Alberi

1 Introduzione agli alberi binari

In teoria dei grafi, un *albero* nella sua accezione più generale corrisponde ad una particolare tipologia di *grafo* (non orientato, connesso e aciclico). Per un'esaustiva comprensione dei concetti più teorici si rimanda dunque al capitolo inerente i grafi. Per gli argomenti trattati in questo capitolo, è sufficiente comprendere come può essere implementata una struttura dati corrispondente ad un albero binario.

Un **albero binario** è composto da **nodi** connessi tra loro mediante certe regole. I nodi della struttura sono i vari oggetti, contenenti un campo chiave e dati satelliti; inoltre i nodi contengono i campi *left*, *right* e *p* che puntano ad altri nodi denominati rispettivamente **figlio sinistro**, **figlio destro** e **padre** del nodo. Ogni nodo di un albero binario può dunque avere *al più due nodi figli e un unico nodo padre*. Un albero binario presenta un nodo denominato radice (*root*), a cui non connesso alcun nodo padre (nodo 1, nella *Figura 1*). Infine, nel caso degli alberi binari, non è solo importante il numero di figli di un nodo (0, 1 o 2), ma anche la *posizione* del figlio: figlio destro o figlio sinistro.

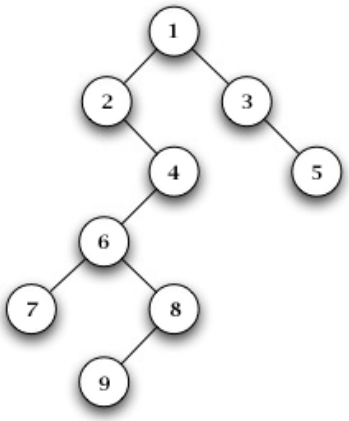

Figure 1: Esempio di albero binario etichettato con interi

Per ciò che riguarda il linguaggio di programmazione C, i nodi possono essere implementati mediante *struct*, tra i cui campi figurano i puntatori che provvedono alla connessione con il figlio sinistro e il figlio destro (il puntatore al padre solitamente non viene inserito).

Codice d'esempio:

```
struct nodo {
   int key;
   struct nodo *left;
   struct nodo *right;
}
```

Questo non è l'unico modo con cui può essere implementato un albero binario in C: anche un semplice array può essere utilizzato, come vedremo nella sezione del capitolo riguardante gli *heap*.

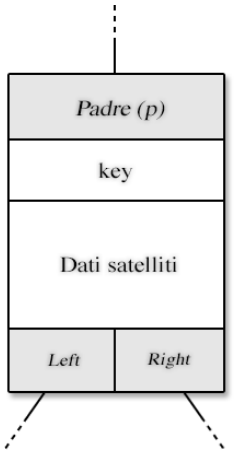

Figure 2: Struttura di un nodo di albero binario, con le celle grigie puntatori ad altri nodi.

Una definizione ricorsiva di albero binario è la seguente: un albero binario è un insieme finito di nodi e può essere:

- un insieme vuoto, che non contiene nodi, oppure
- è composto da tre insiemi disgiunti di nodi: un nodo radice e due alberi binari, rispettivamente *sottoalbero sinistro* della radice e *sottoalbero destro* della radice.

Un *sottoalbero con radice nel nodo* x è l'albero indotto dai discendenti del nodo x e avente come radice x. Se si considera un nodo x di un albero binario T, un nodo qualsiasi

y in un cammino unico da r a x è detto *antenato* di x. Se y è un antenato di x, allora x è un *discendente* di y. Se $x \neq y$, allora si parla di *antenato proprio* e *discendente proprio*.

Vengono ora elencate ulteriori definizioni:

- *Foglia*: un nodo senza figli è definito come foglia;
- *Nodo interno*: un nodo avente almeno un figlio;
- *Profondità di un nodo*: se la profondità di un nodo è p i suoi figli non vuoti hanno profondità $p + 1$. La radice r ha profondità 0, i suoi figli sinistro e destro hanno profondità 1, i nipoti profondità 2 e così via;
- *Altezza di un albero*: massima profondità raggiunta dalle sue foglie o, in altri termini, profondità del cammino più lungo;
- *Albero binario completo*: albero binario in cui tutte le foglie hanno la stessa profondità e tutti i nodi interni hanno grado 2.

In un **albero binario completo**, la radice ha 2 figli a profondità 1, ciascuno dei quali ha 2 figli alla profondià 2 e così via. Quindi, il numero di foglie alla profondità h è 2^h. Il numero totale di nodi risulta essere $2^{(h+1)} - 1$. L'altezza di un albero binario completo con p foglie è $\log_2(p)$.

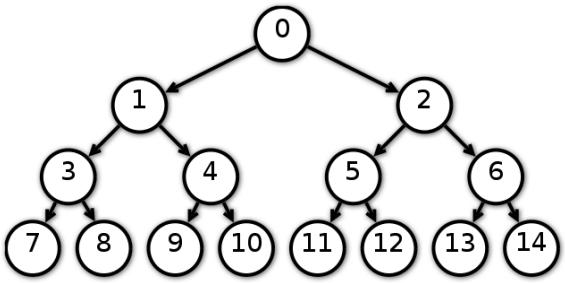

Figure 3: Esempio di un albero binario completo

2 Albero binario di ricerca

Un **albero binario di ricerca** è un albero binario in cui i nodi sono disposti ordinatamente a seconda del valore chiave che li identifica, di modo che valori minori di quelli del nodo di partenza siano memorizzati nei figli a sinistra e i valori più grandi nei figli a destra.

Sia x un nodo dell'albero binario di ricerca. Se y è un nodo nel sottoalbero sinistro di x allora il valore *key* di y deve essere minore del valore *key* di x. Viceversa, se y è un nodo

nel sottoalbero destro di x allora il valore *key* di y deve essere maggiore del valore *key* di x.

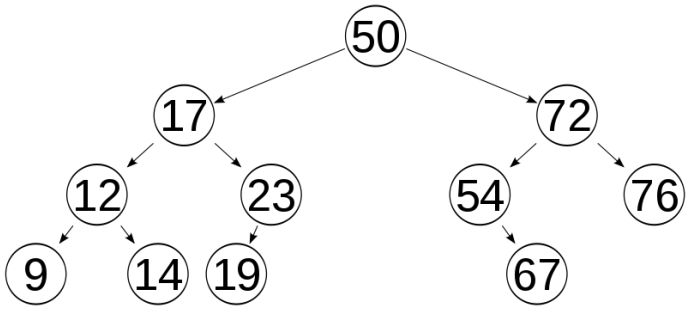

Figure 4: Esempio di un albero binario di ricerca

Gli alberi di ricerca binari rappresentano una struttura di dati che supporta in modo efficiente le operazioni SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, DELETE. La proprietà di ordinamento che contraddistingue un albero binario di ricerca permette di eseguire le operazioni più comuni, come ad esempio la ricerca di un elemento, con una complessità proporzionale all'altezza h dell'albero. Per un albero completo e bilanciato con n nodi la complessità risulta essere $\Theta(\lg\ n)$ nel caso peggiore. Tuttavia, la struttura dell'albero può degenerare in una catena lineare di n elementi, con conseguente deterioramento delle prestazioni a $\Theta(n)$ nel caso peggiore.

2.1 Attraversamento

È possibile elencare ordinatamente tutte le chiavi di un albero binario di ricerca con un semplice algoritmo ricorsivo chiamato *attraversamento simmetrico di un albero* (inorder). Tale algoritmo segue la regola per la quale, dato un nodo di un albero binario, prima si visita il nodo sinistro e se tale nodo non esiste si visita la radice (stampandone il valore *key*) prima di proseguire con la visita del nodo destro (se esiste).

INORDER-VISIT (da implementare)
1
2
3
4

Con una minima modifica all'algoritmo sopra riportato, è possibile definire anche un'*attraversamento anticipato di un albero* (preorder), con cui la radice viene elencata

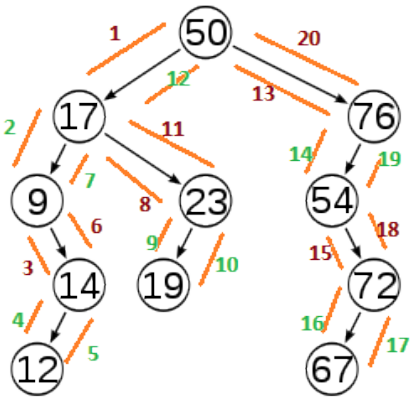

Figure 5: Attraversamento simmetrico di un albero binario

prima dei valori dei suoi sottoalberi, e un'*attraversamento posticipato di un albero* (postorder), il quale elenca la radice dopo i valori dei suoi sottoalberi.

Considerando l'albero binario di ricerca della *Figura 2.1*, gli elenchi risultanti dai tre metodi di attraversamento sono i seguenti:

ORDINE	SEQUENZA
In-order	9, 12, 14, 17, 19, 23, 50, 54, 67, 72, 76
Pre-order	50, 17, 9, 14, 12, 23, 19, 76, 54, 72, 67
Post-order	12, 14, 9, 19, 23, 17, 67, 72, 54, 76, 50

Table 1: Attraversamenti dell'albero binario di ricerca di Figura 2.1

2.2 Ricerca