Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Esercizio 1

Esercizio. Si risolva la seguente equazione di ricorrenza con due metodi diversi, per ciascuno dettagliando il procedimento usato:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \textit{se } n > 1$$

$$T(1) = \Theta(1)$$

tenendo conto che n è una potenza di 2.

Soluzione: Utilizziamo dapprima il metodo principale, che è il più rapido, e poi verifichiamo la soluzione trovata con il metodo iterativo.

Esercizio 1 svolto

$T(n) = T(n/2) + \Theta(n^2)$

Metodo principale:

- Le costanti *a* e *b* del teorema principale valgono qui 1 e 2, rispettivamente, quindi $\log_b a = \log_2 1 = 0$, per cui, ponendo $\epsilon = 2$, si ha $f(n) = \Theta(n^2) = \Omega(n^{0+2}) = \Omega(n^{\log_b a + \epsilon})$ e siamo nel caso 3. del teorema a patto che valga anche $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ per una qualche costante $c < 1$ e n sufficientemente grande.

- Poiché *f(n)* è espressa tramite notazione asintotica, per verificare la disuguaglianza, dobbiamo eliminare tale notazione, e porre ad esempio $f(n) = \Theta(n^2) = h \cdot n^2 + k$.

- Ci chiediamo quindi se esiste $c < 1$ tale che $h\left(\frac{n}{2}\right)^2 + k \leq c(hn^2 + k)$.

- Risolvendo si ha : $hn^2\left(\frac{1}{4} - c\right) + k(1 - c) \leq 0$ che è vera ad esempio per *c* = 1/2.

- Ne possiamo dedurre che $T(n) = \Theta(n^2)$.

Esercizio 1 svolto

$T(n) = T(n/2) + \Theta(n^2)$

Metodo iterativo:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$= T\left(\frac{n}{2^2}\right) + \Theta\left(\left(\frac{n}{2}\right)^2\right) + \Theta(n^2)$$

$$= T\left(\frac{n}{2^3}\right) + \Theta\left(\left(\frac{n}{2^2}\right)^2\right) + \Theta\left(\left(\frac{n}{2}\right)^2\right) + \Theta(n^2)$$

$$= \dots$$

$$= T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \Theta\left(\left(\frac{n}{2^i}\right)^2\right)$$

Esercizio 1 svolto

$T(n) = T(n/2) + \Theta(n^2)$

$$T(n) = T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \Theta\left(\left(\frac{n}{2^i}\right)^2\right)$$

quando $k = \log n$ si ha $\frac{n}{2^k} = 1$ e ci fermiamo ed otteniamo

$$T(n) = \Theta(1) + \sum_{i=0}^{\log n - 1} \Theta\left(\left(\frac{n}{2^i}\right)^2\right)$$

$$= \Theta(1) + n^2 \sum_{i=0}^{\log n - 1} \Theta\left(\left(\frac{1}{4}\right)^i\right)$$

$$= \Theta(1) + n^2 \cdot \Theta\left(\frac{1 - \left(\frac{1}{4}\right)^{\log n}}{1 - \frac{1}{4}}\right)$$

$$= \Theta(n^2)$$

nota: ho usato che $\sum_{i=0}^{b} a^i = \frac{a^{b+1} - 1}{a - 1}$

Esercizio 2

Dato un vettore che contiene solo numeri negativi e positivi (nessun valore pari a zero), riorganizzarlo in modo che tutti i numeri negativi stiano a sinistra di quelli positivi.

IDEA

Ordinare il vettore risolve il problema, ma è inutilmente costoso.

Basta adattare la *Partition*() del *Quicksort* in modo che scambi un positivo puntato da *i* con un negativo puntato da *j*

soluzione 1 Esercizio 2:

```
def separaPosNeg(A):
    i, j = 0, len(A)-1
    while i < j:
        while A[j] > 0 and i <= j:
            j -= 1
        while A[i] < 0 and i <= j:
            i += 1
        if i < j:
            A[i], A[j] = A[j], A[i]
```

```
>>> A=[3,4,-1,5,6,-2,-7,-8,9]
>>> separaPosNeg(A)
>>> A
[-8, -7, -1, -2, 6, 5, 4, 3, 9]
```

Costo computazionale: $\Theta(n)$

soluzione 2 Esercizio 2:

IDEA: Il problema di separare gli elementi positivi dai negativi si può ridurre a quello di ordinare *n* numeri interi nel range [0,1]; si può quindi applicare una modifica dell'algoritmo di *Counting Sort*, in cui non serve il vettore C, perché non abbiamo dati satellite:

```
def separaPosNeg2(A):
    n=len(A)
    B=[0]*n
    neg, pos = 0, n-1
    for x in A:
        if x>0:
            B[pos]= x
            pos -= 1
        else:
            B[neg]= x
            neg += 1
    return B
```

```
>>> A=[3,4,-1,5,6,-2,-7,-8,9]
>>> SeparaPosNeg2(A)
[-1, -2, -7, -8, 9, 6, 5, 4, 3]
```

Costo computazionale: $\Theta(n)$

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizio 3.

Progettare un algoritmo che, dato in input un vettore che rappresenta un heap massimo di n elementi, restituisca il valore minimo.

L'algoritmo deve avere complessità $\Theta(n)$ ed il numero di elementi dell'heap esaminati deve essere al più $\left\lceil \frac{n}{2} \right\rceil$.

Esercizio 4

Scrivere una funzione che dati un max Heap A, il suo size_heap e un valore k inserisce k nel Max-heap.

La funzione deve avere complessità $O(\log size\_heap)$.

Esercizio 5

Scrivere una funzione che dati un Max Heap A, ed il suo size_heap restituisce l'elemento massimo dell'heap dopo averlo cancellato dall'heap.

La funzione deve avere complessità $O(\log size\_heap)$.

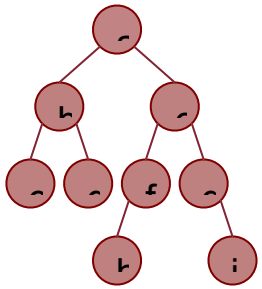
Esercizio 6

Scrivere una funzione che dati un Max Heap A, il suo size_heap e l'indice i di un posizione dell'heap, cancella dall'heap l'elemento A[i].

La funzione deve avere complessità $O(\log size\_heap)$.

Esercizio 7

Scrivere una funzione che dato in input un albero binario A, con n chiavi e memorizzato con la notazione posizionale ne stampa le chiavi in preorder . La procedura deve avere complessità $O(n)$.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	b	e	c	d	f	g	-	-	-	-	h	-	-	i

stampaPreorder: a b c d e f h g i

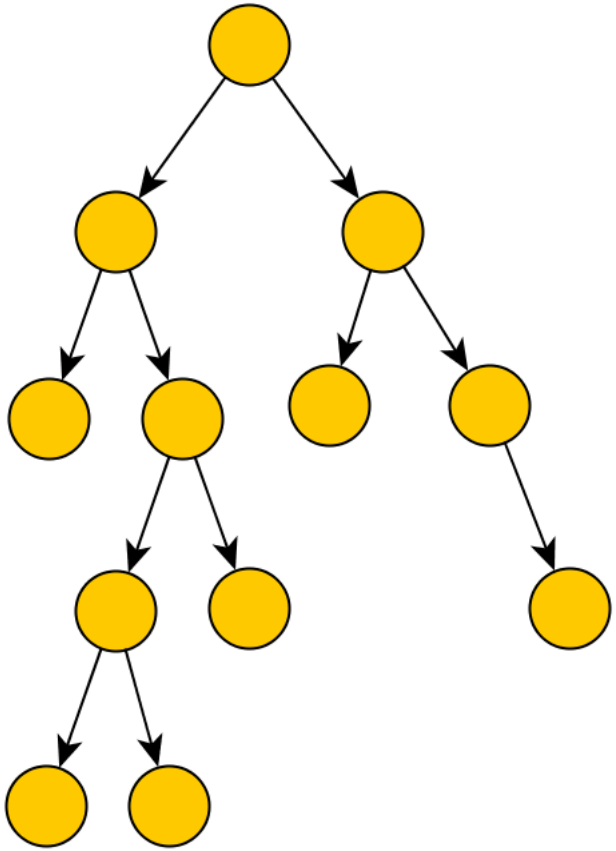
Esercizio 8

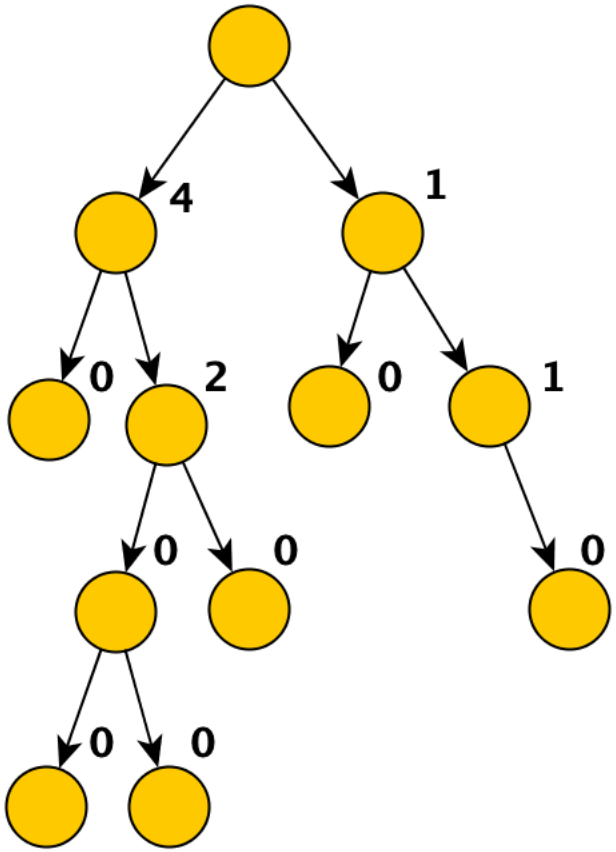
In un albero binario lo **sbilanciamento** di un nodo è il valore assoluto tra il numero di nodi nel suo sottoalbero sinistro ed il numero di nodi nel suo sottoalbero destro.

Dato un albero binario di n nodi si conosce il puntatore alla radice, trovare il massimo tra gli sbilanciamenti dei suoi nodi.

La funzione deve essere ricorsiva e richiedere tempo $O(n)$.

esempio Esercizio 8





Il massimo sbilanciamento dell'albero è 4