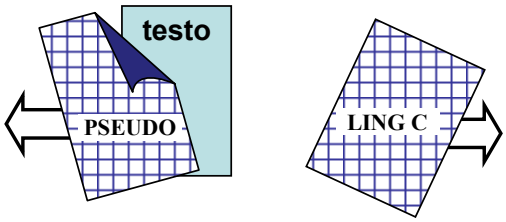
Algoritmi e Strutture di Dati – A.A. 2016-2017
Prova scritta del 4 settembre 2017 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

☐ Note (es: Ho bisogno di una correzione veloce in quanto…) ……
……

Cognome: _____ **Nome:** _____ **Matricola:** _____



- **CONSEGNA PSEUDOCODIFICA E LINGUAGGIO C SU DUE FOGLI PROTOCOLLO SEPARATI**
- **METTI IL TESTO DENTRO LA PARTE DI PSEUDOCODIFICA**
- **PUOI SCRIVERE (E CONSEGNARE) A MATITA**

PSEUDOCODIFICA

Negli esercizi seguenti un grafo non orientato è rappresentato con un array A in cui ogni elemento A[*u*] è un riferimento al primo elemento della lista di adiacenza doppiamente concatenata del nodo *u* (con i campi `prev`, `info` e `next`). Essendo il grafo non orientato esiste un arco (*u*,*v*) per ogni arco (*v*,*u*).

Esercizio 1

Scrivi lo pseudocodice della procedura **CONNESSE-DIVERSE**(A) che accetti in input un grafo non orientato A e ritorni true se tutte le componenti connesse del grafo hanno un numero di nodi differente. Se il grafo ha una sola componente connessa, oppure se è vuoto, la funzione ritorna `true`. Se il grafo ha almeno due componenti connesse con lo stesso numero di nodi la funzione ritorna `false`.

Esercizio 2

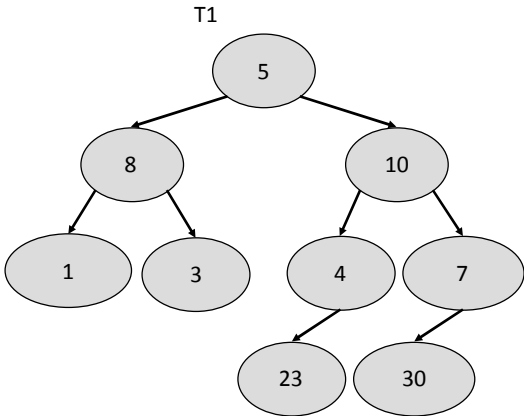
Discuti la complessità computazionale nel caso peggiore (in termini di O-grande, Omega e Theta) della seguente procedura in funzione del numero n di elementi dell'albero. Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA fa un numero di operazioni proporzionali alla lunghezza della lista corrente.

```
FUNZIONE(T)       /* T è un albero binario di interi */

L.head = NULL     /* L è una nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L,0)
return L

FUNZ-RIC(v,L,depth)
 if(v==NULL) return
 if(depth > 1000)
      AGGIUNGI-IN-CODA(L,v.info)
 else
      AGGIUNGI-IN-TESTA(L,v.info)
 FUNZ-RIC(v.left,L,depth+1)
 FUNZ-RIC(v.right,L,depth+1)
```

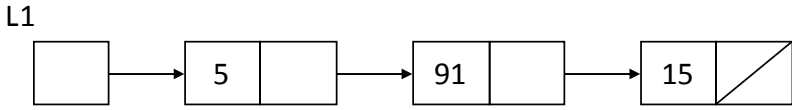
LINGUAGGIO C

Si consideri la libreria **numeri.h** che implementa quanto segue:

```
typedef struct elem {
      int info;
      struct elem* sx;
      struct elem* dx;
   } nodo_albero;
typedef nodo_albero* BTree;
```




```
typedef struct elem2 {
      int info;
      struct elem2* next;
   } nodo_lista;
typedef nodo_lista* pList;
```




```
int esiste(pList L, int x);
int lunghezza(pList L);
int altezza(BTree T);
```

Tale libreria implementa alberi binari i cui nodi contengono numeri interi nel tipo **BTree**, e liste semplicemente concatenate i cui nodi contengono numeri interi nel tipo **pList**. Il metodo **esiste** restituisce 1 se esiste nella lista **L** un nodo contenente il valore **x**, 0 altrimenti. Il metodo **lunghezza** restituisce la lunghezza (numero di nodi) della lista **L** (se la lista è vuota ritorna 0). Il metodo **altezza** restituisce l'altezza dell'albero T (restituisce -1 se l'albero è vuoto).

Utilizzando la libreria **numeri.h**, si richiede di implementare in linguaggio C i seguenti metodi:

1) `int esiste_somma(BTree T, pList L)`, che dato un albero binario di interi T e una lista L, restituisca 1 se esiste nella lista L un nodo contenete la somma dei valori contenuti nei nodi dell'albero T, 0 altrimenti. Ad esempio `esiste_somma (T1, L1)` restituirà il valore `1` (la somma dei valori contenuti nei nodi di T1 è 91 e nella lista L1 esiste un nodo contenete 91).
2) `int verifica_somma_foglie (BTree T, pList L)`, che dato un albero binario `T` e una lista L, restituisca 1 se la somma dei valori contenuti nelle foglie in T è un valore contenuto in un nodo della lista L, 0 altrimenti. Ad esempio `verifica_somma_foglie (T1, L1)` restituirà il valore `0` (nell'albero `T1` la somma dei valori contenuti nelle foglie è 57 e non esiste in L un nodo contenente il valore 57).
3) `int verifica_somma_livello (Btree T, int x, pList L)` che dato un albero binario di interi `T` un intero x e una lista `L`, restituisca 1 se esiste in L un nodo contenete un valore corrispondente alla somma dei valori contenuti nei nodi posizionati al livello x in T, 0 altrimenti. Ad esempio `verifica_somma_livello(T1, 2, L1)` restituirà il valore `1;` nell'albero `T1` la somma dei valori contenuti nei nodi posizionati al livello 2 è 15, valore contenuto nel terzo nodo della lista L1.

È possibile utilizzare qualsiasi libreria nota e implementare qualsiasi metodo di supporto a quelli richiesti.