Introduzione agli Algoritmi
27 gennaio 2015
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)

Parte I

Le risposte non motivate non saranno prese in considerazione. Negli esercizi di progettazione, prima di passare allo pseudocodice descrivete l'idea algoritmica soggiacente. Per gli algoritmi progettati è necessario analizzare tempo di esecuzione e correttezza.

Esercizio 1

Si considerino le seguenti operazioni su un array di *n* valori interi arbitrari:

a. Trovare il valore massimo
b. Calcolare la media aritmetica
c. Calcolare la moda (valore che appare più frequentemente)

Per ciascuna operazione, rispondere alle seguenti domande:

- Occorre applicare un algoritmo di ordinamento per implementare l'operazione in modo efficiente?
- Se si, quale algoritmo di ordinamento conviene applicare?
- Si descriva un algoritmo che esegue il calcolo richiesto nel modo più efficiente possibile.

Soluzione

a. Per calcolare il massimo sono necessari e sufficienti n confronti, quindi non conviene ordinare l'array, cosa che richiede $\Omega(n \lg n)$ confronti.
Per trovare il massimo in $\Theta(n)$ si può procedere come segue:
conservato il primo elemento in una variabile MaxProvv, si confronta MaxProvv con gli elementi successivi. Se un elemento è maggiore di MaxProvv quest'ultimo ne prende il valore, altrimenti si prosegue con il confronto con l'elemento successivo.

b. Come nel caso del massimo la media aritmetica si calcola in $\Theta(n)$ e non conviene ordinare gli elementi.
L'algoritmo in $\Theta(n)$ è il seguente:
detta sum una variabile inizializzata a 0, si scorrono gli elementi dell'array sommando ogni valore in sum. La media si ottiene dividendo il valore di sum per il numero degli elementi.

c. Per calcolare la moda invece conviene ordinare l'array, per esempio con il mergesort o l'heapsort che hanno una complessità $\Theta(n \lg n)$.
Su elementi non ordinati infatti non c'è altro modo che quello di contare le occorrenze di ogni elemento confrontandolo con i rimanenti, processo che porta a un algoritmo di complessità quadratica.
Se invece l'array è ordinato le occorrenze di uno stesso elemento sono consecutive e quindi basta esaminare l'array mantenendo in una variabile jm il massimo numero di duplicati di un elemento trovati fino a quel momento e quell'elemento in una variabile Moda. Le variabili jm e Moda vengono aggiornati se il numero dei duplicati trovati successivamente supera il valore di jm. La complessità di quest'ultima parte è $O(n)$.
Quindi tutto l'algoritmo ha complessità $\Theta(n \lg n)$.

MODA(A)

```
prec.: A è una sequenza non vuota e  ordinata di numeri
postc: restituisce il valore più frequente
    i = 1
    jm = 1  % contiene la frequenza più alta  corrente
    Moda = A(1)
    while (i ≤ n)
        j = 0 % contiene il numero di confronti tra elementi uguali all' i-simo, per cui j+1 è il
    numero delle copie di A[i] presenti in A
        while ( i+j+1 ≤ n and A[i+j] == A[i+j+1]):
                j = j+1
    if (jm < j+1):
        jm = j+1;
        Moda = A[i+j]
    i = i+j+1
    return Moda
```

Esercizio 2

Si determini il tempo di esecuzione T(n) della seguente funzione:

```
analizzami(int n)
        c = 1
        m = n*n
        while m>1 do
                for j=1 to m do c++
                m=m-2
        if n>1 then analizzami(n/4)
```

Il ciclo while contiene un ciclo for che viene eseguito m volte la prima volta, m - 2 la seconda, …, fino a un valore ≤ 1. Quindi in totale il ciclo while viene eseguito un numero di volte pari alla somma dei numeri pari da 0 a m se m è pari, dei numeri dispari da 1 a m se m è dispari. Tali somme crescono con il quadrato di m quindi il ciclo while comporta l'esecuzione di $O(n^4)$ istruzioni di costo costante.
Quindi la relazione di ricorrenza è $T(n) = T(n/4) + O(n^4)$.
Per risolverla bisogna esplicitare le costanti e fissare il caso base:
$T(n) = c$ per $n \leq 1$,
$T(n) = T(n/4) + dn^4$, per costanti $c,d>0$
Ipotizziamo che $T(n) = O(n^4)$, quindi dobbiamo dimostrare che esistono un a e un n_0 tali che $T(n) \leq an^4$, per tutti gli $n \geq n_0$, dove a e n_0 sono costanti positive. Sia n>1.
Per ipotesi induttiva sia $T(n/4) \leq a(n/4)^4$ allora $T(n) \leq a(n/4)^4 + dn^4$ e si vuole determinare se esistono un a e un n_0 tali che $a(n/4)^4 + dn^4 \leq an^4$ per ogni $n \geq n_0$. $an^4/4^4 + dn^4 \leq an^4 \iff a/4^4 + d \leq a \iff a + 4^4 d \leq 4^4 a \iff 4^4/(4^4 - 1)d \leq a$.

Si può concludere che prendendo a=2d e per ogni $n_0 = 2$ la tesi è vera.

Introduzione agli Algoritmi
27 gennaio 2015
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)
Parte II

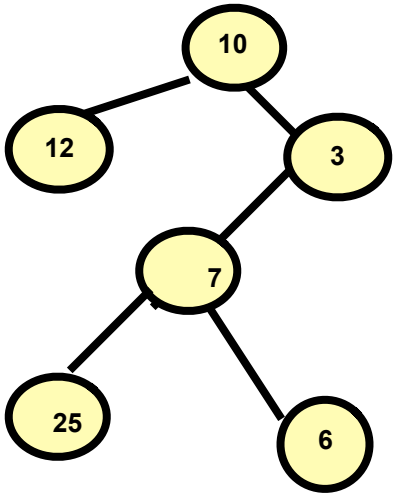
Esercizio 3

A. Si esprima l'altezza h di un max_Heap in termini del numero n di nodi. Si dimostri la verità della risposta.
B. Si esprima l'altezza h di un AVL (o rosso nero) in termini del numero n di nodi. Si dimostri la verità della risposta.

Vedere libro di testo o lucidi.

Esercizio 4

Dato un albero binario, definiamo distanza tra due nodi il numero di archi da attraversare per andare da un nodo all'altro, per esempio nell'albero sottostante i nodi a distanza 2 da 3 sono 25, 6 e 12, solo il nodo 10 è a distanza 3 dal nodo 6.



Si scriva un algoritmo per la stampa dei nodi a distanza k da un nodo x in un albero binario T. Si suggerisce di trattare separatamente e ricorsivamente il caso in cui i nodi a distanza k si cercano nel sottoalbero radicato nel nodo x. Successivamente questa procedura sarà applicata nella ricerca dei nodi a distanza k risalendo nell'albero. Si assuma che la rappresentazione in memoria scelta per i nodi comprenda, oltre al campo chiave, i campi puntatori ai figli e al padre.

Soluzione.

Per la ricerca di nodi a distanza k nel sotto albero radicato in x si usi la seguente semplice procedura, ottenuta modificando la visita in preorder di un albero binario, inserendo il parametro k e decrementandolo a ogni chiamata su un figlio. Quando k è diventato 0 si è trovato un nodo a distanza k e quindi lo si stampa.

```
NodiDistkSottoAlbero(x,k)
if x == NIL  then return
if k == 0  print x.key
if (k>0) then
      NodiDistkSottoAlbero(x.left,k-1)
      NodiDistkSottoAlbero(x.right,k-1)
```

Nel caso generale innanzi tutto si usa la precedente procedura per il calcolo di quelli a distanza k nel sotto albero radicato in x, poi si cercano i nodi a distanza k risalendo dal nodo x verso la radice, ogni volta chiamando la procedura NodiDistkSottoAlbero sul fratello del nodo su cui si sta risalendo, tenendo conto della distanza percorsa in salita. Contiamo i passi di risalita nella variabile s.

```
NodiDistk(T,x,k)
if T == NIL return 0
NodiDistkSottoAlbero(x,k)
% qui si stampano i nodi a distanza k nel sotto albero radicato in x
s = 1
z = x
y = x.p
while  (y≠NIL and s≤k) do
      if (s==k) then print(y.key)
% y è il padre di z e potrebbe essere a distanza  k, in tal caso non si deve procedere oltre
      else
      if (y.left ≠ NIL and z == y.left)  then NodiDistkSottoAlbero(y.right,k-s-1)
% z è figlio sinistro e k>s, la ricerca dei nodi a distanza k si fa nel sotto albero destro,
tenendo conto che la risalita di s passi + la discesa sul fratello contano s + 1 sulla distanza
       else if (y.right ≠ NIL  and z == y.right)  NodiDistkSottoAlbero(y.left,k-s-1)
       z=y
       y=y.p
       s = s+1
```

La correttezza della procedura si basa sulle seguenti osservazioni.
Se non si entra nel ciclo vuol dire che x è la radice e non si deve stampare altro.
Prima e durante l'esecuzione del ciclo y è il padre di z e s è la distanza di y da x.
Supponiamo corretta l'esecuzione del ciclo fino a questa nuova esecuzione. Se s è uguale a k si stampa la chiave di y, che è l'antenato di x a distanza k. Altrimenti, cioè se s<k si chiama la procedura di ricerca dei nodi a distanza k-s-1 nel sotto albero radicato nel figlio fratello di quello da cui si risale. Questo è corretto perché s tiene conto del numero di nodi attraversati nel cammino da x a y e questo valore va sottratto a k-1 per tener conto della "ridiscesa" al fratello. All'inizio del ciclo s contiene di nuovo la distanza tra x e y che è il padre di z.
Dal ciclo si esce o perché si è giunti alla radice o perché s supera k.
Nel primo caso, cioè se y==NIL e s≤k vuol dire che z è la radice e i controlli su z e l'eventuale sotto albero sono stati fatti e non c'è altro da fare.
Se y≠NIL ma s>k analogamente non ci sono altri nodi da stampare.
La complessità è O(n), raggiunta per esempio nel caso in cui x è la radice e k è pari all'altezza dell'albero.