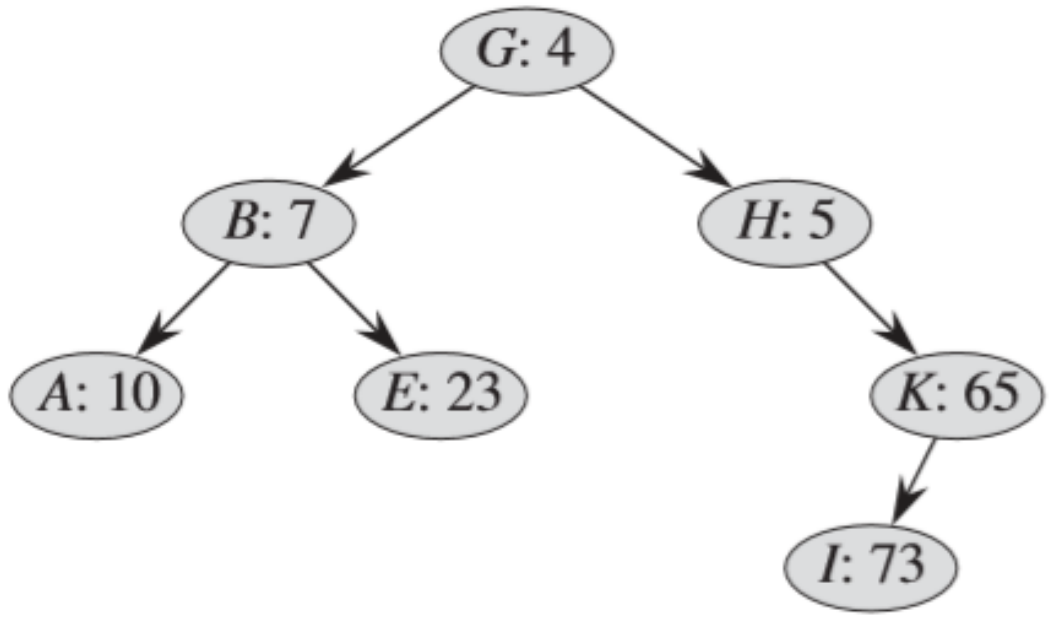
Exercise 1

If we insert a set of n items into a binary search tree, the resulting tree may be horribly unbalanced, leading to long search times. However, randomly built binary search trees tend to be balanced. Therefore, one strategy that, on average, builds a balanced tree for a fixed set of items would be to randomly permute the items and then insert them in that order into the tree. What if we do not have all the items at once? If we receive the items one at a time, can we still randomly build a binary search tree out of them?

Treap

We will examine a data structure that answers this question in the affirmative. A treap is a binary search tree with a modified way of ordering the nodes. As usual, each node x in the tree has a key value x.key. In addition, we assign x.priority, which is a random number chosen independently for each node. We assume that all priorities are distinct and also that all keys are distinct. The nodes of the treap are ordered so that the keys obey the binary-search tree property and the priorities obey the min-heap order property:



A treap. Each node x is labeled with x.key: x.priority. For example, the root has key G and priority 4.

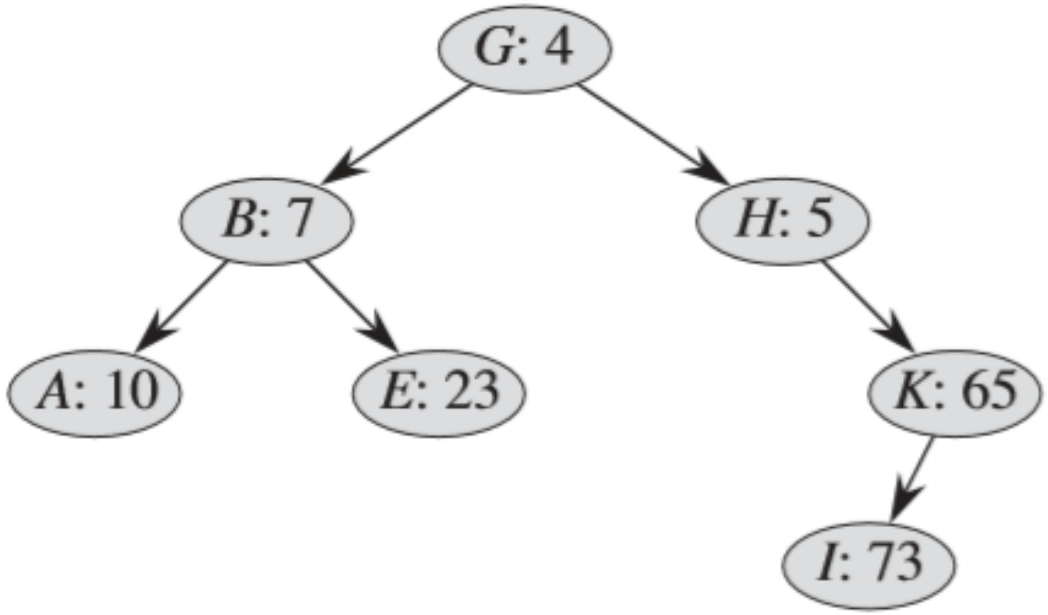
If v is a left child of u, then v.key < u.key.

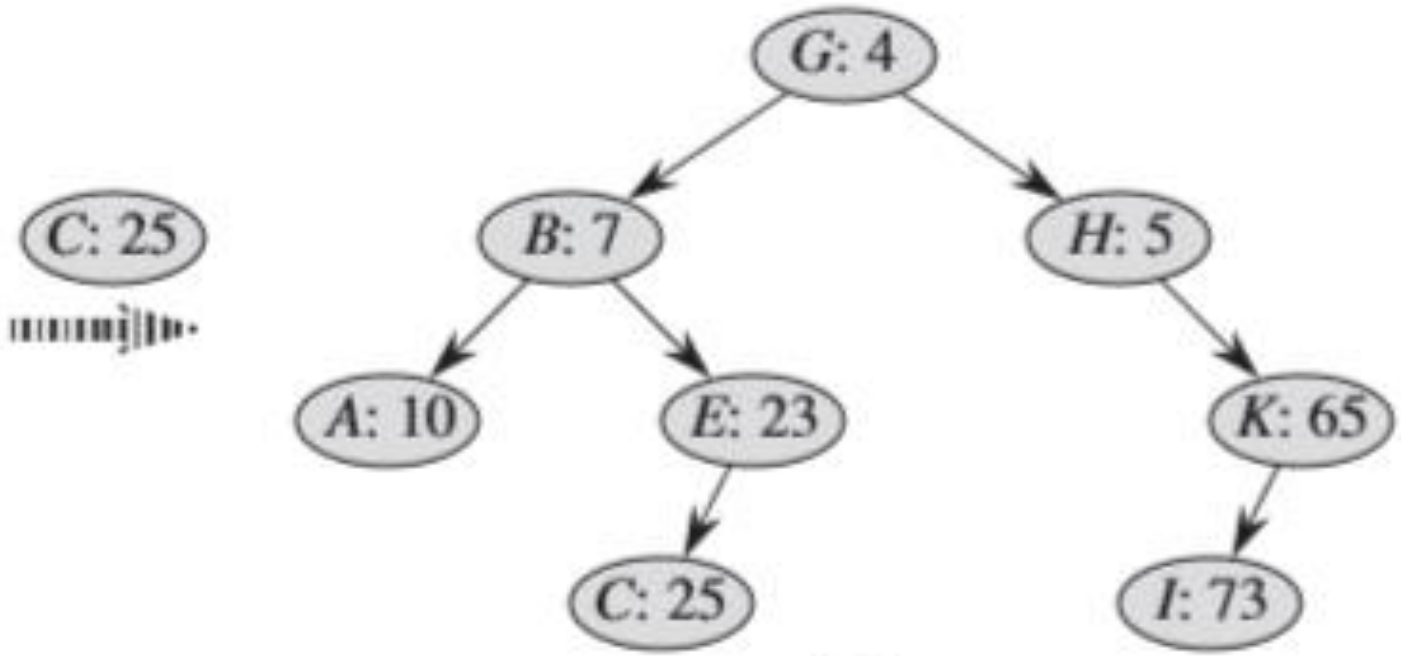
If v is a right child of u, then v.key > u.key.

If v is a child of u, then v.priority > u.priority.

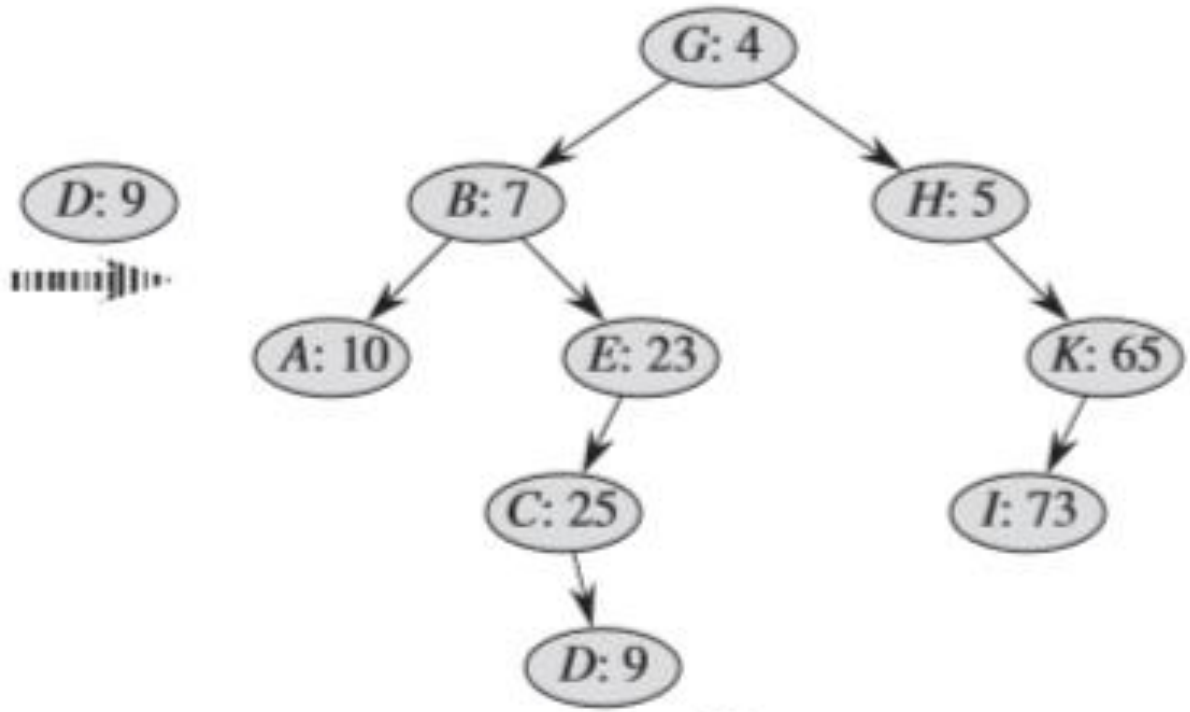
(This combination of properties is why the tree is called a “treap”: it has features of both a binary search tree and a heap.)

a. how to insert a new node into an existing treap?

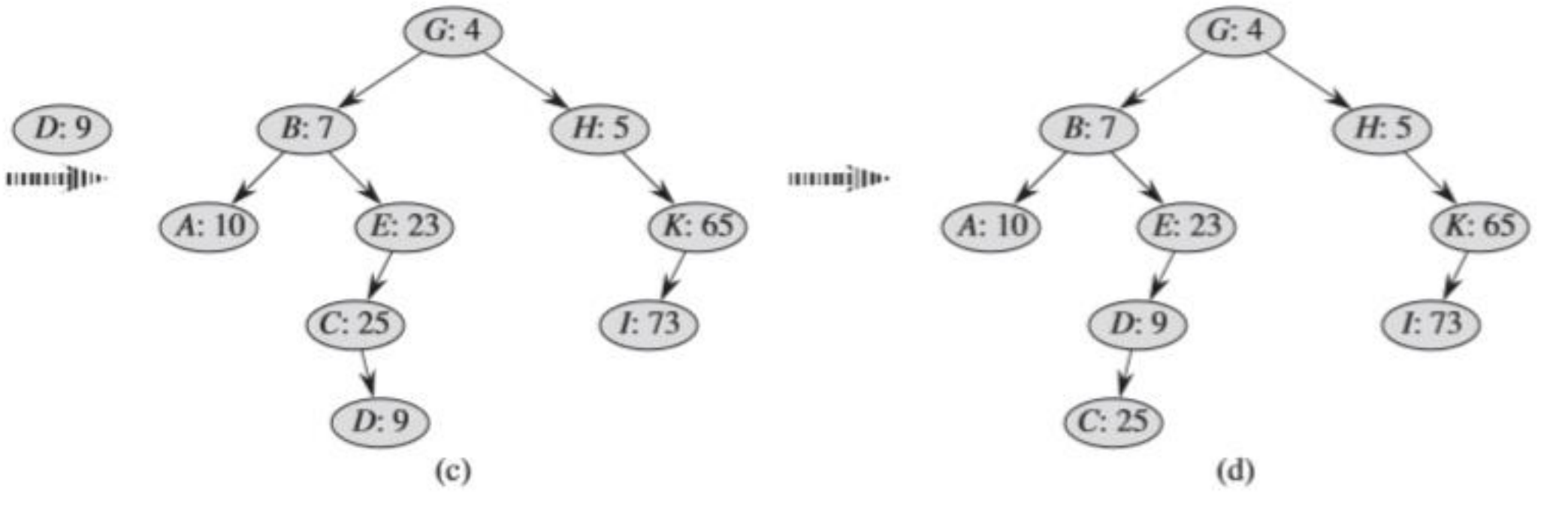


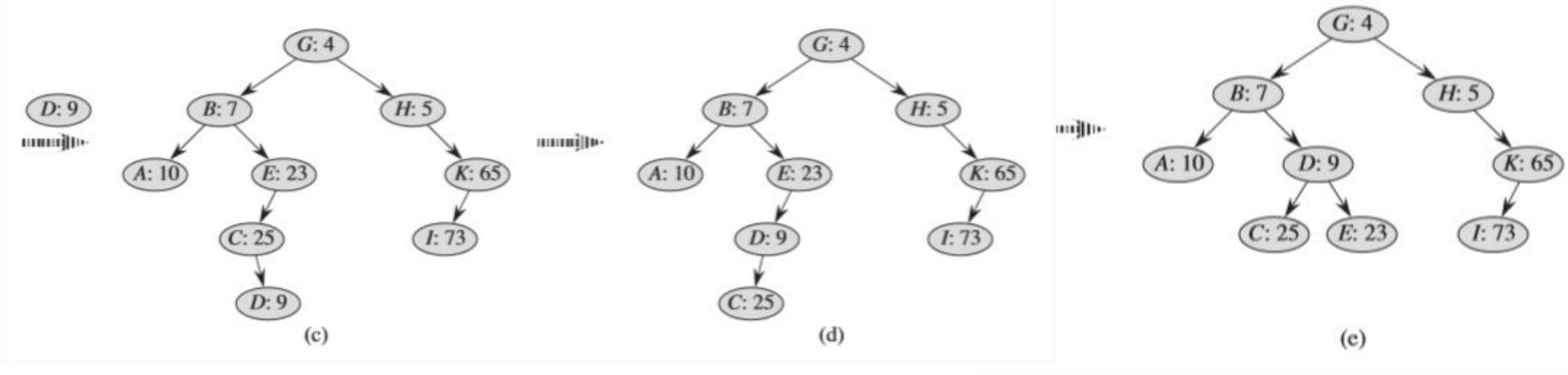


(b)



(c)


D: 9
G: 4
B: 7
H: 5
A: 10
E: 23
K: 65
C: 25
I: 73
D: 9
(c)
G: 4
B: 7
H: 5
A: 10
E: 23
K: 65
D: 9
I: 73
C: 25
(d)

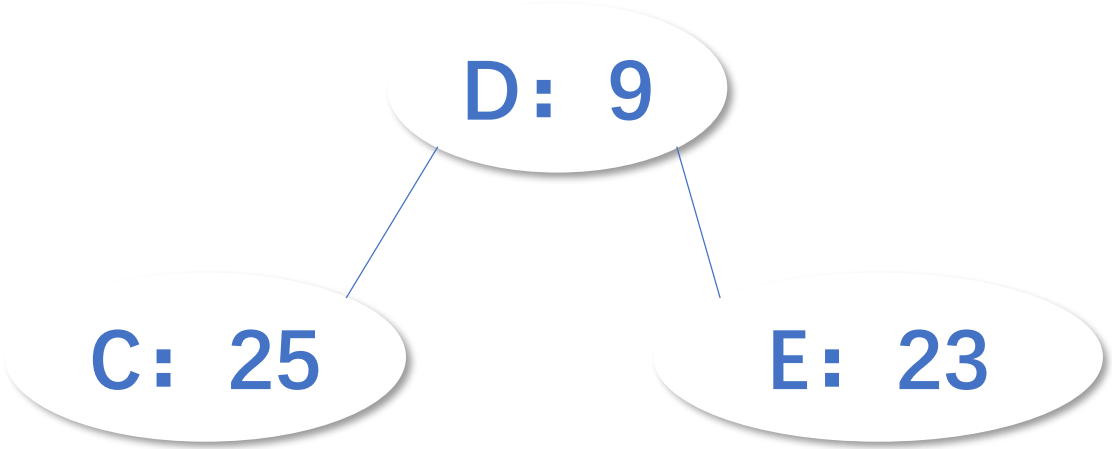

D: 9
G: 4
B: 7
H: 5
A: 10
E: 23
K: 65
C: 25
I: 73
D: 9
(c)
G: 4
B: 7
H: 5
A: 10
E: 23
K: 65
D: 9
I: 73
C: 25
(d)
G: 4
B: 7
H: 5
A: 10
D: 9
K: 65
C: 25
E: 23
I: 73
(e)

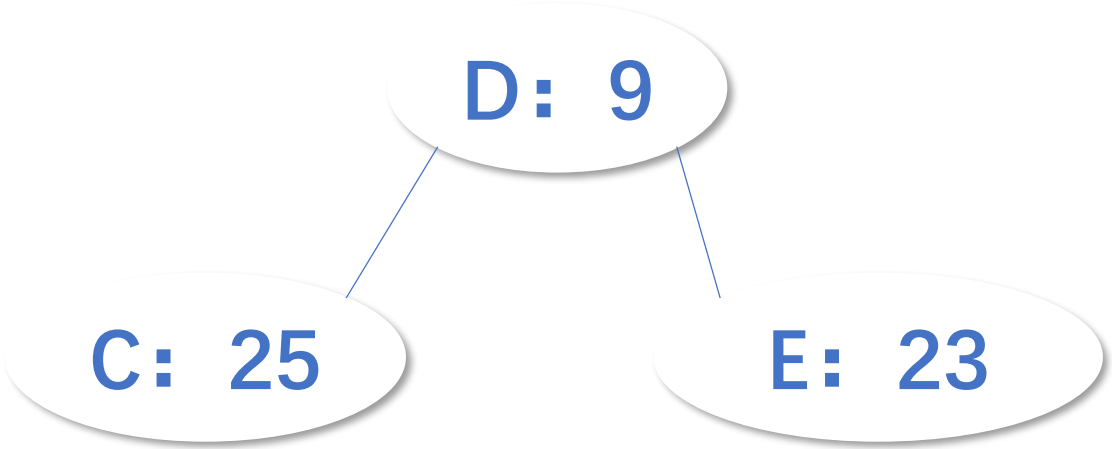
b. Show that given a set of nodes $x_1, x_2, \ldots, x_n$, with associated keys and priorities, all distinct, the treap associated with these nodes is unique.

D:9,E:23,C:25

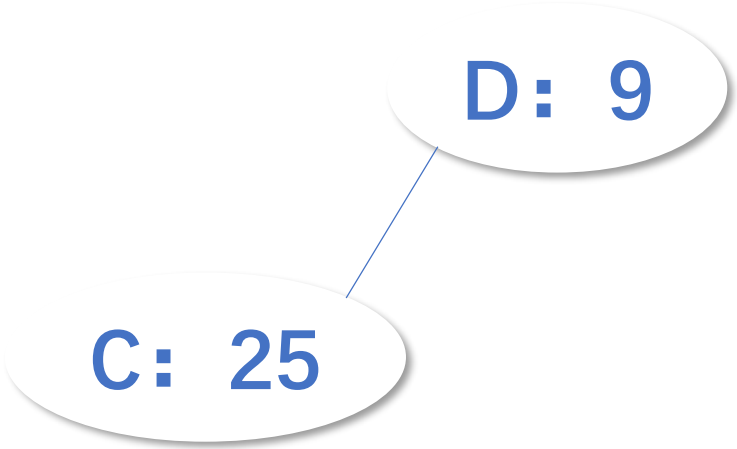
D: 9

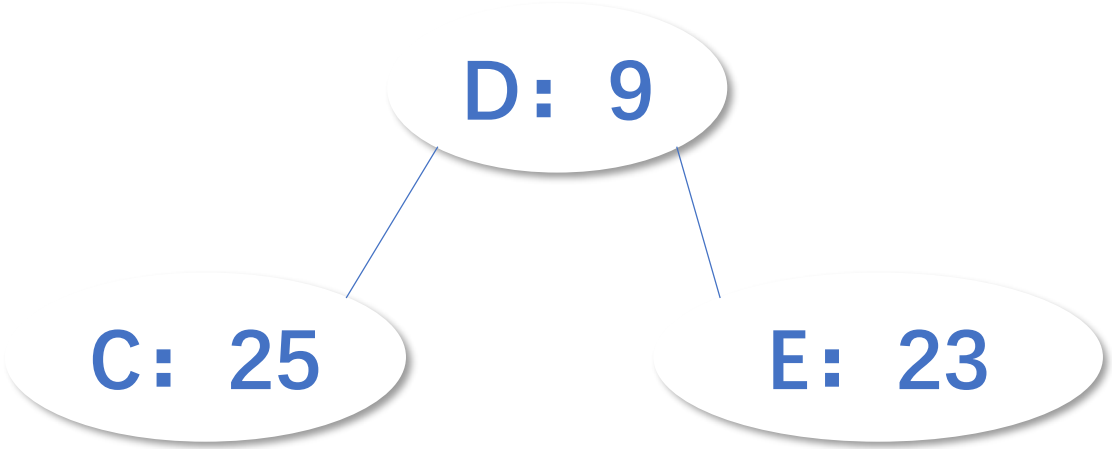

D：9
E：23


D：9
C：25
E：23


D：9
C：25
E：23

C: 25


D：9
C：25


D：9
C：25
E：23

The root r is uniquely determined because it must contain the smallest priority. Then we partition the set of nodes into those which have key values less than r and those which have values greater than r. We must make a treap out of each of these and make them the left and right children of r. By induction on the number of nodes, we see that the treap is uniquely determined.