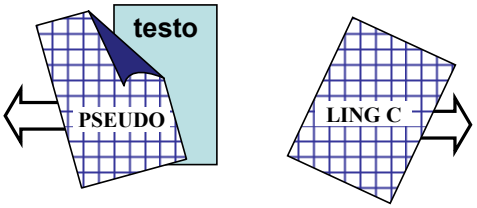
Algoritmi e Strutture di Dati – A.A. 2015-2016
Prova scritta del 13 settembre 2016 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

☐ 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

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-SPARSE**(A) che accetti in input un grafo non orientato A e produca in output il numero delle componenti connesse di A che hanno il numero minimo possibile di archi (una componente connessa con k nodi ha il numero minimo possibile di archi se ha k-1 archi non orientati). Puoi assumere (senza doverlo verificare) che il grafo in input non abbia loop (cioè archi i cui due estremi sono lo stesso nodo).

Esercizio 2

Discuti la complessità computazionale della seguente procedura nel caso peggiore fornendo O-grande, Omega e Theta in funzione del numero n di elementi dell'albero.

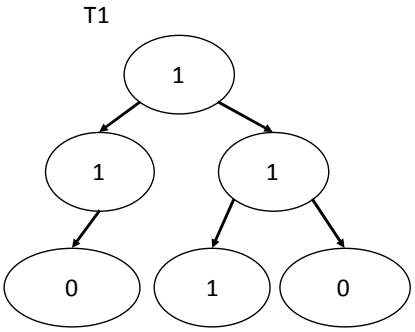
```
FUNZIONE(T)      /* T è un albero binario di interi */
L.head = NULL    /* L è una nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L)
return L

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

Supponi che AGGIUNGI-IN-TESTA abbia complessità Theta(1) e AGGIUNGI-IN-CODA abbia complessità Theta(x) dove x è la lunghezza della lista

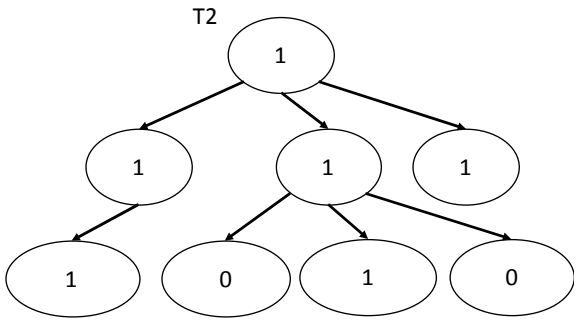
LINGUAGGIO C

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

```
typedef structelem {
            intvalore;
            structelem* sx;
            structelem* dx;
   } nodo_albero;
typedef nodo_albero* BTree;
```




```
typedef struct elem2 {
            intvalore;
            struct elem2* primofiglio;
            struct elem2* fratello;
   } nodo_alberoN;
typedef nodo_alberoN* BNTree;
```



Tale libreria implementa valori booleani (1 o 0) di tipo **`int`**, alberi binari i cui nodi contengono booleani nel tipo **`BTree`**, e alberi di grado arbitrario i cui nodi contengono booleani nel tipo **`BNTree`**.

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

1) `int` **`cammino_vero`**`(BTree T)`, che restituisca `1` se esiste in `T` un cammino radice-foglia in cui tutti i nodi contengono il booleano 1. Se l'albero `T` è vuoto oppure non esiste un cammino radice-foglia come richiesto, la funzione restituirà il valore `0`. Ad esempio **`cammino_vero`**`(T1)` restituirà il valore `1`.
2) `int` **`livello_vero`**`(Btree T, int h)` che dato un albero binario `T` e un intero `h`, verifica se tutti i nodi al livello h in T contengono il booleano 1. Se l'albero `T` è vuoto oppure non esiste il livello h la funzione restituirà il valore 0. Ad esempio **`livello_vero`**`(T1, 2)` restituirà il valore `0` (nell'albero `T1` il livello 2, che è l'ultimo in basso, contiene degli zeri).
3) `int` **`conta_booleani`**`(BNTree T)`, che dato un albero di grado arbitrario `T` conta quanti nodi in T hanno tutti figli contenenti il booleano 1. Se un nodo N non ha figli allora **`conta_booleani`**`(N)` ritorna 0. Ad esempio **`conta_booleani`**`(T2)` restituirà il valore `2`.

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