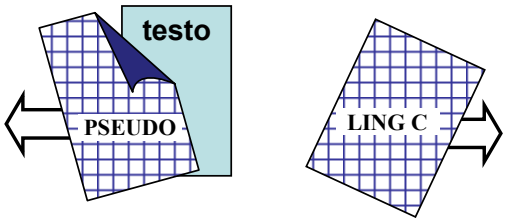
Algoritmi e Strutture di Dati – A.A. 2016-2017
Prova scritta del 12 luglio 2017 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

☐ 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-UGUALI**(A) che accetti in input un grafo non orientato A e ritorni true se tutte le componenti connesse del grafo hanno lo stesso numero di nodi. 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 un diverso numero di nodi la funzione ritorna `false`.

Esercizio 2

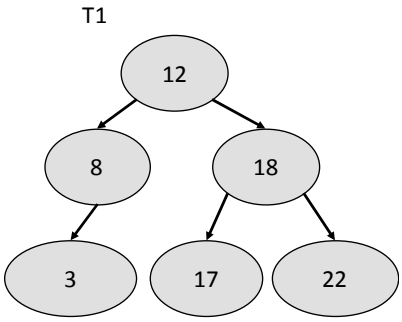
Discuti la complessità computazionale nel caso peggiore (in termini di O-grande, Omega e Theta) della seguente procedura in funzione del numero n di elementi dell'albero. Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA fa un numero di operazioni proporzionali 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 == 3)
      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)
```

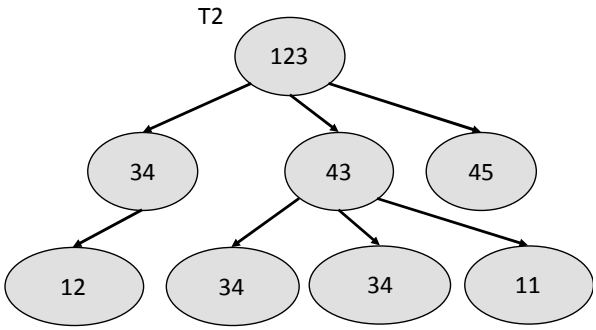
LINGUAGGIO C

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

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




```
typedef struct elem2 {
            int data;
            struct elem2* primofiglio;
            struct elem2* fratello;
   } nodo_alberoN;
typedef nodo_alberoN* BNTree;
int distanza(Btree T, int x);
```



Tale libreria implementa alberi binari di ricerca i cui nodi contengono numeri interi nel tipo **BTree**, e alberi di grado arbitrario i cui nodi contengono numeri interi nel tipo **BNTree**. Nel tipo **BTree** i numeri interi che ricorrono nei nodi sono valori distinti mentre nel tipo **BNTree** lo stesso numero intero può ricorrere più di una volta in più nodi. Il metodo **distanza** restituisce la profondità del nodo contenente il valore **x** nell'albero **T**. Se l'albero è vuoto o non esiste nodo contenete x, la funzione restituirà -1. Ad esempio **livello** (T1,17) restituisce il valore 2.

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

1) `int` **`livello_min`**`(BTree T)`, che restituisca la profondità del nodo in `T` contente il valore minimo. Se l'albero `T` è vuoto la funzione restituirà il valore `-1`. Ad esempio **`livello_min`** `(T1)` restituirà il valore `2` (essendo 3 il valore minimo, la profondità del nodo contenente 3 è 2).
2) `int` **`diff_livello`**`(Btree T, int x)` che dato un albero binario di ricerca `T` e un intero `x`, restituisca 1 se la profondità del nodo in T contenente x è pari alla differenza tra x e la somma di tutti i valori contenuti negli altri nodi in T posizionati allo stesso livello del nodo contenente x, 0 altrimenti. Se l'albero T è vuoto o non esiste un nodo contenete x, la funzione restituirà 0. Ad esempio **`diff_livello`**`(T1, 22)` restituirà il valore `1;` nell'albero `T1` la profondità del nodo contenente 22 è 2 così come la differenza 22 – (17+3).
3) `int` **`conta_occorrenze`**`(BNTree T, int x, int n)`, che dato un albero ennario `T`, un intero `x` e un intero n, conta quanti nodi in T hanno un numero di figli contenenti il valore x compreso tra 1 (incluso) ed n (incluso). Se l'albero T è vuoto restituisce 0. Ad esempio **`conta_occorrenze`**`(T2, 34, 2)` restituirà il valore `2` (nell'albero `T2` sia il nodo contenente `"123"` che il nodo contenente `"43"` hanno almeno un figlio e al massimo 2 figli in cui ricorre il valore 34).

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