Corso “Programmazione 1”
Capitolo 12: Strutture Dati Astratte

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 24 novembre 2021

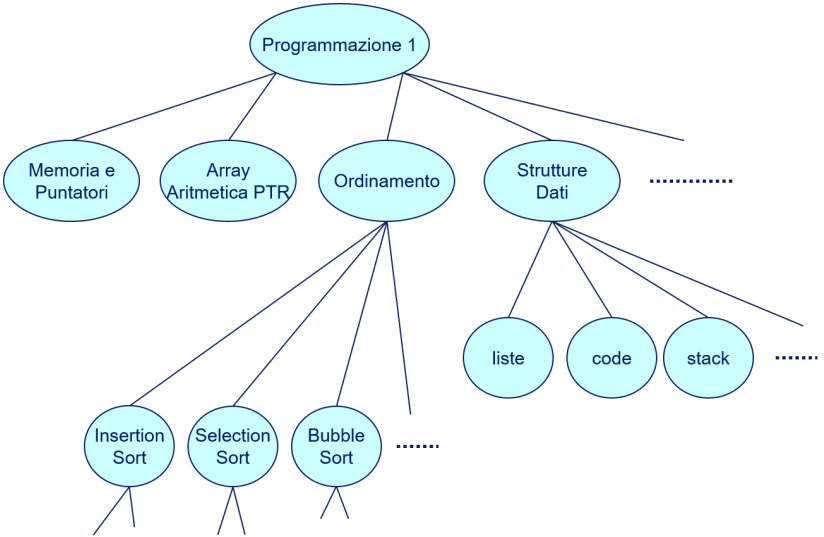
Terms of Use and Copyright

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Alberi

- Gli alberi sono una struttura matematica che gioca un ruolo molto importante nella progettazione e nell'analisi di algoritmi:
  - Sono spesso utilizzati per descrivere proprietà dinamiche degli algoritmi.
  - Spesso utilizziamo strutture dati che rappresentano implementazioni concrete di alberi.
- Questo tipo di ADT lo incontriamo nella vita di tutti i giorni:
  - L'albero genealogico della propria famiglia (da cui deriva la maggior parte della terminologia impiegata nella teoria degli alberi).
  - Nei tornei sportivi.
  - Per rappresentare l'organigramma di aziende.
  - Per rappresentare l'analisi sintattica dei linguaggi di programmazione.
  - Il file system di un sistema operativo.
  - Gerarchie
  - ...

Alberi


Programmazione 1
Memoria e Puntatori
Array Aritmetica PTR
Ordinamento
Strutture Dati
.....
liste
code
stack
.....
Insertion Sort
Selection Sort
Bubble Sort
.....

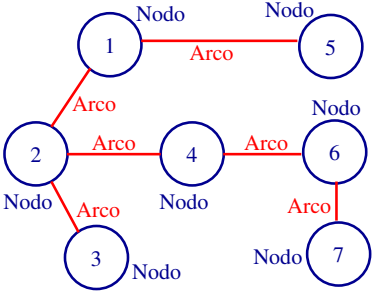
Tipi di Alberi

- Esistono diversi tipi di alberi, ed è importante distinguere tra modello astratto e modello concreto (ovvero tra modello matematico e implementazione).
- In ordine di generalità decrescente distinguiamo:
  - Alberi generici.
  - Alberi con radice.
  - Alberi ordinati.
  - Alberi M-ari.
  - Alberi binari come caso particolare di albero M-ario.

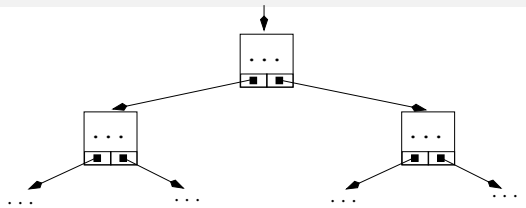
Alberi (Teoria dei grafi)

In teoria dei grafi un albero è un grafo non orientato nel quale due vertici qualsiasi sono connessi da uno e un solo cammino

- Definizione: Un albero è un insieme non vuoto di vertici ed archi (grafo) che soddisfa alcune proprietà:
  - Un vertice (o nodo) è un oggetto semplice che può essere dotato di un nome, e di una informazione associata (denominata spesso chiave o key).
  - Un arco è una connessione tra due nodi.
  - Un grafo non orientato, connesso e privo di cicli



Alberi binari



- (Nella sua versione più semplice) una albero binario `t` di oggetti di tipo `T` è definito come segue:
  - `t` è un puntatore **`NULL`** (albero vuoto) oppure
  - `t` è un puntatore ad un nodo (**`struct`**) contenente:
    - un campo `value` di tipo `T`
    - due campi `left` e `right` di tipo albero

```
struct node;
typedef node * albero;
struct node { T value;  albero left, right; };
```

Un albero binario è una struttura dati dinamica.

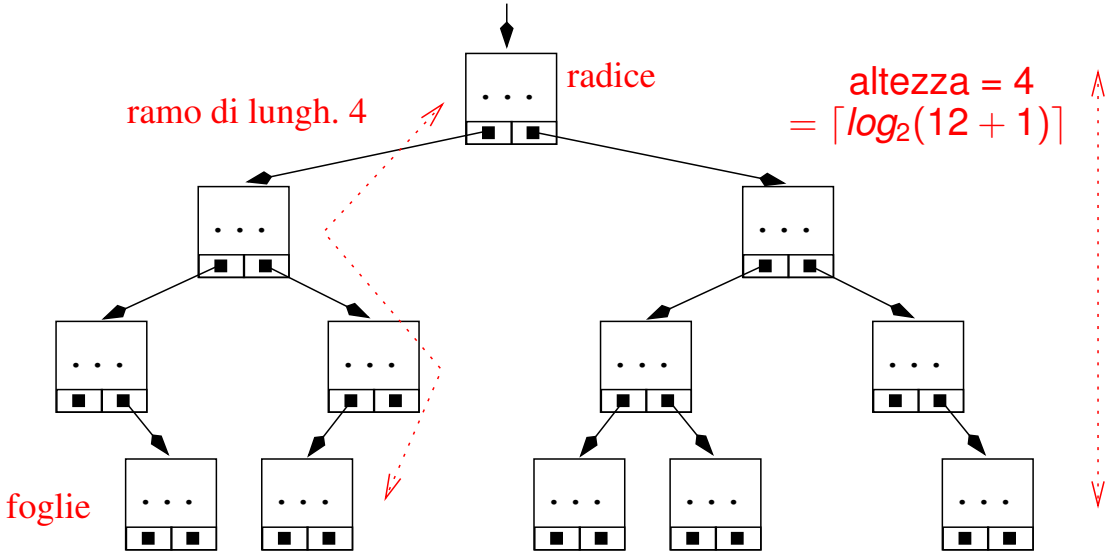
Alberi binari: terminologia

- I sottoalberi (possibilmente vuoti) di un nodo `N` sono detti sottoalbero sinistro e sottoalbero destro di `N`
- Se un nodo `N` punta nell'ordine a due (eventuali) nodi `N1`, `N2`
  - `N1` e `N2` sono detti rispettivamente figlio sinistro e figlio destro di `N`
  - `N` è detto nodo padre di `N1` e `N2`
- In un albero binario ci possono essere tre tipi di nodi:
  - Il nodo radice, che non ha padre
  - I nodi foglia, che non hanno figli
  - I nodi intermedi, che hanno padre e almeno un figlio

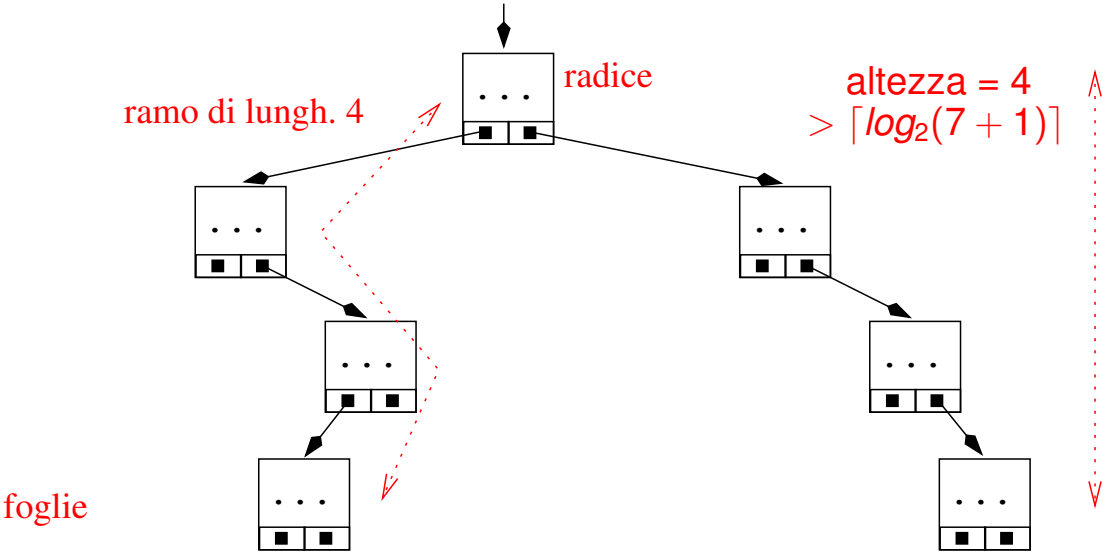
Alberi binari: terminologia - II

- Una catena di nodi dalla radice a una foglia è detta ramo
  - Il numero di nodi in un ramo è detto lunghezza del ramo
  - La massima lunghezza di un ramo è detta altezza dell'albero
  - L'altezza di un albero binario di N elementi è $h \in [\lceil log_2(N+1) \rceil, N]$
- Un albero binario di N elementi è bilanciato se la sua altezza è $h = \lceil log_2(N+1) \rceil$
  $\Longrightarrow$ tutti i rami hanno lunghezza `h` o `h-1`
- Un albero binario di N elementi è completo se la sua altezza è tale che $N = 2^h - 1$
  $\Longrightarrow$ tutti i rami hanno lunghezza `h`

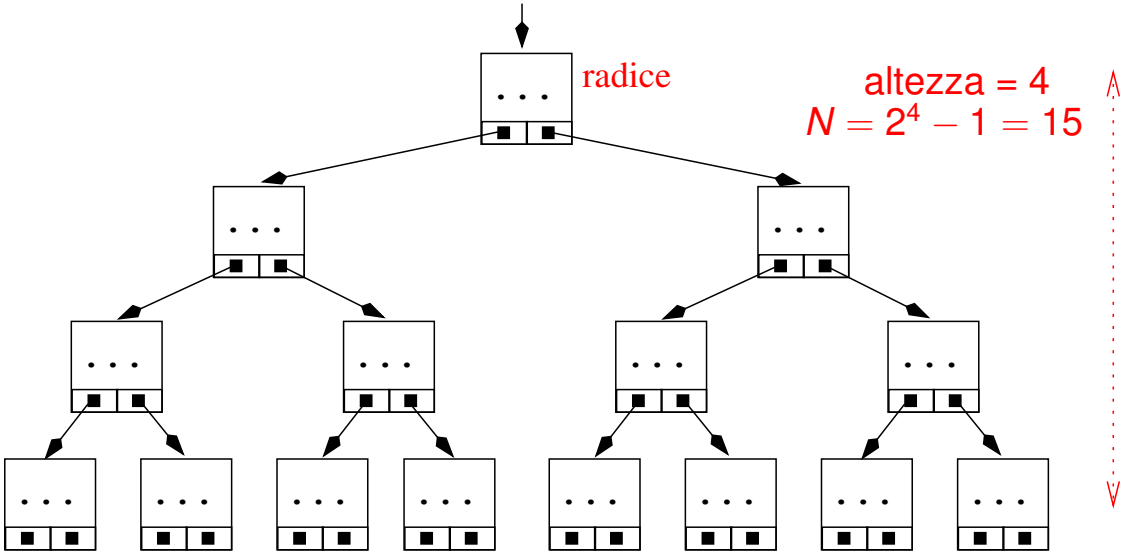
Esempio: albero binario bilanciato



Esempio: albero binario non bilanciato



Esempio: albero binario completo



Esempio: albero binario completo (2)

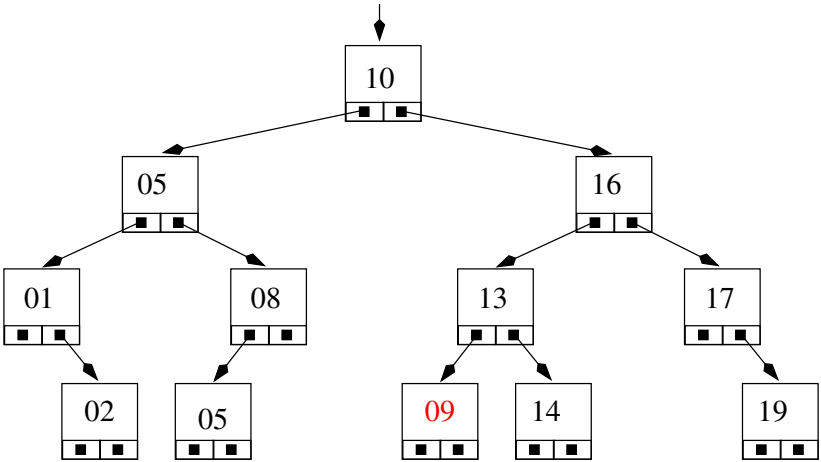
Albero di ricerca binaria

- Un albero di ricerca binaria è una struttura dati utile a mantenere dati ordinati.
- Assumiamo una relazione di ordine totale di precedenza "$\preceq$" tra gli elementi `T`
  - Es: ordine numerico, ordine alfabetico del campo "cognome", ecc.
- Un albero binario è un albero di ricerca binaria se ogni nodo `N` dell'albero verifica la seguente proprietà:
  - Tutti i nodi del sottoalbero di sinistra precedono strettamente `N`
  - Tutti i nodi del sottoalbero di destra sono preceduti da `N`

  (è possibile invertire lo "strettamente" tra sinistra e destra)

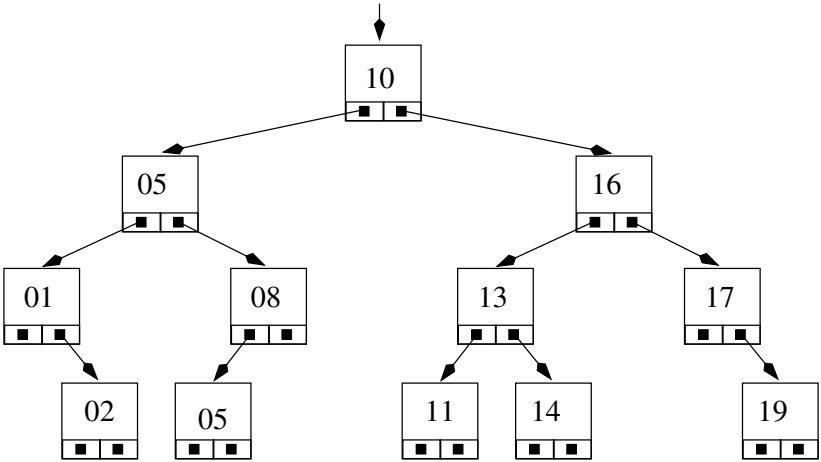
Nota: in alcuni casi non è previsto che ci possano essere due valori uguali nel valore valutato dalla relazione di precedenza (valore chiave)

Esempio: albero di ricerca binaria



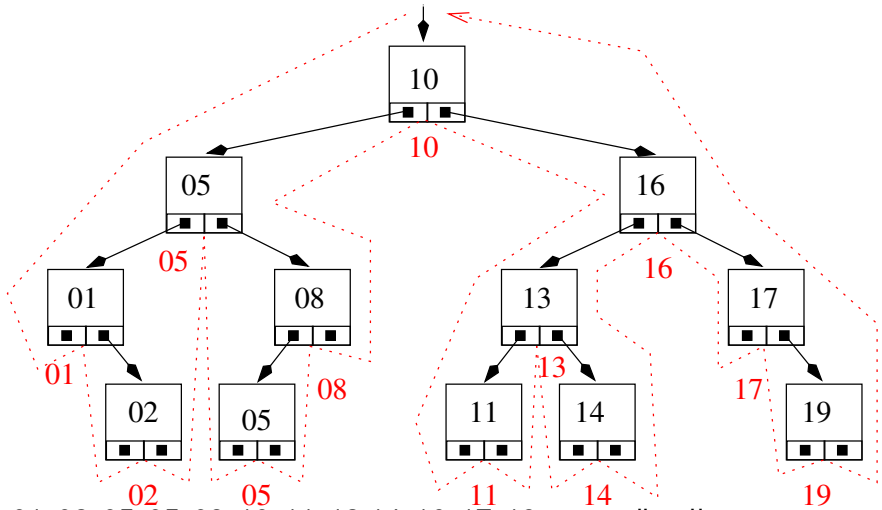
- Questo è un albero di ricerca binaria?
- No, 09 non può stare nel sottoalbero di destra di 10

Esempio: albero di ricerca binaria



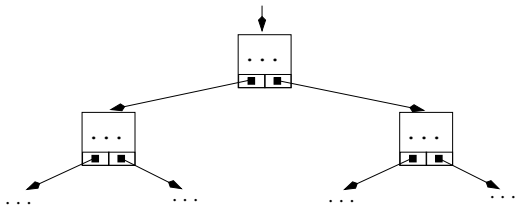
- Questo è un albero di ricerca binaria?
- Sì

Esempio: Visita ordinata di un albero di ricerca binaria



- Visita: $01, 02, 05, 05, 08, 10, 11, 13.14, 16, 17, 19 \Longrightarrow$ ordinati!

Implementazione di un albero di ricerca binaria

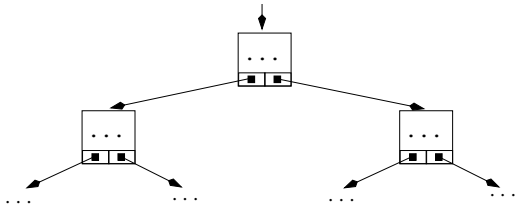


- Dati: un albero di ricerca binaria `t`
  - `t` punta al primo elemento inserito nell'albero (inizialmente **`NULL`**)
  - albero vuoto: `t==`**`NULL`**
  - albero pieno: `out of memory`

  $\Longrightarrow$ numero di elementi contenuti nell'albero limitato solo dalla memoria

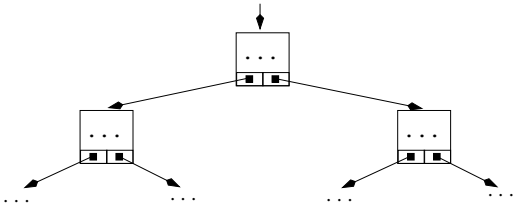
N.B.: allocati solo gli `n` nodi necessari a contenere gli elementi

Implementazione di un albero di ricerca binaria II



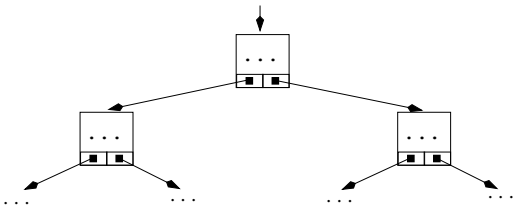
- Funzionalità:
  - `init`: pone `t`=**`NULL`**
  - `search` (cerca un elemento `val` in `t`):
    1. se `t == NULL`, restituisce **`NULL`**
    2. se `val == t->value`, restituisce `t`
    3. se `val < t->value`, cerca ricorsivamente in `t->left`
    4. se `val > t->value`, cerca ricorsivamente in `t->right`
  - ...

Implementazione di un albero di ricerca binaria III



- Funzionalità:
  - ...
  - `insert` (inserisce un elemento `val` in `t`):
    1. se `t` è vuoto, `t == NULL`:
       - crea un nuovo nodo per il puntatore `tmp`
       - pone `tmp->value=val`, `tmp->left=NULL`, `tmp->right=NULL`,
       - pone `t=tmp`
    2. se `val < t->value`, inserisci ricorsivamente in `t->left`
    3. se `val >=t->value`, inserisci ricorsivamente in `t->right`
  - ...

Implementazione di un albero di ricerca binaria III



- Funzionalità:
  - ...
  - `print` (stampa in modo ordinato l'albero `t`): Se l'albero non è vuoto
    - stampa ricorsivamente il sottoalbero sinistro `t->left`
    - stampa il contenuto del nodo puntato da `t`: `t->value`
    - stampa ricorsivamente il sottoalbero destro `t->right`
  - `deinit`: se l'albero non è vuoto:
    - applica ricorsivemente `deinit` ai sottoalberi sinistro `t->left` e destro `t->right`
    - applica **`delete`** al nodo puntato da `t`
  - [`remove` non analizzata qui]

Esempi su alberi di ricerca binaria su interi

- albero di **`char`**:
  - `TREE/tree.h`
  - `TREE/tree.cc`
  - `TREE/tree_main.cc`
- variante della precedente:
  - `TREE/tree1.h`
  - `TREE/tree1.cc`
  - `TREE/tree1_main.cc`

Esempi su alberi di ricerca binaria su tipi generici

- albero di "qualsiasi" tipo si voglia:
  - `MODULAR_TREE/tree.h`
  - `MODULAR_TREE/tree.cc`
  - `MODULAR_TREE/tree_main.cc`

Esercizi proposti

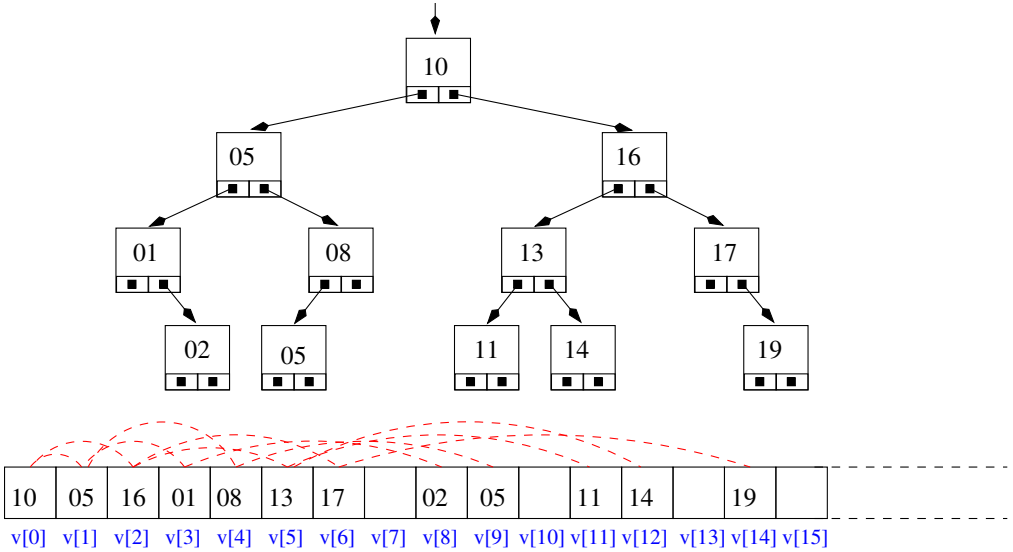
Vedere file `ESERCIZI_PROPOSTI.txt`

Implementazione di un albero binario tramite array

- Dati: un array `v` di `dim` elementi di tipo `T`
  - un (sotto)albero è dato da un puntatore a `v` e un indice `i`
    `struct tree { T * v; int i; };`
  - `v` allocato dinamicamente
  - l'elemento radice è in `v[0]`
  - se un elemento è in posizione `v[i]`, i suoi due figli sono in posizione `v[2*i+1]` e `v[2*i+2]`
  - necessaria una nozione ausiliaria di "elemento vuoto"
- Funzionalità: come nell'implementazione precedente, cambia solo la nozione di figlio sinistro/destro

N.B.: allocati `dim` nodi $\Longrightarrow$ efficace solo se ben bilanciato

Implementazione di un albero binario tramite array II



Esempi su alberi di interi

- albero di **`char`**:
  - `TREE_ARRAY/tree.h`
  - `TREE_ARRAY/tree.cc`
  - `TREE_ARRAY/tree_main.cc`
- albero di "qualsiasi" tipo si voglia:
  - `MODULAR_TREE_ARRAY/tree.h`
  - `MODULAR_TREE_ARRAY/tree.cc`
  - `MODULAR_TREE_ARRAY/tree_main.cc`

N.B. Il "`main`" e gli header delle funzioni identici a quelli in `TREE/MODULAR_TREE` $\Longrightarrow$ Tipo di Dato Astratto