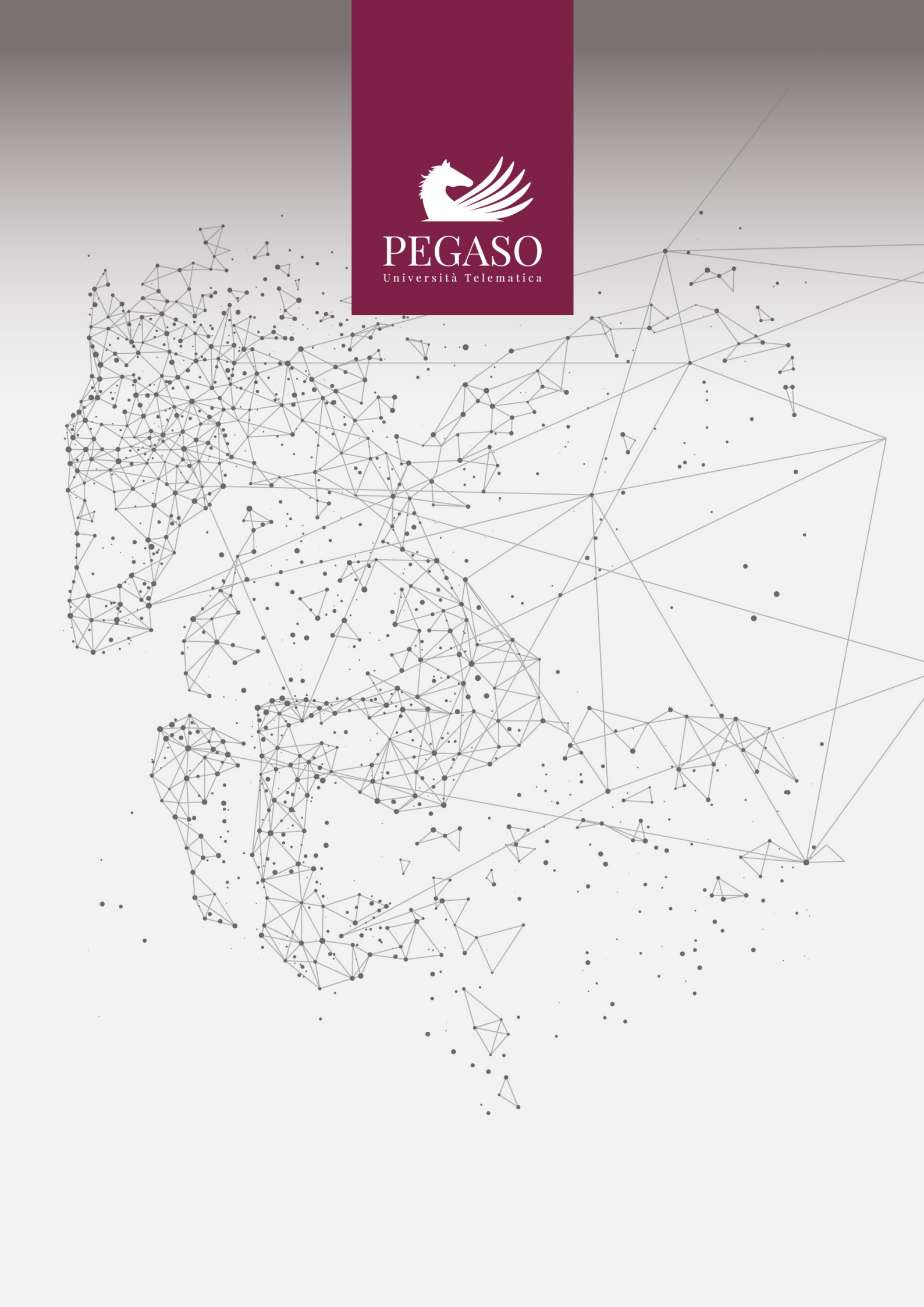
PEGASO
Università Telematica

# Indice

# 1. Albero binario

Un albero consiste di un insieme di **nodi** e un insieme di **archi** orientati che connettono coppie di nodi, con le seguenti proprietà:

- Un nodo dell'albero è designato come nodo **radice** (**root**)
- Ogni nodo n, a parte la radice, ha esattamente un arco entrante
- Esiste un **cammino** unico dalla radice ad ogni nodo
- Ogni nodo che non presenta archi uscenti è detto **foglia** (**leaf**)
- L'albero è **connesso** (è connesso se per ogni coppia di nodi x, y, esiste un cammino da x ad y)

Un albero è dato da:

- un insieme vuoto, oppure
- un nodo radice e zero o più **sottoalberi**, ognuno dei quali è un albero; la radice è connessa alla radice di ogni sottoalbero con un arco orientato.

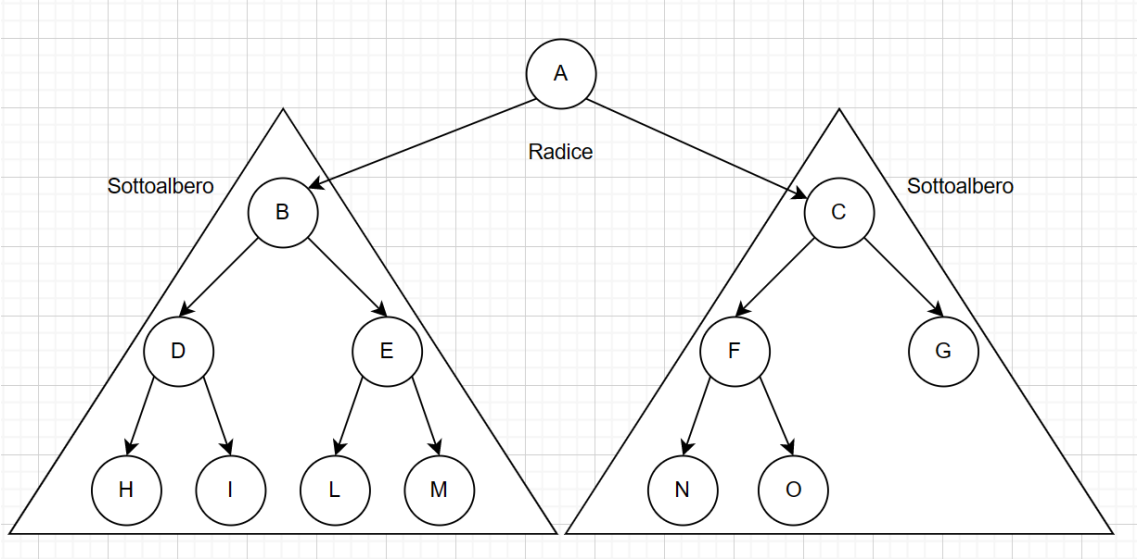


**Figura 1:** Radice e sottoalberi

Intendiamo per proprietà di un albero le sue caratteristiche che possono essere misurate o quantificate. Riportiamo di seguito alcune caratteristiche di cui si deve tener conto quando si parla di "alberi":

- **Radice**: è il nodo principale che non ha genitori
- **Foglia**: è un nodo che non ha figli.
- **Profondità di un nodo (Depth)**: la lunghezza del cammino semplice dalla radice al nodo, cioè il numero di archi che separano il nodo dalla radice dell'albero

- **Livello (Level)**: il livello di un nodo è il numero di livelli tra la radice dell'albero e il nodo stesso; il livello della radice è zero e il livello di ogni altro nodo è uno in più rispetto alla profondità l'insieme di nodi alla stessa profondità
- **Altezza albero (Height)**: la profondità massima delle sue foglie, cioè il numero massimo di archi tra la radice dell'albero e una delle sue foglie
- **Grado**: è il numero di sottoalberi del nodo, che è uguale al numero di figli del nodo

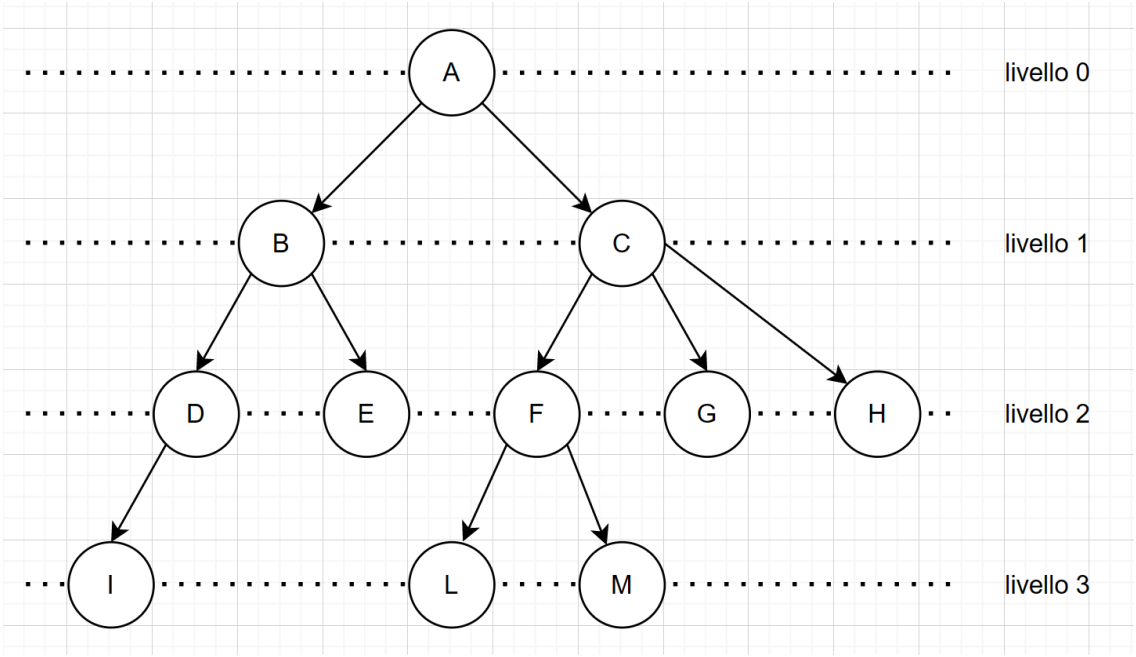


**Figura 2:** Livelli in un albero

Un albero binario è un albero radicato in cui ogni nodo ha al massimo due figli, identificati come figlio sinistro e figlio destro.

Oltre al "classico" albero binario indicato di seguito:

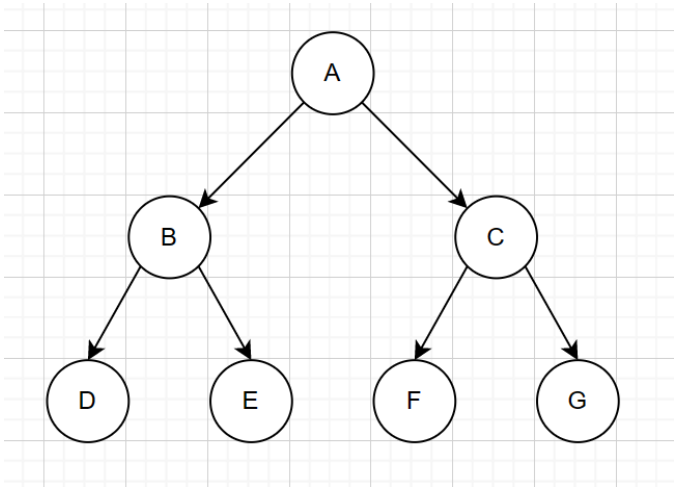


**Figura 3:** Albero binario

Anche i seguenti sono alberi binari:

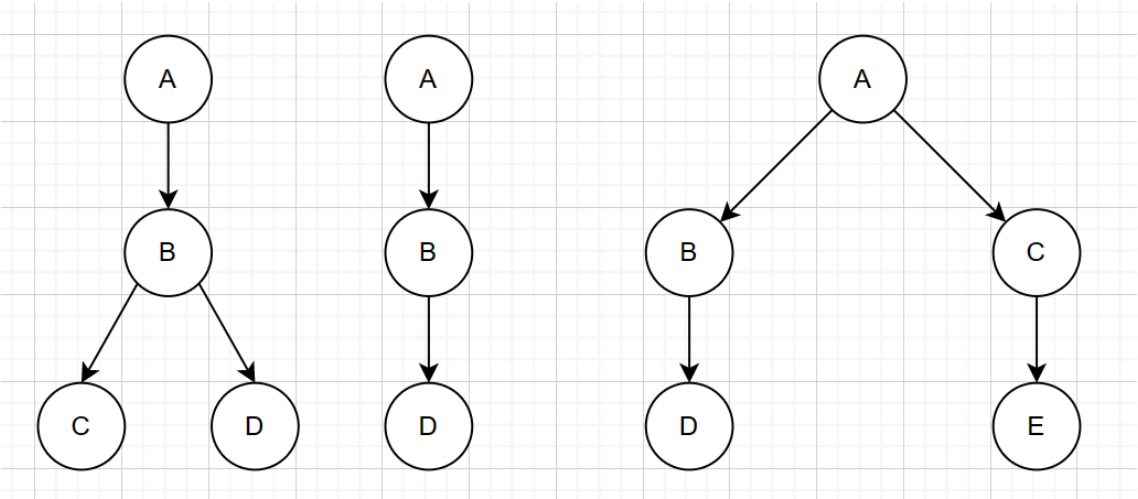


**Figura 4:** Esempi di alberi binari

# 2. Implementazione in C

Immaginiamo di voler costruire una struttura dati che rappresenti il generico nodo dell'albero; tale struttura dovrà avere i seguenti campi:

- Valore memorizzato nel nodo (**item**)
- Riferimento al padre (**parent**)
- Riferimento al figlio sinistro (**left**)
- Riferimento al figlio destro (**right**)

Nel momento in cui viene creato dunque un nuovo albero binario si dovrà creare un nodo "radice" che non avrà parent valorizzato (e continuerà a non averlo valorizzato anche successivamente essendo un nodo radice), avrà un item settato e figlio sinistro e destro inizialmente nulli (successivamente si potranno "agganciare" i figli alla radice).

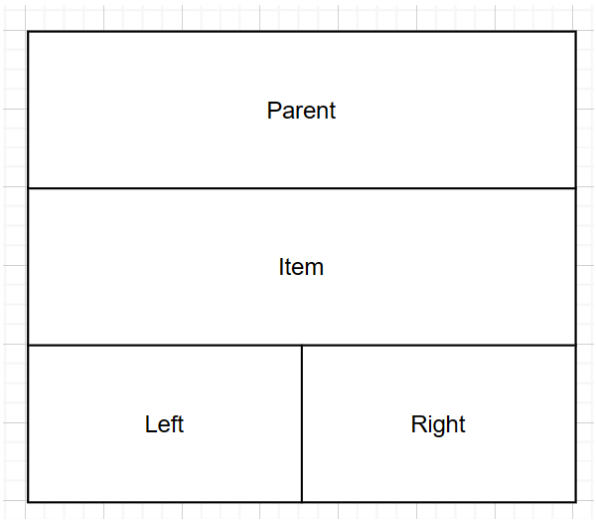


**Figura 5:** Generico Nodo

Il generico Nodo avrà dunque la seguente struttura:

```
struct node {
  int data;
  struct node *left;
  struct node *right;
  struct node *parent;
};
```

La dichiarazione del nostro albero sarà:

```
struct node *root = NULL;
```

La creazione del nodo sarà come di seguito:

```
struct node* create_node(int data, struct node *parent) {
 struct node *new_node = (struct node*)malloc(sizeof(struct node));
 new_node->data = data;
 new_node->left = NULL;
 new_node->right = NULL;
 new_node->parent = parent;
 return new_node;
}
```

Relativamente all'inserito, consideriamo il seguente codice:

```
void insert(struct node **root, int data) {
 if (*root == NULL) {
  *root = create_node(data, NULL);
 } else {
  struct node *temp = *root;
  if (data <= temp->data) {
   if (temp->left == NULL) {
    temp->left = create_node(data, temp);
   } else {
    insert(&temp->left, data);
   }
  } else {
   if (temp->right == NULL) {
    temp->right = create_node(data, temp);
   } else {
    insert(&temp->right, data);
   }
  }
 }
```

La logica di inserimento è la seguente: la radice dell'albero è rappresentata da un puntatore alla struttura node che viene inizializzato come NULL e viene modificato nel metodo insert. Quando si esegue una insert, il valore viene inserito nel primo nodo vuoto disponibile a sinistra o a destra del nodo corrente, a seconda se il valore da inserire è minore o maggiore del valore corrente.

```
insert(&root, 5);
insert(&root, 3);
insert(&root, 8);
insert(&root, 1);
insert(&root, 4);
insert(&root, 6);
insert(&root, 9);
```

Questo determina il seguente albero:

```
    5
  /  \
 3    8
/\   /\
1 46  9
```

È inoltre possibile usare le funzioni di insertLeft ed insertRight per inserire il nuovo valore a sinistra o destra del nodo genitore:

```
void insertLeft(struct node *parent, int value) {
   struct node *newNode = (struct node*) malloc(sizeof(struct node));
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
   parent->left = newNode;
}

void insertRight(struct node *parent, int value)
{
   struct node *newNode = (struct node*) malloc(sizeof(struct node));
```

```
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    parent->right = newNode;
}
```

Il seguente codice determinerebbe il seguente risultato:

```
struct node *root = NULL;
insert(&root, 5);
insertLeft(root, 3);
insertRight(root, 8);
insertLeft(root->left, 1);
insertRight(root->left, 4);
insertLeft(root->right, 6);
insertRight(root->right, 9);
```

```
    5
   / \
  3   8
 / \ / \
1  4 6  9
```

È possibile avere una funzione che stampi l'albero in maniera gerarchica nella seguente modalità:

```
void print_tree(struct node *root, int space) {
    if (root == NULL)
        return;
    space += 10;
    print_tree(root->right, space);
    for (int i = 10; i < space; i++)
        printf(" ");
    printf("%d\n", root->data);
    print_tree(root->left, space);
}
```

Con il seguente codice:

```
struct node *root = NULL;
insert(&root, 5);
insertLeft(root, 3);
insertRight(root, 8);
insertLeft(root->right, 6);
insertRight(root->right, 9);
print_tree(root,0);
```

L'output grafico sarebbe il seguente:

```
        9
    8
        6
5
    3
```

Con il seguente codice:

```
struct node *root = NULL;
insert(&root, 1);
insertLeft(root, 10);
insertRight(root, 11);
insertLeft(root->left, 100);
insertRight(root->left, 101);
insertLeft(root->right, 110);
insertRight(root->right, 111);
insertLeft(root->left->left, 1000);
insertRight(root->left->left, 1001);
insertLeft(root->left->right, 1001);
insertRight(root->right->left, 1110);
insertRight(root->right->right, 1111);
print_tree(root,0);
```

L'output grafico sarebbe il seguente:

```
                1111
            111
        11
                1110
            110
1
            101
                1001
        10
                1001
            100
                1000
```

Un'alternativa di visualizzazione può essere la seguente:

```
void print_tree_graphical(struct node *root, int level) {
   if (root == NULL)
      return;
   printf("%*s%d\n", level * 2, "", root->data);
   if (root->left != NULL) {
      for (int j = 0; j <= level * 2; j++) {
         printf(" ");
      }
      printf("|--");
      print_tree_graphical(root->left, level + 1);
   }
   if (root->right != NULL) {
      for (int j = 0; j <= level * 2; j++) {
         printf(" ");
      }
      printf("|--");
      print_tree_graphical(root->right, level + 1);
   }
}
```

L'output grafico sarebbe il seguente:

**1**

**|-- 10**

**|-- 100**

**|-- 1000**

**|-- 1001**

**|-- 101**

**|-- 1001**

**|-- 11**

**|-- 110**

**|-- 1110**

**|-- 111**

**|-- 1111**

# 3. Implementazione in Python

Di seguito l'implementazione di un albero binario in Python:

```
class Node:
  def __init__(self, value, left=None, right=None):
    self.value = value
    self.left = left
    self.right = right

class BinaryTree:
  def __init__(self, root=None):
    self.root = root

def print_tree(node, prefix=""):
  if not node:
    return
  print(prefix + str(node.value))
  prefix += "|  "
  print_tree(node.left, prefix)
  print_tree(node.right, prefix)

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
```

# Bibliografia


- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione;
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione;
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley;
- Sedgewick: Algoritmi in C, Pearson, 2015;
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione.