Esercizio 2

ABR: Visita

Scrivere un programma che legga da tastiera una sequenza di N interi distinti e li inserisca in un albero binario di ricerca (senza ribilanciamento).
Il programma deve visitare opportunamente l'albero e restituire la sua altezza.

Esercizio 2

```
typedef struct _node {
  struct _node *left;
  struct _node *right;
  int value;
} node;

int maxdepth(node *n) {
  if (n == NULL) return 0;
  return 1 + max( maxdepth(n->left), maxdepth(n->right) );
}
```

Esercizio 5

Albero Ternario (prova del 01/02/2012)

Scrivere un programma che riceva in input una sequenza di N interi positivi e costruisca un albero **ternario** di ricerca **non** bilanciato. L'ordine di inserimento dei valori nell'albero deve coincidere con quello della sequenza.

Ogni nodo in un albero ternario di ricerca può avere fino a tre figli: figlio sinistro, figlio centrale e figlio destro. L'inserimento di un nuovo valore avviene partendo dalla radice dell'albero e utilizzando la seguente regola. Il valore da inserire viene confrontato con la chiave del nodo corrente. Ci sono tre possibili casi in base al risultato del confronto:

1. se il valore è minore della chiave del nodo corrente, esso viene inserito ricorsivamente nel sottoalbero radicato nel figlio sinistro;

2. se il valore è **divisibile** per la chiave del nodo corrente, esso viene inserito ricorsivamente nel sottoalbero radicato nel figlio centrale;

3. in ogni altro caso il valore viene inserito ricorsivamente nel sottoalbero radicato nel figlio destro.

Il programma deve stampare il numero di nodi dell'albero che hanno **tre** figli.

Esercizio 5

```
typedef struct _Nodo {
  int key;
  struct _Nodo* left;
  struct _Nodo* central;
  struct _Nodo* right;
} Nodo;
```

Esercizio 5

```
void insert(Nodo **t, int k) {
  Nodo *e, *p, *x;

  /* crea il nodo foglia da inserire contenente la chiave */
 e = (Nodo *) malloc(sizeof(Nodo));
 if (e == NULL) exit(-1);
 e->key = k;
 e->left = e->right = e->central = NULL;
 x = *t;

 /* se l'albero è vuoto imposta e come radice dell'albero */
 if (x == NULL) {
     *t = e;
     return;
}

continua ...
```

Esercizio 5

```
 /* altrimenti cerca la posizione della foglia nell'albero */
 while (x != NULL) {
   p = x;
   if(k % x->key == 0) x = x->central;
   else {
     if (k < x->key) x = x->left;
     else x = x->right;
   }
 }

 /* ora p punta al padre del nuovo elemento da inserire in t quindi si procede a
collegare p ed e */
 if(k % p->key == 0) p->central = e;
 else {
   if (k < p->key) p->left = e;
   else p->right = e;
 }
```

Esercizio 5

```
int conta(Nodo * node) {
  int r, c, l, curr;

  if (node == NULL) return 0;
  r = c = l = curr = 0;

  if (node->right != NULL) { r = conta(node->right); curr++;}
  if (node->left != NULL) { l = conta(node->left); curr++;}
  if (node->central != NULL) { c = conta(node->central); curr++;}

  if (curr == 3) return r+l+c+1;
  else return r+l+c;
}
```