Esercizio 2 **[punti 12]**

Sia dato un albero binario i cui nodi sono colorati di Rosso, Nero o Giallo. Progettare un algoritmo ricorsivo che conta il numero di sottoalberi i cui nodi sono colorati soltanto con due tipi di colori, e questi appaiono in ugual numero. Analizzarne anche la complessità al caso pessimo.

La procedura verrà invocata sulla radice dell'albero di cui si vuole effettuare il conteggio suddetto. La procedura invocata sul nodo u restituisce una quadrupla <c,nrossi,nneri,ngialli> dove *c* = numero di sottoalberi che discendono da u e soddisfano la proprietà richiesta, *nrossi* = numero di nodi che discendono da u e sono colorati di rosso, *nneri* = numero di nodi che discendono da u e sono colorati di nero, *ngialli* = numero di nodi che discendono da u e sono colorati di giallo. La complessità è O(n) trattandosi di una semplice visita posticipata. Nell'algoritmo useremo [*expr*] per indicare un valore 0 o 1 a seconda che l'espressione booleana *expr* sia falsa o vera, rispettivamente.

```
Conta_sottoalberi(u)
  if (u==NULL) return <0,0,0,0>;

  <c_left,nr_left,nn_left,ng_left> = Conta_sottoalberi(u.left);
  <c_right,nr_right,nn_right,ng_right> = Conta_sottoalberi(u.right);

  nrossi = nr_left + nr_right + [u.color == rosso];
  nneri = nn_left + nn_right + [u.color == nero];
  ngialli = ng_left + ng_right + [u.color == giallo];
  c = c_left + c_right;

  num_col_distinti = (ngialli>0) + (nneri > 0) + (nrossi > 0);
  if ((num_col_distinti == 2) &&
       ((ngialli==nneri) || (ngialli == nrossi) || (nrossi=nneri))
      ) { c++; }
  return <c,nrossi,nneri,ngialli>;
```

Algoritmica

Appello 27/06/2005

Esercizio 1 **[punti 10]**

E' dato un albero binario T in cui ciascun nodo è colorato di rosso oppure di nero. Ogni nodo è composto da tre attributi: oltre a *left* e *right* che puntano al figlio sinistro e al figlio destro, rispettivamente, esiste l'attributo *color* che memorizza il colore del nodo. Per un arbitrario intero $k > 0$, si vuole verificare se esiste un cammino in T, dalla radice a un nodo interno di profondità k, in cui tutti i nodi attraversati sono colorati di rosso. (La profondità della radice è 0.)

1. Descrivere a parole la struttura dell'algoritmo di risoluzione.
2. Dare una realizzazione dell'algoritmo in pseudocodice.
3. Valutare la complessità dell'algoritmo spiegando il risultato indicato.

Soluzione

```
Red_path(u,k,prof)
        if( (u==NULL) || (u.color==nero) ) return FALSE;
        if(prof == k) return TRUE;
        x= Red_path(u.left,k,prof+1);
        y= Red_path(u.right,k,prof+1);
        return(x || y);
```

L'algoritmo viene chiamato ricorsivamente sia sul figlio destro che su quello sinistro di ogni nodo u, posto che il nodo corrente u sia rosso, altrimenti
la visita si interrompe siccome il cammino discendente non risolverà sicuramente il problema. L'algoritmo segue lo schema della visita anticipata e quindi la sua complessità in tempo al caso pessimo è O(|T|). I parametri in input sono il nodo corrente da visitare *u*, la sua profondità *prof*, e la profondità k indicata nel problema. L'algoritmo mantiene la seguente induzione: il cammino che congiunge la radice al padre di u è tutto rosso.

ALGORITMI E STRUTTURE DATI - II modulo

Terza verifica – 11 aprile 2001

COGNOME: **NOME:**

Esercizio 1 *(14 punti)*
Sia A un albero binario. Progettare un algoritmo efficiente che cancella il figlio sinistro di ogni nodo se è una foglia e ha la stessa chiave memorizzata nel nodo padre.

1. Descrivere a parole la struttura dell'algoritmo.
2. Dare un programma in pseudocodice per l'algoritmo proposto.
3. Discutere la complessità dell'algoritmo proposto.

```
Cancella_fsf (r)
  if r = NiL then return;
  else
    if ((foglia (left[r]) and (key[left[r]] = = key[r]))
    then Tree_delete (left[r]);
    cancella_fsf (left[r]);
    cancella_fsf (right[r]);
```

ORDINE IMPORTANTE

```
foglia (u)
  if ((u ≠ NiL) and (left[u] = NiL) and (right[u] = NiL))
    then return True
    else return false
```

ALGORITMICA Verifica del 16 Maggio 2002

COGNOME **NOME** **CORSO**

Esercizio 3. *(10 punti)* È dato un albero binario T in cui ciascun nodo è colorato di rosso oppure di nero. Ogni nodo, oltre agli attributi LEFT e RIGHT che puntano al figlio sinistro e al figlio destro, ha l'attributo COLOR che memorizza il colore del nodo (e non ci sono altri attributi disponibili nel nodo). Per un arbitrario intero k, si vuole verificare se esiste un sottoalbero in T di k nodi tutti colorati di rosso.

1. Descrivere a parole la struttura dell'algoritmo di risoluzione.
2. Dare una realizzazione dell'algoritmo in pseudocodice.
3. Valutare la complessità dell'algoritmo spiegando il risultato indicato.

Siano date le seguenti costanti:

```
DISCENDENTE_NERO = -1;
TROVATO_SOTT = -2;
```

La procedura che sqve restituisce un intero che può essere:

1) DISCENDENTE_NERO → se da u discende un nodo nero;
2) TROVATO_SOTT → se da u discende un sottoalbero che soddisfa le condizioni
3) intero positivo → sottoalbero radicato in u è tutto rosso e ha la dimensione specificata da tale intero.

La procedura verrà invocata come:

Sottoalbero_rosso (T.root, k)

```
Sottoalbero_nero (u, K)
    if (u== NULL) return 0;
    l = Sottoalbero_nero (u.left, K);
    r = Sottoalbero_nero (u.right, K);
    if ((l = TROVATO_SOTT) || (r== TROVATO_SOTT)) return (TROVATO_SOTT);
    if ((u.color== NERO) || (l==DISC_NERO) || (r== DISC_NERO))
                                        return (DISC_NERO);

    if ((l+r+1) == K) return (TROVATO_SOTT);
    return (l+r+1);
```

la complessità in tempo è $O(n)$.

Una soluzione semplicissima ma di complessità al caso peggiore $= O(n^2)$ consisterà nell'utilizzare le procedure

size (u) = calcola la dimensione del sottoalbero radicato in u

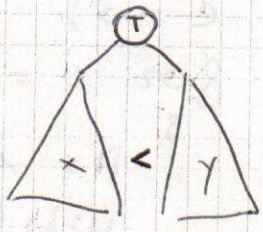
nero (u) = restituisce TRUE se tutti i nodi che discendono da u sono neri.

* Eseguendo il check: (size(u) == k) && (nero(u)) su ogni nodo u si potrà risolvere il problema con una semplice visita.

24/11/2003

Esercizio 1 Siano x e y due chiavi in un albero binario di ricerca con $x < y$. Trovare la distanza (= n° archi) tra x e y in T

```
Distanza (T, x, y)
{ LCA = TroveLCA (T, x, y);        // T.root
  d1 = Conte (LCA, x);
  d2 = Conte (LCA, y);
  return d1 + d2;
}
```




```
TroveLCA (π, x, y)    // (x < y)
{ if (π.key ≥ x)
        Then if (π.key ≤ y)
                Then return π;
                else return TroveLCA (π.left, x, y);
        else return TroveLCA (π.right, x, y);
}
```

```
Conte (π, z)
{ if (π.key == z) return 0;
  if (π.key > z)
        Then return (1 + Conte (π.left, z));
        else return (1 + Conte (π.right, z));
}
```

Costo in Tempo $O(h)$

ALGORITMICA Appello del 10 Febbraio 2006

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

Esercizio B1. *(15 punti)* Dato un albero binario T in cui ogni nodo contiene un intero, diciamo che la *media* di un suo sottoalbero non vuoto è la somma degli interi contenuti nei suoi nodi diviso il numero di tali nodi. Scrivere lo pseudocodice di un algoritmo ricorsivo che, preso in ingresso T, ne identifica il sottoalbero la cui media è massima rispetto a quella degli altri sottoalberi.

```
<somma, nnodi, max> ←

Media (u)
{
    if (u == NULL) return <0, 0, -∞>

    <sl, nl, ml> = Media (u.left);
    <sr, nr, mr> = Media (u.right);

    somma = sl + sr + u.key;
    nnodi = nl + nr + 1;
    [illegible] > max (ml, mr)
    nnodi

    return <somma, nnodi, max (ml, mr, somma/nnodi)>;
```

Esercizio 2 **[punti 10]**

Si vuole inserire la sequenza di chiavi intere 3, 20, 22, 116, 23, 70, 29, 51, 32, 12, 31, 28 in una tabella hash inizialmente vuota di dimensione m = 16, usando l'indirizzamento aperto con scansione quadratica mediante la seguente funzione hash definita su ciascuna chiave k:

$$h(k, i) = (h'(k) + 2 * i + i^2) \bmod m$$

dove i = 0, 1, ..., m-1, e h'(k) = k mod m. 16

Stabilire se h(k,i) genera una *sequenza di probing* che inserisce correttamente nella tabella hash tutte le chiavi su indicate. Mostrare inoltre l'esecuzione delle inserzioni, elencando per ciascuna chiave k le posizioni h(k,i) esaminate nella tabella hash durante la sua scansione quadratica.

Chiave k	Posizioni h(k,i) esaminate nella tabella hash
3	3
20	4
22	6
116	4, 7
23	7, 10
70	6, 9
29	13
51	3, 6, 11
32	0
12	12
31	15
28	12, 15, 4, 11, 4, 15, 12, 11, 12, 15, 4, 11, 4, 15, 12, 11

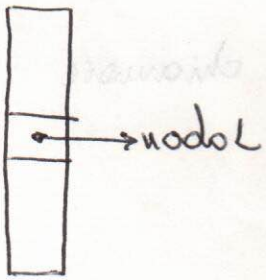
Tabella hash, indirizzamento aperto,

②

È data una tabella hash di dim = m, collisioni gestite mediante liste di traboccio e funzione hash basata sul metodo della divisione.

Scrivere una procedura di ricerca che passi le chiave ricercate in testa alla sua lista di traboccio.

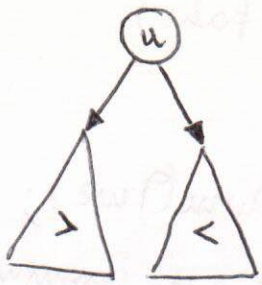
```
nodoL {
  int Key;
  nodoL prev, succ;
  }
```



```
mtf_hash (T, k)
    i = k mod m;
    x = T[i];                          && (x ≠ NULL))
    while ((x.key ≠ k) \/ x = x.succ
    if ((x ≠ NULL) && (x ≠ T[i])) {
            y = x.prev;        /* esiste */
            z = x.succ;        /* forse */
            y.succ = z;
            if (z ≠ NULL)  z.prev = y;
            x.succ = T[i];  x.prev = NULL;
            T[i] = x;
            }
    return x;
```

potremmo chiamare $t = T[i]$ per la testa.

Progettare un algoritmo che "inverte" un albero binario di ricerca.



chiavi sono Tutte distinte

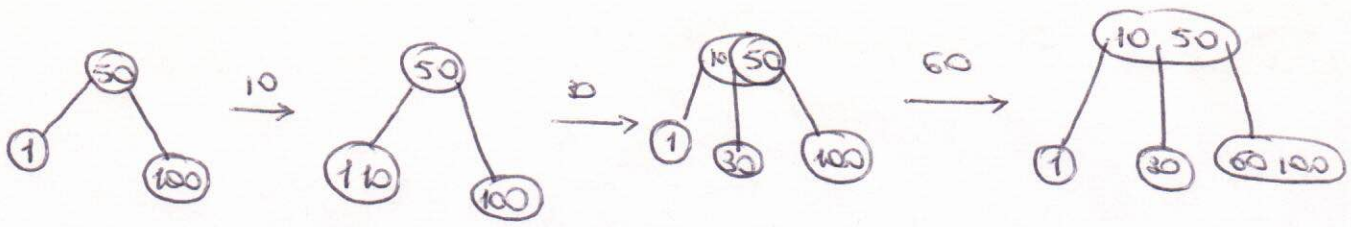
⇒ inversione agisce sui figli ⇒ puntatori
⇒ post-order

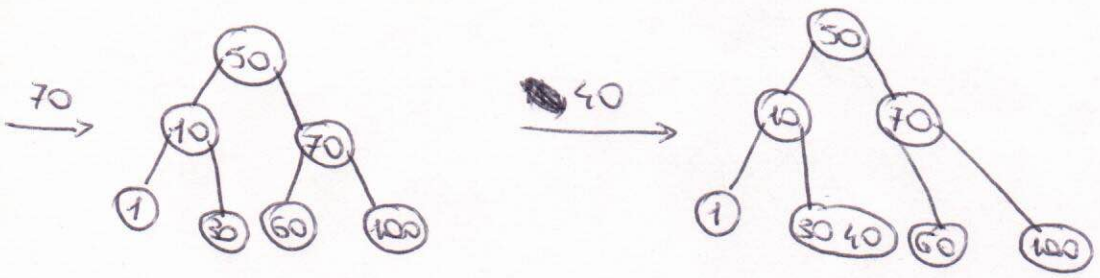
```
Inverti (u)
  if (u ≠ NULL) {
    Inverti (u.left);
    Inverti (u.right);
    t = u.left;
    u.left = u.right;
    u.right = t;
    }
```

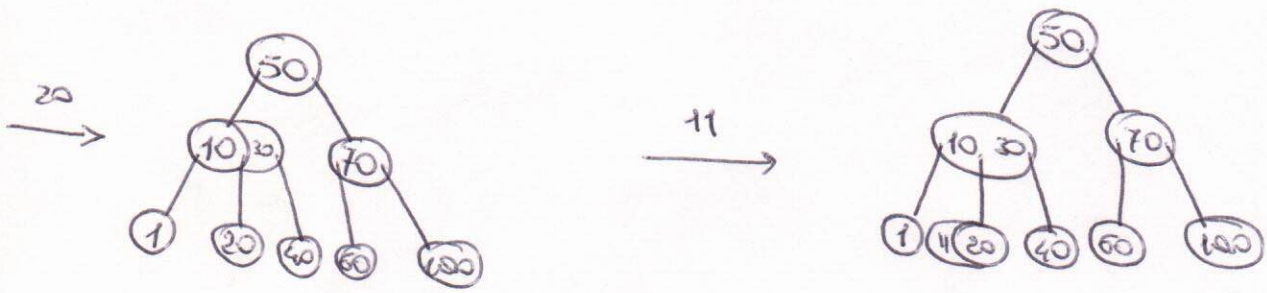
costo = O(n)

Esercizio 6 2-3

- Inserire le chiavi : 1, 50, 100, 10, 30, 60, 70, 40, 20, 11







- cancellare 10 ⇒ rotazione

- cancellare 60