Utilizzo dei simboli

1. La freccia `->` :

Nel contesto dei tipi (`Intero, Albero → Albero`): significa:

- Questa funzione prende in ingresso un intero e un albero e restituisce un nuovo albero.

Nel contesto delle variabili (`zii(y, T) → I`):

- Se chiamo la funzione con il nodo `u` e l'albero `T` , il risultato sarà la variabile `I` .

Le utilizziamo per dichiarare le funzioni.

2. Parentesi regolari `<...>`

Si usano per definire una tupla o una coppia ordinata, cioè un oggetto composto da più parti fisse.

Per esempio: `T = <N, A>`

- Un albero (o grafo) non è un numero singolo, è na struttura composta da due insiemi: l'insieme dei nodi (N) e l'insieme degli archi (A).
- Scrivendo `T = <N, A>` , significa: L'albero T è composta dalla ocoppia N, A.

Si utilizzano quando devi descrivere la struttura interna di un oggetto matematico (come l'albero o grafo o record).

3. Il simbolo €

Il simbolo € indica l'appartenenza.

- `u € N` : L'elemento u appartiene a N, u è un nodo dell'albero.
  Da utilizzare quando devi dire che una variabile è un elemento di un insieme.

4. Le parentesi graffe `{...}` e la `|`

Questa è la notazione per costruire gli insiemi:
`I = { x € N | PADRE (x, T) = PADRE(PADRE(u,T),T)}`
Si legge:

- `{` : l'insieme composto da: ...

- `x € N` : tutti i nodi di `x` che appartengono a `N`
- `|` : significa TALI CHE
- `PADRE(x...) = PADRE(PADRE(u...))` : il padre di `x` è uguale a nonno si `u` .
- `}` : Fine dell'insieme.
  Quando il risultato della tua funzione è un insieme di cose (come nel caso degli "zii").
  Se il risultato è un singolo numero o un albero, non usi le graffe.

5. Il meno `-` (differenza tra insiemi)

`... } – {PADRE(u,T)}` : prendi l'insieme che hai calcolato e sottrai l'elemento che sta nel secondo insieme.

- Gli zii sono i fratelli del padre. I fratelli del padre sono tutti i figli del nonno meno il padre stesso.
  Si utilizza quando devi escludere specifici elementi da un gruppo.

6. L'apice `'` (es. `T’`)

`min_a_radice_k(k, T) → T’` : indica lo stato modificato dell'oggetto.

- `T` : è l'albero prima dell'operazione.
- `T'` : è l'albero come diventa dopo l'operazione.
  Si utilizzano nelle funzioni che modificano la struttura dati.
  Serve per distinguere il "prima" dal "dopo".

7. Abbreviazioni logiche `t.c.` , `AND` , `esiste`

Questi sono operatori logici per scrivere le condizioni (PRE e POST). `t.c.` : Sta per "Tale Che". È sinonimo della barretta verticale `|` . Collega una variabile a una condizione che deve rispettare.

- `esiste u ... t.c. livello(u) = k` : Esiste un nodo u tale che il suo livello è k.
- `AND` : Congiunzione logica. Tutte le condizioni devono essere vere contemporaneamente.
- `esiste` / `non esiste` : Quantificatori.
  - `non esiste i ... t.c. leggi(i) < leggi(x)` : È il modo formale per dire che `x` è il minimo.
    Significa: "Non c'è nessuno più piccolo di x".

8. Le parentesi angolari per le sequenze `<a1, ..., an>`

A differenza delle parentesi regolari usate per le coppie fisse (tupla), qui indicano una lista o una sequenza dove l'ordine conta.
`L = <a1, a2, ..., an>` si legge:

- `L` : è la lista risultato.
- `<...>` : indica che è una sequenza ordinata di elementi.
- `a1, a2` : sono gli elementi della lista (in questo caso i nodi figli).

Si utilizza quando l'output è una Lista, una Coda o una Pila, dove la posizione degli elementi è importante (es. `figli_ordinati`).

9. Il simbolo di sequenza vuota `<>`

`POST: se foglia(u,T), allora L = <>` Si legge:

- `L = <>` : La lista L è vuota, non contiene nessun elemento.

Si utilizza per definire i casi base, ad esempio quando un nodo non ha figli o una lista non ha elementi.

10. Le barre verticali per la cardinalità `|...|`

Da non confondere con il "tale che", queste barre racchiudono un insieme intero. `dato m = |{x € N | padre(x,T) = u}|` Si legge:

- `| ... |` : "Il numero di elementi dell'insieme..." (Cardinalità).
- Nel contesto specifico: `m` è uguale al numero totale di nodi `x` che sono figli di `u` .

Si utilizza quando devi contare quanti elementi soddisfano una certa proprietà (es. quanti figli ha un nodo).

11. Gli indici e i range `1 <= i < n`

Utilizzati per scorrere le sequenze o confrontare elementi adiacenti. `per ogni j, 1 <= j < n, leggiNodo(aj,T) <= leggiNodo(a(j+1),T)` Si legge:

- `per ogni j` : scorrendo l'indice `j` .
- `1 <= j < n` : partendo dal primo elemento fino al penultimo.
- `aj` : l'elemento alla posizione `j` .
- `a(j+1)` : l'elemento alla posizione successiva.

Si utilizza per specificare l'ordinamento (verificare che un elemento sia minore o uguale al successivo) o per dire che una proprietà vale per tutti gli elementi di una lista.

12. Il simbolo diverso `!=`

`con i != j, ai != aj` Si legge:

- `i != j` : se gli indici sono diversi...

- `ai != aj` : ...allora anche i nodi devono essere diversi.

Si utilizza per garantire che nella lista non ci siano duplicati (ogni nodo appare una sola volta).

13. La parola chiave `OPPURE`

Utilizzata nelle POST condizioni per gestire casi alternativi. `POST: S' = S U {<d,<>>} OPPURE S' = INSERISCI(...)` Si legge:

- La post-condizione può essere soddisfatta in un modo **oppure** in un altro, a seconda dello stato iniziale o dell'implementazione scelta.

Si utilizza quando l'operazione può avere esiti diversi o può essere descritta in modi equivalenti (es. usare l'operatore unione insiemistica o un operatore funzionale come `INSERISCI`).

Operatori per tenere a mente:

Operatori Aritmetici

Operatore	Nome	Esempio (a=10, b=3)	Risultato	Descrizione
`+`	Addizione	`a + b`	`13`	Somma due operandi.
`-`	Sottrazione	`a - b`	`7`	Sottrae il secondo dal primo.
`*`	Moltiplicazione	`a * b`	`30`	Moltiplica due operandi.
`/`	Divisione	`a / b`	`3`	Divide. **Nota:** Se i numeri sono interi, tronca la virgola (10/3 fa 3, non 3.33).
`%`	Modulo (Resto)	`a % b`	`1`	Restituisce il **resto** della divisione intera (10 diviso 3 fa 3 con resto 1).
`++`	Incremento	`a++` o `++a`	`11`	Aumenta il valore di 1.
`--`	Decremento	`a--` o `--a`	`9`	Diminuisce il valore di 1.

Operatori Relazionali (Confronto)

Operatore	Significato	Esempio (a=10, b=20)	Risultato
`==`	Uguale a	`a == b`	`false`
`!=`	Diverso da	`a != b`	`true`
`>`	Maggiore di	`a > b`	`false`
`<`	Minore di	`a < b`	`true`
`>=`	Maggiore o uguale	`a >= b`	`false`
`<=`	Minore o uguale	`a <= b`	`true`

Operatori Logici (Algebra Booleana)

Operatore	Nome	Esempio	Descrizione
`&&`	AND (E)	`(x > 5) && (x < 10)`	Restituisce `true` solo se **entrambe** le condizioni sono vere.
`		`	OR (O)
`!`	NOT (Non)	`!(x == 5)`	Inverte il risultato: se è vero diventa falso e viceversa.

Operatori di Assegnazione

Operatore	Esempio	Equivalente a	Descrizione
`=`	`x = 5`	-	Assegna il valore 5 a x.
`+=`	`x += 3`	`x = x + 3`	Aggiunge e assegna.
`-=`	`x -= 3`	`x = x - 3`	Sottrae e assegna.
`*=`	`x *= 3`	`x = x * 3`	Moltiplica e assegna.
`/=`	`x /= 3`	`x = x / 3`	Divide e assegna.
`%=`	`x %= 3`	`x = x % 3`	Fa il modulo e assegna.

Operatori di Accesso ai Membri (Fondamentali per Classi e Struct)

Operatore	Nome	Esempio	Descrizione
`.`	Punto	`studente.nome`	Accede a un attributo/metodo di un **oggetto**.
`->`	Freccia	`puntatore->nome`	Accede a un attributo/metodo tramite un **puntatore**. È una scorciatoia per `(*puntatore).nome`.

Operatore	Nome	Esempio	Descrizione
`[]`	Indice	`array[0]`	Accede all'elemento in una specifica posizione di un array o vettore.

Operatori per Puntatori e Memoria

Operatore	Nome	Descrizione
`&`	Indirizzo di (Reference)	`&x` ti dà l'indirizzo di memoria dove si trova `x`.
`*`	Dereferenza (Value at)	`*p` legge il valore contenuto all'indirizzo puntato da `p`.
`new`	Allocazione	Crea un oggetto nella memoria dinamica (Heap).
`delete`	Deallocazione	Distrugge un oggetto creato con `new` per liberare memoria.

Altri Operatori Utili

Operatore	Nome	Esempio	Descrizione
`sizeof`	Dimensione	`sizeof(int)`	Restituisce la grandezza in byte di una variabile o tipo (es. 4 byte per int).
`?:`	Ternario	`(a > b) ? x : y`	Un `if-else` in una riga. Se `a>b` è vero restituisce `x`, altrimenti `y`.
`::`	Risoluzione di scope	`std::cout`	Indica l'appartenenza (es. `cout` appartiene al namespace `std`, oppure una funzione appartiene a una classe).

Algoritmi e complessità

Algoritmo	Caso Ottimo	Caso Medio	Caso Pessimo	Memoria	Note e Funzionamento
--- ORDINAMENTO ---					
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Scambia adiacenti. Lento, ma $O(n)$

Algoritmo	Caso Ottimo	Caso Medio	Caso Pessimo	Memoria	Note e Funzionamento
					se già ordinato.
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Inserisce carte in mano. Ottimo per n piccoli o quasi ordinati.
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Cerca sempre il minimo. Lento anche se ordinato.
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Divide et Impera. Stabile, ma usa memoria extra per gli array.
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	Divide tramite Pivot. Il più veloce in pratica, ma $O(n^2)$ se pivot scelto male.
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	Usa la struttura Heap. Costante e in-place, ma non stabile.
Tree Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$	Inserisce tutto in un BST e fa visita simmetrica. Lento se BST sbilanciato.
--- RICERCA ---					
Ricerca Lineare	$O(1)$	$O(n)$	$O(n)$	$O(1)$	Scorre tutto. Unica opzione per liste non ordinate.
Ricerca Binaria	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$	Divide a metà. **Solo** su

Algoritmo	Caso Ottimo	Caso Medio	Caso Pessimo	Memoria	Note e Funzionamento
					array ordinati.
Hashing (Chaining)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	Accesso diretto tramite chiave. Pessimo se collisioni totali.
BST (Ricerca/Ins)	$O(1)$	$O(\log n)$	$O(n)$	$O(h)$	Efficienza dipende dal bilanciamento dell'albero.
--- STRUTTURE DATI & GRAFI ---					
Visita Alberi (DFS/BFS)	$O(n)$	$O(n)$	$O(n)$	$O(h)$	Visita completa. Costo lineare sul numero di nodi.
Heapify (Costruzione)	$O(n)$	$O(n)$	$O(n)$	$O(1)$	Algoritmo per trasformare un array in Heap.
Bilanciamento AVL	$O(1)$	$O(1)$	$O(1)$	$O(1)$	Rotazioni (LL, RR, LR, RL) per correggere l'albero.
Parentesi Bilanciate	$O(n)$	$O(n)$	$O(n)$	$O(n)$	Usa una Pila. Una passata sulla stringa.
--- MATEMATICA ---					
Euclide (MCD)	-	Logaritmico	-	$O(1)$	Molto efficiente per trovare il divisore comune.
Fibonacci (Ricorsivo)	-	$O(2^n)$	$O(2^n)$	$O(n)$	Costo esponenziale senza ottimizzazioni (memoization).

Gerarchia delle Complessità

Notazione	Nome	Esempio Tipico	Note
$O(1)$	Costante	Accesso array, Hash Map ideale	Il tempo non cambia mai. Il migliore assoluto.
$O(\log n)$	Logaritmico	Ricerca Binaria, Alberi Bilanciati	Cresce lentissimamente. Ottimo.
$O(n)$	Lineare	Scansione lista, Visita albero	Cresce pari ai dati. Standard.
$O(n \log n)$	Linearitmico	Merge Sort, Heap Sort, Quick Sort	Il miglior tempo possibile per ordinare basandosi su confronti.
$O(n^2)$	Quadratico	Bubble Sort, Doppio ciclo annidato	Lento. Accettabile solo per n piccoli.
$O(2^n)$	Esponenziale	Fibonacci ricorsivo, Enumerazione	**Disastroso**. Il tempo esplode. Da evitare.

Pattern "Pronti all'Uso" per PRE/POST Condizioni

Frasi logiche standard da copiare/incollare per descrivere situazioni comuni negli esercizi di specifica (come quello del Social Network o dell'Albero).

1. Dire che tutti gli elementi sono unici (niente duplicati):

> `per ogni i, j con 1 <= i, j <= n: se i != j allora a_i != aj`

2. Dire che una lista/array è ordinata (crescente):

> `per ogni i con 1 <= i < n: ai <= a{i+1}`

3. Dire che "x" è il massimo nell'insieme S:

> `x ∈ S AND (non esiste y ∈ S t.c. y > x)`

4. Sommare valori (es. contare articoli o pesi):

> `Totale = Sommatoria per ogni x ∈ S di valore(x)`

5. Definire un cammino tra due nodi u e v (Grafi):

> `esiste una sequenza <n1, ..., nk> t.c. n1 = u AND nk = v AND per ogni i esisteArco(ni, n{i+1})`

Glossario Rapido: Proprietà di Alberi e Grafi

Definizioni che servono spesso per le condizioni (es. "se il nodo è una foglia...").

- **Radice:** L'unico nodo che non ha padre (`padre(u) = null` o non definito).
- **Foglia:** Un nodo che non ha figli (`primofiglio(u) = null` o insieme figli vuoto).
- **Grado di un nodo:** Il numero di figli che ha (o numero di archi collegati).
- **Livello/Profondità:** Distanza dalla radice (Radice = livello 0).
- **Altezza dell'albero:** Il livello massimo raggiungibile (il cammino più lungo dalla radice a una foglia).
- **Albero n-ario:** Ogni nodo può avere un numero indefinito di figli.
- **Albero Binario:** Ogni nodo ha al massimo 2 figli (sinistro e destro).

Simbologia Base Ricorrente

Prima delle definizioni, ricorda che:

- **T**: Indica l'Albero corrente (Tree).
- **u,v,z**: Indicano generici **nodi** dell'albero ($u \in N$).
- **N**: È l'insieme di tutti i nodi dell'albero.
- **padre(u)**: La funzione che restituisce il genitore del nodo u.
- **NIL / ø /** `null` : Indica l'assenza di un nodo (es. il padre della radice o il figlio di una foglia).

Glossario: Proprietà di Alberi e Grafi (con notazione)

- **Radice (r o root):**
  - L'unico nodo che non ha un padre. È il punto di ingresso dell'albero.
  - **Condizione:** $padre(u)=NIL$ (oppure $u=radice(T)$).
  - *In codice:* Solitamente accessibile con `T.radice()` .
- **Foglia (u):**
  - Un nodo che non ha figli (o successori).
  - **Condizione (Albero n-ario):** $primofiglio(u)=NIL$ (la lista dei suoi figli è vuota).
  - **Condizione (Albero Binario):** $sinistro(u)=NIL \wedge destro(u)=NIL$.
  - *Nota:* In molti esercizi ricorsivi, le foglie rappresentano uno dei **casi base**.
- **Nodo Interno:**
  - Qualsiasi nodo che non è né la radice (solitamente) né una foglia. Ha almeno un figlio.
  - **Condizione:** primofiglio(u)ˆI=NIL.
- **Grado di un nodo (deg(u)):**

    - Il numero di sotto-alberi (figli diretti) che possiede u.
    - **Condizione:** Se $deg(u)=0$, allora u è una foglia.
    - *Esempio:* In un albero binario, il grado può essere solo 0, 1 o 2.
- **Livello (liv(u)) o Profondità:**
    - La distanza del nodo u dalla radice (numero di archi da attraversare).
    - **Definizione ricorsiva:**
        - Se $u=radice(T) \rightarrow liv(u)=0$.
        - Altrimenti $\rightarrow liv(u)=liv(padre(u))+1$.
    - *Utilità:* Serve per esercizi tipo "stampa nodi a livello k".
- **Altezza dell'albero (h(T)):**
    - Il massimo livello raggiungibile tra tutti i nodi dell'albero (il cammino più lungo radice-foglia).
    - **Formula:** $h(T)=\max\{liv(u) \mid u \in N\}$.
    - *Nota:* Un albero con solo la radice ha altezza 0. Un albero vuoto ha altezza -1 (convenzione comune).
- **Albero n-ario (Generico):**
    - Ogni nodo u può avere un numero indefinito di figli.
    - **Navigazione:** Si usa la logica "Primo Figlio - Fratello Successivo".
        - $v=primofiglio(u)$ (scende di livello).
        - $w=succfratello(v)$ (scorre allo stesso livello).
- **Albero Binario:**
    - Ogni nodo u ha al massimo 2 figli, distinti in **sinistro** e **destro**.
    - **Navigazione:**
        - $sx=sinistro(u)$
        - $dx=destro(u)$
    - *Tipico:* Alberi di Ricerca (BST) o Heap.
- **Fratelli (Siblings):**
    - Due nodi u e v sono fratelli se hanno lo stesso padre.
    - **Condizione:** $padre(u)=padre(v)$.
    - *In codice n-ario:* v è raggiungibile da u chiamando ripetutamente `succfratello`.