Binary Search Trees

Chapter 12 of Cormen's book

Giulia Bernardini
giulia.bernardini@units.it

Algorithmic Design
a.y. 2024/2025

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)

```
1   while x ≠ NIL and k ≠ x.key
2       if k < x.key
3           x = x.left
4       else x = x.right
5   return x
```

Binary Search Trees

```
TREE-INSERT(T, z)
 1   y = NIL
 2   x = T.root
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z            // tree T was empty
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

Binary Search Trees

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

Binary Search Trees

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

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

Binary Search Trees

```
TREE-SUCCESSOR(x)
1   if x.right ≠ NIL
2       return TREE-MINIMUM(x.right)
3   y = x.p
4   while y ≠ NIL and x == y.right
5       x = y
6       y = y.p
7   return y
```

Binary Search Trees

```
TRANSPLANT(T, u, v)
1   if u.p == NIL
2       T.root = v
3   elseif u == u.p.left
4       u.p.left = v
5   else u.p.right = v
6   if v ≠ NIL
7       v.p = u.p
```

Binary Search Trees

```
TREE-DELETE(T, z)
 1   if z.left == NIL
 2       TRANSPLANT(T, z, z.right)
 3   elseif z.right == NIL
 4       TRANSPLANT(T, z, z.left)
 5   else y = TREE-TREE-SUCCESSOR(z)
 6       if y.p ≠ z
 7           TRANSPLANT(T, y, y.right)
 8           y.right = z.right
 9           y.right.p = y
10       TRANSPLANT(T, z, y)
11       y.left = z.left
12       y.left.p = y
```