Incremento+confronto (2 op. elementari).

Ognuna di queste operazioni (righe) vengono eseguite più di una volta, quindi il costo sarà maggiore, andiamo ad esprimerlo:

- 2) Costo = 1 (fuori dal ciclo)
- 3) La testa viene eseguita $n+1$ volte poiché abbiamo anche l'ultima operazione per uscire dal ciclo, quindi Costo = $2 * (n+1) = 2 * \sum_{i=1}^{N+1} 1$
- 4) Questo for verrà ripetuto N volte poiché il corpo del for viene eseguito N volte, quindi il suo costo sarà:

$$\underbrace{2}_{\text{costo dell'operazione}} * \underbrace{\sum_{i=1}^{N}}_{\text{for esterno}} \underbrace{\sum_{j=1}^{N+1}}_{\text{for interno}} 1$$

- 5) L'if stando in entrambi i for avrà un costo di: $3 * \sum_{i=1}^{N} \sum_{j=1}^{N+1} 1$
- 6) Questa operazione non ha un numero fisso di volte di esecuzione. Pertanto e necessario stabilirne un algoritmo per decretarne il numero. Pensandoci il numero di volte che questa operazione esegue dipende da N e dall' i fissate in precedenza. Calcolando, anche banalmente a mano, quante operazioni vengono eseguite ci troveremo con:

$$N - i + 1 \text{volte che l'operazione viene eseguita.}$$

- 7) Costo = 1 (fuori dal ciclo)

Dopo che viene effettuata l'analisi, possiamo andare a sommare tutti i risultati che abbiamo ottenuto in termini di unita (correggere accento) di tempo.
La funzione $T(n)$ e(correggere) la funzione che ci tiene traccia della complessita dell'algoritmo.

Andiamo semplicemente a sommare i nostri risultati di ogni riga:

$$T(n) = 1 + 2 * (N) + 2 * (N^2 + N) + 3 * \frac{N(N+1)}{2}$$

Questo risultato e ottenuto semplificando le nostre sommatorie:

- 3) $2 * \sum_{i=1}^{N+1} 1 = 2 * (N+1)$
- 4) $2 * \sum_{i=1}^{N} \sum_{i=1}^{N+1} 1 = 2 * \sum_{i=1}^{N} N + 1 = 2 * (N^2 + N)$
- 5) $3 * \sum_{i=1}^{N} \sum_{i=1}^{N} 1 = 3 * \sum_{i=1}^{N} N = 3N^2$
- 6) $\sum_{i=1}^{N} (N-i+1) = N-(k-1)$ cioe ad ogni ciclo il numero delle volte che viene eseguita questa operazione diminuisce costantemente di 1 (cioe dipendente dal salire di i).

$$T(n) = \frac{13}{2}N^2 = \frac{9}{2}N + 4$$

Come vediamo questa funzione e quadratica, quindi cresce esponenzialmente nel tempo, molto pesante e lenta come funzione.

1.1.1 Algoritmo di conteggio v2

Dopo aver ottenuto i risultati dell'analisi sopra, possiamo dire che e sicuramente possibile semplificare il nostro codice in modo tale da far eseguire meno operazioni al nostro processore e quindi utilizzare meno tempo.

```
Conta(N):
  ris = 0;
  for i=1 to N do
    ris = ris + (N-i+1)
  return ris
```

Cosi facendo abbiamo semplicemente detto al nostro codice che deve sommare soltanto gli elementi che nel momento in cui i e fissato, sono $\leq$ di se stesso.
Cosi facendo si dovrebbero eliminare molte operazion inutili, analizziamo.

- 2)sempre 1 operazione
- 3) $2 * \sum_{i=1}^{N+1} 1$
- 4) $\sum_{i=1}^{N} 7 = 7 \cdot N$

Come possiamo osservare abbiamo eliminato il secondo for, dunque abbiamo eliminato la quadraticita, ora l'operazionea riga 4 viene eseguita solamente N volte, e il numero di operazioni semplici che esegue e fissato.

$$T(n) = 1 + 2(N+1) + 7N + 1 = 9N + 4$$

1.1.2 Algoritmo di conteggio v3

Da come possiamo notare e possibile di nuovo semplificare l'ultima sommatoria della riga 4 dello scorso algoritmo.

Da

$$\sum_{i=1}^{N} N - i + 1 \rightarrow \sum_{i=1}^{N} i$$

Il risultato della sommatoria e lo stesso, se andiamo a semplificarlo.

$$\sum_{i=1}^{N} N - i + 1 = \sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

```
Conta(N):
  ris = 0
  ris = ris+(N-i+1)
  return ris
```

In questo modo abbiamo eliminato qualsiasi ciclo e quindi il risultato sara un numero fisso di operazioni.

- 1) 1 operazione
- 2) 5 operazioni elementari

$$T(n) = 5 + 1 = 6$$

In questo caso la funzione tempo per eseguire queste operazioni e costante, non dipendente da nessun N, dunque e la migliore soluzione possibile per questo algoritmo.

1.2 Algoritmo della massima sottosequenza contigua

Preso un array di n elementi, vorremmo provare a trovare la sottosequenza la cui somma di tutti i valori e massima.

Come fare? La soluzione piu naive possibile e effettuare 3 cicli for innestati:

```
int Max_seq_sum_1(int N, array a[])
  maxsum = 0
    for i=1 to N
      for j=i to N
        sum = 0
        for k=i to j
```

```
        sum = sum + a[k]
      maxsum = max(maxsum,sum)
  return maxsum
```

Come possiamo notare l'avere 3 for innestati ci fa avere una complessita computazionale di N^3, comunemente rappresentato dalla formula della notazione asintotica $O(N^3)$.

1.2.1 Algoritmo v2

Come e facile notare, nel terzo for innestato c'e una ripetizione abbastanza inutile di operazioni che effettuiamo per andarci a sommare i valori delle sottosequenze. In particolare andiamo a ripetere scorrere piu volte lo stesso numero di celle, solamente per trovare la sottosequenza poco piu grande (a volte anche di una cella).

Gli stessi valori delle sottosequenze possono essere semplicemente trovati scorrendo avanti l'indice e sommando alla sequenza precedente il valore successivo. Analiticamente ci troveremmo che:

$$\sum_{k=i}^{j+1} a_k = a_{j+1} + \sum_{k=i}^{j} a_k$$

Il valore sum rimarra inalterato, ma verra solamente aggiornato del valore successivo. Il codice si presentera in questo modo:

```
int Max_seq_sum_1(int N, array a[])
  maxsum = 0
    for i=1 to N
      sum = 0
      for j=i to N
        sum = sum + a[j]
        maxsum = max(maxsum,sum)
  return maxsum
```