Algoritmi e Strutture Dati
31 agosto 2021

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 e motivarne la correttezza.
3. L'efficienza e l'aderenza alla traccia sono criteri di valutazione delle soluzioni proposte.
4. Si consegnano i fogli di bella copia.

Domande

Domanda A (7 punti) Si consideri la funzione ricorsiva `search(A,p,r,k)` che dato un array A, ordinato in modo decrescente, un valore k e due indici p, q con $1 \leq p \leq r \leq A.length$ restituisce un indice i tale che $p \leq i \leq r$ e $A[i] = k$, se esiste, e altrimenti restituisce 0.

```
search(A,p,r,k)
   if p <= r
      q = (p+r)/2
      if A[q] = k
         return q
      elseif A[q] > k
         return search(A,q+1,r,k)
      else
         return search(A,p,q-1,k)
   else
      return 0
```

Dimostrare induttivamente che la funzione è corretta. Inoltre, determinare la ricorrenza che esprime la complessità della funzione e risolverla con il Master Theorem.

Soluzione: La prova di correttezza avviene per induzione sulla lunghezza $l = r - p + 1$ del sottoarray $A[p..r]$ di interesse. Se $l = 0$, ovvero $p > r$, la funzione ritorna correttamente 0, dato che non ci sono elementi nel sottoarray e quindi non ci possono essere elementi $= k$. Se invece $l > 0$ si calcola $q = \lfloor (p + r)/2 \rfloor$ e si distinguono tre casi:

- se $A[q] = k$ si ritorna correttamente k
- se $A[q] > k$, dato che l'array è ordinato in modo decrescente certamente $A[j] \geq A[q] > k$ per $p \leq j \leq q$. Quindi il valore k può trovarsi solo nel sottoarray $A[q + 1, r]$. La chiamata $search(A, q + 1, r, k)$, dato che il sottoarray ha lunghezza minore di l, per ipotesi induttiva restituisce un indice i tale che $q + 1 \leq i \leq r$ e $A[i] = k$, se esiste, e altrimenti restituisce 0. Per l'osservazione precedente, questo è il valore corretto da restituire anche per $search(A, p, r, k)$ e si conclude.
- se $A[q] < k$ si ragiona in modo duale rispetto al caso precedente.

Per quanto riguarda la ricorrenza, ignorando i casi base, dato che la funzione ricorre su di un array la cui dimensione è la metà di quello originale, si ottiene:

$$T(n) = T(n/2) + c$$

Rispetto allo schema standard del master theorem ho $a = 1$, $b = 2$ e $f(n) = c$. Ho dunque che $f(n) = 1 = \Theta(n^{\log_2 1} = n^0 = 1$. Pertanto si conclude che $T(n) = \Theta(n^0 \log n) = \Theta(\log n)$.

Domanda B (6 punti) Si consideri una tabella hash di dimensione $m = 8$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \ mod \ m$ e $h_2(k) = 1 + 2(k \ mod \ (m - 2))$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 13, 29, 19, 27, 8.

Soluzione: Si ottiene

0	29
1	-
2	27
3	19
4	-
5	13
6	-
7	8

Esercizi

Esercizio 1 (9 punti) Si consideri un albero binario T, i cui nodi x hanno i campi $x.l$, $x.r$, $x.p$ che rappresentano il figlio sinistro, il figlio destro e il padre, rispettivamente. Un *cammino* è una sequenza di nodi $x_0, x_1, \ldots, x_n$ tale che per ogni $i = 0, \ldots, n-1$ vale $x_{i+1}.p = x_i$. Il cammino è detto *nil-terminated* se $x_n.l = nil$ oppure $x_n.r = nil$. Dato un certo k, diciamo che l'albero è k-bilanciato se tutti i cammini nil-terminated che iniziano dalla radice hanno lunghezze che differiscono al più di k. Scrivere una funzione `bal`(T, k) che dato in input l'albero T e un valore k verifica se T è k-bilanciato e ritorna un corrispondente valore booleano. Valutarne la complessità.

Soluzione: La soluzione può basarsi su di una funzione ricorsiva che calcola per un nodo x la lunghezza massima e minima dei cammini che iniziano da x e sono nil-terminated

```
bal(T,k)    // verifica se l'albero radicato in x e' k-bilanciato,
             // ovvero i cammini dalla radice terminati da nil hanno
            // lunghezze che differiscono al piu di k

          // si basa su di una funzione ricorsiva che calcola la
          // lunghezza minima e massima dei cammini nil-terminated
          // da un certo nodo.
 min,max = nilTerm(T.root)

 return max-min <= k

nilTerm(x)
 if x = nil
    return (0,0)
 else
    (left_min,left_max) = nilTerm(x.left)
    (right_min,r_right_max) = nilTerm(x.right)
    return min(left_min, right_min) + 1, max(left_max,right_max) + 1
```

Per quanto riguarda la complessità si osservi che $T(n) = c + T(k) + T(n-k-1)$ per un'opportuna costante c e quindi, la complessità è $O(n)$.

Esercizio 2 (9 punti) Data una stringa di numeri interi $A = (a_1, a_2, \ldots, a_n)$, si consideri la seguente ricorrenza $z(i,j)$ definita per ogni coppia di valori (i,j) con $1 \le i, j \le n$:

$$z(i,j) = \begin{cases} a_j & \text{if } i = 1, 1 \le j \le n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \le n, \\ z(i-1,j) \cdot z(i,j+1) \cdot z(i-1,j+1) & \text{altrimenti.} \end{cases}$$

1. Si fornisca il codice di un algoritmo iterativo bottom-up Z(A) che, data in input la stringa A restituisca in uscita il valore $z(n,1)$.

2. Si valuti il numero esatto $T_Z(n)$ di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

Soluzione:

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione "reverse column-major", in cui calcoliamo gli elementi della tabella in ordine decrescente di indice di colonna e, all'interno della stessa colonna, in ordine crescente di indice di riga. Il codice è il seguente.

   ```
   Z(A)
   n = length(A)
   for i=1 to n do
      z[1,i] = a_i
      z[i,n] = a_{n+1-i}
   for j=n-1 downto 1 do
      for i=2 to n do
         z[i,j] = z[i-1,j] * z[i,j+1] * z[i-1,j+1]
   return z[n,1]
   ```

   Si osservi che un altro modo corretto di riempire la tabella è attraverso una scansione "reverse diagonal", che scansiona per diagonali parallele alla diagonale principale partendo da quella contenente solo $z[1, n]$.

2. Ogni iterazione del doppio ciclo dell'algoritmo esegue due moltiplicazioni tra interi, e quindi

$$\begin{aligned} T_Z(n) &= \sum_{j=1}^{n-1} \sum_{i=2}^{n} 2 \\ &= \sum_{j=1}^{n-1} 2(n-1) \\ &= 2(n-1)^2. \end{aligned}$$

   Equivalentemente, basta osservare che l'algoritmo esegue due moltiplicazioni per ogni elemento di una tabella $(n-1) \times (n-1)$.