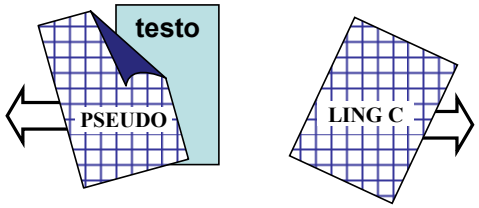
Algoritmi e Strutture di Dati – A.A. 2015-2016
Prova scritta del 4 febbraio 2016 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

☐ 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-DENSE**(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 massimo possibile di archi (una componente connessa con k nodi ha il numero massimo possibile di archi se ha k(k-1)/2 archi non orientati). Puoi assumere (senza doverlo verificare) che il grafo in input non abbia loop.

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(TEST(v))
     AGGIUNGI-IN-CODA(L,v.info)
FUNZ-RIC(v.left,L)
FUNZ-RIC(v.right,L)
```

Supponi che:

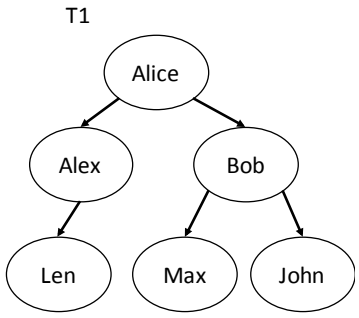
- TEST(v) abbia complessità Theta(x), dove x è il numero dei nodi del sottoalbero radicato in v
- AGGIUNGI-IN-CODA abbia complessità Theta(x) dove x è la lunghezza della lista

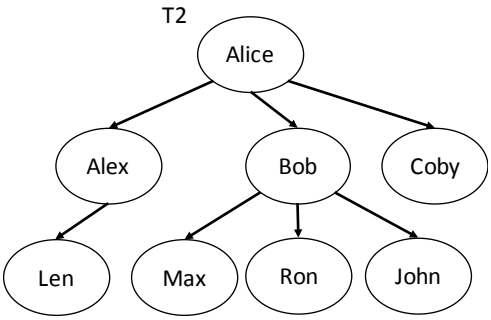
LINGUAGGIO C

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* NTree;
int size_genealogia(Btree T, stringa S);
```





Tale libreria implementa stringhe di lunghezza massima 19 caratteri nel tipo **stringa** (terminate con '\0'), il tipo **BTree** (alberi binari i cui nodi contengono stringhe) e il tipo **NTree** (alberi di grado arbitrario i cui nodi contengono stringhe). Sia il tipo **BTree** che il tipo **NTree** rappresentano 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 **size_genealogia** restituisce la dimensione (numero di nodi) del sottoalbero radicato nel nodo contenente la stringa **S** (tale nodo è incluso nel conteggio) all'interno dell'albero binario **T.** Ad esempio **size_genealogia**(T1, "Bob") restituisce il valore 3.

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

1) `int` **discendenti**`(BTree T, stringa S1, stringa S2)`, che restituisca `1` se esiste in `T` un cammino dal nodo contenente `S1` al nodo contenente `S2`. Se l'albero `T` è vuoto oppure non esiste il nodo contenente `S1` oppure `S2`, la funzione restituirà il valore `0`. Ad esempio **discendenti**`(T1, "Alice", "Max")` restituirà il valore `1`.
2) `int` **conta_genealogia**`(Btree T, int x)` che dato un albero binario `T` e un intero `x`, conta quanti nodi in T sono radici di sottoalberi la cui dimensione (numero di nodi compresa la radice) è strettamente maggiore di `x`. Ad esempio **conta_genealogia**`(T1, 2)` restituirà il valore `2` (nell'albero `T1` solamente i nodi contenenti `"Alice"` e `"Bob"` sono radici di alberi che contengono rispettivamente 6 e 3 nodi).
3) `int` **conta_discendenti**`(NTree T, int x)`, che dato un albero di grado arbitrario `T` e un intero `x` conta quanti nodi in T hanno esattamente `x` figli (diretti discendenti). Ad esempio **conta_discendenti**`(T2, 3)` restituirà il valore `2` (nell'albero `T2` solamente i nodi contenenti `"Alice"` e `"Bob"` hanno esattamente 3 figli, che sono diretti discendenti).

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

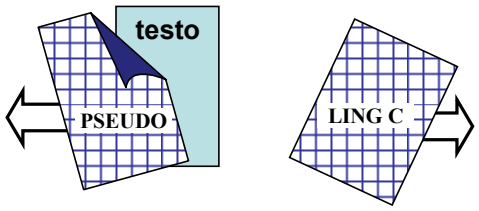
B

Algoritmi e Strutture di Dati – A.A. 2015-2016
Prova scritta del 4 febbraio 2016 – D.M. 270
Libri e appunti chiusi
Tempo = 2:00h

☐ 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-SENZA-CICLI**(A) che accetti in input un grafo non orientato A e produca in output il numero delle componenti connesse di A che non hanno cicli (una componente connessa con k nodi non ha cicli se ha k-1 archi non orientati). Puoi assumere (senza doverlo verificare) che il grafo in input non abbia loop.

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(TEST(v))
      FUNZ-RIC(v.left,L)
else
      FUNZ-RIC(v.right,L)
AGGIUNGI-IN-CODA(L,v.info)
```

Supponi che:

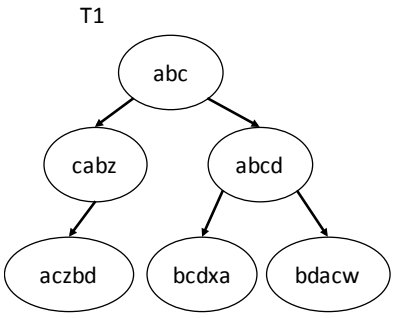
- TEST(v) abbia complessità Theta(x), dove x è il numero dei nodi del sottoalbero radicato in v
- AGGIUNGI-IN-CODA abbia complessità Theta(x) dove x è la lunghezza della lista

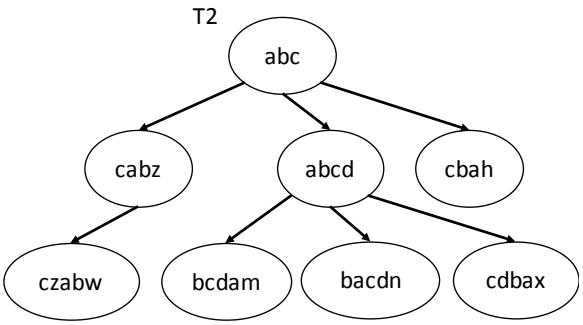
LINGUAGGIO C

Si consideri la libreria **parole.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* NTree;
int conta_occorrenza(Btree T, char c);
```





Tale libreria implementa stringhe di lunghezza massima 19 caratteri nel tipo **stringa** (terminate con '\0'), il tipo **BTree** (alberi binari i cui nodi contengono stringhe) e il tipo **NTree** (alberi di grado arbitrario i cui nodi contengono stringhe). Sia il tipo **BTree** che il tipo **NTree** implementano un dizionario (l'arco da un nodo **n1** ad un nodo **n2** rappresenta la relazione di contenimento di tutte le lettere di **n1** in **n2**). Il metodo **conta_occorrenza** conta quanti nodi dell'albero binario **T** contengono la lettera **c**. Ad esempio **conta_occorrenza**(T1, 'b') restituisce il valore 6.

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

1) `int` **distanza**`(BTree T, stringa S1, stringa S2)`, che restituisca la lunghezza del cammino (numero di archi) in `T` dal nodo contenente `S1` al nodo contenente `S2`. Se l'albero `T` è vuoto oppure non esiste il nodo contenente `S1` oppure `S2`, oppure non esiste un cammino da `S1` a `S2`, la funzione restituirà il valore `0`. Ad esempio **distanza**`(T1, "abc", "aczbd")` restituirà il valore `2`.
2) `int` **conta_nodi**`(Btree T, int x, char c)` che dato un albero binario `T`, un intero `x` e un carattere `c`, conta quanti nodi in T sono radici di sottoalberi che contengono almeno x nodi in cui occorre il carattere c nella stringa del nodo. Ad esempio **conta_nodi**`(T1, 3, 'a')` restituirà il valore `2` (nell'albero `T1` solamente i nodi contenenti `"abc"` e `"abcd"` sono radici di alberi che contengono rispettivamente 6 e 3 nodi nella cui stringa occorre la lettera 'a').
3) `int` **conta_figli**`(NTree T, int x, char c)`, che dato un albero di grado arbitrario `T`, un intero `x` e un carattere c, conta quanti nodi in T hanno esattamente `x` figli la cui stringa contiene il carattere c. Ad esempio **conta_figli**`(T2, 3, 'd')` restituirà il valore `1` (nell'albero `T2` solamente il nodo contenente `"abcd"` ha esattamente 3 figli in cui occorre la lettera 'd').

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