Soluzione Progetto ASD

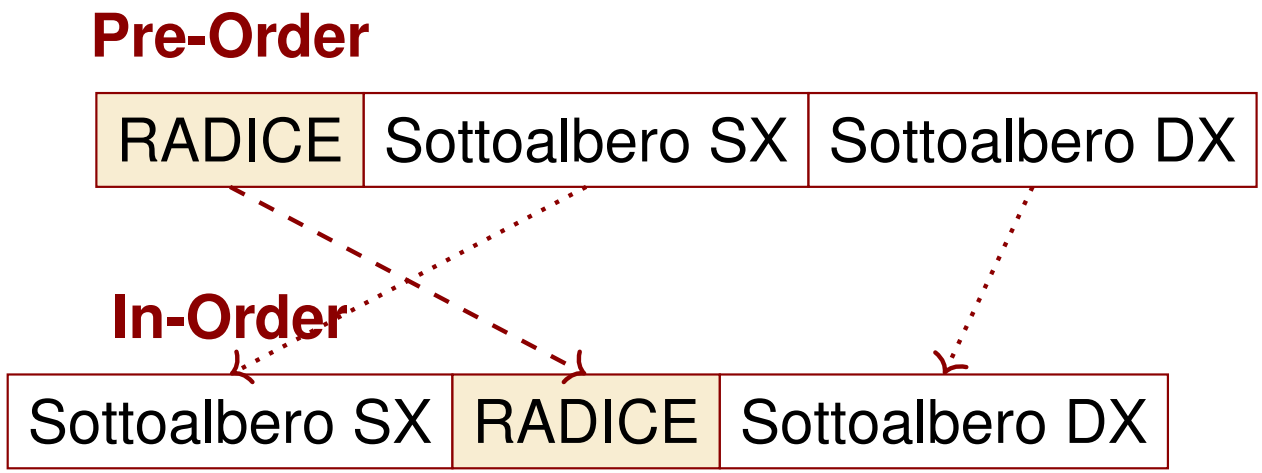
Ricostruzione di alberi dalle visite

Anno Accademico 2025/2026

LA CHIAVE DI LETTURA

Per ricostruire l'albero, dobbiamo identificare univocamente **la radice** e la dimensione dei **sottoalberi** sinistro e destro.

- **Pre-order** / **Post-order**: Ci dicono *chi* è la radice.
- **In-order**: Ci dice *dove* si "divide" l'albero tra destra e sinistra.



Strategia Risolutiva (Divide et Impera)

L'approccio è naturalmente **ricorsivo**.

Passi dell'algoritmo (Pre + In)

1. Prendiamo il **primo** elemento del vettore `Pre` come **Radice**.
2. Cerchiamo la posizione della radice nel vettore `inIndex[]`.
3. Calcoliamo la dimensione del sottoalbero sinistro (L_{size}) basandoci sull'indice trovato in `inIndex[]`.
4. **Ricorsione SX**:
   - Pre: dal prossimo elemento per L_{size} elementi.
   - In: dall'inizio fino alla radice esclusa.
5. **Ricorsione DX**:
   - Pre: dopo il blocco sinistro fino alla fine.
   - In: dopo la radice fino alla fine.

Panoramica della Strategia per i 100pt

La soluzione ottima deve gestire $N = 250.000$ in tempi stretti.

I 3 Pilastri della Soluzione

1. **Mappatura O(1)**: Utilizziamo un'`unordered_map` per trovare istantaneamente la posizione di un nodo nella visita In-order.
2. **Indici per Riferimento**: Evitare copie di vettori. Passare un indice globale ('int& idx') che avanza/indietreggia nel vettore della visita preorder/postorder.
3. **Ordine di Costruzione**:
   - Pre-order: Radice $\rightarrow$ Sinistra $\rightarrow$ Destra.
   - Post-order: Radice (dal fondo) $\rightarrow$ **Destra** $\rightarrow$ Sinistra.

Strutture Dati Globali

```cpp
struct Node {
    int label, left, right;
    Node(int v) : label(v),
                  left(-1), right
                      (-1) {}
    Node() : label(0),
             left(-1), right(-1)
                 {}
};

// Mappe per accesso rapido
unordered_map<int, int> inIndex;
unordered_map<int, Node> nodes;
```

Perché unordered_map?

- `inIndex`: Sostituisce la ricerca lineare dell'indice nel vettore della visita inorder. In questo modo la complessità totale scende da $O(N^2)$ a $O(N)$.
- `nodes`: Permette di salvare i nodi anche se gli ID non fossero contigui .

Caso 1: Preorder + Inorder

Algoritmo

La radice è all'inizio del Pre-order. Costruiamo ricorsivamente.

```
int buildFromPre(..., int inL, int inR, int& preIdx) {
    if (inL > inR) return -1;

    // 1. Prendi Radice e avanza indice
    int rootVal = preorder[preIdx++];
    // 2. Trova punto di taglio in O(1)
    int mid = inIndex[rootVal];

    // 3. Ricorsione Standard: SX poi DX
    int leftChild  = buildFromPre(..., inL, mid - 1, preIdx);
    int rightChild = buildFromPre(..., mid + 1, inR, preIdx);

    nodes[rootVal].left = leftChild;
    nodes[rootVal].right = rightChild;
    return rootVal;
}
```

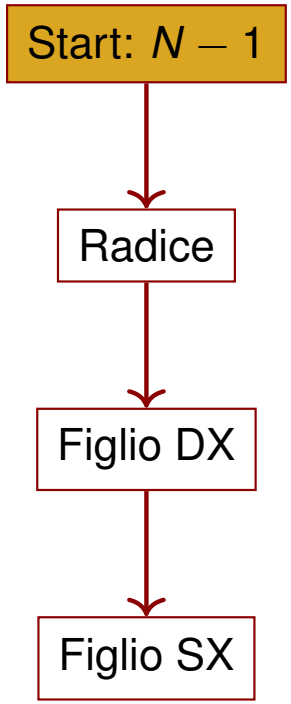
Caso 2: Postorder + Inorder (Il "Trucchetto")

```
int buildFromPost(..., int inL, int
    inR, int& postIdx) {
    if (inL > inR) return -1;

    // 1. Radice è in FONDO
    int rootVal = postorder[postIdx
        --];
    int mid = inIndex[rootVal];

    // 2. IMPORTANTE: Prima DX poi SX!
    int rightChild = buildFromPost
        (..., mid + 1, inR, postIdx);
    int leftChild  = buildFromPost
        (..., inL, mid - 1, postIdx);

    nodes[rootVal].left = leftChild;
    nodes[rootVal].right = rightChild;
    return rootVal;
}
```



Gestione Input e Output

Input

- Leggere N.
- Identificare quale vettore è In-order e quale è Pre/Post.
- Riempire `inIndex` mappa: `inIndex[val] = i`.

Output

- L'output richiede ordine per ID $(0 \ldots N-1)$.
- Le mappe non sono ordinate!
- **Soluzione:** Estrarre le chiavi, ordinare ('sort') e stampare iterando.

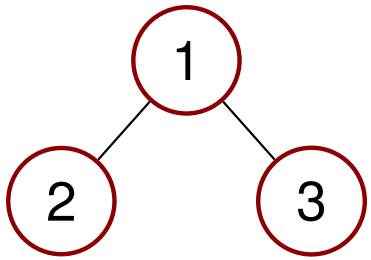
Complessità Totale $O(N)$

Perché è necessaria l'In-Order?

Il problema dell'ambiguità

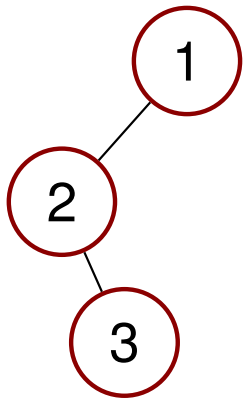
Con solo Pre-order o Post-order, alberi diversi possono produrre la stessa sequenza.

Albero A



Pre: `1, 2, 3`

Albero B



Pre: `1, 2, 3`

Domanda: Il nodo 3 è figlio della radice o del nodo 2?

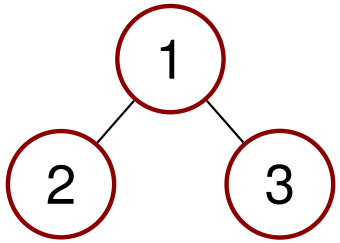
Senza ulteriori informazioni, **non possiamo saperlo**!

L'In-Order risolve l'ambiguità

Il potere separatore dell'In-Order

L'In-order divide univocamente i nodi tra sottoalbero sinistro e destro.

Albero A



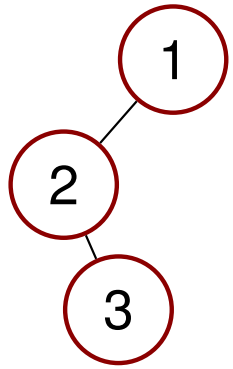
Pre: `1, 2, 3`

In: `2, 1, 3`

Radice 1 separa:

SX: {2}, DX: {3}

Albero B



Pre: `1, 2, 3`

In: `2, 3, 1`

Radice 1 separa:

SX: {2, 3}, DX: {}

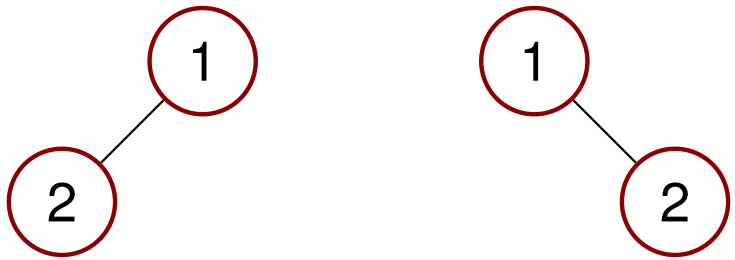
L'In-order elimina ogni ambiguità!

Pre-Order + Post-Order funziona?

Altra combinazione possibile?

Abbiamo Pre (radice all'inizio) e Post (radice alla fine). Basta?

NO! Persiste l'ambiguità per nodi con un solo figlio.



Entrambi: Pre: `1, 2` | Post: `2, 1`

Stessa ambiguità! Il nodo 2 è a sinistra o a destra?

Vincoli alternativi

Quando basta una sola visita?

Serve aggiungere vincoli strutturali forti.

1. Binary Search Tree

Se l'albero è di ricerca:

- $Val < Radice \rightarrow SX$
- $Val > Radice \rightarrow DX$

I valori determinano la struttura!
Basta solo Pre-order o Post-order.

2. Serializzazione con `null`

Marcare esplicitamente i puntatori vuoti:

Pre: `1, 2, #, #, 3, #, #`

I marcatori `#` descrivono completamente la struttura.

Nel nostro problema

Nessun vincolo sulla struttura $\Rightarrow$ **In-order indispensabile!**

FASTEST 100 POINTS SOLUTION

Jannik Sipper

Premiazioni

Fastest 100 Points Solution

Jannik Sipper

Best Team Names

1. **Jannik Sipper**
2. **O(so(n*n)o)**
3. **AssemBRIE**

Premiazioni

Fastest 100 Points Solution

Jannik Sipper

Best Team Names

1. **Jannik Sipper**
2. **O(so(n*n)o)**
3. **AssemBRIE**

Menzioni Speciali

- **Fatine Ricorsine**: per aver ottenuto il maggior numero di compilation error
- **LePatate**: per essere arrivati alla soluzione con una sola sottomissione

Domande?

Università degli Studi di Trento
Dipartimento di Ingegneria e Scienze dell'Informazione

Anno Accademico 2025/2026