EECS2011: Fundamentals of Data Structures

Assignment 4

Name: Yang Wang

EECS ID: infi999

Student ID: 213894167

December 5th, 2016

Problem 1:

For this problem, we need make a method to find the kth smallest element. There are 2 sorted arrays S and T with 2n distinct positive integers. We need to find kth smallest element in the union of array S and T and the worst case time should be O (log n).

1. When k = 6, the answer is 16. When k =10, the answer is 35.
2. If k = n, we can use Search(S, 0, S.length - 1, T, 0, T.length - 1, n) to calculate the k-th smallest value.
3. If k < n, we can use Search(S, 0, k, T, 0, k, k) to calculate the k-th smallest value.

Algorithm Search(int[]S, int startS, int endS, int[]T, int startT, int endT, int k):
begin
if k is equal to 1
return min{a[startS], b[startT]}
end if
if startS > endS **then**
return T [startT + k - 1]   // S has done, go find smallest element in T
end if
if startT > endT **then**
return S [startS + k - 1]   // T has done, go find smallest element in S
end if
Sl ← endS - startS + 1
Tl ← endT - startT + 1
// if the number of elements in S greater than T, swap S and T, put less one in front
if Sl > Tl **then**
// return Search (T, startT, endT, S, startS, endT, k)
swap (S, T)
swap (startS, startT)
swap (endS, endT)
end if
ms ← 0

ns ← 0

if (endS - startS + 1) < k / 2) **then**

ms ← endS - startS + 1

else

ms ← k / 2

end if

ns ← k - ms

m ← ms - 1 + startS

n ← ns - 1 + startT

// recursively till find the kth smallest element

if S[m] is equal to T[n] **then**

return S[m]

else if S[m] > T[n] **then**

return Search(S, startS, m, T, n + 1, endT, k - ns)

else

return Search(S, m + 1, endS, T, startT, n, k - ms)

end if

end algorithm

Above is my algorithm, I used reclusive to help me solve this problem and cut the k into half. The worst case time is O(log(S) + log(T)) is O (log n). I also attached the java code for problem 1, the output is below.

The java code and output is on next page.

```java
 * EECS2011: Fundamentals of Data Structures, Fall 2016
package A4sol;

public class MedianFinal {
    public static void main(String[] args) {
        // set 2 sorted arrays a and b
        int[] a = new int[] { 3, 5, 9, 15, 27, 33, 35, 41, 57, 65 };
        int[] b = new int[] { 2, 16, 18, 42, 44, 46, 48, 50, 52, 54 };

        // find the kth smallest element
        int k = 10;
        System.out.println("The " + k + "th" + " smallest element is " + Search(a, 0, a.length - 1, b, 0, b.length - 1, k));
    }

    public static int Search(int[] a, int startA, int endA, int[] b,
            int startB, int endB, int k) {
        if (k == 1) {
            if (a[startA] <= b[startB]) {
                return a[startA];
            } else {
                return b[startB];
            }
        }

        // if startA>endA, it means A is done, go find smallest element on B
        if (startA > endA) {
            return b[startB + k - 1];
        }
        // if startB>endB, it means B is done, go find smallest element on A
        if (startB > endB) {
            return a[startA + k - 1];
        }

        int al = endA - startA + 1;
        int bl = endB - startB + 1;


        if (al > bl) {
            return Search(b, startB, endB, a, startA, endA, k);
        }
```

```java
        if ((endA - startA + 1) < k / 2) {
            ms = endA - startA + 1;
        } else {
            ms = k / 2;
        }

        ns = k - ms;

        int m = ms - 1 + startA;
        int n = ns - 1 + startB;

        if (a[m] == b[n]) {
            return a[m];
        } else if (a[m] > b[n]) {
            return Search(a, startA, m, b, n + 1, endB, k - ns);
        } else {
            return Search(a, m + 1, endA, b, startB, n, k - ms);
        }
```

Problems @ Javadoc Declaration Console

```
<terminated> MedianFinal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Cor
The 10th smallest element is 35
```

Problem 2:

For this problem, we need make a method countRange(k1, k2) to find the number of keys of a sorted map fall in the specified range. The worst case time should be O (h).

```
Algorithm Position<Entry<K,V>> countRange(Position<Entry<K,V>> p, K key1, K key2):
begin
Static int totalsize
// Returns the number of values in the range(k1,k2)
if isExternal(p) then
return 0                                    // the final leaf，  return 0
end if
int comp1 ← compare(key1, p.getElement( ))      // compare the key1 with the key of p
int comp2 ← compare(key2, p.getElement( ))      // compare the key2 with the key of p
if comp1 >0 && comp2<0 then               // key1<key(p)<key2
totalsize ++;
countRange(left(p), key1, key2)
countRange(right(p), key1, key2)
else if comp1 < 0                         // key(p)<key1, only the right sub tree is considered
countRange(right(p), key1, key2)
else if comp1 >0                          // key(p)>key2, only the left sub tree is considered
countRange(left(p), key1, key2)
else if comp2 ==0 && isInternal(left(p))
// key(p)=key2, only the left sub tree is considered, 1 is the p itself
totalsize ← totalsize+1+left(p).getSize()
else if comp2 ==0 && isExternal(left(p))
totalsize ← totalsize+1
else if comp1 ==0 && isInternal(right(p))
// key(p)=key1, only the right sub tree is considered, 1 is the p itself
totalsize ← totalsize+1+right (p).getSize()
```

else if comp1 ==0 && isExternal(right (p))

totalsize ← totalsize+1

end if

return totalsize // search right subtree

end algorithm

The insert process is changed by the following two algorithms:

// Returns the position in p's subtree having given key (or else the terminal leaf)

Algorithm Position<Entry<K,V>> treeSearchForInsert(Position<Entry<K,V>> p, K key)

begin

if isExternal(p) **then**

return p // key not found; return the final leaf

end if

int comp ← compare(key, p.getElement( ))

p.setSize(p.getSize() + 1)

// insert a node in the subtree of p, therefore the size of p increase by 1

if comp is equal to 0 **then**

return p // key found; return its position

else if comp < 0

return treeSearchForInsert (left(p), key) // search left subtree

else

return treeSearchForInsert (right(p), key) // search right subtree

end if

end algorithm

Algorithm V put(K key, V value) throws IllegalArgumentException

begin

checkKey(key) // may throw IllegalArgumentException

Entry<K,V> newEntry ← new MapEntry<>(key, value)

Position<Entry<K,V>> p ← treeSearchForInsert(root( ), key)

if isExternal(p) **then** // key is new

expandExternal(p, newEntry)

rebalanceInsert(p) // hook for balanced tree subclasses

return null

else // replacing existing key

V old ← p.getElement( ).getValue( )

set(p, newEntry)

rebalanceAccess(p) // hook for balanced tree subclasses

return old

end if

end algorithm

The delete process is changed by the following two algorithms:

Algorithm V remove(K key) throws IllegalArgumentException

begin

checkKey(key) // may throw IllegalArgumentException

Position<Entry<K,V>> p ← treeSearch(root( ), key)

if isExternal(p) **then** // key not found

rebalanceAccess(p) // hook for balanced tree subclasses

return null

else

V old ← p.getElement( ).getValue( )

If isInternal(left(p)) && isInternal(right(p)) // both children are internal

Position<Entry<K,V>> replacement ← treeMax(left(p))

replacement.setSize(p.getSize()-1) // one node is deleted, so decrease the size by 1

set(p, replacement.getElement( ))

p = replacement

end if

// now p has at most one child that is an internal node

Position<Entry<K,V>> leaf ← (isExternal(left(p)) ? left(p) : right(p))

Position<Entry<K,V>> sib ← sibling(leaf)

remove(leaf)

remove(p) // sib is promoted in p's place

rebalanceDelete(sib) // hook for balanced tree subclasses

return old

end algorithm

The data structure is changed by the following:

protected static class BSTNode<E> extends Node<E>

int size ← 0

BSTNode(E e, Node<E> parent, Node<E> leftChild, Node<E> rightChild)

super(e, parent, leftChild, rightChild)

public int getSize( )

return size

public void setSize(int value)

size = value

//end of nested BSTNode class

end algorithm

As we can see from countRange method as above, no matter the key is only one or more, the worst case time is O (n).

Problem 3:

For this problem, we need to write an algorithm to find who wins the election. For the n element in sequence S. Each element represents a vote which given by an integer represents a candidate. And we also know that the number of k (k<n) of candidates running. The running time should be O (nlogk).

Algorithm:

Set sequence S

Set AVL tree to store candidate ID

// set A for array of key k

A ← array of k integer with value 0

//check sequence S not empty

while S is not empty **do**

// set first element in sequence S to be d

d ← S.first()

// if position p is 0

for (p,0) ← S.remove(d) **do**

A[p] = A[p] + 1

win ← 0

count ← 0

end for

// keep track the count of votes

for (i ← 0; i ← k-1; i++) **do**

// check if count is more than index of array A

if A[i] < count **then**

count = A[i]

win = i

end if

return win

end for

end while

end algorithm

Since this data structure is hold k elements for each search. The total running time is O (nlogk).