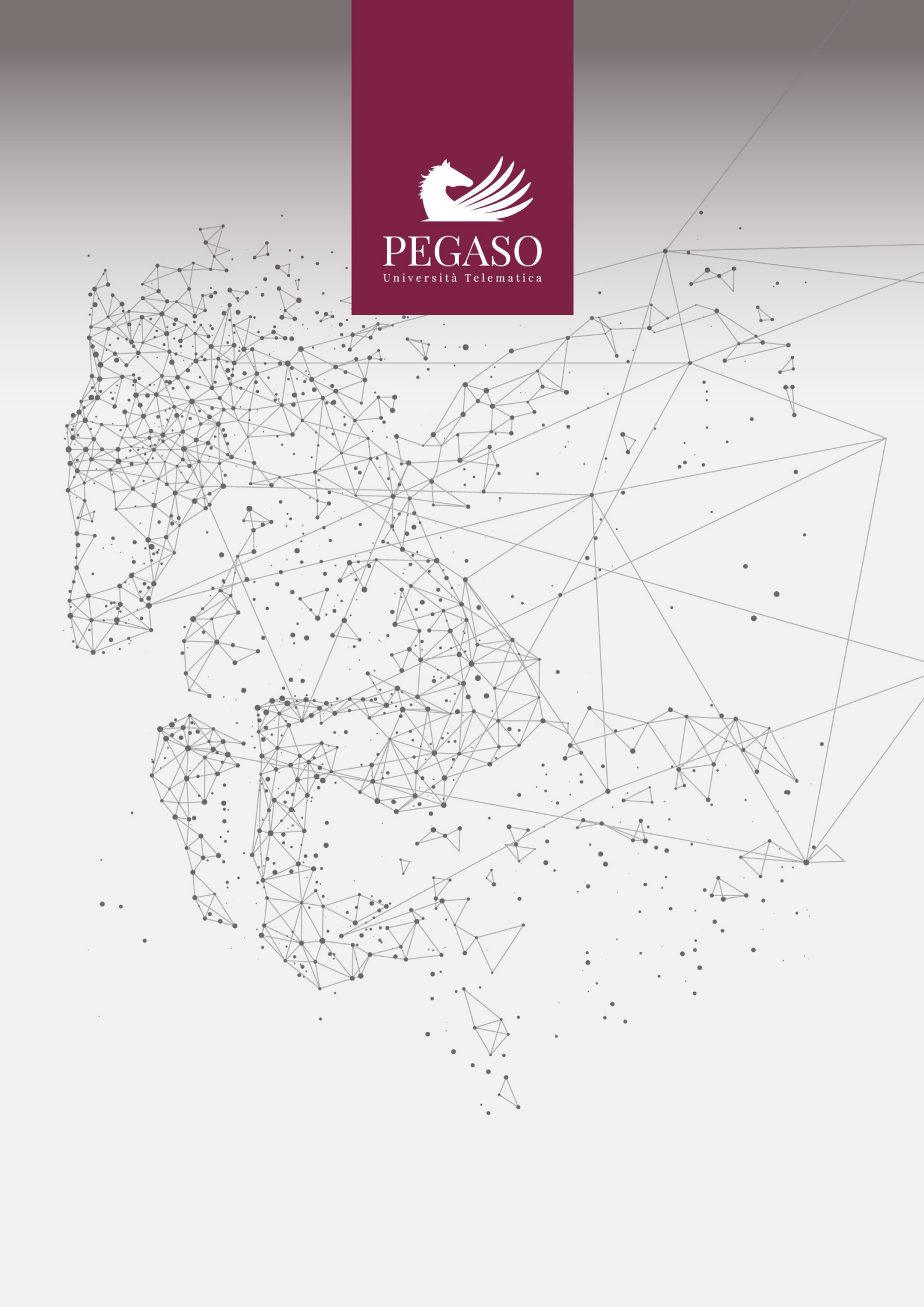
PEGASO
Università Telematica

Indice

1. Visita di un 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.

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

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

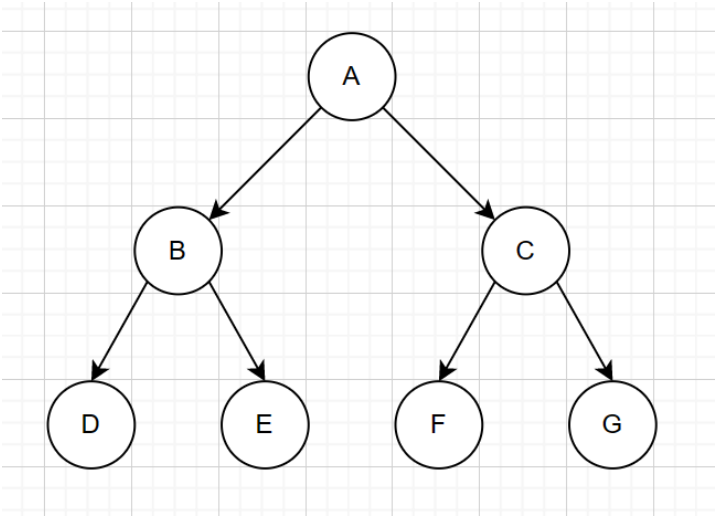

Figura 1: Albero binario

Un albero può essere attraversato (visitato) in vari modi. Distinguiamo tra:

- Visita in profondità – **Depth-First Search (DFS)**
- Visita in ampiezza – **Breadth First Search (BFS)**

La differenza principale tra la Visita in Profondità (DFS) e la Visita in Ampiezza (BFS) è l'ordine con cui i nodi dell'albero vengono visitati:

- la DFS esplora prima i nodi più profondi dell'albero
- la BFS esplora prima i nodi più vicini alla radice

La DFS utilizza uno **stack** per tenere traccia dei nodi da visitare, mentre la BFS utilizza una **coda**. Questo comporta una differenza nella strategia di esplorazione, con la DFS che esplora profondamente ogni ramo prima di passare al prossimo, mentre la BFS esplora prima tutti i nodi ad un certo livello prima di scendere a quello successivo.

Entrambe le strategie hanno le loro applicazioni a seconda delle esigenze dell'algoritmo o della situazione specifica. Ad esempio, la BFS è spesso utilizzata per la risoluzione di problemi di ricerca in larghezza, mentre la DFS è utilizzata per la risoluzione di problemi di ricerca in profondità, come la ricerca di un cammino minimo in un grafo.

In relazione alla **DFS** la strategia prevede dunque:

- La visita ricorsiva di ognuno dei sottoalberi mediante 3 possibili varianti: pre-ordine, in-ordine e post-ordine
- L'utilizzo di uno stack

In relazione alla **BFS** la strategia prevede dunque:

- La visita di ogni livello dell'albero, uno dopo l'altro, partendo dalla radice
- L'utilizzo di una queue

2. Visita in profondità (DFS)

Distinguiamo tra 3 strategie di visita:

- Visita in ordine simmetrico (in-order)
- Visita in ordine anticipato (pre-order)
- Visita in ordine posticipato (post-order)

Consideriamo il seguente albero ed analizziamo le 3 strategie:

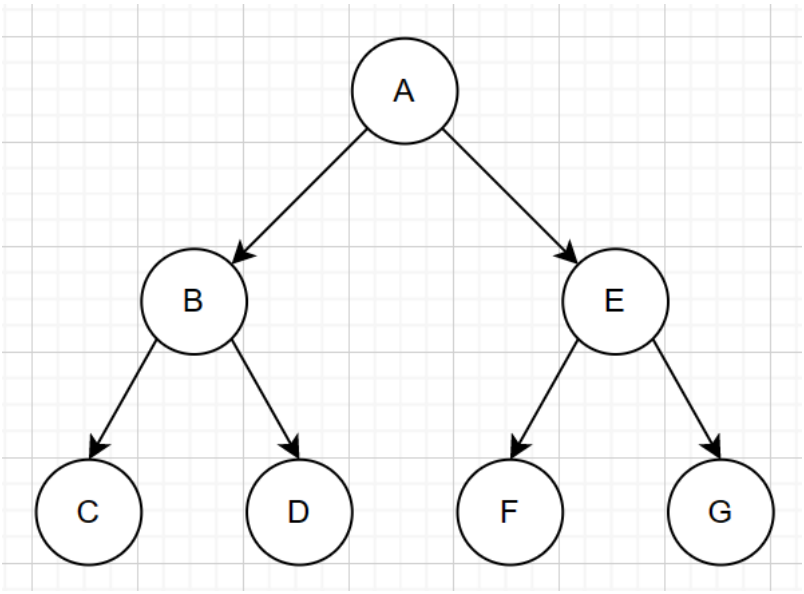

Figura 2: Albero binario di analisi

Consideriamo inoltre la seguente struttura per la gestione del singolo nodo:

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

L'algoritmo per la visita in ordine anticipato (in pre-order) consiste nella visita rispettivamente di:

- radice
- sottoalbero sinistro
- sottoalbero destro

Tale visita prevede dunque l'attraverso dei seguenti nodi: **A B C D E F G**

L'implementazione della procedura di visita è dunque la seguente:

```
void preOrder(struct node *root) {
   if (root == NULL)
      return;
   printf("%d ", root->data);
   preOrder(root->left);
   preOrder(root->right);
}
```

L'algoritmo per la visita in ordine simmetrico (in-order) consiste nella visita rispettivamente di:

- sottoalbero sinistro
- radice
- sottoalbero destro

Tale visita prevede dunque l'attraverso dei seguenti nodi: **C B D A F E G**

L'implementazione della procedura di visita è dunque la seguente:

```
void inOrder(struct node *root){
   if (root == NULL)
      return;
   inOrder(root->left);
   printf("%d ", root->data);
   inOrder(root->right);
}
```

L'algoritmo per la visita in ordine posticipato (post-order) consiste nella visita rispettivamente di:

- sottoalbero sinistro
- sottoalbero destro
- radice

Tale visita prevede dunque l'attraverso dei seguenti nodi: **C D B F G E A**

L'implementazione della procedura di visita è dunque la seguente:

```
void postOrder(struct node *root) {
   if (root == NULL)
      return;
   postOrder(root->left);
```

```
    postOrder(root->right);
    printf("%d ", root->data);
}
```

Di seguito un esempio di attraversamento di un albero binario nelle 3 strategie:

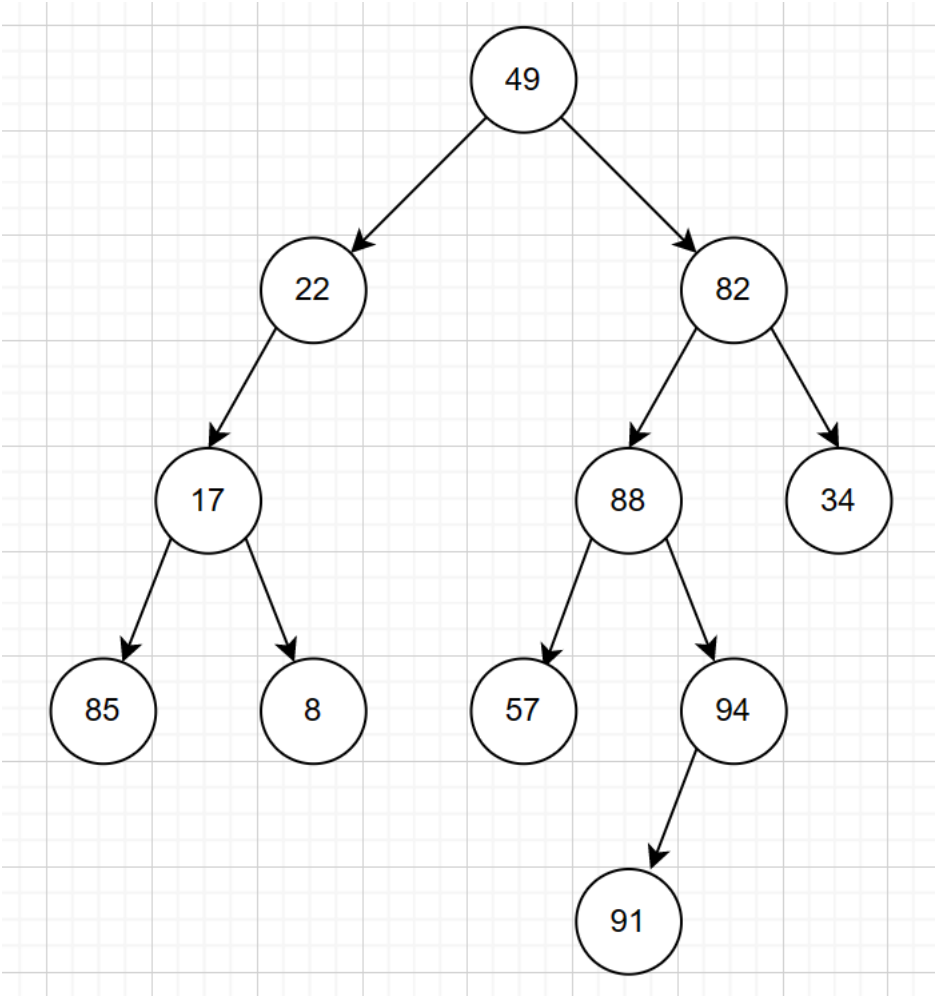

Figura 3: Albero binario di esempio

- Visita in pre-ordine: **49, 22, 17, 85, 8, 82, 88, 57, 94, 91, 34**
- Visita in post-ordine: **85, 8, 17, 22, 57, 91, 94, 88, 34, 82, 49**
- Visita simmetrica: **85, 17, 8, 22, 49, 57, 91, 94, 88, 82, 34**

Indichiamo di seguito la procedura per contare i nodi di un albero:

```
int contaNodi(struct node *root) {
  if (root == NULL)
    return 0;
  else {
    int C1=contaNodi(root->left);
    int C2=contaNodi(root->right);
    return C1+C2+1;
}
```

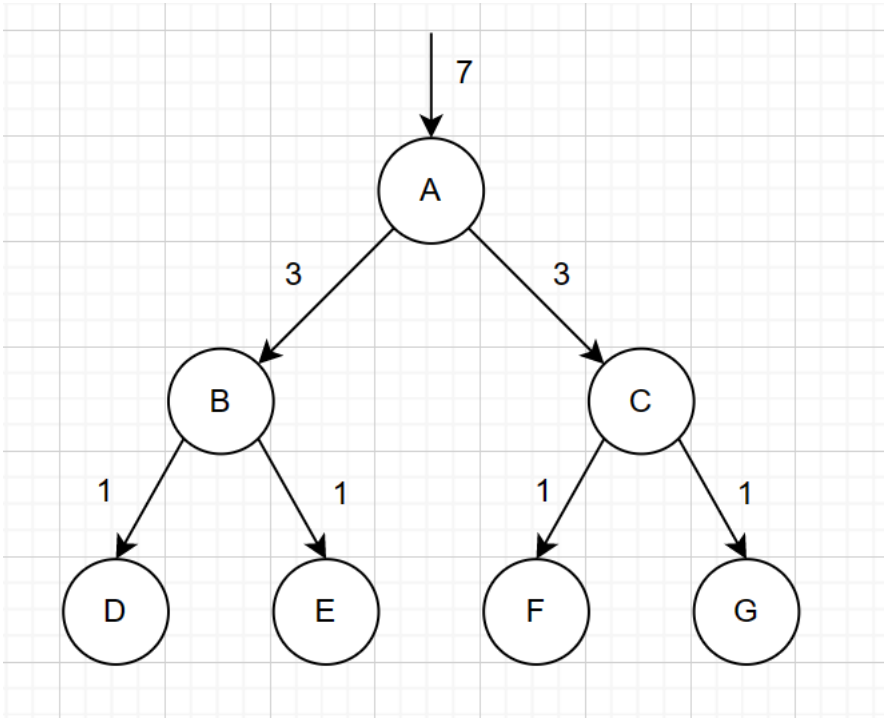

Figura 2: ContaNodi

Ogni ramo in figura contiene il totale dei nodi che provengono dal sottoalbero:

- l'arco che congiunge A con B contiene il valore 3 che è il totale dei nodi del sottoalbero con radice B (B, D ed E)
- l'arco che congiunge A con C contiene il valore 3 che è il totale dei nodi del sottoalbero con radice C (C, F ed G)
- l'arco che congiunge B con D contiene il valore 1 che è il totale dei nodi del sottoalbero con radice D (solamente D)
- […]

3. Visita in profondità (DFS) – iterativa

È possibile visitare un albero in DFS anche in modalità iterativa, utilizzando una pila / stack.

Di seguito mostriamo l'implementazione in Python delle varie visite mediante l'utilizzo di uno stack:

```
def dfs_preorder(root):
    stack = []
    stack.append(root)
    while stack:
        node = stack.pop()
        print(node.value)
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)

def dfs_postorder(root):
    if root is None:
        return
    stack = [root]
    result = []
    while stack:
        node = stack.pop()
        result.append(node.value)
        if node.left is not None:
            stack.append(node.left)
        if node.right is not None:
            stack.append(node.right)
    return result[::-1]

def dfs_inorder(root):
    if root is None:
        return
```

```
stack = []
result = []
current = root
while current is not None or stack:
    while current is not None:
        stack.append(current)
        current = current.left
    current = stack.pop()
    result.append(current.value)
    current = current.right
return result
```

Qual è il senso di utilizzare uno stack ed una visita iterativa?

Consideriamo la visita in pre-ordine: la radice dell'albero viene inserita nello stack all'inizio, e quindi viene eseguito un ciclo che continua finché lo stack non è vuoto. Ad ogni iterazione, l'elemento in cima allo stack viene estratto (utilizzando stack.pop()) e il suo valore viene stampato. Quindi, se il nodo estratto ha figli sinistro o destro, questi vengono inseriti nello stack (utilizzando stack.append()), in modo che vengano visitati in un secondo momento. In questo modo, l'algoritmo visita l'albero in modo profondo (DFS), utilizzando uno stack per tenere traccia dei nodi da visitare.

Sebbene per la maggior parte delle visite di un albero binario, la soluzione più semplice e comune è quella di utilizzare la ricorsione, che ha una complessità lineare, tuttavia, esistono situazioni in cui potrebbe essere necessario utilizzare uno stack per eseguire una visita DFS dell'albero.

Ad esempio, in alcune implementazioni di sistemi di elaborazione delle informazioni, potrebbe essere necessario eseguire una visita DFS in modo iterativo, senza utilizzare la ricorsione. In questi casi, l'utilizzo di uno stack è utile per mantenere traccia dei nodi che devono ancora essere visitati.

Inoltre, l'utilizzo di uno stack per la visita DFS può essere preferibile in termini di prestazioni, poiché in alcune implementazioni la ricorsione potrebbe consumare troppa memoria e causare un sovraccarico del sistema. In questi casi, l'utilizzo di uno stack può aiutare a ridurre l'utilizzo della memoria.

In sintesi, l'utilizzo di uno stack per la visita DFS di un albero binario è una soluzione alternativa alla ricorsione che può essere utile in determinate situazioni, come ad esempio quando è necessario eseguire la visita in modo iterativo o quando la ricorsione consuma troppa memoria.

4. Complessità

La complessità delle visite pre-order, in-order e post-order in un albero binario è $\Theta(n)$, dove n è il numero di nodi nell'albero.

$\Theta(n)$ indica una stima asintotica della complessità, che fornisce una descrizione approssimata del comportamento dell'algoritmo in relazione alla grandezza dell'input. In questo caso, la complessità è esattamente lineare, il che significa che la quantità di lavoro richiesto per esplorare l'albero aumenta proporzionalmente con il numero di nodi nell'albero.

Abbiamo infatti che il limite superiore è $O(n)$ mentre quello inferiore è $\Omega(n)$ ed essendo quindi limite superiore ed inferiore lineare, possiamo concludere che la complessità è lineare: $\Theta(n)$.

Il costo di una visita di un albero contenente n nodi è $\Theta(n)$, in quanto ogni nodo viene visitato al massimo una volta.

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.