

Parte 4 – Alberi

Visite di alberi



Gustave Klimt *Albero della vita* 1909

Visite di alberi

- In molti casi è necessario attraversare un albero visitandone tutti i nodi
- Due principali tipologie di visite:

Visita in profondità (depth-first search DFS)

- Visito i nodi dalla radice verso le foglie
- Tre varianti: anticipata o in preordine, posticipata o in postordine, simmetrica o in inordine

Visita in ampiezza (breadth-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 iterattiva*
- Vedremo l'implementazione della DFS ricorsiva e l'implementazione della BFS iterattiva

Visita DFS ricorsiva

Algoritmo visitaDFSRicorsiva(nodo n)

1. visita il nodo n
2. for each child n' of n
visitaDFSRicorsiva(n')

- L'algoritmo implementa una visita in *preordine*
- Sia T un albero non vuoto con radice n e k figli T_1, \dots, T_k
 - In **preordine** visito n e poi nell'ordine T_1, \dots, T_k
 - In **postordine** visito T_1, \dots, T_k e poi n
 - In **inordine** visito T_1, \dots, T_i , visito n , visito T_{i+1}, \dots, T_k per un prefissato $i \geq 1$
- La **visita** di un nodo consiste nell'accesso al nodo e nella sua elaborazione come richiesto dall'applicazione che si vuole implementare

Serializzazione di un albero

- Vediamo un esempio di applicazione di visita DFS ricorsiva

La serializzazione di un albero

- Un albero è una struttura non lineare e la stampa del suo contenuto a video o su file richiede di applicare un meccanismo (o funzione) di *serializzazione* non ambiguo ovvero che consenta di individuare i sottoalberi di ogni albero
- La funzione di serializzazione deve quindi essere invertibile (ovvero biunivoca):
$$f: \text{tree} \rightarrow \text{stringa}$$
$$f^{-1}: \text{stringa} \rightarrow \text{tree}$$
- È possibile stabilire una *corrispondenza biunivoca* tra alberi e sequenze di parentesi bilanciate

Funzione di serializzazione

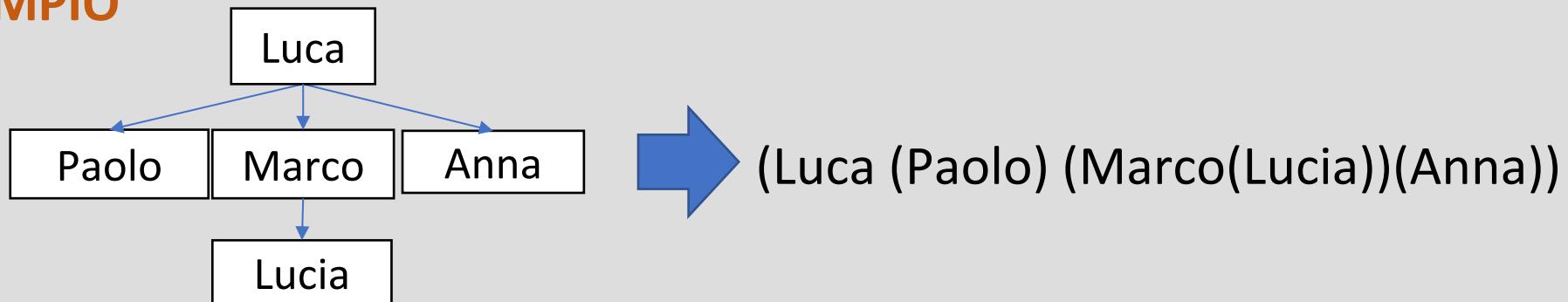
- Ogni nodo n è associato a una coppia di parentesi bilanciate (...)
- I suoi sottoalberi sono ricorsivamente serializzati ciascuno con una coppia di parentesi bilanciate

FORMALMENTE

Sia T un albero non vuoto con radice n e k figli T_1, \dots, T_k

serializzazione(n): (n serializzazione(T_1)... serializzazione(T_k))

ESEMPIO



La tipologia di visita DFS richiesta è in preordine e la visita del nodo consiste nella stampa dell'apertura di parentesi e del contenuto del nodo

Esercizio

Rivedere il modulo «main» del progetto sugli alberi nella directory *albero*

1. Implementando la procedura **serializza** che stampa a video il contenuto dell’albero secondo la funzione di serializzazione appena descritta
 - La procedura deve implementare una visita DFS in preordine
 - La procedura deve usare le primitive di accesso all’albero
2. Sostituire le righe di stampa nel main con una chiamata alla funzione

SOLUZIONE Vedi cartella *visitaDFS_albero*

La procedura serializza

```
void serialize(tree t) {  
    cout<<"(";  
    print(get_info(t));  
  
    tree t1 = getFirstChild(t);  
    while(t1!=NULL) {  
        serialize(t1);  
        t1 = getNextSibling(t1);  
    }  
    cout<<") ";  
}
```

Visita del nodo

Chiamata
ricorsiva ai figli

Calcolo dell'altezza di un albero

- Vogliamo calcolare l'**altezza** o **profondità** di un albero
- L'altezza misura il massimo delle lunghezze dei cammini che vanno dalla radice alle sue foglie

OSSERVAZIONE

- Se T è un albero composto da un solo nodo n (ovvero n è una foglia) allora la sua altezza è 0:
$$\text{altezza(foglia)}=0$$
- Sia T un albero con radice n e k figli T_1, \dots, T_k allora la sua altezza è il massimo dell'altezza di T_1, \dots, T_k incrementata di 1:
$$\text{altezza}(T)=\max(\text{altezza}(T_1), \dots, \text{altezza}(T_k))+1$$
- Il calcolo dell'altezza di un albero può essere implementato attraverso la DFS in post-ordine

Esercizio

- Estendere il codice del modulo “main” del progetto *visitaDFS_albero* aggiungendo la funzione **int altezza(tree)** che restituisce l’altezza dell’albero

SOLUZIONE Vedi file *main_altezza*

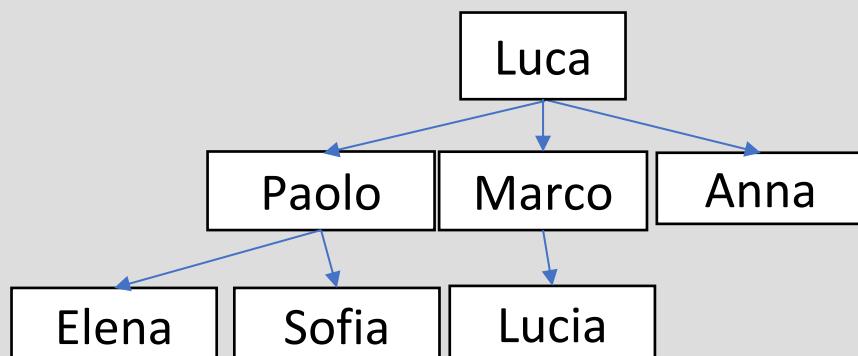
Osservazione sulla visita degli alberi

- 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** ovvero nodi che rappresentano i punti di ramificazione rimasti in sospeso e da cui la visita deve proseguire

Visita DFS

- Nel caso della *visita DFS* i nodi «aperti» sono i figli di un nodo padre da cui riprende la visita quando un suo sottoalbero è stato esplorato

ESEMPIO



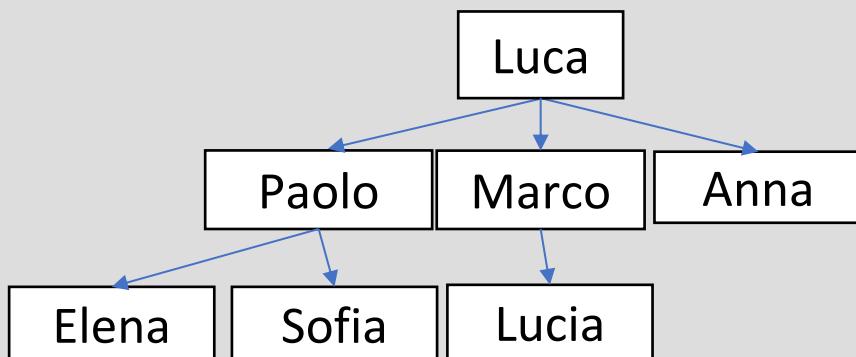
1. Visito Luca
Push Anna, Marco, Paolo
Nodi aperti: Paolo, Marco, Anna
2. Pop nodi aperti: Visito Paolo
Push Sofia, Elena
Nodi aperti: Elena, Sofia, Marco, Anna
3. Pop Nodi aperti: Visito Elena
...

- La sequenza dei nodi da visitare è gestita implicitamente dallo *stack dei RDA* delle chiamate ricorsive

Visita BFS

- Nel caso della visita BFS la lista dei nodi deve essere gestita attraverso una **coda**

ESEMPIO



1. Visito Luca
Enqueue Paolo, Marco, Anna
Nodi aperti: Paolo, Marco, Anna
2. Dequeue Nodi aperti: Visito Paolo
Enqueue Elena Sofia
Nodi aperti: Marco, Anna, Elena, Sofia
3. Dequeue Nodi aperti: Visito Marco
Enqueue Lucia
Nodi aperti: Anna, Elena, Sofia, Lucia
4. Dequeue Nodi aperti: Visito Anna
5. Dequeue Nodi aperti: Visito Elena
6. Dequeue Nodi aperti: Visito Sofia
7. Dequeue Nodi aperti: Visito Lucia

- Possiamo implementare la visita BFS con un **algoritmo iterattivo** che gestisce esplicitamente la coda

Visita BFS iterattiva

Algoritmo visitaBFSIterattiva(nodo n)

1. coda C
2. enqueue(C,n)
3. while ($C \neq \text{NULL}$)
 - a. $n = \text{dequeue}(C)$
 - b. visita nodo n
 - c. for each child n' of n
enqueue(n')

Il modulo «coda-bfs» per visita BFS iterattiva

- La visita BFS iterattiva fa uso di una coda dove vengono gestiti i nodi aperti
- Realizziamo il modulo «coda-bfs» che implementa una *coda di nodi dell'albero* come un tipo particolare di lista dove le politiche di accesso sono di tipo FIFO
- Il tipo di dato codaBFS è definito come segue:

```
struct elemBFS
{
    node* inf;
    elemBFS* pun ;
};

typedef elemBFS* lista;

typedef struct{
    lista head;
    elemBFS* tail;} codaBFS;
```

Il modulo «coda-bfs» per visita BFS iterattiva

- Le primitive del modulo sono le seguenti:

```
codaBFS enqueue(codaBFS, node*) ;  
node* dequeue(codaBFS&) ;  
node* first(codaBFS) ;  
bool isEmpty(codaBFS) ;  
codaBFS newQueue() ;  
static elemBFS* new_elem(node*) ;
```

- Il file sorgente `coda-bfs.cc` include
 - `coda-bfs.h`
 - `tree.h`
 - `tipo.h`

Vedi directory `coda-bfs`

Dimensione di un albero

- La dimensione di un albero corrisponde al numero di nodi contenuti nell’albero
- Estendere il progetto «visitaDFS_albero»
 - Aggiungendo il modulo coda-bfs
 - Aggiungendo al codice del modulo “main” la funzione **int dimensione(tree)** che restituisce la dimensione dell’albero dato
 - La funzione deve calcolare la dimensione implementando la visita BFS iterattiva

SOLUZIONE Vedi directory *visita_albero*

Esercizio

- Estendere il modulo «main» del progetto *visita_albero* aggiungendo una seconda implementazione della funzione **int dimensione(tree)** che fa uso della visita DFS ricorsiva
- Aggiungere espressioni per la compilazione condizionale che consentano di compilare la versione DFS o la versione BFS della funzione attraverso
 - la definizione di una macro vuota per la specifica di un identificatore
 - L'introduzione di una direttiva **#ifdef**

SOLUZIONE Vedi *main_dimensione.cc*

Considerazioni sulle implementazioni delle visite

- Abbiamo visto le implementazioni delle visite DFS e BFS più comuni:
 - DFS ricorsiva
 - BFS iterattiva
- L'implementazione *DFS iterattiva* richiede di gestire esplicitamente la pila dei nodi
 - I nodi figli di un nodo devono essere impilati da destra verso sinistra
 - La struttura dati **tree** non è adatta a questo tipo di operazioni e sarebbe necessaria una struttura di appoggio
 - L'implementazione DFS iterattiva risulta molto più semplice in caso di alberi binari perché ogni nodo ha al più 2 figli
- L'implementazione *BFS ricorsiva* risulta molto più complicata della corrispondente iterattiva perché i nodi sono naturalmente gestiti in una coda mentre le chiamate ricorsive fanno uso di una pila e il comportamento dei due tipi di lista è opposta per inserimento e estrazione

Esercizi non risolti

(Provare ad implementare ogni esercizio usando la BFS e la DFS)

- Calcolare il numero di foglie dell'albero
- Calcolare il grado medio dei nodi dell'albero (numero medio di figli di un nodo)
- Data una stringa, verificare se esiste un nodo che contenga quella stringa
- Verificare se esiste un cammino padre-figlio che collega un nodo con valore «s1» dato in input a un nodo con valore «s2» dato in input