ECE250: Algorithms and Data Structures
Binary Search Trees (Part A)

Ladan Tahvildari, PEng, SMIEEE
Professor
Software Technologies Applied Research (STAR) Group
Dept. of Elect. & Comp. Eng.
University of Waterloo

Materials from CLRS: Chapter 12.1 and 12.2
Acknowledgements

- The following resources have been used to prepare materials for this course:
  - MIT OpenCourseWare
  - *Introduction To Algorithms* (CLRS Book)
  - *Data Structures and Algorithm Analysis in C++* (M. Wiess)
  - *Data Structures and Algorithms in C++* (M. Goodrich)

- Thanks to many people for pointing out mistakes, providing suggestions, or helping to improve the quality of this course over the last ten years:
Dictionaries

- **Dictionary ADT** – a dynamic set with methods:
  - Search(S, k), Insert(S, x), Delete(S, x)

- **Ordered Dictionaries** – a dynamic set with the following additional methods:
  - **Min (S)** – an access operation that returns a pointer to the element of S with the smallest key
  - **Max (S)** – an access operation that returns a pointer to the element of S with the largest key
  - **Successor (S, x)** – an access operation that returns a pointer to the next larger element in S for a given x, or NIL if x is the maximum element
  - **Predecessor (S, x)** – an access operation that returns a pointer to the next smaller element in S for a given x, or NIL if x is the minimum element

These operations require that keys are comparable
A List-Based Implementation

- **Unordered list**
  - search, min, max, predecessor, successor: $O(n)$
  - insert: $O(1)$

- **Ordered list**
  - search, insert: $O(n)$
  - min, max, predecessor, successor: $O(1)$
Binary Search

- Narrow down the search range in stages
  - findElement(22)

```
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
```

```
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
```

```
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
```

```
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
```

```
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
```
Running Time

- The range of candidate items to be searched is halved after comparing the key with the middle element

- Binary search runs in $O(lgn)$ time (remember recurrence in Lecture 4-Slide #17)

- What about insertion and deletion?
  - search: $O(lg n)$
  - insert, delete: $O(n)$
  - min, max, predecessor, successor: $O(1)$
Binary Tree ADT

- BinTree ADT:
  - Accessor functions:
    - key(): int
    - parent(): BinTree
    - left(): BinTree
    - right(): BinTree
  - Modification procedures:
    - setKey(k:int)
    - setParent(T:BinTree)
    - setLeft(T:BinTree)
    - setRight(T:BinTree)
Binary Search Trees

- A binary search tree is a binary tree $T$ such that
  - each internal node stores an item ($k$) of a dictionary
  - keys stored at nodes in the left subtree are less than or equal to $k$
  - keys stored at nodes in the right subtree are greater than (or equal) to $k$

- Example sequence 2,3,5,5,7,8
Searching in a BST

To find an element with key $k$ in a tree $T$

- compare $k$ with $T.key()$
- if $k < T.key()$, search for $k$ in $T.left()$
- otherwise, search for $k$ in $T.right()$
Pseudocode for BST Search

- **Recursive version** – *divide-and-conquer algorithm*

```plaintext
Search(T, k)
01 if T = NIL then return NIL
02 if k = T.key() then return T
03 if k < T.key()
04    then return Search(T.left(), k)
05    else return Search(T.right(), k)
```

- **Iterative version**

```plaintext
Search(T, k)
01 x ← T
02 while x ≠ NIL and k ≠ x.key() do
03    if k < x.key()
04      then x ← x.left()
05    else x ← x.right()
06 return x
```
BST Search - Example 1

- Search($T$, 11)
BST Search - Example 2

- Search($T$, 6)
Analysis of BST Search

- *Running time on tree of height* $h$ *is* $O(h)$
- After the insertion of $n$ keys, the worst-case running time of searching is $O(n)$
BST Minimum (Maximum)

- Find the minimum key in a tree rooted at x

```
TreeMinimum(x)
01 while x.left() ≠ NIL do
02 x ← x.left()
03 return x
```

- Running time $O(h)$, i.e., it is proportional to the height of the tree
**BST Successor**

- Given $x$, find the node with the smallest key greater than $x.key()$

- We can distinguish two cases, depending on the right subtree of $x$
BST Successor – Case 1

- Right subtree of x is nonempty
- Successor is the leftmost node in the right subtree
- This can be done by returning $\text{TreeMinimum}(x.\text{right()})$
**BST Successor – Case 2**

- The right subtree of $x$ is empty

- Successor is the lowest ancestor of $x$ whose left child is also an ancestor of $x$
BST Successor Pseudocode

TreeSuccessor(x)
01 if x.right() ≠ NIL
02 then return TreeMinimum(x.right())
03 y ← x.parent()
04 while y ≠ NIL and x = y.right()
05 x ← y
06 y ← y.parent()
03 return y

For a tree of height h, the running time is O(h)