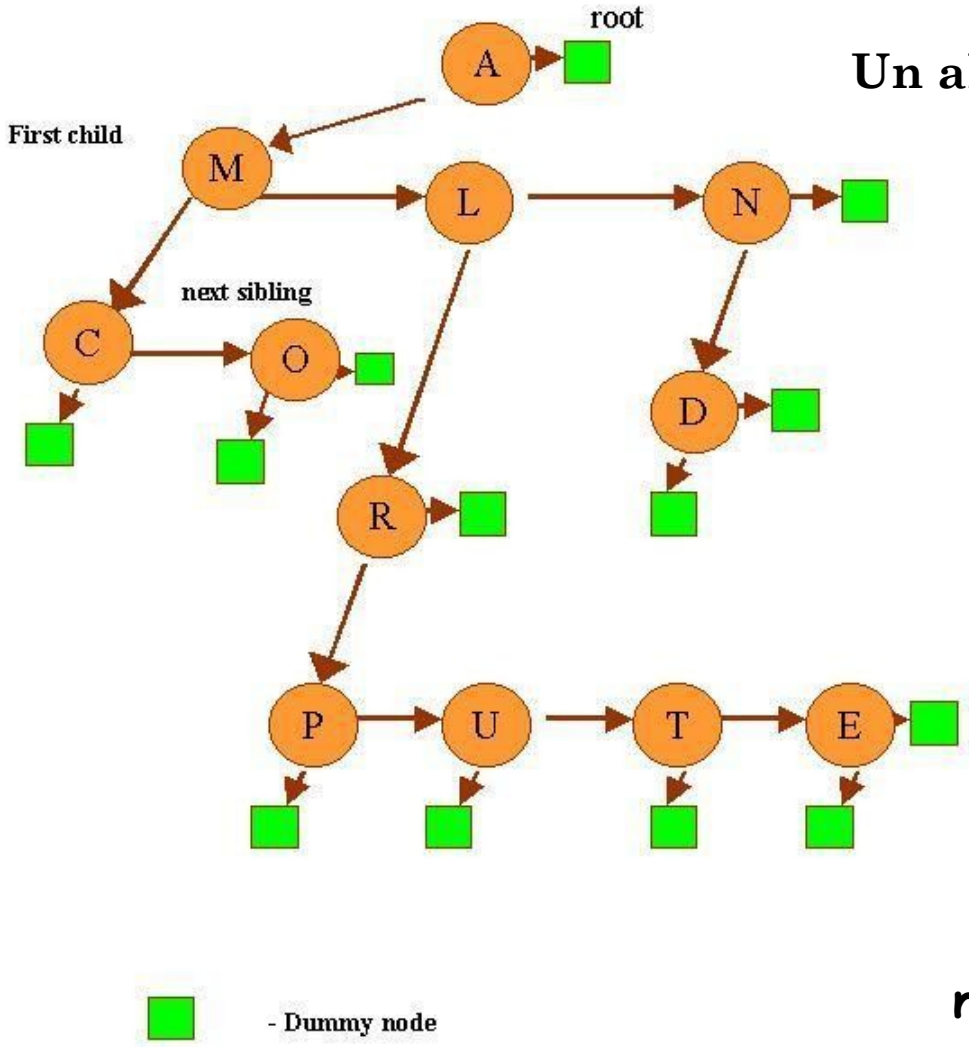
LABORATORIO 7

Algoritmi e strutture dati 2022-2023

ALBERO GENERICO

Implementato con struttura dati 'primo figlio prossimo fratello'



Un albero è un puntatore **a treeNode**

```
struct tree::treeNode {
    string label;
    treeNode *firstChild;
    treeNode *nextSibling;
};
typedef treeNode* Tree;
```

numero arbitrario di figli!

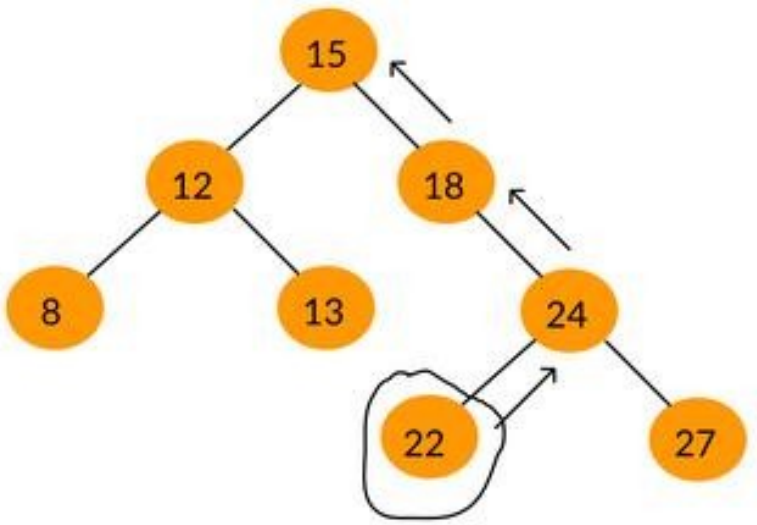
FUNZIONI DA IMPLEMENTARE

```
namespace tree{
  enum Error {OK, FAIL};
  typedef string Label;
  const Label emptyLabel = "$#$#$";
  struct treeNode; // definita nel file tree.cpp
  typedef treeNode* Tree;
  const Tree emptyTree = NULL;
  bool isEmpty(const Tree&);
  Error addElem(const Label, const Label, Tree&);
  Error deleteElemR(const Label, Tree&);
  Error deleteElemI(const Label, Tree&);
  Label father(const Label, const Tree&);
  list::List children(const Label, const Tree&);
  int degree(const Label, const Tree&);
  list::List ancestorsR(const Label, const Tree&);
  list::List ancestorsI(const Label, const Tree&);
  Label leastCommonAncestor(const Label, const Label, const Tree&);
  bool member(const Label, const Tree&);
  int numNodes(const Tree&);
  Tree createEmpty();
}
/* Funzioni che non caratterizzano il TDD Tree, ma che servono per input/output
tree::Tree readFromFile(string);
void printTree(const tree::Tree&);
```

tree-chsib.h

FUNZIONI ANCESTORS (I e R)

- Le funzioni **ancestors (antenati)** restituiscono la lista degli antenati di un nodo
- Per realizzare queste funzioni avrete bisogno del TDD lista e quindi di due file .h e .cpp per implementare le liste.
  - Le liste servono solo per questo scopo e non per implementare i fratelli nell'albero.



VINCOLI IMPLEMENTAZIONE LISTE

- Il codice richiede l'utilizzo delle liste

```
list::List children(const Label, const Tree&);
int degree(const Label, const Tree&);
list::List ancestorsR(const Label, const Tree&);
list::List ancestorsI(const Label, const Tree&);
```

- Nel codice da completare troverete una delle implementazioni viste durante il corso
  - **list-array.h e list-array.cpp**
  - **non i vector!!!!**

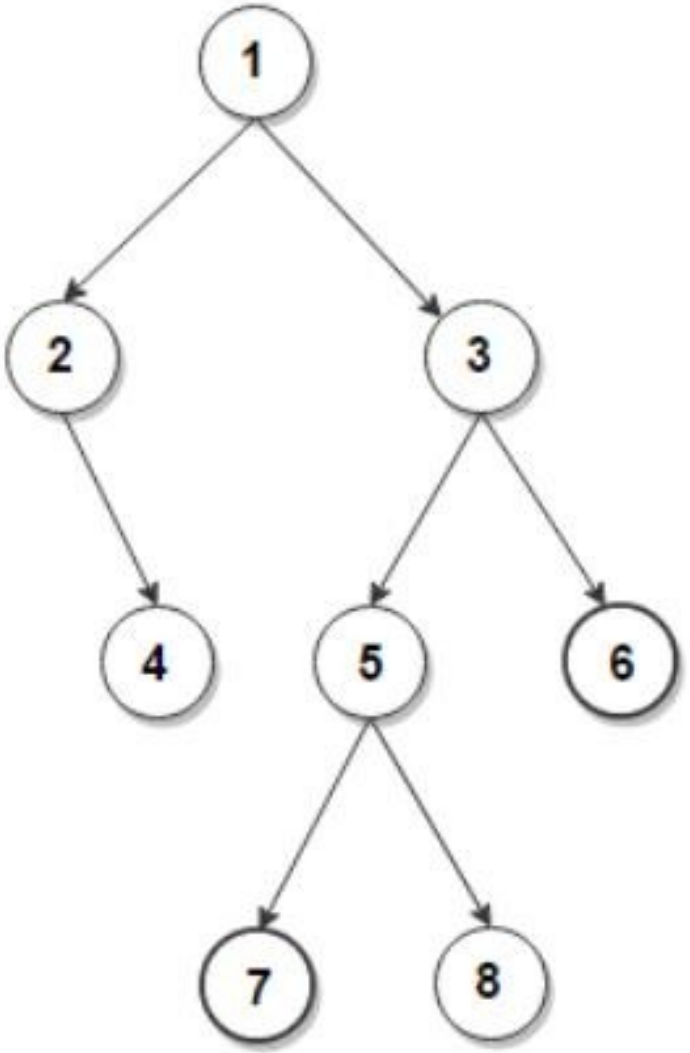
FUNZIONE LEAST COMMON ANCESTOR

LCA(6, 7) = 3

LCA(5, 8) = 5

LCA(3, 6) = 3

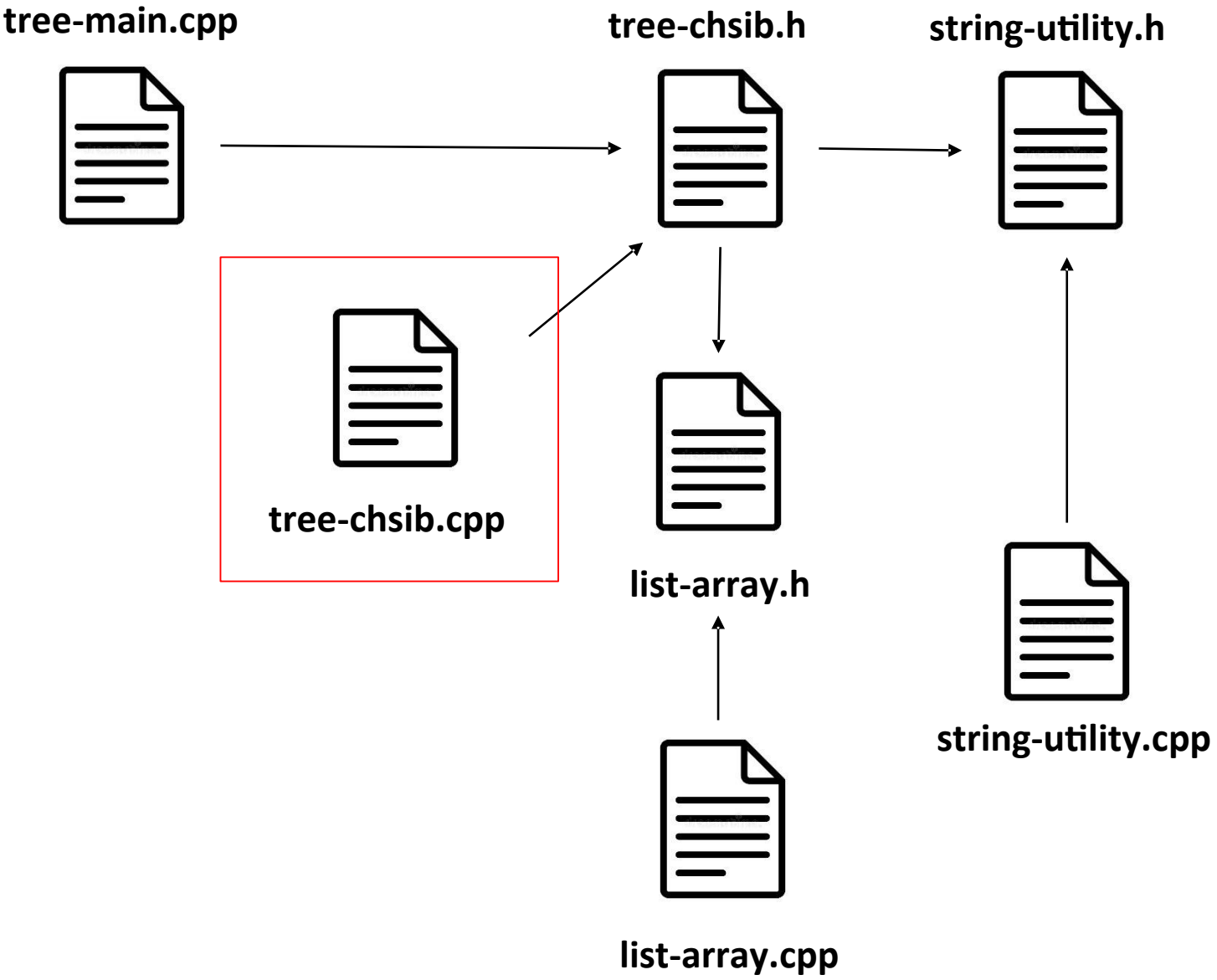
LCA(2, 5) = 1



File .zip (TRACCIA)

*Composta da **diversi** file + 1 file di input*

- **tree-main.cpp**
  - il main contiene il menu che permette di eseguire le varie operazioni sul TDD Albero - **NON MODIFICARE**
- **tree-chsib.h**
  - Contiene **strutture dati** e prototipi delle funzioni che andranno implementate nel file **tree-chsib-hashtable.cpp** e richiamate dal mian
  - Header - **NON MODIFICARE**
- **tree-chsib.cpp**
  - Implementazione dell'albero (primo figlio prox fratello)
    - **Implementare qui le funzioni richieste!**
- **string-utility.cpp, string-utility.h**
  - Questi file contengono delle funzioni per "nomalizzare" le label, rendendo tutti i caratteri minuscoli ed eliminando gli spazi
- **list-array.h e list-array.cpp**
  - Implementazione delle liste

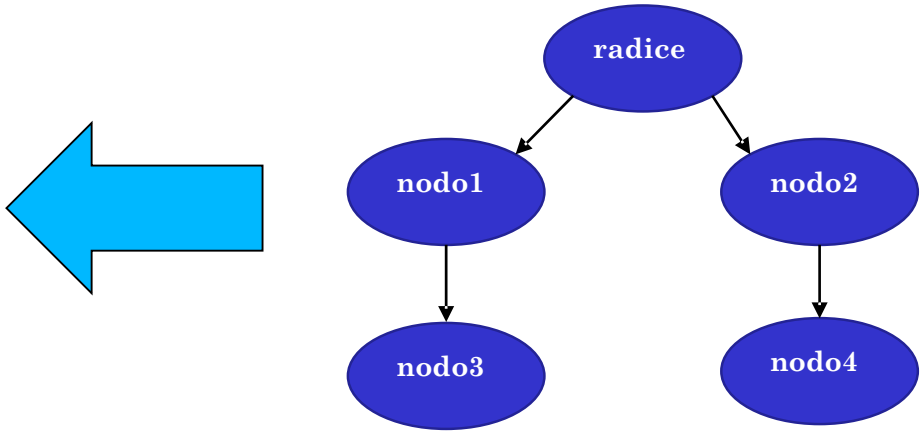


Le frecce come al solito rappresentano gli include

FORMATO FILE INPUT

- Il formato dei file che contengono la rappresentazione degli alberi è il seguente:

radice
radice nodo1 nodo2
nodo1 nodo3
nodo2 nodo4

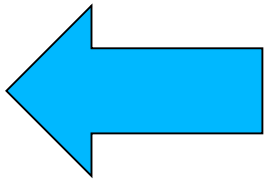


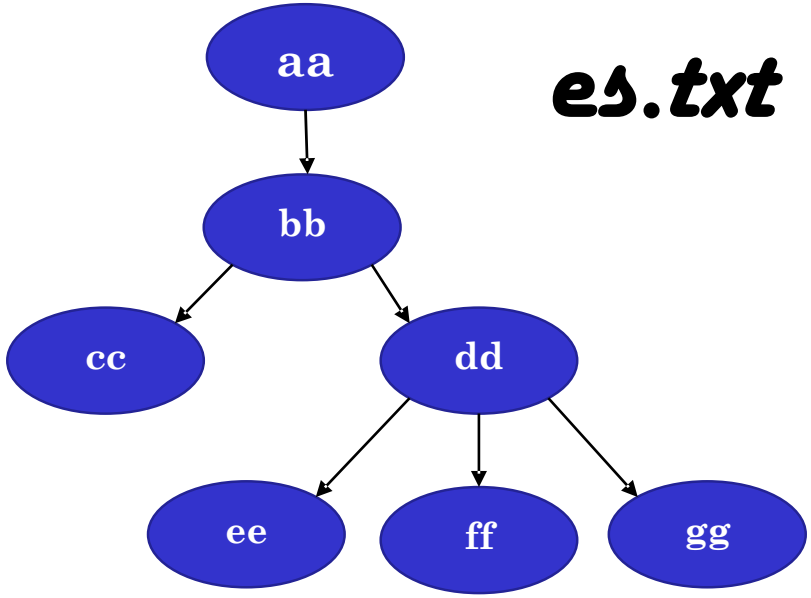
- Ovvero la prima riga del file deve contenere l'etichetta della radice e le righe seguenti contengono come prima etichetta quella di un nodo (che deve già essere stato elencato prima!) seguita dalle etichette dei suoi figli.

MODALITA' DI STAMPA

- Stampare l'albero in maniera strutturata usando l'indentazione per rendere esplicito il livello di un nodo nell'albero:

```
aa
--bb
----cc
----dd
------ee
------ff
------gg
```





es.txt

- Se non riuscite a implementare la stampa strutturata cercate di stampare l'albero in modo che sia chiara la struttura gerarchica

SUGGERIMENTI

- Le prime due funzioni da implementare sono la **addElem,** che si può testare dal main selezionando l'opzione “b”, e la **printTree**

- **addElem** viene richiamata dalla funzione readFromStream(istream& str) ed è quindi necessaria per leggere dati da file

- Si consiglia di implementare poi le funzioni più facili (member, father, degree, numNode) e di affrontare solo all'ultimo la funzione deleteElem, nelle sue due varianti, e le funzioni ancestors, nelle sue due varianti, e leastCommonAncestor.