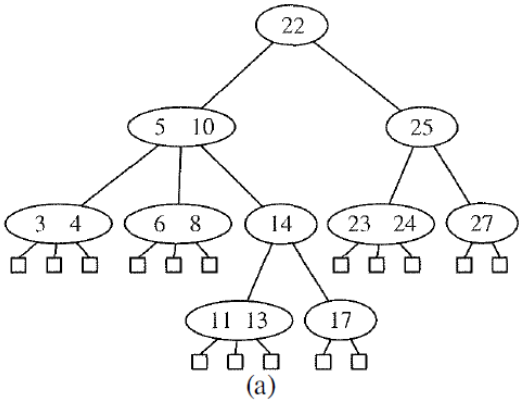
Consider the multi way search tree of Lecture 8, slide 39. Why isn't it a valid (2,4) tree? Justify your answer. What could we do to make it into a valid (2,4) tree? Draw the valid (2,4) tree.

Solution:

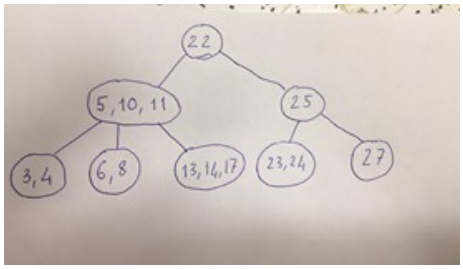
(2,4) Tree has two properties:

- Every node has at most four children.
- All the external nodes have the same depth.



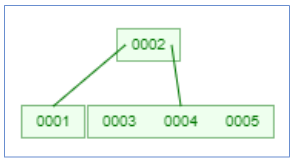
(a)

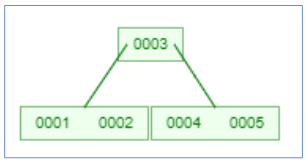
Change to:



R-3.10 A certain Professor Amongus claims that a (2,4) tree storing a set of items will always have the same structure, regardless of the order in which the items are inserted. Show that Professor Amongus is wrong

Solution:

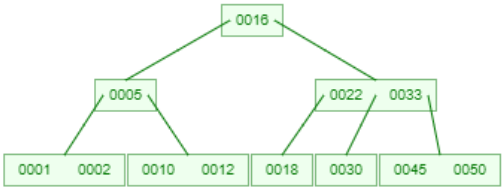




Two trees create from the same set, but they are the different.

Insert the following sequence of keys into an initially empty 2-4 tree in this order: (16, 5, 22, 45, 2, 10, 18, 30, 50, 12, 1, 33)

Solution



C-4.11 Suppose we are given an n-element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Suppose we know who the candidates are and the number of candidates running is k < n. Describe an O(n log k)-time algorithm for determining who wins the election

Solution

Algorithm findWinner(S)	**Running time**
T ← new Dictionary(AVL)	O(1)
count := 0	O(1)
winnerID := -1	O(1)
for i=0 to S.size – 1 do	O(n)
e ← S.elementAtRank(i)	O(1)
p := T.findElement(e)	O(logk)
if element <> NO_SUCH_KEY	O(1)
value = p.value() + 1	O(1)
T.insertElement(p.key(), value)	O(logk)
if count < value	O(1)
count = value;	O(1)
winnerID = p.key()	O(1)
else	
T.insertElement(e, 1)	O(1)
return winnerID	O(1)

➔ **T(n) is O(n log k)**

C-4-22 Let A and B be two sequences of n integers each. Given an integer x, describe an O(n log n)-time pseudo code algorithm for determining if there is an integer a in A and an integer b in B such that x = a + b.

Solution

Algorithm findWinner(A, B, x)	**Running time**
T ← new Dictionary(AVL)	O(1)
for i=0 to A.size – 1 do	O(n)
e ← A.elementAtRank(i)	O(n)
key = x – e	O(n)

```
        p := T.findElement(key)                          O(n*log k)
        if element == NO_SUCH_KEY                        O(n)
                T.insertElement(key, 1)                  O(n*log k)
for i=0 to B.size – 1 do                                 O(n)
        e ← B.elementAtRank(i)                           O(n)
        p = T.findElement(e)                             O(n*log k)
        if p <> NO_SUCH_KEY                              O(n)
                return true                              O(n)
return false
```

➔ **T(n) is O(n log n)**