Binary Search Trees

Chapter 12 of Cormen’s book

Giulia Bernardini
giulia.bernardini@units.it

Algorithmic Design / Algorithms for Scientific Computing

a.y. 2023/2024

Binary Search

Binary search is an efficient algorithm using a divide-and-conquer strategy. Its running time is $O(\log n)$.

Algorithm 3 Binary Search

INPUT: A sorted sequence $s = s[1]s[2]\dots s[n]$ of items from a set X and an item $x \in X$.
OUTPUT: An index $i \in [1, n]$ such that $s[i] = x$; or **FAIL** if no such index exists.
start $\leftarrow 1$, end $\leftarrow n$;
while start $\leq$ end **do**
    mid $\leftarrow \lfloor(\text{start} + \text{end})/2\rfloor$;
    if $s[\text{mid}] = x$ **then**
        return : mid;
    else if $s[\text{mid}] < x$ **then**
        start $\leftarrow$ mid $+ 1$;
    else if $s[\text{mid}] > x$ **then**
        end $\leftarrow$ mid $- 1$;
return : **FAIL**;

Binary Search Trees

BSTs have the following property:

For every node x in the tree, for every node y in the left subtree of x, then $key(x) \leq key(y)$; for every node z in the right subtree of x, $key(z) \geq key(x)$.

```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```

Binary Search Trees

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z          // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

Binary Search Trees

```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

Binary Search Trees

```
TREE-MINIMUM(x)
    while x.left ≠ NIL
        x = x.left
    return x
```

```
TREE-MAXIMUM(x)
    while x.right ≠ NIL
        x = x.right
    return x
```

Binary Search Trees

TREE-SUCCESSOR(x)

```
if x.right ≠ NIL
    return TREE-MINIMUM(x.right)
y = x.p
while y ≠ NIL and x == y.right
    x = y
    y = y.p
return y
```

Binary Search Trees

```
TRANSPLANT(T, u, v)
if u.p == NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

Binary Search Trees

```
TREE-DELETE(T, z)
if z.left == NIL
        TRANSPLANT(T, z, z.right)
elseif z.right == NIL
        TRANSPLANT(T, z, z.left)
else y = TREE-TREE-SUCCESSOR(z)
        if y.p ≠ z
                TRANSPLANT(T, y, y.right)
                y.right = z.right
                y.right.p = y
        TRANSPLANT(T, z, y)
        y.left = z.left
        y.left.p = y
```