Prova_parziale 2013_01_28 soluzione

1. Dare la definizione di albero binario **completo**.
   Scrivere in C un programma **efficiente** per stabilire se un albero binario è **completo** e calcolarne la complessità al caso pessimo indicando, e risolvendo, la corrispondente relazione di ricorrenza.

Definizione di albero binario completo:

- Un albero binario completo consiste in un albero in cui ogni nodo ha al più due figli, ed ogni livello risulti riempito completamente, ad eccezione al massimo dell'ultimo. Tutti i nodi dell'ultimo livello inoltre devono essere posizionati più a sinistra possibile. Quest'ultima proprietà accerta che la struttura sia bilanciata

Bisogna ora definire prima la struct del nodo

```
struct Node {
        int key;
        Node* left;
        Node* right;
}

typedef Node* Pnode;
```

Immaginiamo ora di avere già la classe per l'albero da analizzare **tree**, e che la funziona **tree.getRoot()** ritorni il nodo radice. La funzione si chiama **tCheckComplete(Pnode u)** e ritorna 1 o 0 in base all'esito della funzione stessa.

```
int numNodeCalculator(Pnode u){
        if(u == nullptr) return 0; // ho finito di scorrere

        // cerco sia a destra che a sinistra
        int left = heightCalculator(u->left);
        int right = heightCalculator(u->right);

        // ritorno la somma dei nodi sotto il nodo in cui mi trovo
        return 1 + left + right;
}

int tCheckCompleteAux(Pnode u, int index, int tot){
        if(u == nullptr) return true; // controllo se ho finito il ramo da
scorrere
        if(index >= tot) return false; // controllo se sono out-of-index

        return tCheckCompleteAux(u->left,2*index+1,tot) &&
tCheckCompleteAux(u->right, 2*index+2, tot);
```

```
}

int tCheckComplete(Pnode u){
        // caso base == albero vuoto
        if(u == nullptr){
                return 1;
        }

        // calcolo il numero di nodi totale
        int tot = numNodeCalculator(u);

        int index = 0; // we start from root with value 0

        // Il fulcro dell'algoritmo, si basa tutto sull'index!!
        return tCheckCompleteAux(u->left,2*index+1,tot) &&
tCheckCompleteAux(u->right, 2*index+2, tot);
}
```

Nel codice sopra riportato la complessità nel caso peggiore risulta essere **O(n)** per il semplice fatto che **devo scorrere tutti i nodi dell'albero**, cosa che faccio sempre per esempio nel caso in cui l'albero è effettivamente un albero binario completo.