Introduzione agli Algoritmi
Esame Scritto
I canale e CdL in Teledidattica
con spunti per la soluzione

docente: T. Calamoneri
Sapienza Università di Roma
12 Giugno 2024

Esercizio 1 (10 punti) Si consideri la seguente funzione:

```
def Potenza(n, k):
    if k == 0: return 1
    if k == 1: return n
    if k % 2 ==0:
       pot = Potenza (n, k DIV 2)
       return pot * pot
    else:
       pot = Potenza (n, (k-1) DIV 2)
       return n * pot * pot
```

a) Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.

b) La si risolva utilizzando **due dei metodi studiati**, dettagliando sia i passaggi matematici che quelli logici.

Si ottiene l'equazione di ricorrenza della ricerca binaria, ampiamente discussa a lezione, la cui soluzione è $T(n) = \Theta(\log n)$.

Esercizio 2 (10 punti): Si definisce *punto di sella* di una matrice quell'elemento che gode della proprietà di essere simultaneamente minimo di riga e massimo di colonna.

Si progetti un algoritmo che, data una matrice quadrata M di $n \times n$ interi distinti, restituisca una coppia di indici (i, j) corrispondenti alla posizione del punto di sella in M, e $None$ se la matrice non ha punti di sella.
L'algoritmo deve avere costo computazionale $\Theta(n^2)$.
Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

a) Possiamo applicare al definizione stessa di punto di sella: per ogni riga i calcoliamo la posizione j in cui compare l'elemento minimo e poi calcoliamo la posizione del massimo della colonna j; se questa posizione è i allora abbiamo trovato il punto di sella e restituiamo gli indici (i, j), in caso contrario passiamo alla riga successiva. Se in nessuna delle righe troviamo il punto di sella terminiamo restituendo $None$. Segue un possibile codice Python dell'idea proposta.

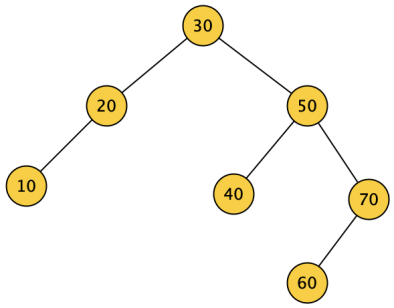
```python
def es2(M):
    n= len(M)
    for i in range(n):
        j= minR(i,M)
        if maxC(j,M)==i:
            return i,j


def minR(i,M):
    n, mj = len(M), 0
    for j in range(1,n):
        if M[i][j]<M[i][mj]: mj = j
    return mj


def maxC(j, M):
    n, mi = len(M), 0
    for i in range(1,n):
        if M[i][j]>M[mi][j]: mi=i
    return mi
```

Qui, la funzione $minR(i, M)$ restituisce la posizione nella riga i di M in cui si trova il minimo, mentre $maxC(j, M)$ restituisce la posizione nella colonna j di M in cui si trova il massimo.

b) Il costo di una chiamata a $minR$ o a $maxC$ è $\Theta(n)$ quindi ogni iterazione del for nel programma principale richiede $\Theta(n)$; poiché le iterazioni sono n, il costo computazionale dell'algoritmo risulterà $\Theta(n^2)$.

Esercizio 3 (10 punti): Siano dati un intero positivo k ed un albero binario di ricerca T di altezza h; si vuole individuare la k-esima chiave di T se queste fossero messe in ordine.

Ad esempio per l'albero in figura:



- con $k = 1$ la risposta è 10,
- per $k = 5$ la risposta è 50
- per $k = 9$ la risposta è $None$ perché l'albero ha meno di k nodi.

L'albero è memorizzato tramite puntatori e record di quattro campi: il campo key contenente il valore, i campi $left$ e $right$ con i puntatori al figlio sinistro e al figlio destro, rispettivamente (questi puntatori valgono $None$ in mancanza del figlio), ed il campo num con l'indicazione del numero dei nodi nel sottoalbero in esso radicato.

Si progetti un algoritmo RICORSIVO che risolva il problema in un tempo computazionale $O(h)$.
Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

NOTA BENE: nello pseudocodice dell'algoritmo ricorsivo è preferibile **non** far uso di variabili globali.

a) **Per ciascun nodo p, grazie al campo num suo e dei suoi figli (in realtà del solo figlio sinistro), p è in grado di scoprire se la chiave da stampare sia la sua, si trovi nel sottoalbero di sinistra, o nel sottoalbero di destra. Il primo caso si ha o quando p non ha figlio sinistro e $k = 1$, oppure quando il figlio sinistro esiste e nel sottoalbero sinistro ci sono esattamente $k - 1$ nodi. Gli altri due casi sono gestiti ricorsivamente. Da notare che, mentre nella chiamata ricorsiva a sinistra k rimane inalterato, nella chiamata a destra bisogna tenere conto del fatto che, andando verso destra, alcune chiavi sono state di fatto già conteggiate e quindi il valore di k nella chiamata cambia.**

Segue una possibile codifica in Python dell'algoritmo descritto che non utilizza variabili globali:

```
def cerca(p,k):
    if p==None or k>p.num:
        return None
    if p.left == None:
        if k==1:
            return p.key
        else:
            return cerca(p.right, k-1)
    if p.left.num >= k:
         return cerca(p.left, k)
    if p.left.num == k-1:
         return p.key
    return cerca(p.right, k-1-p.left.num)
```

b) **Osserviamo che la radice dell'albero ha altezza h e le chiamate ricorsive vengono effettuate su sottoalberi di altezza al più pari ad $h-1$. Ad ogni chiamata ricorsiva, quindi, si discende almeno di un livello nell'albero; di conseguenza, per i tempi di calcolo abbiamo la seguente equazione di ricorrenza:**

- $T(h) \leq T(h-1) + \Theta(1)$
- $T(0) = \Theta(1)$

L'equazione si può risolvere con uno dei metodi studiati (che va esplicitamente svolto) e dà come soluzione $O(h)$. Ogni passaggio va dettagliato.