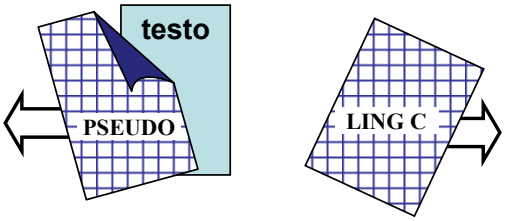
Algoritmi e Strutture di Dati – A.A. 2016-2017
Prova scritta del 9 febbraio 2017 – D.M. 509
Libri e appunti chiusi
Tempo = 1:45h

☐ 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

Nell'esercizio seguente un albero binario `T1` è un oggetto che ha il solo campo `T1.root` che è un riferimento al nodo radice dell'albero, dove ogni nodo ha i campi `parent` (genitore), `left` (figlio sinistro), `right` (figlio destro) e `info`. Un albero di grado arbitrario `T2` è un oggetto che ha il solo campo `T2.root` che è un riferimento al nodo radice dell'albero. Ogni nodo dell'albero di grado arbitrario ha i campi `parent` (genitore), `left` (figlio sinistro), `right` (fratello destro) e `info`.

Esercizio 1

Scrivi lo pseudocodice della procedura **TRASFORMA**(`T1`) che accetti in input un albero binario `T1` e produca in output un albero di grado arbitrario `T2` che ha la stessa struttura di `T1` (l'ordine dei figli non è significativo).

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.

```
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)
return L

FUNZ-RIC(v,L)
 if(v==NULL) return
 if(TEST(v))
     AGGIUNGI-IN-TESTA(L,v.info)
 FUNZ-RIC(v.left,L)
 FUNZ-RIC(v.right,L)
```

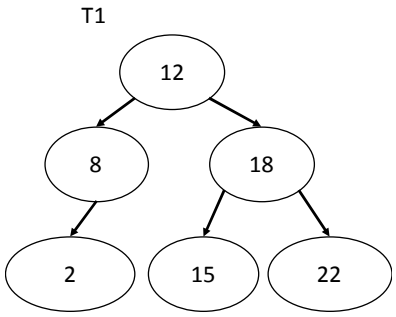
Assumi che TEST(v) abbia un costo proporzionale alla profondità del nodo v e che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante.

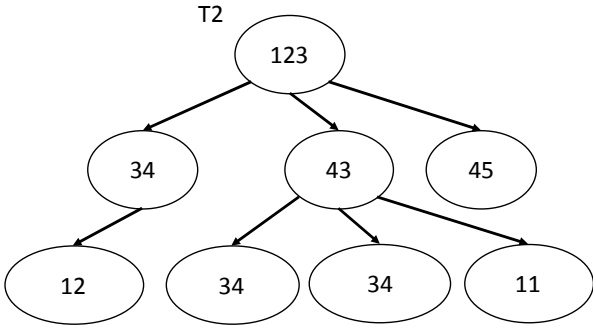
LINGUAGGIO C

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 lunghezza del cammino (numero di archi) dalla radice al 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 **distanza**(T1, 15) restituisce il valore 2.

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

1) int **dist_max**(BTree T), che restituisca la lunghezza del cammino in T dalla radice al nodo contente il valore massimo. Se l'albero T è vuoto la funzione restituirà il valore -1. Ad esempio **dist_max**(T1) restituirà il valore 2 (essendo il valore max 22, la lunghezza del cammino dalla radice al nodo contenente 22 sarà 2).
2) int **dist_livello** (Btree T, int x) che dato un albero binario di ricerca T e un intero x, restituisca 1 se la lunghezza del cammino in T dalla radice al nodo contenente x è pari al numero di nodi posizionati allo stesso livello in T del nodo contenete il valore x (escludendo tale nodo dal conteggio sul livello), 0 altrimenti. Se l'albero T è vuoto o non esiste un nodo contenete x, la funzione restituirà 0. Ad esempio **dist_livello**(T1, 15) restituirà il valore 1 (nell'albero T1 la lunghezza del cammino dalla radice al nodo contenente 15 è 2 così come il numero di nodi al livello del nodo 15 escluso tale nodo).
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 almeno n figli contenenti il valore x. Se l'albero T è vuoto restituisce 0. Ad esempio **conta_occorrenze**(T2, 34, 2) restituirà il valore 1 (nell'albero T2 solamente il nodo contenente "43" ha almeno 2 figli in cui ricorre il valore 34).

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