ALGORITMICA Verifica del 4 Novembre 2005

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

Esercizio 1. *(10 punti)* Sia data una lista *ordinata* L contenente n interi.

1. Scrivere lo pseudocodice per la ricerca di una chiave k nella lista L, valutandone il costo in tempo al caso pessimo.

2. Supponendo di avere a dispozione un ulteriore campo `u.extra` per memorizzare un riferimento dal nodo corrente `u` a un altro nodo della lista (non necessariamente il successore o il predecessore di `u`), descrivere un metodo di assegnare ai campi `u.extra` i riferimenti a nodi strategici in L, in modo che la ricerca di una chiave richieda tempo sublineare nella lista L così estesa (ovvero abbia un costo temporale $o(n)$ strettamente inferiore a n, ignorando le costanti moltiplicative). Descrivere l'algoritmo di ricerca risultante in tal modo, motivando la diminuizione di costo in tempo.

Soluzione 1.1:

```
cerca( k )
u = inizio lista;
while ( u != null && u.info < k )
   u = u.next;
if (u == null || u.info != k )
   return null;
else
   return u;
```

Soluzione 1.2: Ogni riferimento `u.extra` punta a $\sqrt{n}$ nodi in avanti nella lista (e gli ultimi $\sqrt{n}$ nodi della lista puntano alla fine della lista; in particolare, l'ultimo nodo punta a se stesso, per cui `u.extra` non è mai `null`).

```
cerca-veloce( k )
u = inizio lista;
while ( u != null ){
   if (u.extra.info < k )
      u = u.extra;
   else
      u = u.next;
}
if (u == null || u.info != k )
   return null;
else
   return u;
```

La ricerca procede per salti di $\sqrt{n}$ nodi finché può (`u.extra.info` $>$ `k`). In totale esegue $O(\sqrt{n})$ salti in questo modo. Quando la condizione non è più verificata, rimangono al più $\sqrt{n}$ elementi della lista da scandire con `u.next`, a un costo pari a $O(\sqrt{n})$ tempo. Quindi, il tempo totale di ricerca è $O(\sqrt{n}) = o(n)$.

ALGORITMICA Verifica del 4 Novembre 2005

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

Esercizio 2. *(8 punti)* Dato un array A di n campi distinti, progettare un algoritmo che costruisca ricorsivamente, e in tempo $O(n)$, un albero binario *bilanciato* T tale che $A[i]$ sia il campo `u.info` dell'$(i+1)$-esimo nodo `u` in ordine di visita *anticipata* di T (dove $0 \leq i \leq n-1$).

Soluzione: Sia `new` una primitiva per creare un nuovo nodo. La funzione `crea` va invocata con argomenti $i = 0$ e $j = n-1$. Usa l'idea che la radice contiene l'elemento $A[i]$, il sottoalbero sinistro è ricorsivamente costruito con $A[i+1, m]$ e quello destro con $A[m+1, j]$, dove m è la posizione mediana di $A[i+1, j]$.

```
crea( i , j ):
   if ( i > j ) return null;
   m = ((i+1)+j)/2;
   u = new();
   u.info = A[i];
   u.sx = crea( i + 1, m );
   u.dx = crea( m + 1, j );
   return u;
```

ALGORITMICA Verifica del 4 Novembre 2005

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

Esercizio 3. *(7 punti)* Si consideri la funzione `eulero` utilizzata per stampare le profondità dei nodi di un albero binario, percorrendo il suo ciclo Euleriano:

```
eulero( u, p ){
   if (u == null) return;
   print p;
   if (u.sx != null) {
      eulero( u.sx, p+1 );
      print p;
   }
   if (u.dx != null) {
      eulero( u.dx, p+1 );
      print p;
   }
}
```

Descrivere come modificare il codice suddetto per stampare le profondità dei nodi di un albero *ordinale* lungo il suo ciclo Euleriano: a tal fine, usare la memorizzazione binarizzata per tale albero (i riferimenti nei campi `u.sx` e `u.fx`).

Soluzione:

```
eulero( u, p ){
   if (u == null) return;
   print p;
   z = u.sx;
   while ( z != null )
      eulero( z, p+1 );
      print p;
      z = z.fx;
   }
}
```

ALGORITMICA Verifica del 4 Novembre 2005

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

Esercizio 4. *(7 punti)* Si dimostri una delle due proprietà a scelta:

- La ricerca binaria richiede $\Omega(\log n)$ confronti.
- Un albero binario completamente bilanciato di altezza h ha $2^h - 1$ nodi interni e 2^h foglie (per induzione sull'altezza h).

Soluzione:

- Per il limite inferiore sulla ricerca binaria, si vedano le dispense.
- Per induzione su h. Caso base $h = 0$, abbiamo un solo nodo, che è sia radice che foglia, e zero nodi interni. Passo induttivo: indichiamo con t_h l'albero binario completamente bilanciato di altezza h, e supponiamo che t_h abbia $2^h - 1$ nodi interni e 2^h foglie. Prendiamo t_{h+1}, ovvero l'albero completamente bilanciato di altezza $h + 1$: può essere visto come t_h in cui ciascuna foglia diventa un nodo interno dando luogo a due foglie di profondità $h + 1$. Ne deriva che t_{h+1} è ottenibile da t_h aggiungendo 2^{h+1} foglie. Il numero di nodi interni di t_{h+1} è il numero totale di nodi di t_h, ovvero $2^h + 2^h - 1 = 2^{h+1} - 1$. Il passo induttivo è quindi dimostrato per $h + 1$.