Data Structures and Algorithms
(資料結構與演算法)

Lecture 12: Binary Search Tree

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Motivation: Dictionary

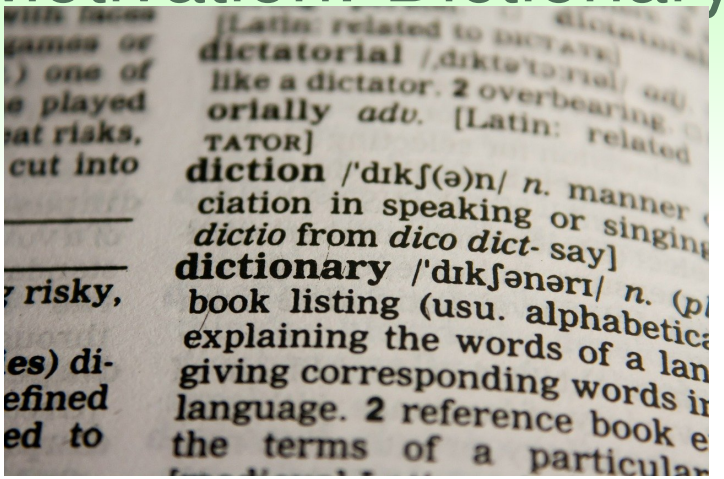


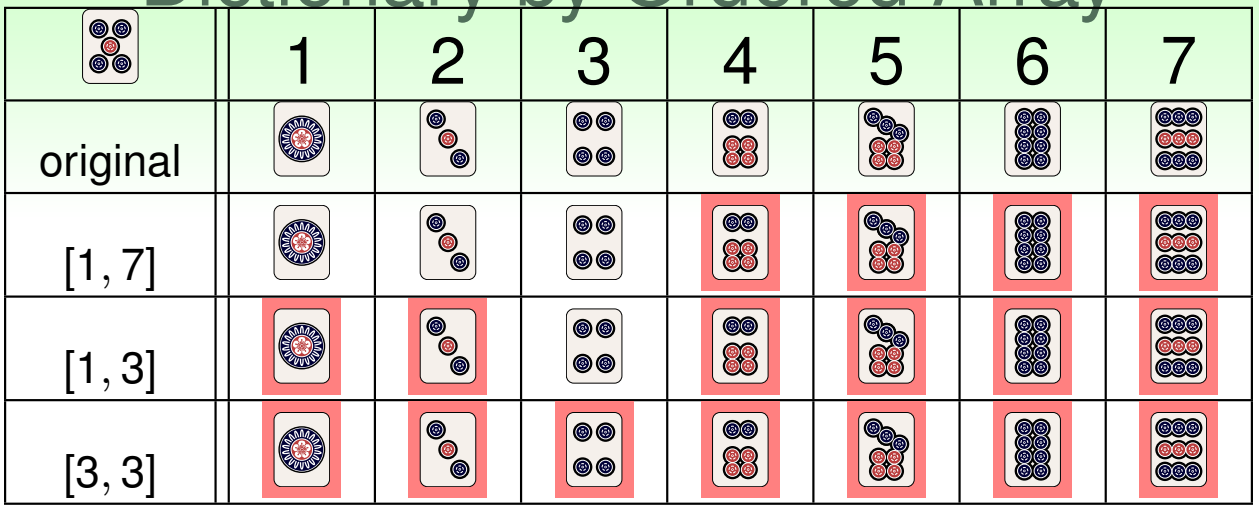
figure by PDPics, licensed for free via Pixabay

key → data, like word → explanation

- key: string, integer, etc.
  - possibly comparable like keys in priority queue
  - usually unique
- also called associative array or map
  - data inserted by $D[key] = data$
  - data accessed by $D[key]$

will illustrate with integer *key* first for simplicity

Dictionary by Ordered Array



BIN-SEARCH(A, key, ℓ, r)

```
while ℓ ≤ r
    m = floor((ℓ + r)/2)
    if A[m] equals key
        return m
    elseif A[m] > key
        r = m − 1 // cut out end
    elseif A[m] < key
        ℓ = m + 1 // cut out begin
return NIL
```

BIN-SEARCH: multiple shortcuts
by quickly checking the middle, remember? :-)

Binary Search Tree

design wishes

- more flexible insertion than array: (binary) tree
- similar algorithmic structure to BIN-SEARCH

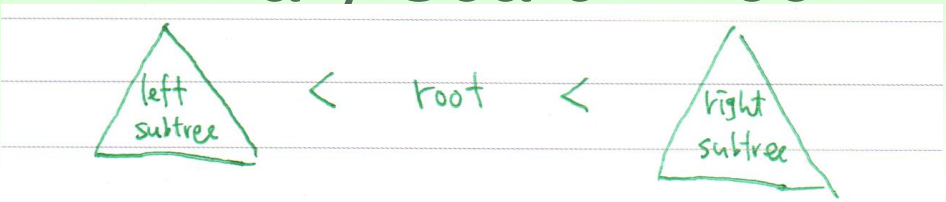
key idea

root $\sim$ middle

- ordered array: [left, middle) < middle < (middle, right]
- **binary search tree**: left sub-tree < root < right sub-tree **for every tree, sub-tree, sub-sub-tree, ...**

binary search tree: binary search with **tree** rather than ordered array

Binary Search Tree



BST-SEARCH(T, key)

```
while T ≠ NIL
        mid = T // root
        if mid.key = key
                return mid
        elseif mid.key < key
                T =??
        elseif mid.key > key
                T =??
return NIL
```

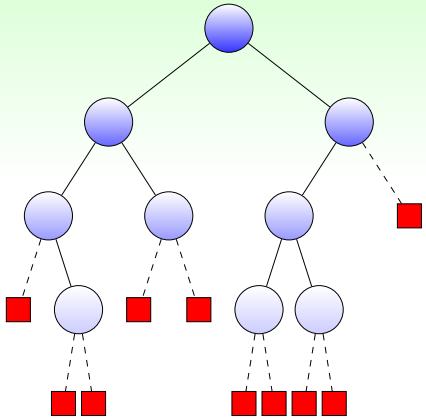
complexity??

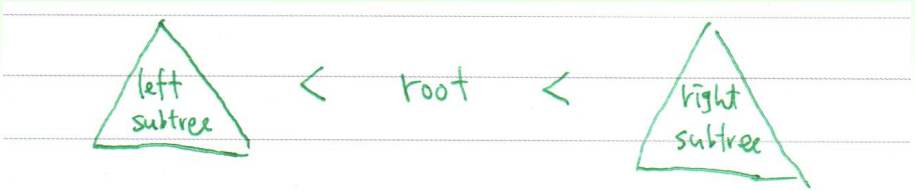
BINARY-SEARCH(A, key)

```
left = 1, right = A.length
while left ≤ right
        mid = floor((left+right)/2)
        if A[mid] = key
                return mid
        elseif A[mid] < key
                left = mid + 1
        elseif A[mid] > key
                right = mid − 1
return NIL
```

complexity: $O(\log n)$

Insertion to Binary Search Tree





BST-INSERT(T, key)

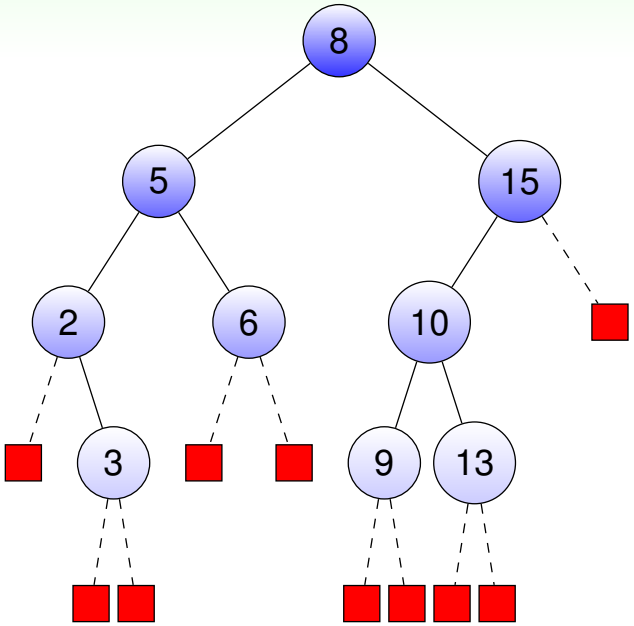
```
while T ≠ ■
    mid = T // root
    if mid.key = key
        return mid
    elseif mid.key < key
        T = T.right
    elseif mid.key > key
        T = T.left
T = NewInternalNode(key)
```

BST-SEARCH(T, key)

```
while T ≠ NIL
    mid = T // root
    if mid.key = key
        return mid
    elseif mid.key < key
        T = T.right
    elseif mid.key > key
        T = T.left
return NIL
```

again, $O(h)$ if not doing anything strategic

In-Order Traversal on BST




```
INORDER-TRAVERSAL(T)
if T is NIL
        return
else

        INORDER-TRAVERSAL(T.left)

        action with T.data

        INORDER-TRAVERSAL(T.right)
```

Heap-Sort: build a heap with $O(n)$, remove with $O(n \log n)$;
Tree-Sort: build a BST with $O(nh)$, traversal with $O(n)$