3.1-Visite Alberi

Visita in profondità (depth-first search DFS)

- Visito i nodi dalla radice verso le foglie
- Sia T un albero non vuoto con radice n e k figli T1.....Tk
  - anticipata o **in preordine** visito n e poi nell'ordine T1.....Tk
  - posticipata o **in postordine** visito T1.....Tk e poi n
  - simmetrica o **in inordine** visito T1.....Ti, visito n, visito Ti+1.....Tk per un prefissato **i>=1**

SERIALIZE

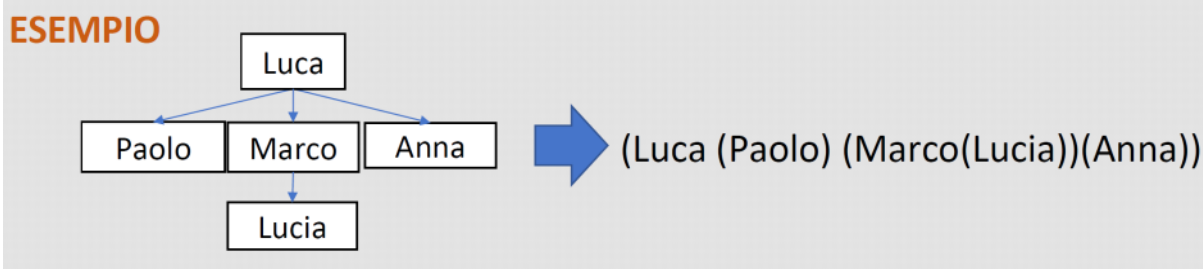
-Esempio di **visita DFS pre-order** che dato un albero t che stampa tutti i nodi di t in modo non ambiguo (è possibile individuare i sottoalberi di ogni albero)

```
void serialize (tree t) {

    cout<<"(";
    print(get_info(t));

    tree t1 = get_firstChild(t);

    while (t1 != NULL){
         serialize(t1);
         t1 = get_nextSiblig(t1);
    }
    cout<<")";
}
```



ALTEZZA

-Funzione che calcola l'altezza o profondità di un albero tramite una **visita DFS post-order**

```
int altezza(tree t){

     if(get_firstChild(t) == NULL)
          return 0;

     int max = 0, max_loc;
     tree t1 = get_firstChild(t);

     while(t1 != NULL){
         max_loc = altezza(t1);
         if(max_loc > max)
              max = max_loc;
         t1 = get_nextSibling(t1);
     }
     return max+1;
}
```

Visita in ampiezza (breath-first search BFS)

- Visito i nodi per livelli, a partire dal livello 0 della radice
- Per ogni tipologia di visita esiste una implementazione ricorsiva e una implementazione iterativa

Un albero è una struttura non lineare:

- La visita di un albero è realizzata come una sequenza di visite ai suoi nodi;
- Ad ogni passo della sequenza ci sono dei nodi aperti (i figli di un nodo padre da cui riprende la visita quando un suo sottoalbero è stato esplorato);

DIMENSIONE

-Funzione che che restituisce la dimensione dell'albero dato calcolata con una BFS iterativa

```
int dimensione(tree t){

     int count = 0;
     codaBFS c = newQueue();
     c = enqueue(c,t);

     while(!isEmpty(c)){
          node* n = dequeue(c);
          count++;
          tree t1 = get_firstChild(n);

          while(t1 != NULL){
               c = enqueue(c,t1);
               t1 = get_nextSibling(t1);
               }
     }
     return count;

 }
```

Primitive BFS-CODA

Struttura elemento coda

```
static elemBFS* new_elem(node* n){

         elemBFS* p = new elemBFS ;
         p->inf = n;
         p->pun = NULL;
          return p;
}
```

Nuova coda

```
 codaBFS newQueue(){

      codaBFS c = {NULL, NULL};
      return c;
 }
```

Metti in coda

```
codaBFS enqueue(codaBFS c, node* i){
     elemBFS *e = new_elem(i);
     if(c.tail!=NULL)
          c.tail->pun = e;
     c.tail = e;
     if(c.head == NULL)
          c.head = c.tail;
     return c;
}
```

Cancella dalla coda

```
node* dequeue(codaBFS& c){

     node* ris = (c.head)->inf;
     c.head = (c.head)->pun;
     return ris;
}
```

Controllo coda vuota

```
bool isEmpty(codaBFS c){

     if(c.head == NULL)
          return true;
     return false;
}
```