Alberi Binari di Ricerca

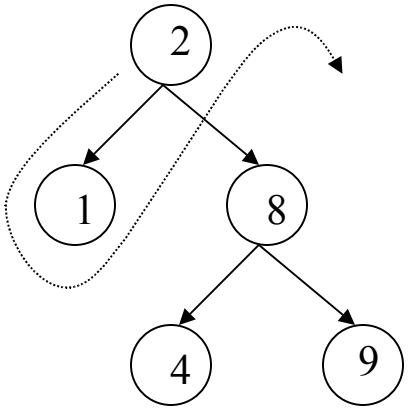
Algoritmi su gli alberi binari: visite

- Dato un puntatore alla radice di un albero vogliamo scandire in modo sistematico tutti i nodi di tale albero
- In una lista abbiamo una unica possibilità: quella di seguire il link al nodo successivo
- Con un albero binario sono possibili 3 strategie:
  - *preordine* o ordine anticipato: si visita prima il nodo e poi i sottoalberi sinistro e destro
  - *inordine* o ordine simmetrico: si visita prima il sottoalbero sinistro e poi il nodo e poi il sottoalbero destro
  - *postordine* o ordine posticipato: si visita prima il sottoalbero sinistro, poi quello destro e poi il nodo

Visita in ordine simmetrico

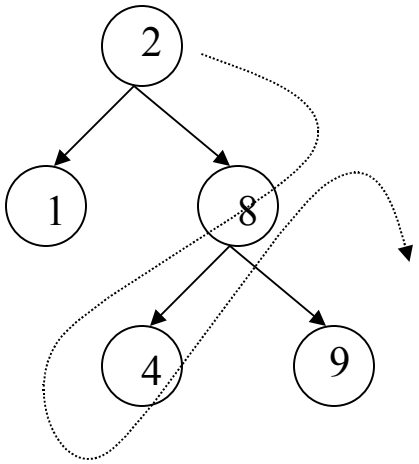
```
Inorder-Tree-Walk(x)
1  if x ≠ NIL
2  then Inorder-Tree-Walk(left[x])
3       stampa(key[x])
4       Inorder-Tree-Walk(right[x])
```

Visualizzazione



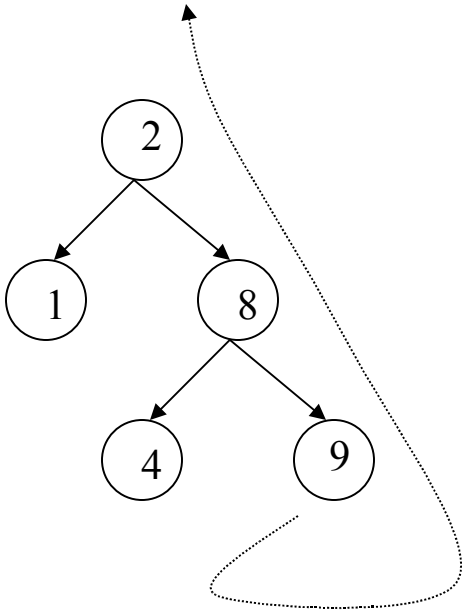
- si parte in 2
- viene chiamata la funzione sul figlio sinistro
- siamo in 1
- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 1
- stampiamo 1
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- torniamo in 1
- la funzione termina
- torniamo in 2
- stampiamo 2
- ….

Visualizzazione



- viene chiamata la funzione sul figlio destro
- siamo in 8
- viene chiamata la funzione sul figlio sinistro
- siamo in 4
- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 4
- stampiamo 4
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- la funzione termina
- torniamo in 8
- stampiamo 8
- viene chiamata la funzione sul figlio destro
- siamo in 9

Visualizzazione

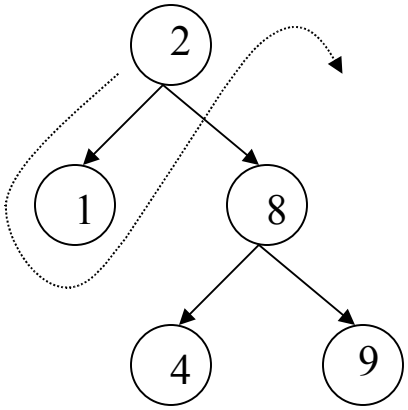


- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 9
- stampiamo 9
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- torniamo in 9
- la funzione termina
- torniamo in 8
- la funzione termina
- torniamo in 2
- la funzione termina

Visita in ordine anticipato

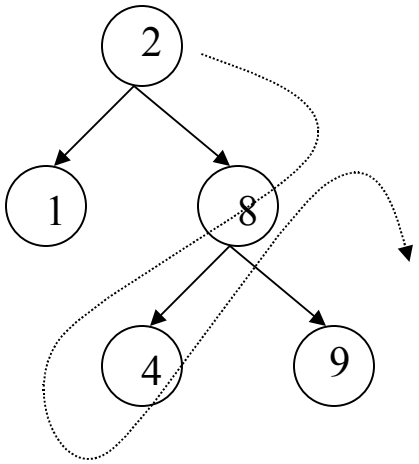
```
Inorder-Tree-Walk(x)
1  if x ≠ NIL
2  then stampa(key[x])
3       Inorder-Tree-Walk(left[x])
4       Inorder-Tree-Walk(right[x])
```

Visualizzazione



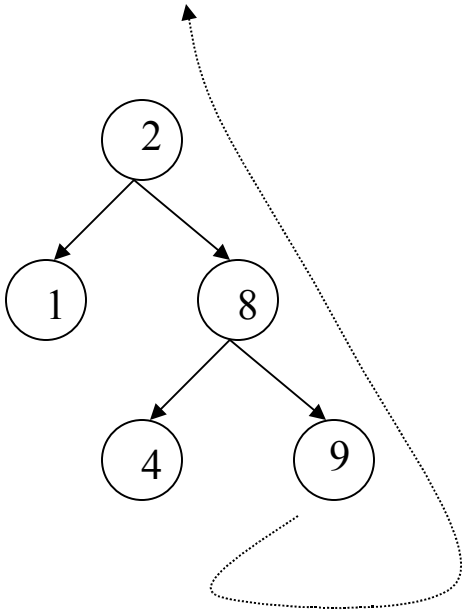
- si parte in 2
- stampiamo 2
- viene chiamata la funzione sul figlio sinistro
- siamo in 1
- stampiamo 1
- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 1
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- torniamo in 1
- la funzione termina
- torniamo in 2

Visualizzazione



- viene chiamata la funzione sul figlio destro
- siamo in 8
- stampiamo 8
- viene chiamata la funzione sul figlio sinistro
- siamo in 4
- stampiamo 4
- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 4
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- la funzione termina
- torniamo in 8
- viene chiamata la funzione sul figlio destro
- siamo in 9

Visualizzazione



- viene chiamata la funzione sul figlio sinistro
- non esiste figlio sinistro e la ricorsione termina
- torniamo in 9
- viene chiamata la funzione sul figlio destro
- non esiste figlio destro e la ricorsione termina
- torniamo in 9
- la funzione termina
- torniamo in 8
- la funzione termina
- torniamo in 2
- la funzione termina

Visita in ordine posticipato

```
Inorder-Tree-Walk(x)
1  if x ≠ NIL
2  then Inorder-Tree-Walk(left[x])
3       Inorder-Tree-Walk(right[x])
4       stampa(key[x])
```

Algoritmi ricorsivi su alberi binari

- Capita di dover determinare dei parametri strutturali di un albero avendo in ingresso solo il link alla radice
- Si può sfruttare la struttura ricorsiva degli alberi ed realizzare versioni ricorsive delle funzioni di interesse
- Consideriamo una funzione per determinare il numero di nodi ed una per determinare l'altezza dell'albero

Funzioni ricorsive

```
int count(link h)
  {
    if (h == NULL) return 0;
    return count(h->l) + count(h->r) + 1;
  }

int height(link h)
  {
    int u, v;
    if (h == NULL) return -1;
    u = height(h->l); v = height(h->r);
    if (u > v) return u+1; else return v+1;
  }
```

Ricerca

- Molte applicazioni richiedono un insieme dinamico che fornisca solo operazioni di:
  - inserimento
  - cancellazione
  - ricerca
  - massimo/minimo
  - predecessore/successore (il più piccolo/grande elemento maggiore/minore di un elemento dato)

Alberi Binari di ricerca

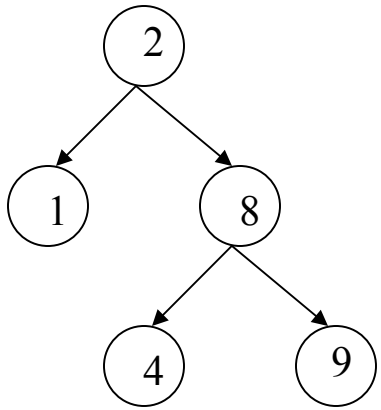
- Gli alberi binari di ricerca sono strutture dati dinamiche che forniscono le operazioni richieste (insert, delete, search, maximum, predecessor, etc) in tempo limitato asintoticamente dall'altezza dell'albero, cioè per le varie operazioni si ha $T(n)=O(h)$

Alberi Binari di Ricerca

- Un albero binario di ricerca è un albero binario in cui le chiavi soddisfano la:

  proprietà dell'albero binario di ricerca

  - sia x un nodo
  - key di nodo in sottoalbero radicato in left[x] ≤ key di x
  - key di nodo in sottoalbero radicato in right[x] ≥ key di x

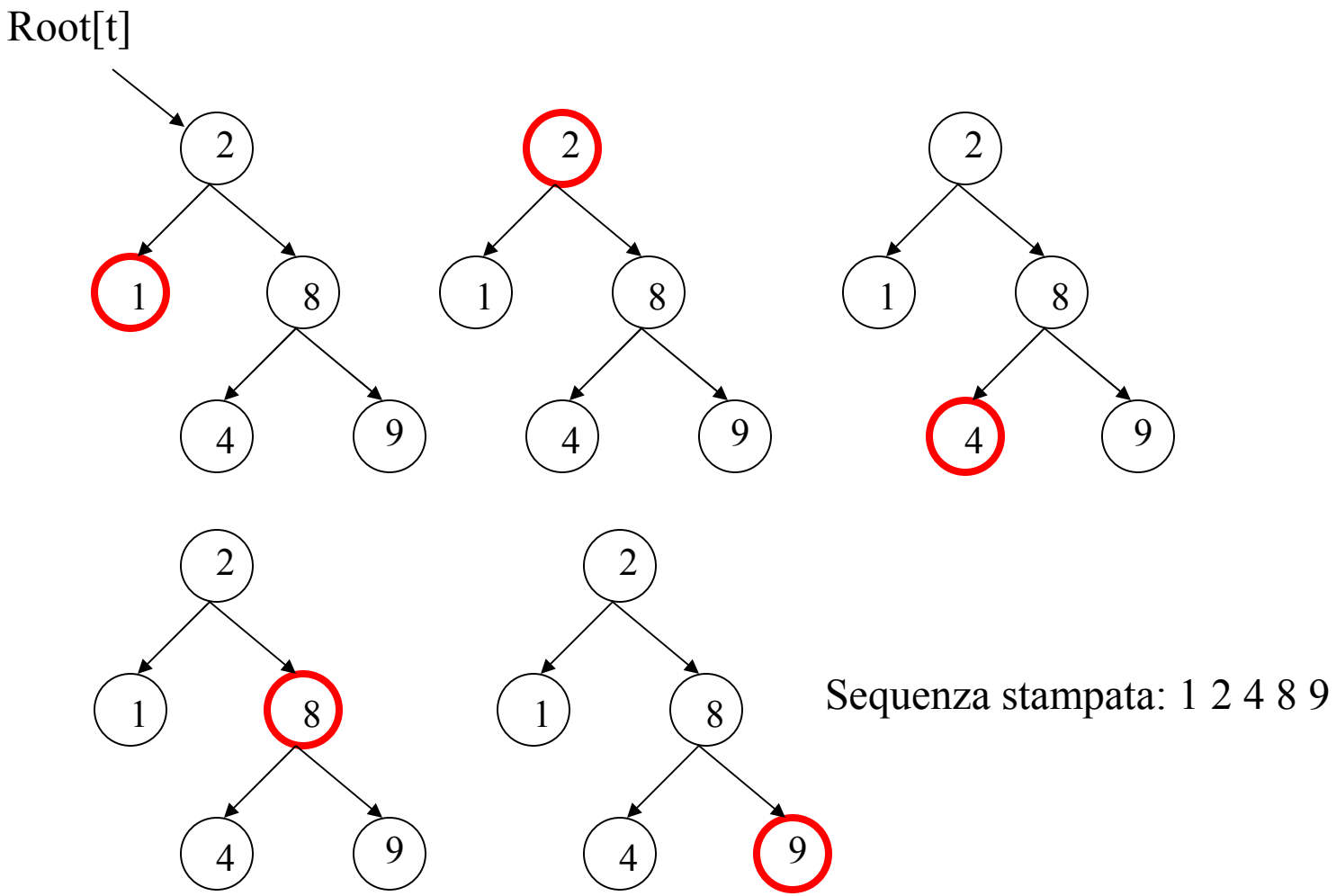


Ordinamento delle chiavi

- In un albero binario di ricerca l'operazione di ordinamento viene eseguita semplicemente attraversando i nodi dell'albero in modo opportuno

- Cerchiamo una visita che stampi prima gli elementi minori e poi quelli maggiori
- ...dato che i nodi nel sottoalbero sinistro hanno chiave minore di quella del nodo radice si dovranno stampare prima questi, poi il nodo radice e infine i nodi con chiave maggiore, cioe' quelli del sottoalbero destro
- -> la visita e' in ordine simmetrico

Visualizzazione



La ricerca

- L'operazionoe di ricerca consiste nel rispondere se e' vero o falso che un dato elemento e' stato memorizzato nella struttura dati e in caso affermativo restituire un puntatore a questo

- L’idea è di confrontare la chiave di un nodo x con la chiave cercata
- nel caso che non coincidano si cerca solo nel sottoalbero che ha la possibilta' di averla ...
- ... ed è possibile sapere quale sia questo sottoalbero perché tutti i nodi del sottoalbero destro (sinistro)

PseudoCodice per la Ricerca (versione ricorsiva)

```
Tree-Search(x,k)
1  if x = NIL or k = key[x]
2  thenreturn x
3  if k < key[x]
4  thenreturn Tree-Search(left[x],k)
5  else return Tree-Search(right[x],k)
```

Tempo di calcolo

- La procedura discende a partire dalla radice l'albero e restituisce un puntatore al nodo la cui chiave coincide con la chiave cercata
- nel caso in cui essa non esista la procedura discende comunque fino ad una foglia e restituisce un puntatore nullo
- Il tempo impiegato è proporzionale alla lunghezza del cammino percorso, ovvero limitato dalla altezza dell'albero
- pertanto $T(n)=O(h)$
  - per albero binario completo $T(n)=O(\lg n)$
  - per albero degenere (lista) $T(n)=O(n)$

Nota

- E’ possibile scrivere qualsiasi procedura ricorsiva in forma non ricorsiva (e viceversa)
- La forma ricorsiva è spesso più elegante e compatta ma non la più efficiente
- Di seguito si da una procedura non ricorsiva per la ricerca in un albero binario
  - invece di effettuare un numero elevato di chiamate a funzioni si aggiorna il valore di un unico puntatore!

PseudoCodice per la Ricerca (versione iterativa)

```
Iterative-Tree-Search(x,k)
1  while x ≠ NIL and k ≠ key[x]
2  do   if k < key[x]
4       then   x ← left[x]
5       else   x ← right[x]
6  return x
```