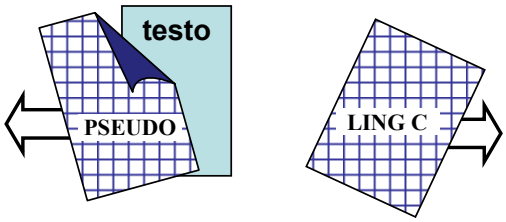
Algoritmi e Strutture di Dati – A.A. 2016-2017
Prova scritta del 9 Febbraio 2017 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

A

☐ 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 **CONNESSA-BIG**(A) che accetti in input un grafo non orientato A e restituisca una lista doppiamente concatenata di interi che contiene gli indici dei nodi della componente connessa più grande del grafo. Se il grafo ha più di una componente connessa di dimensione massima, la lista contiene solo gli indici dei nodi di una di tali componenti (per esempio la prima trovata). Implementa anche la funzione di aggiunta di un elemento alla lista.

Esercizio 2

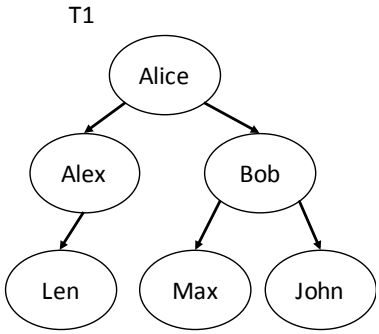
Discuti la complessità computazionale nel caso peggiore (in termini di O-grande, Omega e Theta) della seguente procedura in funzione del numero n di elementi dell'albero. Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante.

```
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 % 2 == 0)
     AGGIUNGI-IN-TESTA(L,v.info)
 FUNZ-RIC(v.left,L,depth+1)
 FUNZ-RIC(v.right,L,depth+1)
```

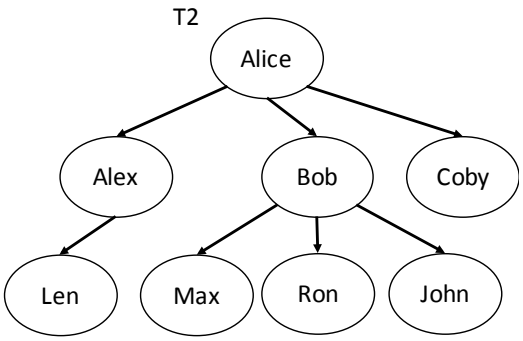
LINGUAGGIO C

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

```
typedef char stringa[20];

typedef struct elem {
            stringa nome;
            struct elem* sx;
            struct elem* dx;
   } nodo_albero;

typedef nodo_albero* BTree;
```




```
typedef struct elem2 {
            stringa nome;
            struct elem2* primofiglio;
            struct elem2* fratello;
   } nodo_alberoN;

typedef nodo_alberoN* BNTree;

int prof_genealogia(Btree T, stringa S);
```



Tale libreria implementa stringhe di lunghezza massima 20 caratteri nel tipo **stringa**, alberi binari i cui nodi contengono stringhe nel tipo **BTree**, e alberi di grado arbitrario i cui nodi contengono stringhe nel tipo **BNTree**. Sia il tipo **BTree** che il tipo **BNTree** implementano un albero genealogico (l'arco da un nodo **n1** ad un nodo **n2** rappresenta la relazione di discendenza diretta di **n2** da **n1**). Il metodo **prof_genealogia** restituisce la profondità del nodo contenente la stringa **S** all'interno dell'albero binario **T.** Ad esempio **prof_genealogia**(T1, "Bob") restituisce il valore 1. Assumi che negli alberi genealogici non ci siano nomi ripetuti due volte.

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

1) `int affini(BTree T, stringa S)`, che restituisca il numero di nodi presenti allo stesso livello del nodo contenente la stringa S (il nodo contenente la stringa S è escluso dal conteggio) all'interno dell'albero binario T. Se l'albero T è vuoto oppure non esiste il nodo contenente S, la funzione restituirà il valore 0. Ad esempio **affini**(T1, "Max") restituirà il valore 2.
2) `int prof_cammino (Btree T, stringa S)` che dato un albero binario T e una stringa S, restituisca 1 se la lunghezza del cammino in T dalla radice al nodo contenente la stringa S è pari al numero di nodi posti alla stessa profondità del nodo contenente S (il nodo contenente la stringa S è escluso dal conteggio dei nodi alla stessa profondità), altrimenti restituisce 0. Se l'albero T è vuoto o non esiste un nodo contenente S, restituisce 0. Ad esempio **prof_cammino**(T1, "Max") restituirà il valore 1.
3) `int conta_figli(BNTree T, int x, int h)`, che dato un albero di grado arbitrario T, un intero x e un intero h conta quanti nodi posizionati a profondità h in T hanno almeno x figli (cioè discendenti diretti). Ad esempio **conta_figli**(T2, 1, 1) restituirà il valore 2 (nell'albero T2 al livello 1 solamente i nodi contenenti "Alex" e "Bob" hanno almeno 1 figlio.

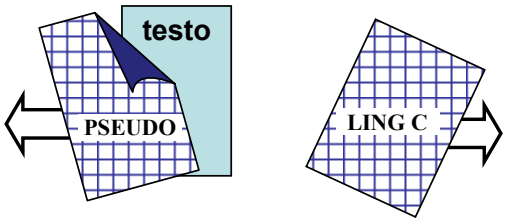
È possibile utilizzare qualsiasi libreria nota e implementare qualsiasi metodo di supporto a quelli richiesti.

Algoritmi e Strutture di Dati – A.A. 2016-2017
Prova scritta del 9 Febbraio 2017 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

B

☐ 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 **NODI-CONNESSE-10**(A) che accetti in input un grafo non orientato A e restituisca una lista doppiamente concatenata di interi che contiene gli indici dei nodi che appartengono a componenti connesse che hanno almeno dieci nodi. Se nessuna componente connessa ha almeno dieci nodi **NODI-CONNESSE-10**(A) restituisce una lista vuota. Implementa anche la funzione di aggiunta di un elemento alla lista.

Esercizio 2

2.1) 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. Assumendo che AGGIUNGI-IN-CODA faccia un numero di operazioni lineare nella lunghezza corrente della lista.

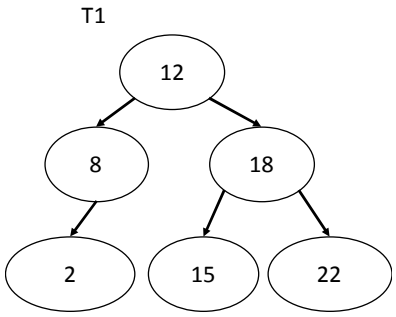
```
FUNZIONE(T)       /* T è un albero binario di interi */

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.info % 2 == 0)
      AGGIUNGI-IN-CODA(L,v.info)
      FUNZ-RIC(v.left,L)
 else
      FUNZ-RIC(v.right,L)
```

2.2) Mostra un albero corrispondente al caso peggiore.

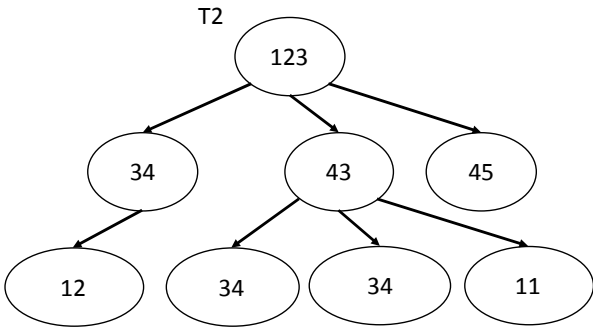
LINGUAGGIO C

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.