

# ALBERI BINARI GENERICI - ESERCIZI

## 1. MAXPATH - CAMMINO DI COSTO MASSIMO

**Problema:** Dato albero binario T, calcolare il costo massimo di un cammino radice-foglia, dove il costo è la somma delle chiavi lungo il cammino.

### Algoritmo Ricorsivo

```
MaxPath(T)
    if T.root == nil
        return 0
    return MaxPathRec(T.root)

MaxPathRec(x)
    if x == nil
        return 0

    if x.left == nil and x.right == nil
        return x.key // foglia

    maxLeft = MaxPathRec(x.left)
    maxRight = MaxPathRec(x.right)

    return x.key + max(maxLeft, maxRight)
```

### Correttezza

#### Caso base:

- Se x è nil, costo = 0
- Se x è foglia, costo = x.key

#### Caso ricorsivo:

- Calcola ricorsivamente max costo per sottoalbero sinistro
- Calcola ricorsivamente max costo per sottoalbero destro
- Somma chiave corrente al massimo tra i due

#### Induzione sull'altezza:

- **Base** ( $h=0$ , foglia): return x.key ✓
- **Passo:** Se MaxPathRec corretto per altezza  $< h$ , allora:
  - $\text{maxLeft} = \text{costo massimo sottoalbero sinistro (altezza } < h\text{)}$
  - $\text{maxRight} = \text{costo massimo sottoalbero destro (altezza } < h\text{)}$

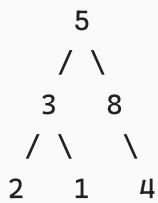
- $x.key + \max(\maxLeft, \maxRight) = \text{costo massimo da } x \checkmark$

## Complessità

$\Theta(n)$

- Ogni nodo visitato esattamente una volta
- Operazioni per nodo:  $O(1)$  (max e somma)
- Totale:  $\Theta(n)$

## Esempio



$\text{MaxPath}(3) = 3 + \max(2, 1) = 5$   
 $\text{MaxPath}(8) = 8 + 4 = 12$   
 $\text{MaxPath}(5) = 5 + \max(5, 12) = 17$

## 2. LEVEL - CONTA NODI CON CHIAVE $\leq$ LIVELLO

**Problema:** Contare quanti nodi  $x$  dell'albero hanno proprietà  $x.key \leq \text{livello}(x)$ , dove  $\text{livello}(\text{radice}) = 0$ .

## Algoritmo Ricorsivo

```

Level(T)
  if T.root == nil
    return 0
  return LevelRec(T.root, 0)

LevelRec(x, level)
  if x == nil
    return 0

  count = 0
  if x.key <= level
    count = 1

  countLeft = LevelRec(x.left, level + 1)
  countRight = LevelRec(x.right, level + 1)
  
```

```
return count + countLeft + countRight
```

## Correttezza

**Invariante:** LevelRec(x, level) conta nodi in sottoalbero di x che soddisfano proprietà, assumendo x sia a profondità level.

**Caso base:** x = nil  $\rightarrow$  return 0 ✓

**Caso ricorsivo:**

1. Verifica se x soddisfa proprietà ( $x.key \leq level$ )
2. Ricorri su figli con level+1
3. Somma risultati

## Complessità

$\Theta(n)$

- Visita DFS di tutto l'albero
- Ogni nodo visitato una volta
- Operazioni per nodo:  $O(1)$

## Esempio



Livello 0: radice=4,  $4 \leq 0$ ? NO  
Livello 1: nodo=2,  $2 \leq 1$ ? NO | nodo=6,  $6 \leq 1$ ? NO  
Livello 2: nodo=1,  $1 \leq 2$ ? SÌ | nodo=3,  $3 \leq 2$ ? NO

Totale: 1 nodo soddisfa proprietà

## 3. 1-BILANCIATO

**Problema:** Verificare se albero è 1-bilanciato: tutti i cammini terminabili dalla radice hanno lunghezze che differiscono al più di 1.

**Cammino terminabile:** Sequenza nodi  $x_0, x_1, \dots, x_n$  con  $x_{i+1}.p = x_i$  e  $x_n.left = \text{nil}$  OPPURE  $x_n.right = \text{nil}$ .

## Algoritmo

```
bal1(T)
    if T.root == nil
        return true

    (minLen, maxLen) = computeLengths(T.root)
    return (maxLen - minLen <= 1)

computeLengths(x)
    if x == nil
        return (0, 0)

    // Se nodo terminabile (almeno un figlio nil)
    if x.left == nil or x.right == nil
        isTerminable = true
    else
        isTerminable = false

    (minLeft, maxLeft) = computeLengths(x.left)
    (minRight, maxRight) = computeLengths(x.right)

    // Calcola min e max per sottoalbero corrente
    if x.left == nil and x.right == nil
        // Foglia: cammino lunghezza 1
        minLen = 1
        maxLen = 1
    else if x.left == nil
        minLen = 1 // cammino termina qui
        maxLen = 1 + maxRight
    else if x.right == nil
        minLen = 1 // cammino termina qui
        maxLen = 1 + maxLeft
    else
        // Entrambi figli presenti
        minLen = 1 + min(minLeft, minRight)
        maxLen = 1 + max(maxLeft, maxRight)

    return (minLen, maxLen)
```

## Definizione Precisa Cammino Terminabile

Un cammino  $x_0, x_1, \dots, x_n$  è terminabile se:

1.  $x_0$  = radice
2.  $\forall i: x_{i+1}$  è figlio di  $x_i$
3.  $x_n$  ha almeno un figlio nil ( $x_n.left = nil$  OR  $x_n.right = nil$ )

**Nota:** Non necessariamente foglia! Nodo con un solo figlio è terminabile.

## Correttezza

**Invariante:** computeLengths(x) restituisce (min, max) dove:

- min = lunghezza minima cammino terminabile da x
- max = lunghezza massima cammino terminabile da x

**Caso base:**

- Foglia: minLen = maxLen = 1 ✓
- Nodo con un figlio nil: min = 1 (termina qui), max = 1 + max sottoalbero altro figlio ✓

**Caso ricorsivo:**

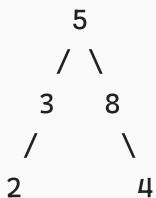
- Se entrambi figli: min = 1 + min(minLeft, minRight), max = 1 + max(maxLeft, maxRight)  
✓

## Complessità

**O(n)**

- Visita DFS di tutto l'albero
- Ogni nodo visitato una volta
- Operazioni per nodo: O(1)

## Esempio 1-bilanciato

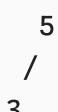


Cammini terminabili:

- 5→3→2: lunghezza 3 (2 è foglia)
- 5→8→4: lunghezza 3 (4 è foglia)
- 5→3: lunghezza 2 (3 ha figlio destro nil)
- 5→8: lunghezza 2 (8 ha figlio sinistro nil)

min = 2, max = 3, diff = 1 ≤ 1 → 1-BILANCIATO ✓

## Esempio NON 1-bilanciato



```
/  
2
```

Cammini terminabili:

- 5→3→2: lunghezza 3
- 5: lunghezza 1 (figlio destro nil)

```
min = 1, max = 3, diff = 2 > 1 → NON 1-bilanciato x
```

## 4. K-BOUNDED

**Problema:** Verificare se albero è k-bounded: per ogni nodo  $x$ , la somma delle chiavi lungo ogni cammino da  $x$  a una foglia nel suo sottoalbero è  $\leq k$ .

### Algoritmo Ricorsivo

```
k_bound(T, k)
    if T.root == nil
        return true
    return k_boundRec(T.root, k)

k_boundRec(x, k)
    if x == nil
        return true

    // Calcola somma massima cammino da x a foglia
    maxPathSum = maxSumPath(x)

    if maxPathSum > k
        return false

    // Ricorri sui figli
    return k_boundRec(x.left, k) and k_boundRec(x.right, k)

maxSumPath(x)
    if x == nil
        return 0

    if x.left == nil and x.right == nil
        return x.key // foglia

    maxLeft = maxSumPath(x.left)
    maxRight = maxSumPath(x.right)

    return x.key + max(maxLeft, maxRight)
```

**Problema:** Questo è  $O(n^2)$ ! maxSumPath costa  $O(n)$  ed è chiamato per ogni nodo.

## Soluzione Efficiente $O(n)$

```
k_bound(T, k)
    if T.root == nil
        return true
    return k_boundRec(T.root, k) != -1

k_boundRec(x, k)
    // Restituisce maxPathSum se k-bounded, -1 altrimenti

    if x == nil
        return 0

    if x.left == nil and x.right == nil
        // Foglia
        if x.key <= k
            return x.key
        else
            return -1

    maxLeft = k_boundRec(x.left, k)
    maxRight = k_boundRec(x.right, k)

    if maxLeft == -1 or maxRight == -1
        return -1 // sottoalbero viola proprietà

    maxPathSum = x.key + max(maxLeft, maxRight)

    if maxPathSum <= k
        return maxPathSum
    else
        return -1
```

## Correttezza

**Invariante:**  $k\_boundRec(x, k)$  restituisce:

- $\text{maxPathSum}$  del sottoalbero di  $x$  se è  $k$ -bounded
- -1 se il sottoalbero viola la proprietà

**Caso base:**

- $x = \text{nil}$ : return 0 (nessun cammino) ✓
- Foglia: return  $x.\text{key}$  se  $\leq k$ , altrimenti -1 ✓

**Caso ricorsivo:**

1. Verifica ricorsivamente sottoalberi sinistro e destro
2. Se uno viola proprietà → return -1
3. Calcola  $\text{maxPathSum} = \text{x.key} + \max(\text{maxLeft}, \text{maxRight})$
4. Se  $\text{maxPathSum} \leq k \rightarrow$  return  $\text{maxPathSum}$ , altrimenti -1

## Complessità

$O(n)$

- Ogni nodo visitato esattamente una volta
- Operazioni per nodo:  $O(1)$
- Una sola visita DFS

## Esempio $k=10$



$k = 10$

Cammini da radice:

- $3 \rightarrow 2 \rightarrow 1$ : somma =  $6 \leq 10$  ✓
- $3 \rightarrow 2 \rightarrow 4$ : somma =  $9 \leq 10$  ✓
- $3 \rightarrow 5$ : somma =  $8 \leq 10$  ✓

Cammini da nodo 2:

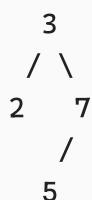
- $2 \rightarrow 1$ : somma =  $3 \leq 10$  ✓
- $2 \rightarrow 4$ : somma =  $6 \leq 10$  ✓

Cammini da nodo 5:

- $5$ : somma =  $5 \leq 10$  ✓ (foglia)

TUTTI  $\leq 10 \rightarrow$  k-bounded ✓

## Esempio NON k-bounded ( $k=8$ )



$k = 8$

Cammini da nodo 7:

-  $7 \rightarrow 5$ : somma = 12 > 8 x

Viola proprietà  $\rightarrow$  NON k-bounded

## RIEPILOGO COMPLESSITÀ

Problema	Complessità	Tecnica
MaxPath	$\Theta(n)$	DFS ricorsivo
Level	$\Theta(n)$	DFS con parametro livello
1-bilanciato	$O(n)$	DFS calcola min/max
k-bounded	$O(n)$	DFS con verifica integrata

## Note Importanti

1. **MaxPath**: Classico problema di ottimizzazione su alberi
2. **Level**: Parametro esplicito per tracciare profondità
3. **1-bilanciato**: Definizione più generale di bilanciamento (non solo foglie)
4. **k-bounded**: Verifica locale ad ogni nodo (non solo dalla radice)

## Pattern Comune

Tutti questi esercizi seguono pattern DFS ricorsivo:

1. **Caso base**: gestione nil e foglie
2. **Caso ricorsivo**:
  - Ricorri su sottoalberi
  - Combina risultati
  - Verifica proprietà corrente
3. **Complessità**:  $O(n)$  con visita singola

## Differenze Chiave

- **MaxPath**: aggrega con max e somma
- **Level**: conta con somma condizionale
- **1-bilanciato**: propaga intervallo (min, max)
- **k-bounded**: verifica con propagazione fallimento (-1)