Esercizi sull'analisi di complessità degli algoritmi

(Fondamenti di Informatica 2 – Walter Didimo)

(Soluzioni)

Esercizio 1 Considera il seguente metodo, che restituisce il massimo valore in un array di interi. Effettuare un'analisi di complessità asintotica del caso peggiore, ed esprimere la complessità con notazione O, rispetto alla dimensione dell'input.

```
public static int massimo (int[] a){
      int max = a[0];
      for (int i=1; i<a.length; i++)
            if (a[i]>max)
                  max=a[i];
      return max;
}
```

Soluzione

Il metodo esegue una visita di tutti gli elementi dell'array a, una ed una sola volta. Ogni volta che un nuovo elemento viene visitato, il metodo effettua un confronto tra il valore di quell'elemento ed il valore della variabile max. Poiché ogni confronto richiede un tempo costante (è un'operazione elementare), e poiché anche le operazioni di assegnazione e l'istruzione return richiedono un tempo costante, segue che la complessità del metodo è O(n), dove n indica il numero di elementi in a, cioè la dimensione dell'input.

Esercizio 2 Per ciascuno dei seguenti problemi, descrivi un algoritmo e la sua complessità asintotica con notazione O:

1) Data una matrice rettangolare di interi di dimensione m x n, visualizzare le somme degli elementi di ogni riga.
2) Data una matrice quadrata di dimensione n, visualizzare la somma degli elementi sulla diagonale principale e quella degli elementi sulla diagonale secondaria.
3) Dato un array di n numeri reali, restituire true se nell'array ci sono due numeri uguali e false in caso contrario.
4) Dato un array a di n numeri interi, con valori compresi tra 0 e 99, restituire true se nell'array ci sono due elementi uguali e false in caso contrario.

Soluzione

1) Un metodo che visualizza le somme degli elementi di ogni riga, deve scandire tutte le righe della matrice e, in ogni riga, tutti i suoi elemento. In altri termini deve scandire tutti gli elementi della matrice, una ed una sola volta. L'operazione dominante del metodo è l'operazione di somma, che deve essere eseguita ogni volta che si visita un nuovo elemento. La complessità del metodo è dunque O(m x n).

2) In questo caso un metodo che risolve il problema non ha bisogno di visitare tutti gli elementi della matrice, ma soltanto gli elementi sulla diagonale principale, cioè n elementi e poi quelli della diagonale secondaria, ossia altri n elementi. Per ciascuno di questi elementi deve effettuare un'operazione di somma, quindi la sua complessità è O(n) + O(n) = O(n).

3) Un metodo che risolve il problema dovrebbe confrontare, nel caso peggiore, ciascun elemento con tutti i suoi successori. Quindi il numero totale di confronti

nel caso peggiore è pari a: (n-1) + (n-2) + … + 1 = n(n-1)/2. Dunque la complessità del metodo è $O(n^2)$.

4) In questo caso, conoscendo il valore massimo che può assumere un elemento dell'array a, si può utilizzare una strategia più efficiente. Precisamente, si definisce un'array di boolean di dimensione 100 e si inizializzano i suoi elementi a false. Chiamiamo b tale array. Poi si visitano tutti gli elementi dell'array a, uno alla volta. Quando si visita il generico elemento a[i] si controlla il valore di b[a[i]]. Se tale valore è false, allora vuol dire che è la prima volta che si incontra il numero a[i], e si pone semplicemente b[a[i]]=true, così da ricordarsi di aver incontrato tale numero nel resto della visita. In questo modo, con una scansione lineare di a è possibile decidere se ci sono due numeri uguali oppure no. Osserva inoltre che l'inizializzazione dell'array b richiede 100 assegnazioni, cioè ha costo costante rispetto alla dimensione dell'input. Il metodo ha dunque complessità O(n).

Esercizio 3 Analizza la complessità asintotica dei seguenti metodi, in funzione della dimensione dell'input.

```
public static void met1 (int[] a){
      for (int i=0; i<a.length; i++){
            int somma = 0;
            for (int j=i; j<a.length; j++)
                  somma += a[j];
            System.out.println (somma);
      }
}

public static void met2 (char c, String s){
      int f = 0;
      for (int i=0; i<s.length(); i++)
            if (s.charAt(i) == c)
                  f++;
      return f;
}

public static void met3 (int[] a){
      for (int i=0; i<a.length-2; i++){
            int somma = 0;
            for (int j=i; j<i+2; j++)
                  somma += a[j];
            System.out.println (somma);
      }
}
```

Soluzione

- L'operazione dominante del metodo è l'operazione di somma nel ciclo più interno. Tale operazione costa O(1) ogni volta che viene eseguita, e viene eseguita un numero di volte pari a: (n-1) + (n-2) + … + 1 = n(n-1)/2. Quindi la complessità del metodo è $O(n^2)$.

- Misuriamo la dimensione dell'input attraverso il numero di caratteri della stringa s; chiamiamo n tale numero (infatti il primo dato di input è sempre un carattere quindi ha una dimensione costante). Indichiamo inoltre con C(n) la complessità asintotica del metodo s.charAt(i) nel caso peggiore. Il metodo effettua una visita di tutti i caratteri di s (una sola volta) ed ogni volta esegue un confronto chiamando il metodo s.charAt(i). Ne segue che la complessità di met2 è O(n) x C(n). Se C(n) = O(1) allora la complessità di met2 è O(n); se invece C(n)=O(n), allora la complessità di met2 è $O(n^2)$.

- L’operazione dominante è la somma nel ciclo più interno. Tale operazione ha costo O(1) e viene eseguita un numero di volte pari a: 2*(n-2), dove n è la dimensione dell’array a. Ne segue che la complessità di met3 è lineare nella dimensione dell’array di input.