Lecture 02/01/17

Lecturer: Xiaodi Wu

Reading Assignment: Chapter 12.2
Search in a sorted table

- Search a key \( k \) in a table of size \( n \). Trivial \( O(n) \).
Search in a sorted table

- Search a key $k$ in a table of size $n$. Trivial $O(n)$.
- In a sorted table (non-decreasing order): $O(\log(n))$. **Binary Search!**
Search in a sorted table

- Search a key $k$ in a table of size $n$. Trivial $O(n)$.
- In a sorted table (non-decreasing order): $O(\log(n))$. **Binary Search!**

How?

- Maintain three pointers: low, high, and mid = (low + high)/2.
Search in a sorted table

- Search a key $k$ in a table of size $n$. Trivial $O(n)$.
- In a sorted table (non-decreasing order): $O(\log(n))$. **Binary Search**!

How?

- Maintain three pointers: low, high, and mid = (low + high)/2.
- Compare $k$ with the key of the mid. If $k = key(mid)$, return mid.
- If $k < key(mid)$, then update the pointer
  $low \leftarrow low,$ $high \leftarrow mid - 1$.
- If $k > key(mid)$, then update the pointer
  $low \leftarrow mid + 1,$ $high \leftarrow high.$
Algorithm BinarySearch($S, k, low, high$)
Input: an ordered vector $S$ storing $n$ items.
Output: an element with key $k$ within $[low, high]$; otherwise, NO_SUCH_KEY.
if $low > high$ then
    return NO_SUCH_KEY
else
    $mid \leftarrow (low + high)/2$
    if $k = key(mid)$ then
        return $mid$.
    else if $k < key(mid)$ then
        return BinarySearch($S, k, low, mid-1$).
    else
        return BinarySearch($S, k, mid+1, high$).
end if
end if
Binary Search: time and correctness

Time

- Watch the difference between low and high. Shrink to half in each recursive call.

\[ O(\log(n)) \]
Binary Search: time and correctness

Time

- Watch the difference between low and high. Shrink to half in each recursive call.
- \(O(\log(high - low)) = O(\log(n))\).

Correctness

- Maintain an invariant: the key is either within \([\text{low}, \text{high}]\) or does not exist.
- Invariant remains during recursive calls.
Binary Search: time and correctness

Time
- Watch the difference between low and high. Shrink to half in each recursive call.
- $O(\log(high - low)) = O(\log(n))$.

Correctness
- Maintain an invariant: the key is either within $[low, high]$ or does not exist.
- Invariant remains during recursive calls.
Binary Search Tree

Definition

- **Binary Search Tree**: for every internal node $e$, the elements in the left subtree are $\leq e$, and the elements in the right subtree are $\geq e$.
- Goal: binary search on a tree data structure.
Binary Search Tree

Definition

- **Binary Search Tree**: for every internal node $e$, the elements in the left subtree are $\leq e$, and the elements in the right subtree are $\geq e$.

- Goal: binary search on a tree data structure.

Search

- Compare $k$ with the key of the root. If $k = \text{key}(\text{root})$, return root.
- If $k < \text{key}(\text{root})$, then search in the left subtree.
- If $k > \text{key}(\text{root})$, then search in the right subtree.
Binary Search Trees
Search in binary search trees

Algorithm TreeSearch\((x, k)\)
Input: a search key \(k\) and a node \(x\) of a binary search tree \(T\).
Output: the node with key \(k\) or NIL.

if \((x == \text{NIL})\) or \((k == x.\text{key})\) then
  return \(x\).
end if

if \(k < x.\text{key}\) then
  return TreeSearch\((x.\text{left}, k)\).
else
  return TreeSearch\((x.\text{right}, k)\).
end if
Search in binary search trees

**Algorithm** TreeSearch(x, k)

Input: a search key k and a node x of a binary search tree T.
Output: the node with key k or NIL.

if (x == NIL) or ( k == x.key)  then
  return x.
end if

if  k < x.key  then
  return TreeSearch(x.left, k).
else
  return TreeSearch(x.right, k).
end if

- Time: $O(h)$ could from $O(\log n)$ to $O(n)$. 
Binary Search Trees: TreeSearch(78, root)
Inorder Traversal of Binary Search Trees

- Inorder Traversal leads to a nondecreasing sequence.

  \[17, 28, 29, 32, 44, 54, 66, 76, 78, 80, 82, 88, 97\]

- Given a binary tree: inorder traversal nondecreasing ⇔ binary search tree.
Algorithm Tree-Minimum(x)
while x.left !=NIL do
    x = x.left
end while
return x.

Algorithm Tree-Maximum(x)
while x.right !=NIL do
    x = x.right
end while
return x.
Algorithm Tree-Minimum($x$)
while $x$.left $\neq$ NIL do
    $x = x$.left
end while
return $x$.

Algorithm Tree-Maximum($x$)
while $x$.right $\neq$ NIL do
    $x = x$.right
end while
return $x$.

Complexity
$O(h)$ which is from $O(\log(n))$ to $O(n)$.
**Algorithm** TreeSuccessor($x$)
Input: a node $x$ of a binary search tree $T$.
Output: the successor of $x$ or NIL.

if ($x$.right $\neq$ NIL) then
    return Tree-Minimum($x$.right).
end if

$y = x$.p

while ($y$ $\neq$ NIL) and ($x$ $==$ $y$.right) do
    $x = y$
    $y = y$.p
end while
return $y$
Algorithm TreeSuccessor(x)
Input: a node $x$ of a binary search tree $T$.
Output: the successor of $x$ or NIL.
if ($x$.right $\neq$ NIL) then
    return Tree-Minimum($x$.right).
end if
$y=x$.p
while ($y\neq$NIL) and ($x==y$.right) do
    $x=y$
    $y=y$.p
end while
return $y$

Time: $O(h)$ could from $O(\log n)$ to $O(n)$. 