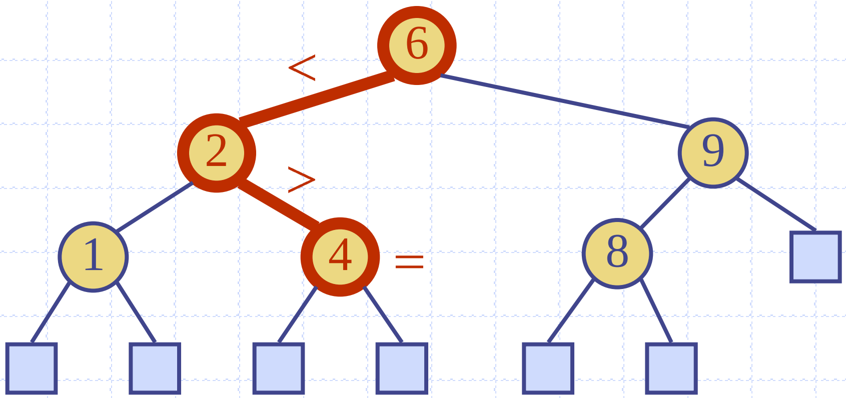
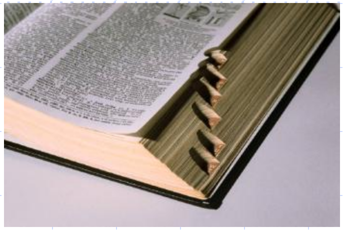
Alberi Binari di Ricerca

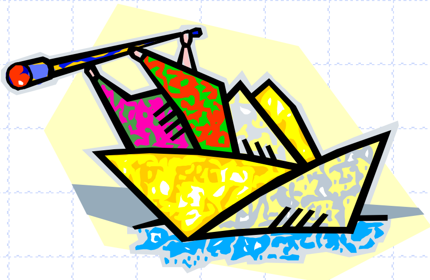


Dizionari ordinati



- Si assume che le chiavi appartengano ad un dominio sul quale è definita una relazione di ordine totale.
- Nuove operazioni:
  - first(): prima entry secondo l'ordinamento definito
  - last(): ultima entry secondo l'ordinamento definito
  - successors(k): iteratore sulle entry con chiavi non inferiori a k (ordine crescente)
  - predecessors(k): iteratore sulle entry con chiavi non superiori a k (ordine decrescente)

Ricerca binaria



- La ricerca binaria realizza l'operazione find(k) su un dizionario implementato come sequenza basata su array, ordinata per chiave
  - simile al gioco "alto-basso"
  - ad ogni passo, il numero di candidati viene dimezzato
  - termina perciò dopo un numero logaritmico (log n) di passi
- Esempio: find(7)

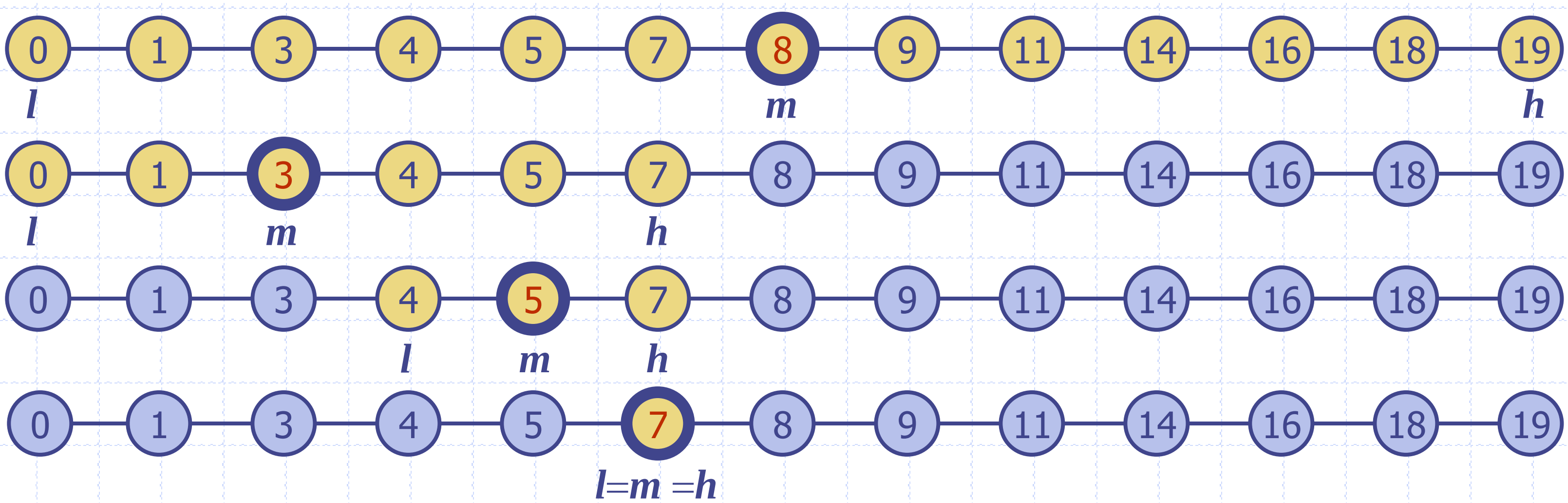
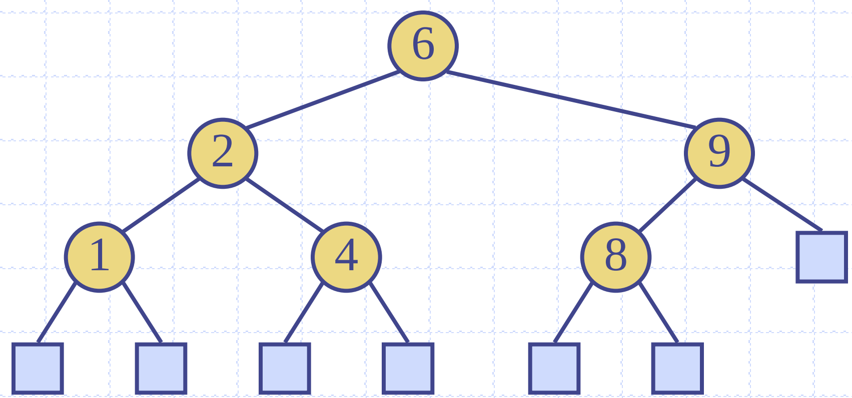


Tabelle di ricerca

- Una tabella di ricerca è un dizionario implementato tramite un array ordinato
  - si memorizzano le entry in una sequenza basata su array, ordinata per chiave
  - si usa apposito oggetto Comparator per le chiavi
- Prestazioni:
  - find richiede tempo $O(\log n)$, usando la ricerca binaria
  - insert richiede tempo $O(n)$ perché nel caso peggiore è necessario spostare n elementi per far spazio al nuovo
  - remove richiede tempo $O(n)$ poiché nel caso peggiore è necessario spostare n elementi per ricompattare l'array dopo l'eliminazione
- Una tabella di ricerca è efficace solo per dizionari di piccole dimensioni o per dizionari dove la ricerca è l'operazione più frequente, mentre inserimenti e rimozioni sono eseguiti raramente (ad es., autorizzazioni per carte di credito)

Alberi binari di ricerca (10.1)

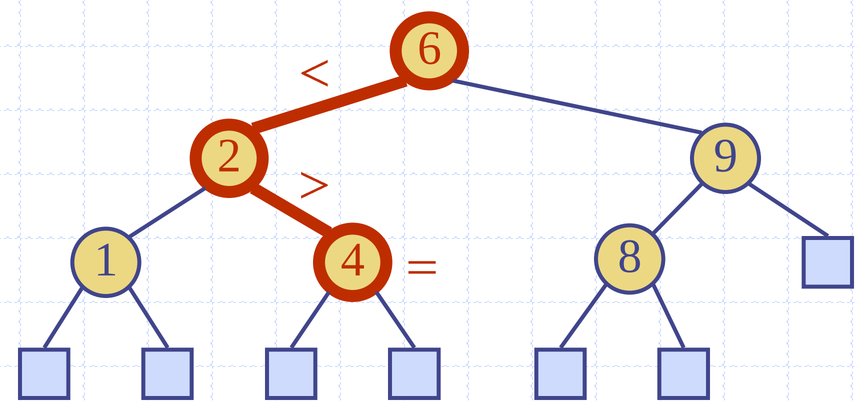
- Un albero binario di ricerca è un albero binario che memorizza chiavi (o entry chiave-valore) nei suoi nodi interni e che soddisfa la seguente proprietà
  - se u, v e w sono tre nodi tali che u è nel sottoalbero sinistro di v e w è nel sottoalbero destro di v, allora, $key(u) \leq key(v) \leq key(w)$
- I nodi esterni non contengono elementi e l'albero è proprio
  - assunzioni semplificative e non realmente necessarie
- L'attraversamento in-ordine di un albero binario di ricerca visita le chiavi in ordine non decrescente



Ricerca (10.1.1)

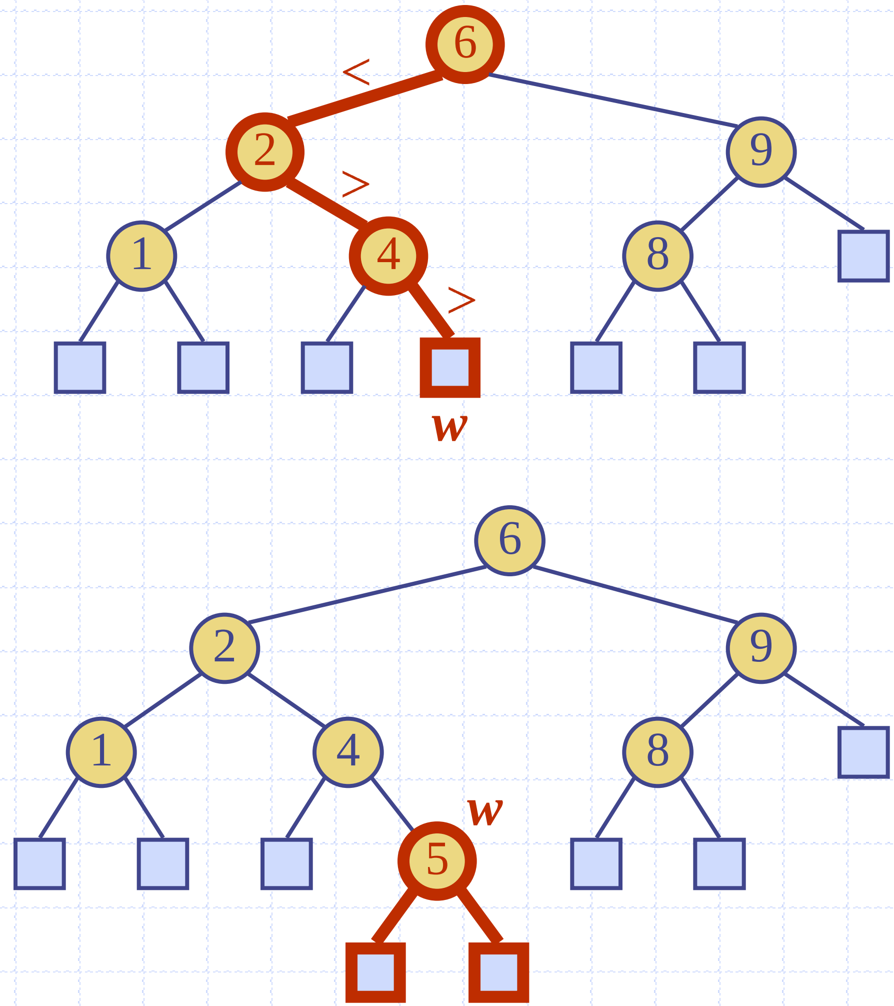
- La ricerca di una chiave ***k*** prevede il tracciamento di un percorso verso il basso, ad iniziare dalla radice
- Alla visita di un nodo, si stabilisce quale sia il prossimo nodo da esaminare in base al risultato del confronto fra ***k*** e la chiave visitata
- Se si raggiunge una foglia, la ricerca fallisce e si restituisce null
- Esempio: find(4):
  - Esecuzione di TreeSearch(4,root)

```
Algorithm TreeSearch(k, v)
    if T.isExternal (v)
        return v { v non contiene entry }
    if k < key(v)
        return TreeSearch(k, T.left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k, T.right(v))
```



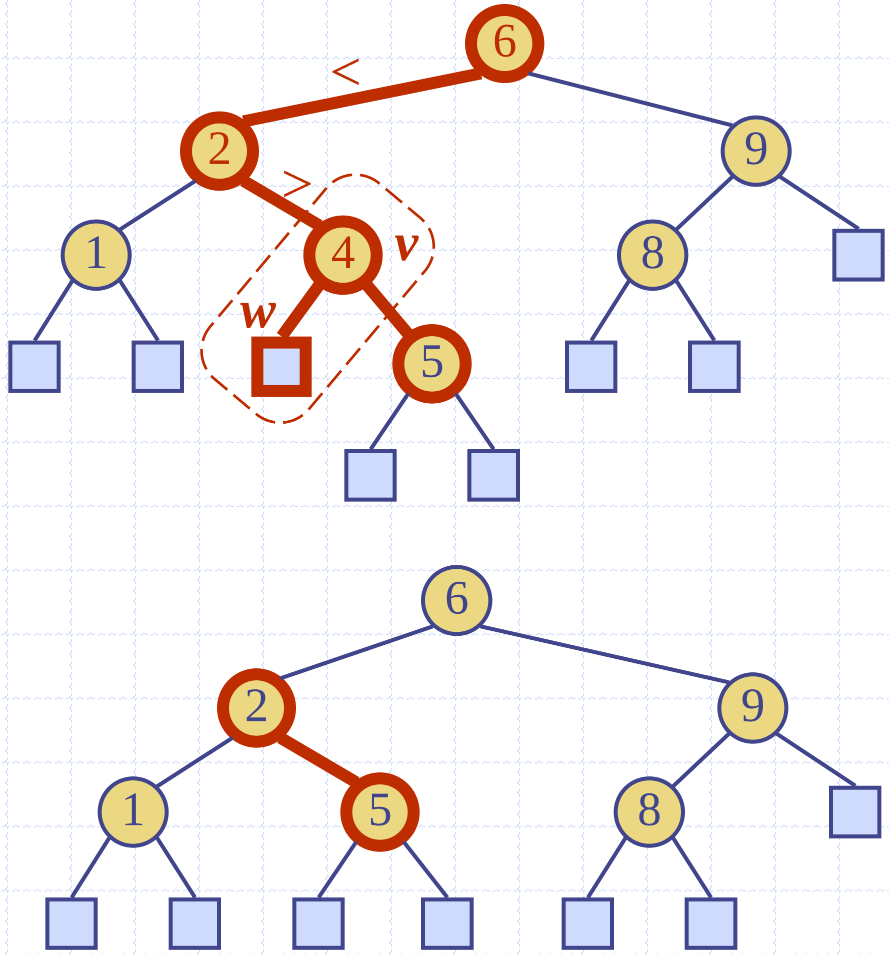
Inserimento

- Per eseguire l'operazione insert(k, o) si cerca dapprima la chiave k (usando TreeSearch)
- Assumendo che k non sia già nell'albero, sia w la foglia raggiunta nella ricerca
- Si inserisce k nel nodo w, che va anche trasformato in nodo interno
- Esempio: insert 5



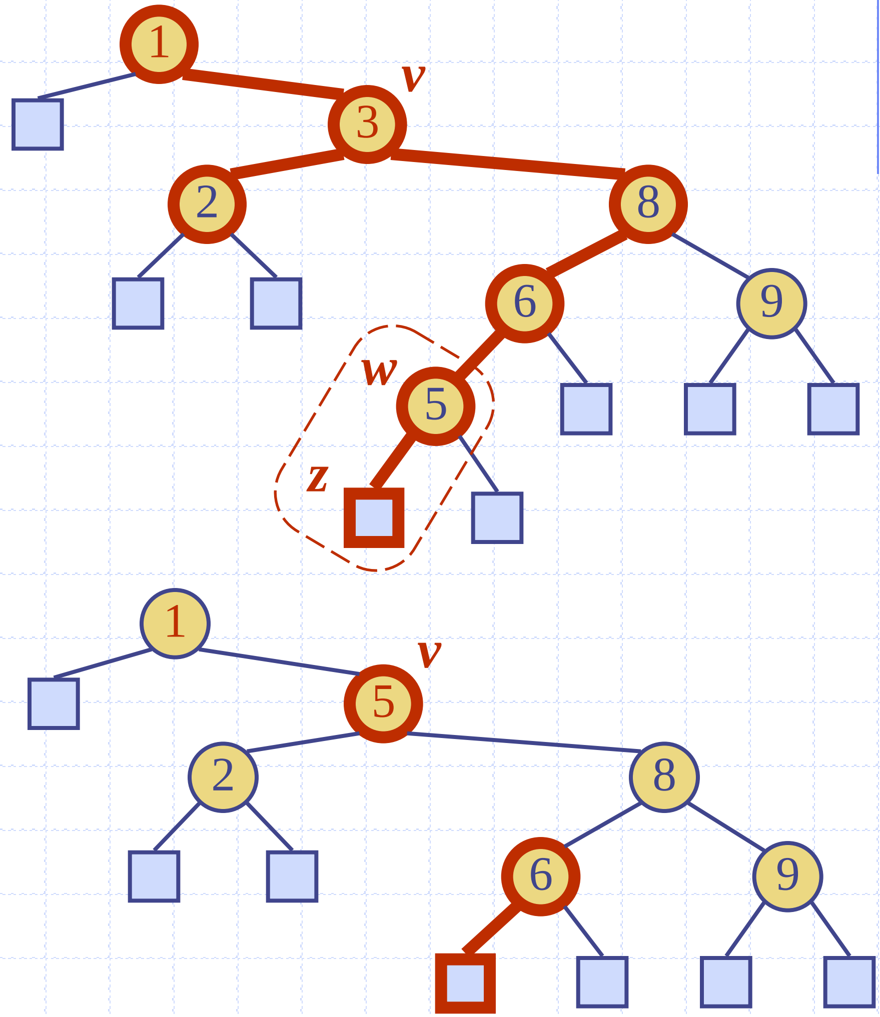
Cancellazione

- Per eseguire l'operazione remove(k), si cerca la chiave k
- Si assuma che k sia nell'albero, nel nodo v
- Se v ha un figlio esterno w, si rimuovono v e w dall'albero tramite l'operazione removeExternal(w), che rimuove w ed il suo genitore
- Esempio: remove 4



Cancellazione (cont.)

- Si considera ora il caso in cui la chiave ***k*** da rimuovere è memorizzata nel nodo ***v*** i cui figli sono entrambi interni
  - si individua il nodo interno ***w*** che segue ***v*** nell'attraverso in-ordine
  - si copia ***key***(***w***) nel nodo ***v***
  - si rimuove il nodo ***w*** ed il suo figlio sinistro ***z*** (che deve essere foglia) tramite l'operazione removeExternal(***z***)
- Esempio: remove 3



Prestazioni

- In un dizionario con n elementi implementato con un albero binario di ricerca di altezza h
  - lo spazio utilizzato è $O(n)$
  - i metodi find, insert e remove richiedono tempo $O(h)$
- L'altezza h è $O(n)$ nel caso peggiore e $O(\log n)$ in quello migliore