Algoritmi e Strutture Dati
14 Giugno 2017

Cognome ..... Nome ..... Matricola .....

Note

1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
2. Quando si presenta un algoritmo è fondamentale spiegare l'idea soggiacente e motivarne la correttezza.
3. L'efficienza è un criterio di valutazione delle soluzioni proposte.
4. Si consegnano tutti i fogli, con nome, cognome, matricola e l'indicazione *bella copia* o *brutta copia*.

Domande

Domanda A (4 punti) Si dimostri che la ricorrenza che segue ha soluzione $T(n) = \Theta(n)$

$$T(n) = \frac{1}{2}T(n-1) + n$$

Domanda B (4 punti) Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
10	6	2	8	19	31	15

Spiegare il processo di costruzione del codice.

Domanda C (6 punti) Sia T un albero binario i cui nodi x hanno i campi *x.left*, *x.right*, *x.key*. L'albero si dice un *sum-heap* se per ogni nodo x, la chiave di x è maggiore o uguale sia alla somma delle chiavi nel sottoalbero sinistro che alla somma delle chiavi nel sottoalbero destro.

Scrivere una funzione `IsSumHeap(T)` che dato in input un albero T verifica se `T` è un sum-heap e ritorna un corrispondente valore booleano. Valutarne la complessità.

Esercizi

Esercizio 1 (7 punti) Si consideri una estensione degli alberi binari di ricerca nei quali ogni nodo x ha anche un campo booleano *x. even* che vale *true* o *false* a seconda che la somma delle chiavi nel sottoalbero radicato in x sia pari o dispari. Realizzare la procedura `Insert(T,z)` di modo che mantenga correttamente aggiornato anche il campo *even*. Valutarne la complessità.

Esercizio 2 (9 punti) Un marinaio ha n pezzi di corda di lunghezze intere $l_1, \ldots, l_n$ ($l_i \geq 2$) e deve annodarli per ottenere una corda di lunghezza esattamente d. Tenendo conto che fare un nodo richiede una lunghezza pari ad 1 (ad es. se annodo due pezzi lunghi 5 e 7 la corda che ottengo è lunga 11), individuare, se esiste, un insieme minimo di pezzi che annodati producano una corda della lunghezza d desiderata.

Più precisamente:

i. Dare una caratterizzazione ricorsiva del numero minimo $m(i, d')$ di pezzi di corda scelti in $l_1, \ldots, l_i$ che possono essere annodati per produrre la lughezza d' ($+\infty$ se non è possibile ottenere d' con nessuna combinazione);

ii. Usare la caratterizzazione al punto precedente per ottenere un algoritmo $Rope(l, n, d)$ che dato l'array delle lunghezze $l[1..n]$ determina il numero minimo di pezzi da annodare per ottenere d (se esiste);

iii. trasformare l'algoritmo in modo che restituisca oltre al numero anche l'indicazione di quali pezzi usare;

iv. valutare la complessità dell'algoritmo.

Nota: Correzione, risultati e visione dei compiti: *Giovedì 29 Giugno, ore 9:30, 1BC/50*

CORREZIONE APPELLO (1° APPELLO 2016|2017)

ES. 1 (Ricorrenza)

DOMANDE

Ricorrenza per sostituzione

- $T(n) = \frac{1}{2} T(n-1) + n = \Theta(n)$

1) $T(n) \leq cn$
2) $T(n) \geq dn$

$\exists c > 0, n_0 \ \forall n > n_0$

$\exists d > 0, n_0 \ \forall n > n_0$

1) $T(n) \leq cn$

Assumendo $\forall n' < n$ mostro che la stessa cosa vale per n.

$T(n) = \frac{1}{2} T(n-1) + n$

$\frac{1}{2} c(n-1) + n \leq cn$

$\frac{1}{2} cn - \frac{1}{2} c + n \leq cn$

$\frac{1}{2} c(n+1) \geq n$

$c \geq 2$

$c \geq 2$

ES. 2 (Huffman)

a	b	c	D	E	F	G
10	6	2	8	9	31	15

A: 100
B: 0001
C: 0000
D: 001
E: 01
F: 11
G: 101

ES. 3 (Is Sum Heap)

$x.key \geq \sum_{y \in L} y.key$

$\sum_{y \in R} y.key$

← [Dato in input un albero dicesse se è V o F la condizione]

Is Sum Heap (T)
  v, _ = Is SumHeap-rec (T.root)

→ ritorna un valore booleano che ci dice se è sum-heap?

→ dist: la somma delle chiavi del corrispondente sottoalbero.

Is SumHeap-rec
  if x = nil
    return true, 0
  else
    IsSumHeapL, sumL = IsSumHeap-rec (x.left)
    IsSumHeapR, sumR = IsSumHeap-rec (x.right)
    Is SumHeap = isSumHeapL and IsSumHeapR and (x.key ≥ sumL)
                 and (x.key ≥ sumR)
    return IsSumHeap, x.key + sumL + sumR

$T(n) = T(k) + T(n-k-1)$ CASO PEGGIORE $2T(n/2) + c$

COMPLESSITÀ:

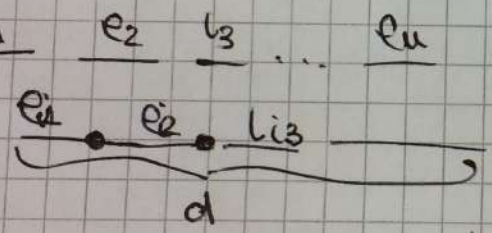
ES. 1

```
Insert (T, z)
  x = T.root
  y = nil
  z.even = (z.key mod 2 == 0)
  while (x <> nil)
    x.even = (x.even == z.even)
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
```

$x.even = \left(\sum_{y.key\ pari} 1\right) + z.key$

COMPLESSITÀ $O(h)$

ES. 2

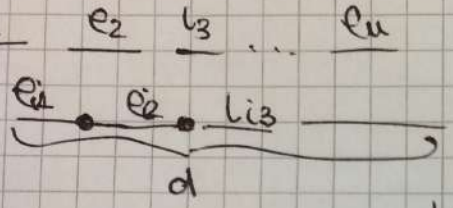


$l_1 \dots l_i$

$m(i, d')$ = * minimo di cordate che ammontano almeno a d'
(+∞ se non è possibile.)

$$m(i,d') = \begin{cases} +\infty & i = 0, d' > 0 \\ 1 & \text{se } i' = d' \text{ (sorta di caso base)} \end{cases}$$

ES. 2



$m(i, d')$ = * minimo di corde che ricoprono $l_1 \dots l_i$ (+∞ se non è possibile.)

$$m(i,d') = \begin{cases} +\infty & i=0, d'>0 \\ 1 \text{ se} & e_i = d' \text{ (sorta di caso base)} \\ m(i-1,d') & e_i > d' \\ \min\{m(i-1,d'-e_i+1)+1, \\ \quad m(i-1,d')\} \end{cases}$$

Rope (l, n, d) $e[1 \dots n]$ lunghezze, $d>0$

```
for d' = 1 to d
    m[0, d'] = +∞        S[1..n, 1..d] = false   iii) S[i,d'] = { true se li è utilizzato; false altrimenti
for i = 1 to n
    for d' = 1 to d
        if l[i] = d'
            m[i,d'] = 1          S[i,d'] = true
        else if
            m[i,d'] = m[i-1,d']
        else e[i] < d'
```

$$m(i,d') = \begin{cases} +\infty & i=0, d'>0 \\ 1 & \text{se } \ell_i = d' \text{ (sorta di caso base)} \\ m(i-1,d') & \ell_i > d' \\ \min\{m(i-1,d'-\ell_i+1)+1,\ m(i-1,d') & \end{cases}$$

Rope (ℓ, n, d) $\ell[1 \dots n]$ lunghezze
$d>0$

```
for d'=1 to d
    m[0,d'] = +∞        S[1..n, 1..d] = false      iii) S[i,d'] = { true se lo i è utilizzato; false altrimenti
for i=1 to n
  for d'=1 to d
    if L[i] = d'
        m[i,d'] = 1          S[i,d'] = true
    else if
        m[i,d'] = m[i-1,d']
    else ℓ[i] < d'
        m_in = m[i-1, d'-ℓ[i]+1) + 1
        m_out = m[i-1, d']
        if m_in <= m_out
            m[i,d'] = m_in          S[i,d'] = true
        else
            m[i,d'] = m_out          S[n,d] = false
return m[i,d'] = m_out                 :
d'=d, i=n                              S[i,d] = true
while d'>0                             S[i-1, d-ℓi+1]
    if (S[i,d']
        print("ℓ%d", i)
        d'=0
    else d' = d'-ℓ[i]+1
```