Elementi di Algoritmi e Strutture Dati
Testo della prova scritta del 25 settembre 2007
docente: Luciano Gualà

Cognome:..... Nome:..... Matr.:..... Corso di Laurea:.....

Esercizio 1 [8 punti] Sia k una costante intera e, per ogni $i \in \{1, \ldots, k\}$, sia $f_i(n) : \mathbb{N} \to \mathbb{R}^+$ una funzione. Inoltre sia $g(n) : \mathbb{N} \to \mathbb{R}^+$ un'altra funzione. Dimostrare o confutare la seguente relazione

$$\sum_{i=1}^{k} f_i(n) = O(g(n))$$

quando

(a) $f_{i-1}(n) = o(f_i(n))$ per ogni $i = 2, \ldots, k$ e $f_1(n) = o(g(n))$.

(b) $f_i(n) = O(g(n))$ per ogni $i = 1, \ldots, k$.

Soluzione esercizio 1

(b) Vera. Basta far vedere che esistono due costanti c e n_0 tale che

$$\sum_{i=1}^{k} f_i(n) \leq cg(n) \ \forall n \geq n_0.$$

Per ipotesi abbiamo che, per ogni $i = 1, \ldots, k$, esistono due costanti c_i e $n_0^{(i)}$ tale che $f_i(n) \leq c_i g(n)$ per ogni $n \geq n_0^{(i)}$. Allora, scegliendo $n_0 = \max_{i=1,\ldots,k} n_0^{(i)}$, e sommando su i, otteniamo:

$$\sum_{i=1}^{k} f_i(n) \leq (c_1 + c_2 + \cdots + c_k)g(n) \ \forall n \geq n_0.$$

Da cui segue la tesi scegliendo $c = \sum_{i=1}^{k} c_i$.

(a) Falsa. Un controesempio per $k = 2$ è il seguente: $f_1(n) = n$, $f_2(n) = n^3$, e $g(n) = n^2$.

Esercizio 2 [8 punti] Realizzare un algoritmo che, preso in input un vettore ordinato A di n interi distinti, e un valore x, restituisca `true` se esistono due elementi distinti in A la cui somma è x, o `false` altrimenti. L'algoritmo deve avere complessità temporale $o(n^2)$. *Attenzione*: l'esercizio sarà valutato solo se corredato da adeguata descrizione del funzionamento dell'algoritmo, in base ai seguenti parametri: correttezza, efficienza e analisi di complessità.

Soluzione esercizio 2 Considereremo il vettore A ordinato in ordine crescente. Un semplice algoritmo è il seguente. Per ogni $i = 1, \ldots, n$, consideriamo l'elemento $A[i]$ e cerchiamo, se esiste, un indice j tale che $A[i] + A[j] = x$. Se troviamo tale indice ritorniamo in output `true`, se viceversa la ricerca fallisce per tutti gli indici i, diamo in output `false`. Fissato un indice i, la ricerca dell'indice j può essere effettuata in tempo $O(\log n)$ tramite una ricerca binaria (questo perché il vettore è ordinato per ipotesi). Tale algoritmo quindi ha complessità $O(n \log n) = o(n^2)$, perché il numero di ricerca binarie nel caso peggiore è $O(n)$.

Lasciamo lo pseudo codice del precedente algoritmo come esercizio, mentre di seguito illustriamo un algoritmo alternativo di complessità lineare (e quindi più efficiente) per il problema in questione. Lo pseudo codice di tale algoritmo è dato di seguito.

```
CercaCoppia(A, x)
n =size[A]
i = 1
j = n
while (i < j) do
    if (A[i] + A[j] = x) then
        return(true)
    end
    else
        if (A[i] + A[j] < x) then
            i = i + 1
        end
        else
            j = j − 1
        end
    end
end
return(false)
```

E' semplice vedere che l'algoritmo ha complessità $O(n)$. La sua correttezza poggia sulla seguente idea. Se ad un certo punto consideriamo una coppia di indici i, j e risulta $A[i] + A[j] > x$, allora la stessa cosa varrà per ogni altra coppia i', j con $i' > i$, tutte coppie che quindi possiamo non controllare (perché la somma dei relativi elementi non darà mai x). Una considerazione del tutto analoga vale quando risulta $A[i] + A[j] < x$, che implica che $A[i] + A[j'] < x$ per ogni $j' < j$. Una prova di questa proprietà è fornita dal seguente lemma, che dimostra formalmente la correttezza dell'algoritmo.

Lemma 1 *Al generico passo k, sia i_k, j_k la coppia di indici correntemente analizzata*

dall'algoritmo. Se $A[i_k] + A[j_k] \neq x$*, allora non esistono due indici* i, j *tale che* $i \in \{1, \ldots, i_k\}$*,* $j \in \{j_k, \ldots, n\}$ *con* $A[i] + A[j] = x$*.*

Proof. La dimostrazione è per induzione su k. Il caso base ($k = 1$) è banalmente verificato. Consideriamo, quindi, un generico passo k con indici correnti i_k e j_k e assumiamo che la tesi sia vera per tutti i passi precedenti. Assumiamo inoltre che $A[i_k] + A[j_k] \neq x$. Dobbiamo dimostrare che $A[i] + A[j] \neq x$ per ogni i, j con $i \in \{1, \ldots, i_k\}$, $j \in \{j_k, \ldots, n\}$. A questo punto, per applicare l'ipotesi induttiva, ci chiediamo quanto valevano i due indici al passo $k - 1$. Siano i_{k-1} e j_{k-1} i valori dei due indici al passo $k - 1$. Possiamo avere due casi:

Caso 1: $i_{k-1} = i_k$**.** Allora $j_k = j_{k-1} - 1$. Per iposesi induttiva $A[i] + A[j] \neq x$ per ogni i, j con $i \in \{1, \ldots, i_k\}$, $j \in \{j_k + 1, \ldots, n\}$. Resta da far vedere che non esiste nessun indice $\bar{i} \in \{1, \ldots, i_k\}$ tale che $A[\bar{i}] + A[j_k] = x$. Supponiamo per assurdo che tale indice $\bar{i}$ esista e chiediamoci quanto valevano gli indici quando è stata analizzata la prima coppia con il primo indice uguale a $\bar{i}$. Sia $\bar{i}, j$ tale coppia di indici. Deve essere $j > j_k$ e quindi $A[\bar{i}] + A[j] > x$ (perché A è ordinato e $A[\bar{i}] + A[j_k] = x$). Inoltre è vero che $A[\bar{i}] + A[j'] > x$ per ogni $j' \in \{j_k + 1, \ldots, j\}$. Questo è un assurdo perché l'algoritmo avrebbe trovato la coppia $(\bar{i}, j_k)$ decrementando sempre il secondo indice.

Caso 2: $i_{k-1} = i_k - 1$**.** Allora $j_{k-1} = j_k$. Se si applica l'ipotesi induttiva (in maniera del tutto analoga all'analisi fatta nel caso 1), dobbiamo da far vedere solo che non esiste nessun indice $\bar{j} \in \{j_k, \ldots, n\}$ tale che $A[i_k] + A[\bar{j}] = x$. Supponiamo per assurdo che tale indice $\bar{j}$ esiste, allora possiamo considerare la prima coppia analizzata dall'algoritmo con il secondo indice uguale a $\bar{j}$. Sia $i, \bar{j}$ tale coppia. Deve risultare $i < i_k$ e $A[i] + A[\bar{j}] < x$, da cui segue l'assurdo poiché l'algoritmo da quel momento in poi avrebbe incrementato sempre il primo indice fino a trovare la coppia $i_k, \bar{j}$.

Esercizio 3 [8 punti] Si risolvano le seguenti relazioni di ricorrenza utilizzando il teorema fondamentale delle ricorrenze.

(a) $T(n) = 27\,T(n/3) + n^3$.

(b) $T(n) = 2\,T(n/2) + 3\,T(n/3) + \sqrt{n}\log n$.

(c) $T(n) = 25\,T(n/5) + n^{\frac{41}{20}}$.

Soluzione esercizio 3

(*a*) Si ha: $a = 27$, $b = 3$, e $f(n) = n^3$, da cui risulta che $n^{log_b a} = n^3 = \Theta(f(n))$. Dal secondo caso del teorema fondamentale delle ricorrenze abbiamo quindi che $T(n) = n^3 \log n$.

(*b*) Non è possibile in questo caso applicare il teorema fondamentale poiché la relazione non ha la forma richiesta dalle ipotesi del teorema (vedere libro di testo).

(*c*) Si ha: $a = 25$, $b = 5$, e $f(n) = n^{\frac{41}{20}} = n^{2+\frac{1}{20}}$, da cui $n^{log_b a} = n^2$. Da ciò deriva $f(n) = n^{2+\frac{1}{20}} = \Omega(n^{2+\epsilon})$, ad esempio per $\epsilon = 1/20$. Per poter applicare il terzo caso del teorema fondamentale, dobbiamo però verificare ancora $a \cdot f(n/b) \leq c \cdot f(n)$ per $c < 1$ e n sufficientemente grande. Ciò è banalmente verificato per $c = \frac{1}{5^{1/20}}$, poiché

$$25 \cdot \left(\frac{n}{5}\right)^{\frac{41}{20}} = 25 \cdot \frac{n^{\frac{41}{20}}}{5^{\frac{41}{20}}} = c \cdot f(n).$$

Esercizio 4 [8 punti] Si descriva in modo sintetito e preciso l'algoritmo `Fibonacci6` che, preso in input un intero n, calcola l'n-esimo numero di Fibonacci in tempo $O(\log n)$. Si analizzi in particolare la sua correttezza e si derivi la sua complessità temporale.

Soluzione esercizio 4 Consultare il libro di testo.