Introduzione agli Algoritmi
Esame Scritto a canali unificati con spunti per la soluzione

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma

14 Febbraio 2024

Esercizio 1 (10 punti):

Dato un albero binario T con n nodi, memorizzato tramite record e puntatori, e dato tramite il puntatore p alla sua radice, si consideri il seguente algoritmo di visita in preordine:

```
def Visita_preordine (p):
  if (p != None):
      # qui istruzioni di costo costante di accesso al nodo
      Visita_preordine (p.left)
      Visita_preordine (p.right)
  return
```

Si scriva l'equazione di ricorrenza che ne descrive il costo computazionale e la si risolva in termini di notazione asintotica stretta.

Si dettaglino i ragionamenti ed i passaggi del calcolo, giustificando ogni affermazione.

Si vedano le dispense.

Esercizio 2 (10 punti):
Sia dato un array A di n elementi memorizzati in ordine qualunque, nel quale si vogliono determinare i k elementi più grandi (dove k è un valore compreso tra 1 ed n).

Si considerino i seguenti due approcci:

1. si ordini A e si prendano i primi k elementi;
2. si costruisca uno heap massimo in tempo lineare, e per k volte si esegua l'estrazione del massimo.

Per ciascun approccio, si calcoli il costo computazionale stretto (cioè in termini di Θ) come funzione di k e di n; poi si discuta per quali valori di k ciascun approccio risulti asintoticamente migliore dell'altro in termini di costo computazionale; infine, si scriva lo pseudocodice della funzione di costruzione dello heap massimo (Build_Heap).

I due approcci hanno il seguente costo computazionale:

1. $\Theta(n \log n)$ **per l'ordinamento di** A **e** $\Theta(k)$ **per prendere i primi** k **elementi, quindi** $\Theta(n \log n)$**, indipendentemente da** k**;**
2. $\Theta(n)$ **per costruire uno heap massimo, e** $O(\log n)$ **per l'estrazione del massimo per** k **volte, quindi** $\Theta(n + k \log n)$**.**

Osservando che il secondo approccio ha un costo $O(n \log n)$ **(perché al massimo** $k = \Theta(n)$**), segue che, tranne nel caso in cui** $k = \Theta(n)$**, in cui i due approcci hanno lo stesso costo, il secondo è sempre preferibile al primo.**

Lo pseudocodice della funzione Build_Heap **si trova sulle dispense.**

Esercizio 3 (10 punti):

Si consideri una lista puntata L contenente chiavi intere positive, memorizzata tramite record e puntatori, ed accessibile tramite il puntatore p alla sua testa.

Definiamo k come un valore strettamente minore della somma di tutte le chiavi della lista puntata. Sia m un valore che rappresenta il minimo numero dei primi elementi della lista puntata la somma delle cui chiavi superi k.

Ad esempio, se la lista è la seguente:

$$p \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$$

e k vale 5, allora il valore restituito m sarà pari a 3, perché $1+2+3 \geq k$ mentre $1+2 < k$.

Si progetti una **funzione ricorsiva** che, dato in input il puntatore p ed il parametro k, restituisca il valore m.

Della funzione proposta:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale.

NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali né di memoria aggiuntiva di dimensione non costante.

Possiamo ragionare in questo modo:
se p.val >=k **allora la risposta e' ovviamente 1**
altrimenti la risposta è 1 + la risposta che si ha per la lista che è accessibile dal puntatore p.next **quando invece che** k **si ha** $k - p.val$**.**

Un possibile codice in python è il seguente.

```
def es3(p,k):
    if p.val >= k:
        return 1
    return 1 + es3(p.next, k-p.val)
```

Il caso peggiore si ha quando si scorre tutta la lista e, in questo caso, il costo dell'algoritmo è dato dalla ricorrenza:
$T(n) = T(n-1) + \Theta(1)$
$T(1) = \theta(1)$

La soluzione della ricorrenza, svolta ad esempio col metodo di sostituzione, è $\Theta(n)$.