**008AA – ALGORITMICA E LABORATORIO**

Verifica del 25 maggio 2009

**Cognome Nome:** **N. Matricola:** **Corso: A B**

**Esercizio 1.** *(4+4+4 punti)* Dato un insieme $S$ di $n$ interi distinti, il mediano è l'intero $k$ tale che $\lfloor n/2 \rfloor$ interi in $S$ sono minori di $k$. Per esempio, il mediano per $S = \{1, 3, 9, 12, 15, 18\}$ è $k = 12$. Dato un albero binario di ricerca $T$ che memorizza gli interi di $S$ nei suoi nodi, definiamo il *nodo mediano* di $T$ come il nodo che contiene il mediano di $S$.

(a) Progettare un algoritmo che, preso in ingresso un albero binario di ricerca $T$ e la sua dimensione $n$, restituisca un puntatore al suo nodo mediano. Discutere la complessità dell'algoritmo proposto.

Soluzione 1 in tempo $O(n)$:

```
/* pmediano e conta sono variabili globali */

MedRec( u ){
  if (u != NULL){
    MedRec( u->sx );
    if (conta-- == 0) pmediano = u;
    MedRec( u->dx );
  }
}

Mediano( radice, dim ){
  pmediano = NULL;
  conta = dim/2;
  if (radice != NULL)
    MedRec( radice );
  return pmediano;
}
```

Soluzione 2 in tempo $O(n)$:

```
Mediano( radice, dim )
    <u,x> = MedRec( radice, dim, 0 );
    return u; // sicuramente x == -1

/*
   Restituisce coppia <nodo, dimS> dove dimS e' la dimensione del sottoalbero di "nodo".
   Se dimS == -1, allora "nodo" e' il mediano.
   numPred = #elementi minori di u.chiave gia' incontrati nella visita simmetrica di T
*/
MedRec( u, n, numPred )
  if (u == NULL) return <NULL, 0>;
  <u', n'> = MedRec( u->sx, n, numPred );
  if (n' == -1) return <u', -1>; // trovato il mediano a sinistra
  if (numPred+n' == n/2) return <u, -1>; // u e' il mediano
  <u'', n''> = MedRec( u->dx, n, numPred + n' + 1 );
  if (n'' == -1) return <u'', -1>; // trovato il mediano a destra
  return <NULL, n' + n'' + 1>;
```

(b) Progettare un algoritmo che, dato un nodo $u \in T$, trovi il suo successore in tempo $O(h)$, dove $h$ è l'altezza dell'albero binario $T$. Il successore di $u$ è quel nodo che contiene la chiave che è la minima tra quelle di $T$ che sono maggiori della chiave contenuta in $u$. Oltre ai campi `sx` e `dx`, è possibile usare il campo `padre` per accedere al padre del nodo corrente.

Soluzione in tempo $O(h)$:

```
Successore( u ){  // hyp: u != NULL
  if (u->dx != NULL) {
    s = u->dx;
    while (s->sx != NULL)
      s = s->sx;
  } else {
    s = u->padre;
    while (s != NULL && u == s->dx) {
      u = s;
      s = s->padre;
    }
  }
  return s;
}
```

(c) Progettare un algoritmo che, presi in ingresso un albero binario di ricerca $T$ (non vuoto), la sua dimensione $n$, una chiave $k'$ e un puntatore al nodo mediano di $T$, inserisca $k'$ in $T$ e restituisca il puntatore al (possibilmente nuovo) nodo mediano di $T$. Il tempo di esecuzione deve essere $O(h)$, dove $h$ è l'altezza di $T$. È possibile ipotizzare che le chiavi siano tutte distinte e che siano disponibili le seguenti funzioni:

- `Inserisci(e)` per l'inserimento di una chiave $e$ in $T$ in tempo $O(h)$;
- `Successore(u)` per recuperare il puntatore al successore di $u$ in tempo $O(h)$;
- `Predecessore(u)` per recuperare il puntatore al predecessore di $u$ in tempo $O(h)$.

Soluzione 1 in tempo $O(h)$:

```
/* variabili globali ausiliarie */
pmediano = Mediano ( T, n );
rmediano = n/2 + 1; // dove il minimo elemento ha rango = 1

1. Inserisci( k' );
2. n = n + 1;
3. if ( k' < pmediano->dato->chiave ) rmediano++;
4. if ( rmediano < n/2 + 1)
     { pmediano = Successore( pmediano ); rmediano++; }
   else if ( rmediano > n/2 + 1)
     { pmediano = Predecessore( pmediano ); rmediano--; }
```

Soluzione 2 in tempo $O(h)$:

```
Inserisci(k',radice, mediano, n) {
    numMinori = n/2; // rango del mediano attuale
    Inserisci( k', radice );
    n++;
    if ( k' < mediano->dato->chiave ) numMinori++; // nuovo rango
    if ( numMinori < n/2 )
       return Successore( mediano );
    else if ( numMinori > n/2 )
       return Predecessore( mediano );
    else return mediano;
}
```

**Cognome Nome:** **N.Matr:**

**Esercizio 2.** *(3+3+3+4 punti)* Si vuole progettare un algoritmo che, ricevuto in ingresso un intero $k > 0$ e uno heap $H$ (per il minimo) contenente $n > 2^k$ interi distinti, stampi i $k$ elementi più piccoli di $H$:

(a) descrivere un algoritmo che richiede $O(k \log n)$ tempo;

Soluzione:

```
for ( i = 0; i < k; i++ ) {
   x = Dequeue( H );
   printf( x );
}
```

(b) fornire un esempio di heap per $n = 9$ e $k = 3$, in cui le tre chiavi più piccole occupano i primi tre livelli dello heap;

Soluzione:

```
H = 1 2 5 3 4 6 7 8 9
```

(c) descrivere un algoritmo che richiede $O(2^k k)$ tempo;

Soluzione: Nel caso pessimo i k elementi piu' piccoli si trovano lungo un cammino di lunghezza k che parte dalla radice di H (come nel punto b). Quindi si applica Heapsort ai primi $2^k$ elementi di H, e quindi richiedere $O(2^k \log(2^k)) = O(k2^k)$ tempo.

(d) descrivere un algoritmo che richiede $O(2^k \log k)$ tempo, raffinando la soluzione al punto (c).

Soluzione: scandiamo i primi $2^k$ elementi mantenendo i $k$ minimi.

```
crea heap-max Q vuoto
for ( i = 0; i < k; i++ )
   Enqueue( Q, H[i] );
for (i = k; i < 2^k; i++ )
   if ( H[i] < First( Q ) )
      { Dequeue( Q ); Enqueue( Q, H[i] );
for ( i = 0; i < k; i++ ) {
   x = Dequeue( Q );
   printf( x );
}
```

Esiste anche una soluzione di costo $O(k \log k)$ che mantiene in uno heap di minimo una frontiera di al massimo $2k$ nodi. Inizialmente l'heap contiene la radice, e a ogni passo si estrae il minimo e si inseriscono in esso i suoi due figli. $k$ passi sono sufficienti!

**Cognome Nome:** **N.Matr:**

**Esercizio 3.** *(7 punti)* A partire da un albero AVL vuoto, simulare l'inserimento delle chiavi 80, 40, 60, 70, 65, 85, disegnando l'albero prima e dopo ciascuna rotazione. Per ciascuna rotazione effettuata, indicarne il tipo (SS, SD, DS, DD) e il nodo critico corrispondente.

Soluzione:
Legenda del nodo: [chiave] (h = altezza, f = fattore di bilanciamento). La struttura dell'albero è ruotata di 90 gradi in senso antiorario.

```
-> Inserimento di: 80

  [80] (h=0, f=0)


**************************************
-> Inserimento di: 40

  [80] (h=1, f=1)
                \ [40] (h=0, f=0)


**************************************
-> Inserimento di: 60

  [80] (h=1, f=2)
                                / [60] (h=0, f=0)
                \ [40] (h=1, f=-1)

Nodo critico : 80 Rotazione: SD

                / [80] (h=0, f=0)
  [60] (h=1, f=0)
                \ [40] (h=0, f=0)


**************************************
-> Inserimento di: 70

                / [80] (h=1, f=1)
                                \ [70] (h=0, f=0)
  [60] (h=2, f=-1)
                \ [40] (h=0, f=0)


**************************************
-> Inserimento di: 65

                / [80] (h=1, f=2)
                                \ [70] (h=1, f=1)
                                                \ [65] (h=0, f=0)
  [60] (h=2, f=-1)
                \ [40] (h=0, f=0)

Nodo critico : 80 Rotazione: SS

                                / [80] (h=0, f=0)
                / [70] (h=1, f=0)
                                \ [65] (h=0, f=0)
  [60] (h=2, f=-1)
                \ [40] (h=0, f=0)


**************************************
-> Inserimento di: 85

                                                / [85] (h=0, f=0)
                                / [80] (h=1, f=-1)
                / [70] (h=2, f=-1)
                                \ [65] (h=0, f=0)
  [60] (h=2, f=-2)
                \ [40] (h=0, f=0)

Nodo critico : 60 Rotazione: DD

                                / [85] (h=0, f=0)
                / [80] (h=1, f=-1)
  [70] (h=2, f=0)
                                / [65] (h=0, f=0)
                \ [60] (h=1, f=0)
                                \ [40] (h=0, f=0)
```