Rappresentazione degli alberi

Gustave Klimt *Albero della vita* 1909

Albero

- Struttura dati *dinamica non lineare* in quanto ogni elemento (o **nodo**) può avere più di un successore (o **discendente**)

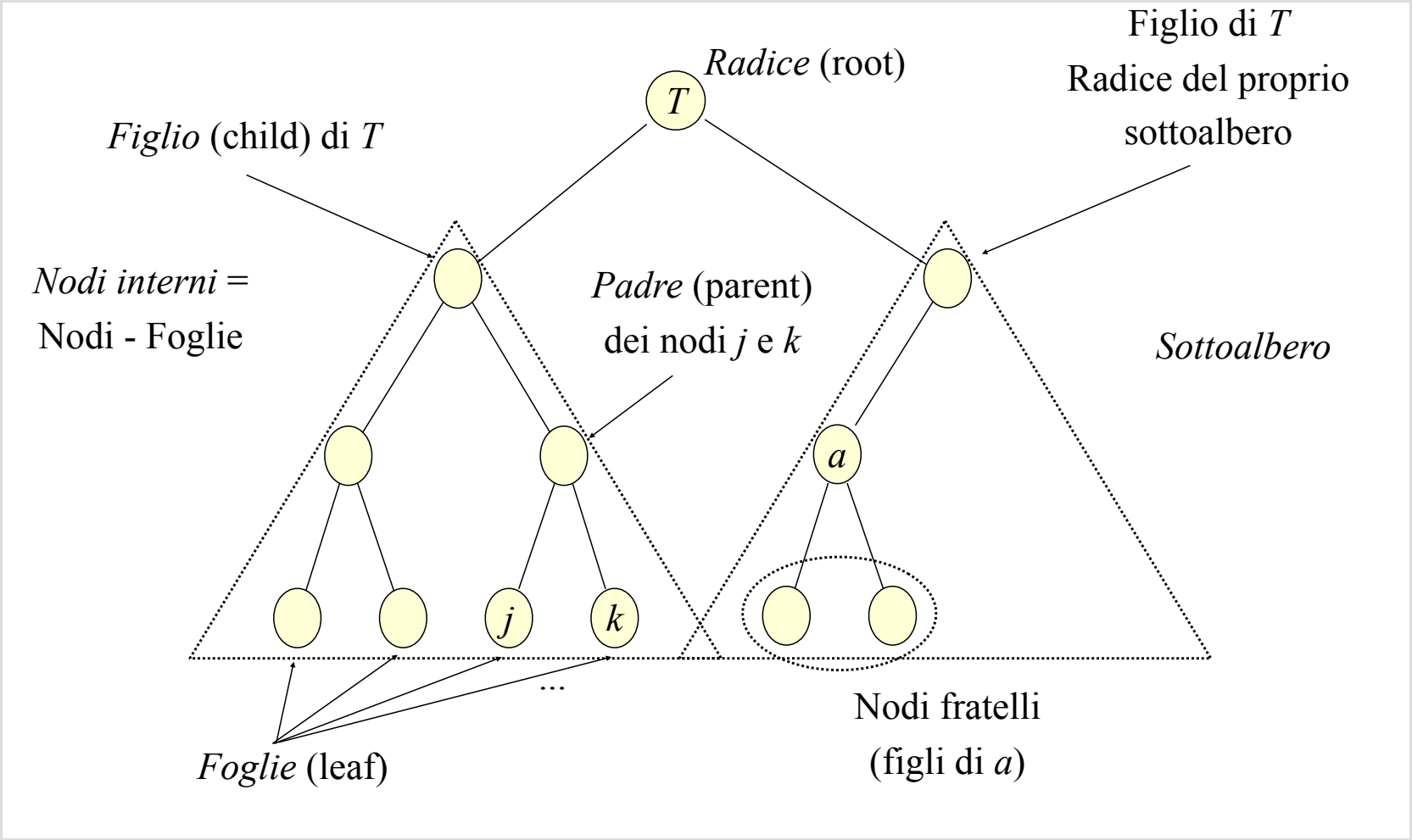
Radice: primo elemento di un albero

- I discendenti diretti di un nodo sono i **figli** del nodo
- Se il nodo *n* è il figlio del nodo *n'*, allora *n'* è il **padre** di *n*
- I nodi con lo stesso padre sono detti **fratelli**
- Un nodo da cui non discende nessun altro nodo si chiama **foglia**
- Ogni nodo che non sia radice è la radice di un albero contenuto nell'albero dato detto **sottoalbero**

Gli **alberi n-ari** possono avere un numero qualsivoglia di figli per ciascun nodo.

Gli **alberi binari** possono avere 0, 1, o al più 2 figli per ciascun nodo

Albero (cont.)


Radice (root)
T
Figlio di T
Radice del proprio sottoalbero
Figlio (child) di T
Nodi interni =
Nodi - Foglie
Padre (parent)
dei nodi j e k
Sottoalbero
a
j
k
...
Foglie (leaf)
Nodi fratelli
(figli di a)

Implementazione degli alberi

- Esistono diverse modalità per rappresentare gli alberi in memoria
- Esistono rappresentazioni che fanno uso di vettori
  - Ad esempio la rappresentazione con *vettore di padri* rappresenta l'albero come un vettore di elementi dove in ogni elemento è mantenuto l'indice del nodo padre
  - I vettori sono strutture dati statiche che presentano i limiti che abbiamo visto quando abbiamo introdotto le liste
- Esistono rappresentazioni che fanno uso di puntatori

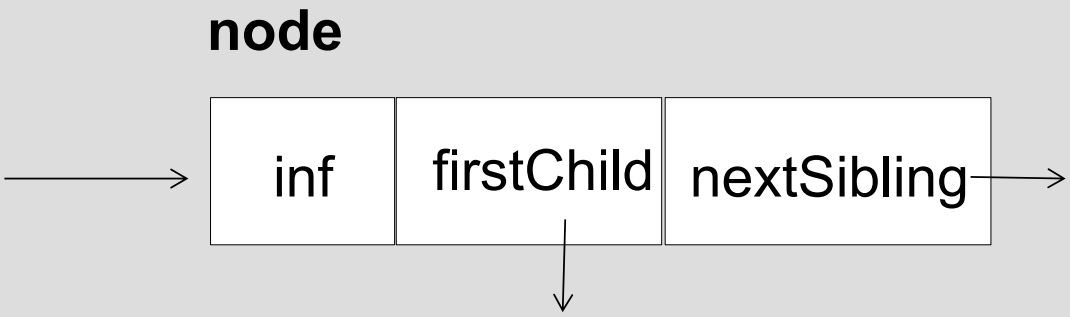
Noi vedremo la rappresentazione

puntatori primo-figlio/fratello

Rappresentazione puntatori primo-figlio/fratello

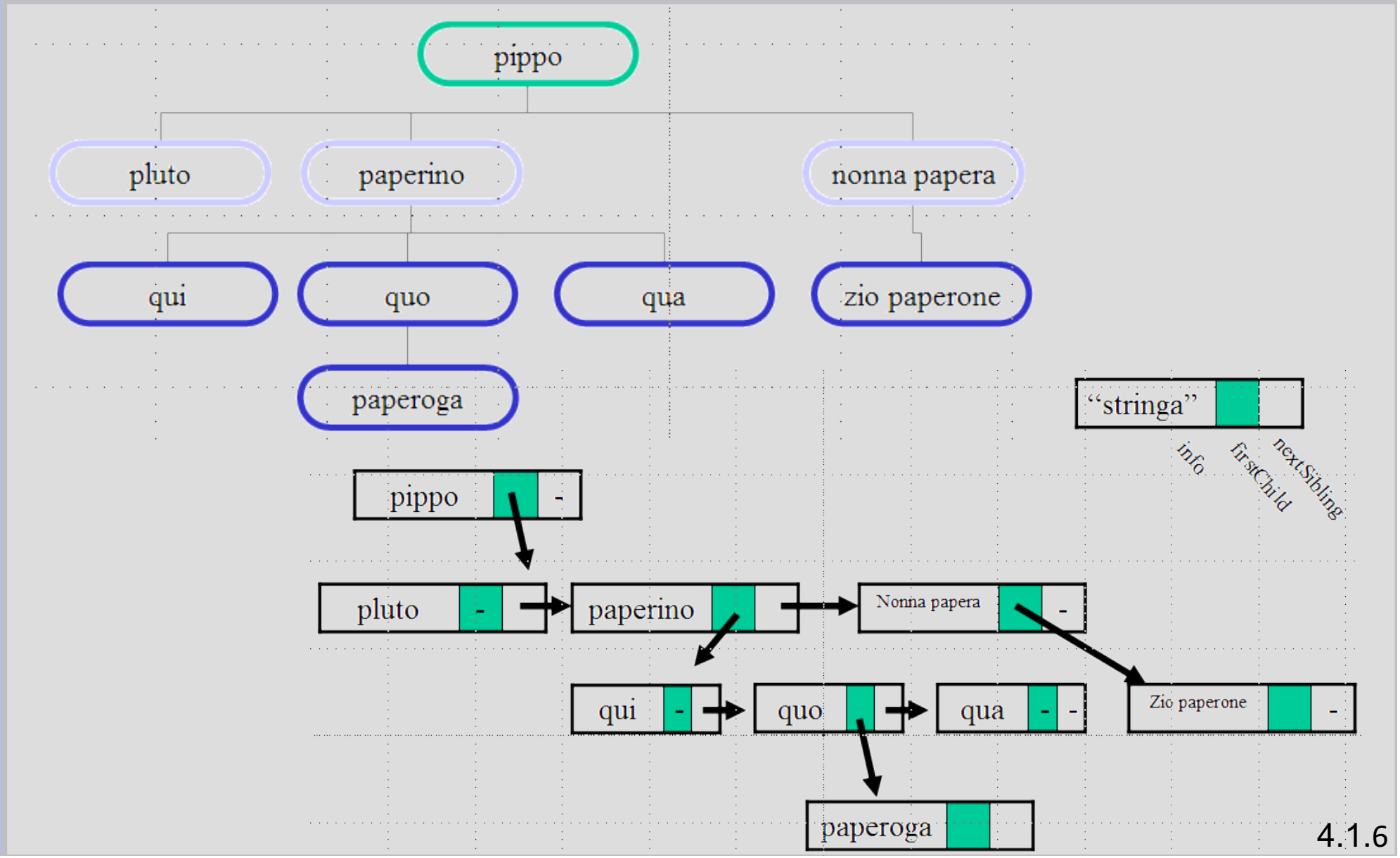
Ciascun elemento (**node**) contiene

- un campo informativo
- un puntatore al primo figlio (**firstChild**)
- un puntatore al fratello successivo (**nextSibling**)



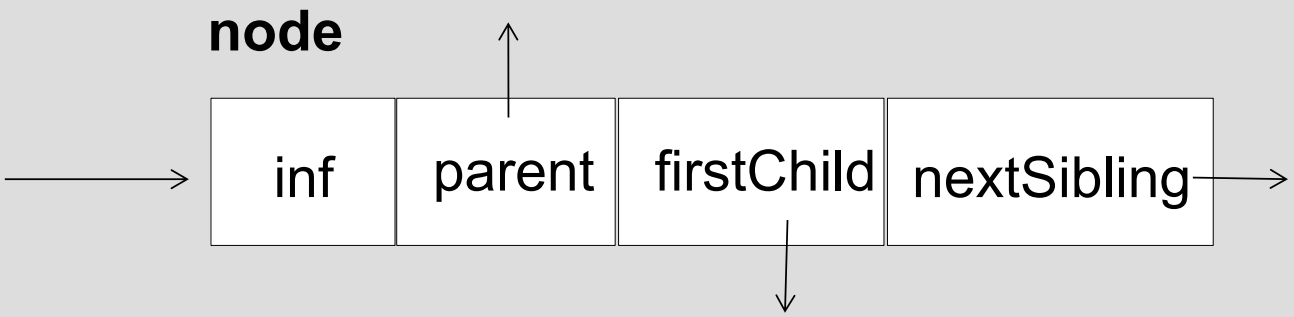
Le *foglie* hanno valore NULL nel campo firstChild

Esempio puntatore primo-figlio/fratello



Osservazioni

- La rappresentazione primo-figlio/fratello supporta direttamente l'esplorazione dell'albero dalla radice alle foglie
  - Ad esempio le diverse modalità di visita di un albero
- Se si vuole navigare l'albero agevolmente in ordine inverso allora possiamo aggiungere un puntatore al padre (**parent**)



Rappresentazione puntatori padre/primo-figlio/fratello

Struct “node” e il tipo di dato “tree”

```
struct node {
    tipo_inf inf;
    node* parent; //opzionale
    node* firstChild;
    node* nextSibling;
};

typedef node* tree; //punta alla radice
                    dell’albero
tree root; //variabile di tipo tree
```

Primitive per la creazione di alberi

`node* new_node(tipo_inf i)`

Crea un nuovo nodo con valore informativo **`i`**

`void insert_child(tree p, tree c)`

Aggiorna **`p`** inserendo il sottoalbero radicato in **`c`** come primo figlio di **`p`**

`void insert_sibling(node* n, tree t)`

Aggiorna **`n`** inserendo il sottoalbero radicato in **`t`** come fratello successivo di **`n`**

Perché il primo parametro di **`insert_child`** e **`insert_sibling`** non viene passato per riferimento?

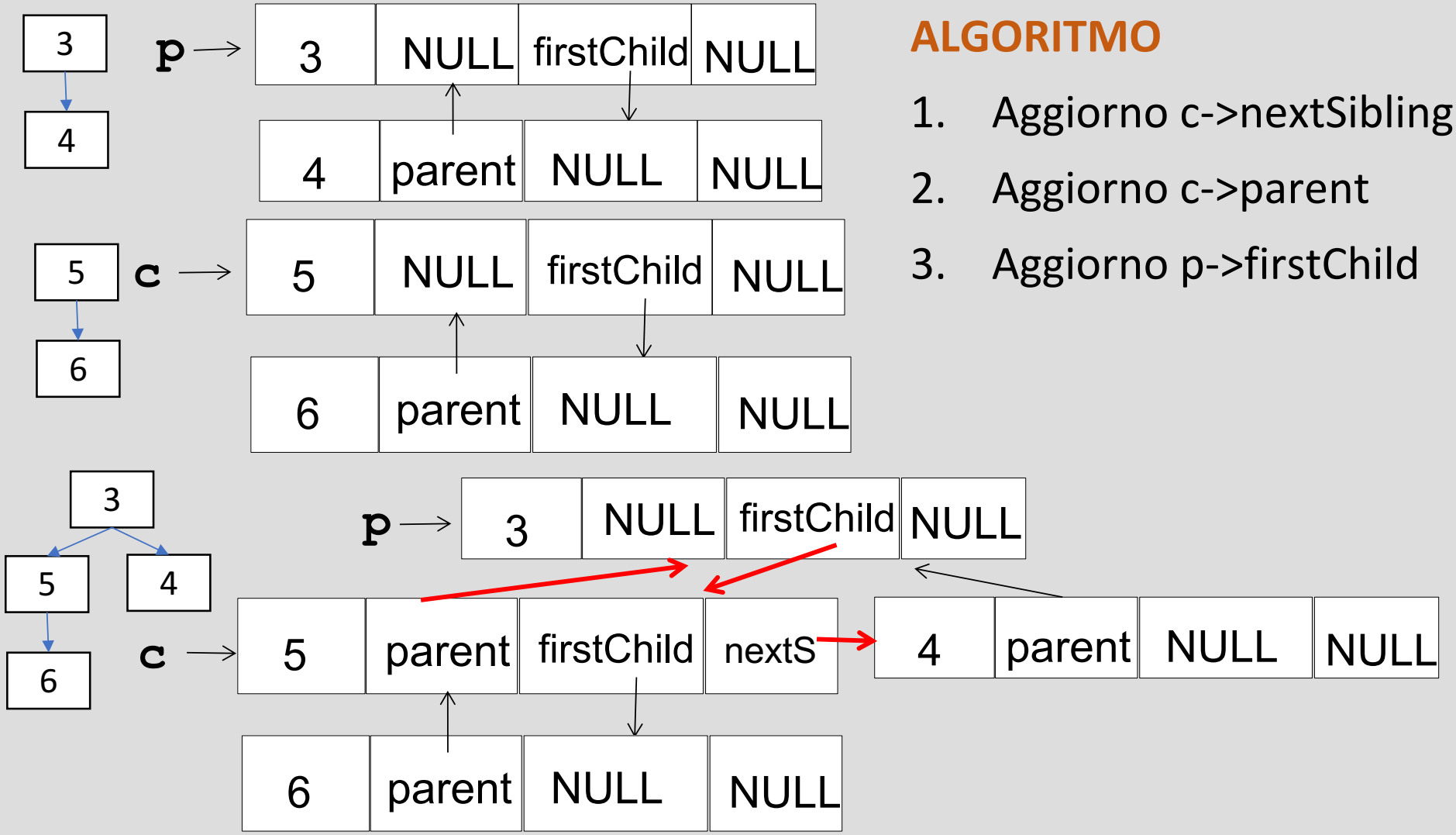
Perché viene modificato l'oggetto puntato dal parametro formale che si trova nell'area heap

Quando è necessario il passaggio per riferimento in questi casi?

Quando dobbiamo modificare il valore del puntatore

La primitiva

`void insert_child(tree p, tree c)`

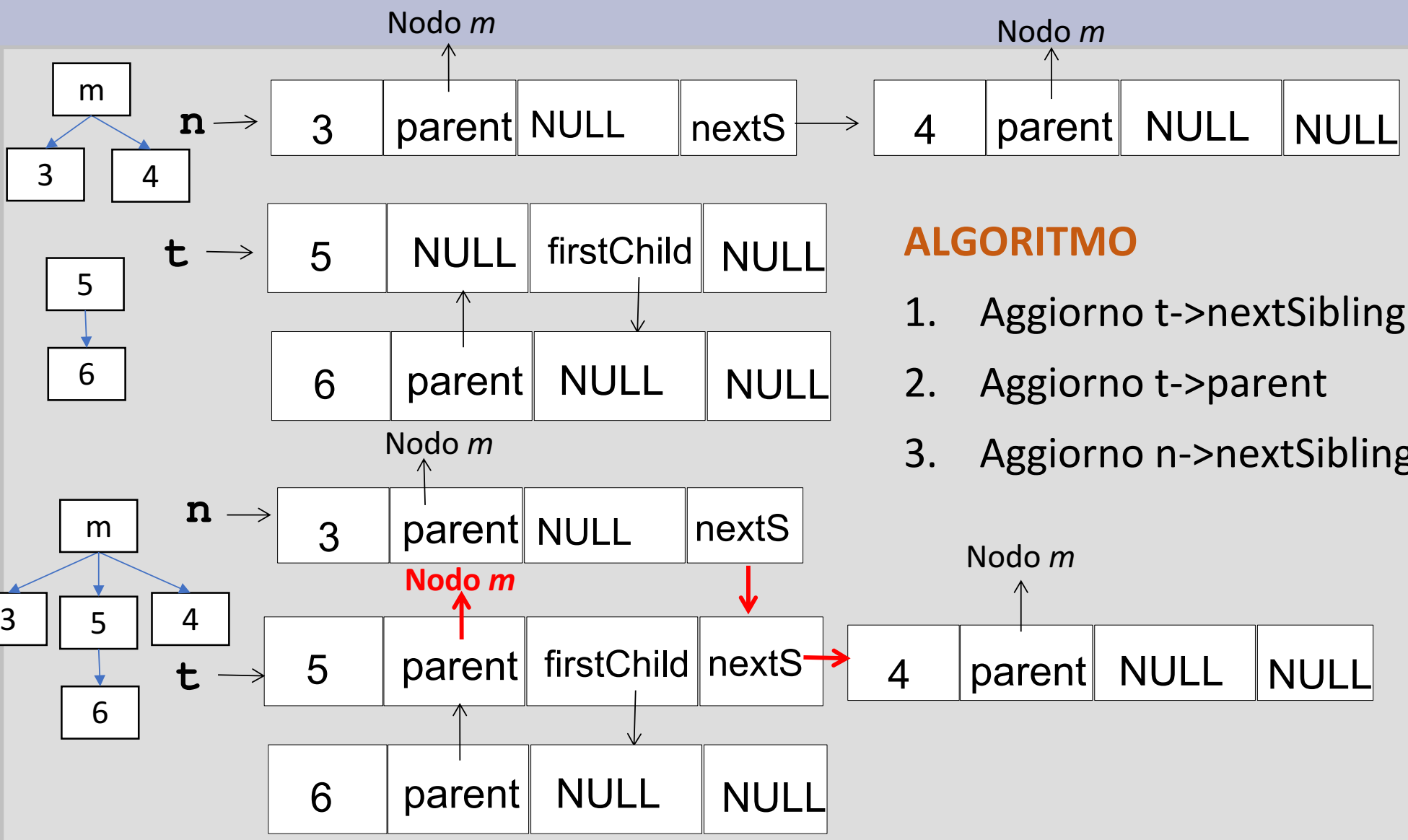


ALGORITMO

1. Aggiorno c->nextSibling
2. Aggiorno c->parent
3. Aggiorno p->firstChild

La primitiva

`void insert_sibling(node* n, tree t)`

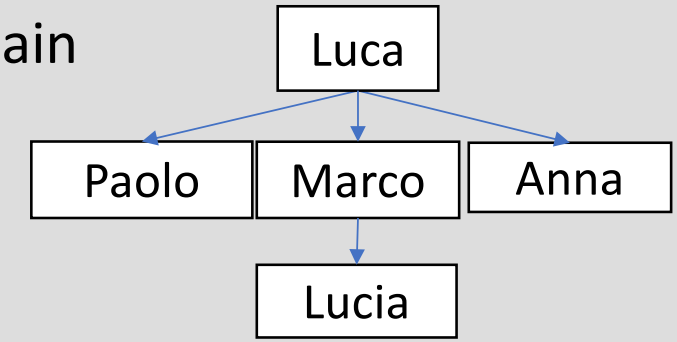


ALGORITMO

1. Aggiorno t->nextSibling
2. Aggiorno t->parent
3. Aggiorno n->nextSibling

Esercizio

- Creare il modulo “albero” che contiene
  - La struct “node” e il tipo di dato “tree”
  - Le primitive per la creazione di alberi
- Creare un progetto per la creazione di un albero di stringhe
- Il progetto si compone di tre moduli: il modulo “albero”, il modulo “stringa” che implementa il tipo tipo_inf come stringa e il modulo “main”
- Nel modulo “main” scrivere un piccolo main di prova che crea il seguente albero:



SOLUZIONE Vedi cartella *crea_albero*

Primitive per l'accesso a alberi

`tipo_inf get_info(node* n)`

Restituisce il contenuto informative del nodo **`n`**

`node* get_parent(node* n)`

Restituisce il padre del nodo **`n`**

`node* get_firstChild(node* n)`

Restituisce il primo figlio del nodo **`n`**, se esiste

`node* get_nextSibling(node* n)`

Restituisce il fratello successive del nodo **`n`**, se esiste

Esercizio

- Estendere il modulo “albero” con le primitive per l’accesso all’albero
- Aggiornare il progetto crea_albero sostituendo nel main l’acceso diretto ai campi di node con chiamate alle corrispondenti primitive

SOLUZIONE Vedi cartella *albero*