Gli alberi binari

Anna Corazza

aa 2023/24

Dove studiare

- Sha'13, cap. 5 fino al 5.3 incluso

Sha'13 Clifford A. Shaffer, Data Structures & Algorithm Analysis in C++, (edition 3.2), 2013
`https://people.cs.vt.edu/shaffer/Book/C++3elatest.pdf`

Definizioni e proprietà

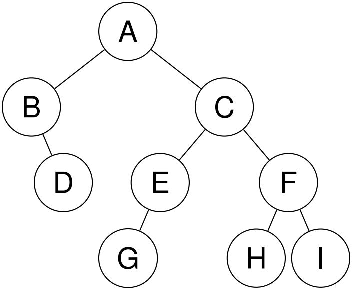
- Un **albero binario** è costituito da un **insieme finito di nodi**, che può essere:
  - vuoto
  - composto da una **radice** e due alberi binari: il **sottoalbero destro** e il **sottoalbero sinistro**, distinti tra loro e dalla radice.
- Nel secondo caso, le radici di ciascun sottoalbero sono figli della radice.
- Un arco (edge) congiunge ogni nodo ai suoi figli.
- Un nodo è detto **genitore** dei figli.

Cammini

- Sia data una sequenza di nodi $n_1, n_2, \ldots, n_k$ tale che n_i è genitore di n_{i+1} per $1 \leq i < k$: tale sequenza è detta **cammino** (path) da n_1 a n_k.
- La lunghezza del cammino è $k - 1$.
- Se esiste un cammino dal nodo R al nodo M, allora R è un **antenato** (ancestor) di M, e M un **discendente** di R.
- Quindi tutti i nodi di un albero sono discendenti della radice e la radice è antenato di tutti i nodi.
- Dato un nodo M, la sua **profondità** (depth) è data dalla lunghezza del cammino dalla radice a M.
- L'**altezza** di un albero è data dalla massima profondità tra i nodi + 1
- I nodi a profondità *d* formano il **livello** (level) *d* dell'albero: la radice è il solo nodo al livello 0 dell'albero.

Foglie e nodi interni

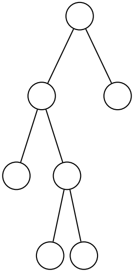
- Un nodo con entrambi i figli vuoti è detto **foglia** (leaf).
- Un nodo con almeno un figlio non vuoto è detto **interno**.



- Il nodo B ha due figli, di cui uno è l'albero vuoto: tutti i nodi di un albero binario hanno due figli, possibilmente vuoti.
- Livello 2 dell'albero: D, E, F
- Lunghezza del cammino A,C,E,G = 3
- Altezza dell'albero 4, massima profondità 3 (nodo I).

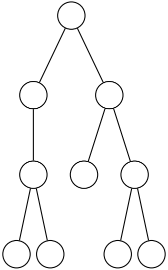
Alberi con nodi pieni

Nodi pieni : ogni nodo è o interno con due figli non vuoti oppure una foglia.



Alberi completi

Completo : in un albero completo di altezza h, tutti i nodi interni, **tranne al più uno** devono avere entrambi i figli, e le foglie si trovano tutte a livello $h-1$ o $h-2$.

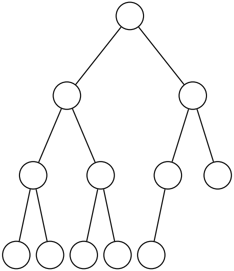


Ma non è pieno.

Alberi completi da sinistra

Definizione applicativa [Sha'13]

Completo [Sha'13]: in un albero binario completo di altezza h, tutti i livelli devono essere completi eccetto al più il (h-1): quet'ultimo livello deve essere riempito da sinistra verso destra (intuitivo: presuppone un ordine nei sottoalberi).



Nemmeno questo è pieno.

Numero di nodi e foglie

- Sia dato un albero con n nodi interni: quante sono le foglie? dipende ...
- Proviamo a capire qual è il numero massimo e minimo di foglie in un albero binario con n nodi interni.
- Il minimo è facile: 1 (l'albero è una catena di n+1 nodi).
- Il massimo è $n+1$: tutti gli alberi binari con nodi pieni con n nodi interni hanno $n+1$ foglie (si dimostra per induzione).
- Ogni albero binario non vuoto ha un numero di sottoalberi vuoti pari al numero dei suoi nodi più uno (si dimostra sostituendo ogni sottoalbero vuoto con una foglia)

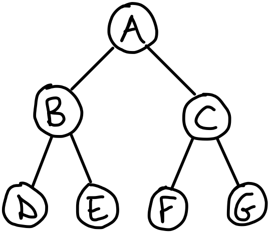
Attraversamenti

- **Traversal**: ogni processo per attraversare **tutti** i nodi di un albero binario.
- **Enumerazione**: un attraversamento che passa per ogni nodo una sola volta.
- Gli attraversamenti possono seguire diversi ordini, tra cui:

  Preorder : ogni nodo viene visitato prima dei suoi figli; quindi si comincia dalla radice, poi si procede prima con il sottoalbero sinistro e poi con il destro.

  Postorder : ogni nodo viene visitato dopo che sono stati visitati i suoi figli (e quindi anche i relativi sottoalberi); es. se voglio cancellare i nodi.

  Inorder : prima il figlio (e il sottoalbero) sinistro, poi il nodo e infine il figlio (e sottoalbero) destro.



IN-ORDER

Andiamo il più a sinistra possibile

D → B → E → A → C → F → G

POST-ORDER

Ricorda: left → right → root

D → E → B → F → G → C → A

PRE-ORDER

Ricorda: root → left → right

A → B → D → E → C → F → G

BREADTH

In ampiezza. Si visita l'albero per livelli

A → B → C → D → E → F → G

* Da notare la posizione della radice

Visitor design pattern

- Come implementare la funzione che fa la visita?
- Lo abbiamo già visto nel primo progetto: l'albero implementa la visita prendendo la funzione come parametro: naturalmente il prototipo di questa funzione deve essere fissato.
- Esempio per contare nodi in un albero:

```
template <typename E>
int count(BinNode<E>* root) {
  if (root == NULL) return 0; // Nothing to
     count
    return 1 + count(root->left()) +
       count(root->right());
}
```

Implementazione di un albero binario

Implementazione del nodo basata sui puntatori

- Un campo per il valore + due puntatori per i figli.
- Il puntatore al genitore di solito non è necessario se si usa bene la ricorsione.
- Stessa implementazione per nodi interni e foglie o è meglio differenziare?
- Classe astratta generica (BinNode) che poi viene derivata in una per nodi interni e una per foglie (ad esempio per espressioni algebriche).

Spazio richiesto

- **Overhead** spazio richiesto oltre ai dati per salvare la struttura.
- Dipende da:
  - in quale tipo di nodi memorizzo i dati (tutti, solo le foglie, ...)
  - se le foglie contengono comunque puntatori nulli
  - se l'albero è a nodi pieni

Spazio richiesto

Due puntatori + dato

- Prima possibilità: due puntatori ai figli (eventualmente nulli) più uno slot per il dato.

$$\begin{aligned} \text{Tot:}\ & n(2P + D) \\ \text{OH:}\ & 2nP \\ \text{perc.:}\ & 2P/(2P + D);\ 2/3 \text{ se } P = D, \end{aligned}$$

- Nelle foglie i puntatori sono inutili, perché nulli.
- In ogni albero binario con n nodi, ci sono $n + 1$ alberi vuoti $\Rightarrow$ puntatori nulli.
- Se l'albero è a nodi pieni (ogni nodo o ha due figli o nessuno), allora se ha n nodi interni, ha $n + 1$ foglie.

Spazio richiesto
Tre puntatori

- Se il dato è grande, nel nodo metto solo il puntatore
- Ho quindi tre puntatori, tutti di overhead

$$\begin{aligned} \text{Tot:}\;& n(3P + D) \\ \text{OH:}\;& 3nP \\ \text{perc.:}\;& 3P/(3P + D);\ 3/4 \text{ se } P = D, \end{aligned}$$

Spazio richiesto

Dati solo nelle foglie

- Se il dato è solo nelle foglie,
  - se l'albero non è a nodi pieni, posso avere una catena e quindi un overhead arbitrariamente alto
  - se l'albero è a nodi pieni, la percentuale di nodi interni è circa la metà e l'overhead è minimo
  - nelle situazioni intermedie dipende da quanto mi avvicino all'albero a nodi pieni

Spazio richiesto

Niente puntatori nelle foglie

- Se l'albero è a nodi pieni il vantaggio è notevole.
- Ogni nodo interno: $2P + D$ e sono circa $n/2$
- Ogni foglia: D e sono circa $n/2$
- Percentuale di overhead:

$$\frac{\frac{n}{2}(2P)}{\frac{n}{2}(2P) + nD} = \frac{P}{P + D}$$

- Se $D = P$ la percentuale di overhead è di circa $1/2$.
- Se però i dati sono solo nelle foglie, lo spazio D nei nodi interni è inutilizzato e la percentuale di overhead sale a $3/4$

Spazio richiesto

Niente puntatori nelle foglie, niente dato nei nodi interni

- Nodi interni: solo puntatori ai figli
- Foglie: puntatore al dato
- Totale
$$\frac{n}{2}2P + \frac{n}{2}(P + D)$$
- Se $P = D$ ottengo $3P/(3P + D) = 3/4$
- Lo spazio total è diminuito, mentre è aumentato il rate di overhead perché adesso il dato sta solo nelle foglie

Spazio richiesto

Differenti tipi di nodo

- Occorre implementare anche un metodo virtuale `isLeaf()` con l'implementazione nelle sottoclassi.
- Se proprio necessario, si può ridurre ad un bit, ma lo si paga in leggibilità e portabilità: di solito non ne vale la pena

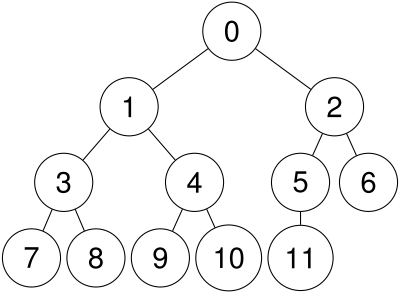
Alberi Binari Completi

Implementazione basata su array

- Se conosciamo la struttura dell'albero, possiamo ridurre l'overhead legato ai puntatori.
- Se l'albero binario è completo, dato il numero di nodi n la sua struttura è unica.
- Numero i nodi partendo dalla radice e scorrendo ciascun livello da sinistra verso destra in modo univoco.
- Questa numerazione mi dà la posizione all'interno dell'array.
- $\text{Parent}(r) = \lfloor (r-1)/2 \rfloor$ se $r \neq 0$
- $\text{LeftChild}(r) = 2r + 1$ se $2r + 1 < n$
- $\text{RightChild}(r) = 2r + 2$ se $2r + 2 < n$
- $\text{LeftSibling}(r) = r - 1$ se r è pari
- $\text{RightSibling}(r) = r + 1$ se r è dispari e $r + 1 < n$

Esempio

n=11



Posizione	0	1	2	3	4	5	6	7	8	9	10	11
Genitore	-	0	0	1	1	2	2	3	3	4	4	5
Figlio sin.	1	3	5	7	9	11	-	-	-	-	-	-
Figlio dx.	2	4	6	8	10	-	-	-	-	-	-	-
Fratello sin.	-	-	1	-	3	-	5	-	7	-	9	-
Fratello dx.	-	2	-	4	-	6	-	8	-	10	-	-