Report

* Implemention of fuctions

1. Lookup:

In order to implement look up key, I mainly use BST property. Note boundary conditions if node is null, return value 0. Then go as BST property :if keys eauql, return value. If key to find greater then node’s key, search right subtree,else search left subtree.

1. Insert:

Insert is implemented as PDF says, first insert a node obey BST property, and then use rotate\_up to make the tree obey HEAP property. When rotating up, I implement left rotation and right rotation together, and end condition with return type bool. So that when inserting, only use rotate\_up function,it will keep rotate untill satisfy HEAP property.

1. Remove:

Remove is implemented as PDF says, I rotate\_down the node to the leaf and then remove it. Rotate\_down is similar to raotate\_up , exchange father node and son node, exchange the continue condition to rotate to leaf. Then like insert judeg differrnt situation, and rotate\_down to leaf finally free.

Because of conflic with C’s stdio.h so I change the name to \_remove.

1. InitTreap:

I firstly build a new tree,which is null and the insert a node with key and value. Then we get a tree with the node.

1. Split:

Insert a node with key and max prioprity then its left subtree and right subtree is treaps by split.

1. Join:

Initial the treap with the node of middle key between left max key and right min key. Make its prioprity to max\_prioprity. Then connect it and two treaps because of prioprity and key. Only need to set left son and right son and two old roots’ new father. Finally remove the initial node. And I handle the case if left tree’s max key equals right tree’s min key to avoid repeat.

1. BalanceFactor:

We compute the hight and node numbers of the tree by recursion. Then compute balbaceFactor deirectly.

1. Meld:

To implement meld in time require, I use recursive function. Base case is T1 or T2 is null treap. We split T2 with T1’s root in order to obey join’s requirement. And recurce in left and right finally join left ,key-value,right together.

1. Difference:

Difference is same with meld in time requirement, so I take same thought that base case is T1 is null and T2 is null. Different between meld and difference is if we find same key in T1 and T2, difference don’t need it so we check it and use join to avoid key-value pair.

* Difficult parts of code

I think the difficult parts of the code is boundary conditions. As for insert I add a situation of keys equal. It uses for split function. We can’t guarantee our max key of T1 didn’t equal min key of T2, so if keys equal then use priority to rank. Also node is null or recursive function’s base case is important. Meld and difference ask for expected time , it can be satisfied by recursive inequality , so if find base case it’s easy to code. In test I find that if data range is small, the balance of tree maybe break, expecially meld and difference. Because of same key in join influent meld and split, so it’s important to think of this, I waste many time in this. I think I can do better in keep balance of tree. And when rotate, I meet problem of end condition, finally I choose judge in rotate\_up function and directly call it in insert. Well, the parameter of function is kind of complex to recursion, so I write some subfunction to help. And because of key can’t be same so I ignore many boundary conditions in my implemention, I think I should do better.