Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 14 Gennaio 2009 —

Esercizio 1 - ASD

Si risolvano le seguenti ricorrenze, giustificando la risposta.

- $T(n) = 9T(\frac{n}{3}) + 3n^2 + 2\sqrt{n}$
- $T(n) = \frac{5}{2}T(\frac{n}{2}) + n$

Soluzione

- Poiché $\lg_3 9 = 2$ e $f(n) = 3n^2 + 2\sqrt{n} = \Theta(n^2)$ siamo nel caso 2 del MT e la soluzione è $T(n) = \Theta(n^2 \lg n)$.
- Poiché $\lg_2 \frac{5}{2} > 1$ esiste $\epsilon > 0$ tale che $1 = \lg_2 \frac{5}{2} - \epsilon$. Quindi $f(n) = n = O(n^{\lg_2 \frac{5}{2} - \epsilon})$. Siamo nel caso 1 del MT e la soluzione è $T(n) = \Theta(n^{\lg_2 \frac{5}{2}})$.

Esercizio 2 - ASD

Date le seguenti procedure A e B, si determini la complessità asintotica della procedura A(n) su input n.

A(n)
```
1  s ← 0
2  for i ← 1 to n
3        do s ← s + B(i)
4  return s
```

B(m)
```
1  s ← 0
2  for j ← 1 to m
3        do s ← s + 1
4  return s
```

Soluzione

La complessità di B(i) è lineare in i pertanto ad ogni iterazione del ciclo la terza istruzione di A ha costo ki per una qualche costante k. Sommando su tutti i valori che assume la variabile i otteniamo $T_A(n) = \sum_{i=1}^{n}(ki) = \Theta(n^2)$

Esercizio 3 - ASD

Scrivere un algoritmo che dato un albero binario di ricerca T, bilanciato e contenente chiavi tutte distinte, e due chiavi $k_1 < k_2$ restituisce **true** se e solo se T soddisfa anche la seguente proprietà:
- per ogni nodo x di T se $key[x] = k_1$ allora non esiste alcun nodo $y \neq x$ in T tale che $k_1 < key[y] < k_2$.
Dire qual è la complessità dell'algoritmo rispetto al numero delle chiavi memorizzate nell'albero e spiegare perché è corretto.

Soluzione

Precondizione: x è la radice di un BST bilanciato.

Postcondizione: L'algoritmo risponde **true** se l'albero radicato in x gode della proprietà data, **false** altrimenti.

CHECK$(x, k1, k2)$

```
1  z ← BSTSEARCH(x, k1)
2  if ((z = NIL) ∨ (SUCC(z) = NIL)
3     then return true
4     else  return (key[SUCC(z)] ≥ k2)
```

Correttezza: Se la chiave k_1 non compare nell'albero (la BSTSEARCH ritorna NIL) allora la condizione è soddisfatta; idem se compare ma è la più grande (SUCC(z) ritorna NIL); altrimenti (compare e non è la più grande), è sufficiente verificare che la chiave del successore di z sia maggiore o uguale a k_2. Infatti la proprietà BST ci assicura che tutti i predecessori di un nodo hanno chiave minore o uguale a quella del nodo mentre i suoi successori hanno chiave maggiore o uguale.

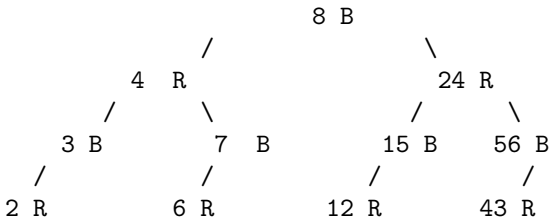
Complesità: La complessità è logaritmica nel numero delle chiavi. Infatti sia la ricerca di una chiave che l'individuazione del successore sono operazioni lineari nell'altezza dell'albero, e l'albero è bilanciato.

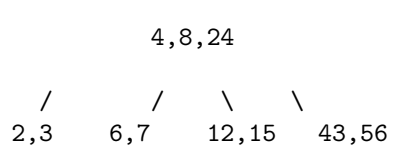
Esercizio 4 - ASD

Si disegni un albero R/B che contiene le seguenti chiavi: 6, 15, 8, 2, 24, 56, 3, 43, 12, 4, 7. Lo si trasformi poi in un albero 2-3-4.

Soluzione

Una possibile soluzione è la seguente.





Esercizio 5 - ASD

Si sviluppi un algoritmo che, dati un albero generale T e un intero positivo $k > 0$, conta il numero di nodi di grado k in T. Si supponga che l'albero generale sia rappresentato tramite gli attributi: `fratello` e `figlio`. (Ricordiamo che il grado di un nodo di un albero è pari al numero dei suoi figli.)

Soluzione

```
CONTAGRADO(x, k)
 1  if (x = NIL)
 2      then return 0
 3      else  z ← figlio[x]
 4            n ← 0
 5            while (z ≠ NIL)
 6                  do n ← n + 1
 7                     z ← fratello[z]
 8            if (n = k)
 9               then return 1 + CONTAGRADO(fratello[x], k) + CONTAGRADO(figlio[x], k)
10               else  return CONTAGRADO(fratello[x], k) + CONTAGRADO(figlio[x], k)
```

Esercizio 1 (Laboratorio)

Una sequenza ordinata è una collezione in cui gli elementi compaiono in modo ordinato e sono ammesse più copie dello stesso elemento. Si vuole realizzare una classe *SequenzaOrdinata* per rappresentare una sequenza ordinata di elementi di tipo stringa. La struttura dati scelta per memorizzare la sequenza ordinata è l'array.

```
public class SequenzaOrdinata {
    private static final int defaultSize = 100;
    private String[] S = new String[defaultSize];    // la sequenza e' un array
    private int count;     // tot. elementi della sequenza


    // pre:  s non nulla
    // post: aggiunge la stringa s alla sequenza ponendola nella
    //       posizione corretta rispetto all'ordine e aggiornando count.
    //       Ritorna true se l'operazione e' riuscita, false se non c'e' piu'
    //       spazio libero nell'array
    public boolean insert(String s) {...}

    ...
}
```

Si richiede di completare l'implementazione del metodo *insert* della classe *SequenzaOrdinata* riportata sopra e di dimostrarne la correttezza.

Soluzione

- Una possibile implementazione è la seguente:

```
public class SequenzaOrdinata {
    private static final int defaultSize = 100;
    private String[] S = new String[defaultSize];    // la sequenza e' un array
    private int count;     // tot. elementi della sequenza


    // pre:  s non nulla
    // post: aggiunge la stringa s alla sequenza ponendola nella
    //       posizione corretta rispetto all'ordine e aggiornando count.
    //       Ritorna true se l'operazione e' riuscita, false se non c'e' piu'
    //       spazio libero nell'array
    public boolean insert(String s) {

         if (count == defaultSize)
            return false;

         // INV: gli elementi originariamente in S[j+1,count-1] sono maggiori
         //      di s e sono stati spostati in S[j+2,count]
         int j;
         for (j = count-1; j >= 0 && S[j].compareTo(s) > 0 ; j--)
             S[j+1] = S[j];
         S[j+1] = s;
         count++;
         return true;
    }

   ...
}
```

- Verifichiamo l'invariante riportato nel codice:

  Inizializzazione: All'inizio j=count-1 e quindi la porzione di array S[cont,count-1] e' vuota. Lo stesso per S[count+1,count]. L'invariante è banalmente verificato.

Mantenimento: Sia INV vero per j fissato. Allora gli elementi originariamente in S[j+1, count-1] sono maggiori di s e sono stati spostati in S[j+2,count] . Si entra nel ciclo solo se S[j] > s e, in tal caso l'elemento S[j] viene copiato in S[j+1]. Allora, dopo l'esecuzione del corpo del ciclo e' vero che gli elementi originariamente in S[j...count-1] sono maggiori di s e sono stati spostati in S[j+1,count]. Quindi l'invariante viene mantenuto al decrementare di j.

Terminazione: il ciclo termina se j >= 0 e S[j] < s oppure se j=-1. In entrambi i casi è vero che gli elementi originariamente in S[j+1..count-1] sono maggiori di s e sono stati spostati in S[j+2, count]. L'invariante garantisce quindi la correttezza dell'inserimento di s in posizione j+1.

Esercizio 2 (Laboratorio)

1. Si consideri una tabella hash T[0,...,6] = [-, 1, C, -, 11, 19, 26] in cui le posizioni 0 e 3 sono libere e la posizione 2 risulta marcata in seguito ad un'operazione di cancellazione. La tabella è stata costruita utilizzando la funzione hash
$$h(k) = k \bmod 7$$
e tecnica di gestione delle collisioni ad indirizzamento aperto con scansione lineare. Si consideri il problema di inserire la chiave 67 nella tabella. Dire se la chiave viene inserita, eventualmente in quale posizione e quali posizioni della tabella vengono esaminate con insuccesso.

2. Si consideri una tabella hash T[0...22] in cui si vogliono memorizzare dati relativi agli studenti che sostengono questo appello d'esame. Gli studenti vengono identificati dal numero di matricola, che diventa la chiave della tabella hash. Si consideri la seguente funzione hash basata sul metodo del ripiegamento:
$$h(K) = h(k_1k_2k_3k_4k_5k_6) = (k_1k_2 \ - \ k_3k_4 \ + \ k_5k_6) \bmod 23$$
dove $k_1k_2k_3k_4k_5k_6$ sono le cifre decimali che compongono K. Ad esempio, h(816145) = (81 - 61 + 45) mod 23 = 65 mod 23 = 19. Partendo dalla tabella contenente solamente la chiave utilizzata come esempio (816145), si richiede di inserire in tabella le seguenti chiavi specificando per ciascuna la posizione di inserimento e il numero di accessi effettuati:
   (a) 837120
   (b) 829133
   (c) 818326
   (d) 792364

   Per la soluzione di eventuali collisioni utilizzare la scansione quadratica
$$c(k,i) = (h(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod 23 \quad 0 \le i \le 22$$
   con $c_1 = 1$ e $c_2 = 3$.

Soluzione

1. La chiave 67 viene inserita in posizione 0 dopo aver esaminato le posizioni 4, 5 e 6.

2. (a) h(837120) = (83 - 71 + 20) mod 23 = (103 - 71) mod 23 = 32 mod 23 = 9, libera. Un accesso.
   (b) h(829133) = (82 - 91 + 33) mod 23 = (115 - 91) mod 23 = 24 mod 23 = 1, libera. Un accesso
   (c) h(818326) = (81 - 83 + 26) mod 23 = 24 mod 23 = 1, occupata.
   Per i = 1 la nuova posizione è ((1 + 1 + 3) mod 23) = 5. Due accessi.
   (d) h(792364) = (79 - 23 + 64) mod 23 = 120 mod 23 = 5, occupata.
   Per i = 1 la nuova posizione è ((5 + 1 + 3) mod 23) = 9, occupata.
   Per i = 2 la nuova posizione è ((5 + 2 + 3*4) mod 23) = 19, occupata.
   Per i = 3 la nuova posizione è ((5 + 3 + 3*9) mod 23) = 12, libera. Quattro accessi.