Lecture 6 Notes

Professor Amir
CSCI - Fundamental Algorithms

October 11 2017

Recap

- Lower Bounds on Comparison based sorting
- Counting Sorts, O(n) run time when all values in a "small" interval.
- Linear time algorithm for median, k-th element.

1 Binary Search Tree (BST)

Introduction. BST is a data strucutre for maintaining dynamic-sets (can be used, e.g. , for priority queues. We shall discuss the following:

- define BST and give algorithms for Basic operations.
- Compare to heaps.

1.1 Properties

- $x.key$ is larger or equal to all keys in its left subtree and is smaller or equal than all keys in its right subtree.
- Also, each node in BST has a x.key, x.left, x.right, x.parent and x can store more satellite data.
- BST is not necessarily close to being balanced!
- Use NIL node to represent no child or no parent.

1.2 Traversals

1.2.1 Inorder-tree-walk:

used for printing elements in sorted order

```
def Inorder_BST_walk(x):
  if x!= None:
    Inorder_tree_walk(x.left)
    print(x.key)
    Inorder_tree_walk(x.right)
```

Correctness By induction.
Running Time: For some k,

$$T(n) = T(k) + T(n-k-1) + O(1)$$

$$\implies \text{Solution:} T(n) = O(n)$$

1.2.2 Preorder-tree-walk:

```
def Inorder_BST_walk(x):
  if x!= None:
    print(x.key)
    Inorder_tree_walk(x.left)
    Inorder_tree_walk(x.right)
```

Correctness By induction.
Running Time: For some k,

$$T(n) = T(k) + T(n-k-1) + O(1)$$

$$\implies \text{Solution:} T(n) = O(n)$$

1.2.3 Postorder-tree-walk:

```
def Inorder_BST_walk(x):
  if x!= None:
    Inorder_tree_walk(x.left)
    Inorder_tree_walk(x.right)
    print(x.key)
```

Correctness By induction.
Running Time: For some k,

$$T(n) = T(k) + T(n-k-1) + O(1)$$

$$\implies \text{Solution:} T(n) = O(n)$$

1.3 Methods on BST

1.3.1 Recusrive Search:

```
def recusrive_BST_search(x, k):
  if x!= None or x.key ==k:
    return x
  if x.key < k:
    return tree_search(x.right , k)
  else:
    return tree_search(x.left , k)
```

Running Time: $O(h)$, where h is the height of the tree. The more balanced a BST, the better is the search performance.

1.3.2 Iterative Search:

```
def iterative_BST_search(x, k):
  while x!= None or x.key !=k:
    if x.key < k:
      x = x.left
    else:
      x = x.right
  return x
```

Running Time: $O(h)$.
Correctness: follows from BST property

1.3.3 BST Minimum:

```
def BST_min(x):
  while x.left!= None:
    x = x.left
  return x
```

Running Time: $O(h)$.
Correctness: follows from BST property

1.3.4 BST Succesor:

Where would x's succesor be?
1. If x has right then it will be the minimum of the right subtree (Note: in this case, it is not x's parent/ancestor).
2. But what if x.right == NIL?

$$\Longrightarrow \text{ if x == x.parent.left then return x.parent}$$

3. But what if x == x.p.right? then we go up until we are at left child.

```
def BST_successor(x):
  if x.right != None:
    return BST_min(x)
  y = x.parent
```

```
    while y!= None and x== y.right:
        x = y
        y = y.parent
    return y
```

Running Time: $O(h)$.
Correctness: follows from BST property

Theorem 6.1. We can implement the dynamic-seoperations: Search, min, max, successor , predecssor in a BST in O(h) runnning time

1.3.5 Insertion:

Insertion is "easy": Simply use the binary search to find the right location for insertion.

```
# T is BST. T.root
# z is the node to be inserted
def BST_insert(T, z):
    y = None
    x = T.root
    while x!= None:
        y = x
        if z.key < x.key:
            x = x.left
        else:
            x = x.right
    z.parent = y
    if y == None:
        T.root = z
    if z.key < y.key:
        y.left = z
    else:
        y.right = z
    return y
```

Running Time: $O(h)$.
Correctness: follows from BST property

1.3.6 Deletion:

1. if z leaf, then simply delete ✓
2. if z has only 1 child, then ✓

```
def BST_transplant(T, u, v):
    ''' Put v instead of u and connect s parents accordingly.
    '''
    if u.p == None:
```

```
    T.root = v
  elif u== u.parent.left:
    u.parent.left = v
  else:
    u.parent.right = v
  if v!= None:
    v.parent = u.parent

def BST_delete(T,z):
  if z.left == None:
    BST_transplant(T, z, z.right)
  elif z.right == None:
    Transplant(T, z, z.left)
  else:
    y = BST_min(z.right)
    if y.parent != z:
      BST_transplant(T, y, y.right)
      y.right = z.right
      y.right.parent = y
    BST_transplant(T, z, y)
    y.left = z.left
    y.left.parent = y
```

Running Time: $O(h)$.
Correctness: follows from BST property

□