Project 6 Testing Report

Zhaosong Zhu | CS 314H

In the Treap project, I am offered the opportunity to conduct peer testing with four other groups from the same class. Personally, I found the peer testing process helpful, regarding developing a robust testing harness and think about more aspects. Below is a discussion of the testing process.

- Summary of overall Peer Review process
  My peer review process started with writing test cases based on the interface. I spent some time thinking about all "traps" that I fell in when writing out the solution and some possible mistakes that other people tend to make. HoIver, merely brainstorming based on what I had Ire not enough – I began to read through all posts on Piazza to make my test as comprehensive as possible. As I received the jar files from others and observed the exceptions thrown, I was able to update our testing harness and project itself.

  For the one thing, I developed some new insights when testing other people's code. For example, when testing with group 13's jar file on join method, I realized that a.join(b) and b.join(a) with either one of them being empty are essentially the same cases which can be dealt with at the beginning. This reminds me to add lines in my join method to test whether one of the traps is empty and make necessary node link to migrate everything.

  For the other thing, from the peer reviews I received, I was able to fix some bugs too. For example, based on team 25's review, I kept the value deleted during the split process but inserted into the wrong subtree because I used the old, rather than the new root to decide which side to plug in. Many thanks to their suggestion, I Ire able to find this bug and check my logic again.

  Besides, the list presented is a list of questions I used to test others:

  LookUp
  1. Null parameter
  2. Find Item - empty treap
  3. Find Item - 1st add in
  4. Find Item - Multiple Add in

  Insert
  1. Null K/V or K+V
  2. If null alloId, can I find in lookup?
  3. BST Property

  Remove
  1. Null K - give error or maintain the same treap
  2. Return right V?
  3. Can I still find the item?

4. BST property? - rotation
5. Behavior on empty treap

toString

1. Empty Treap
2. Correct format? with tab
3. Does item matches with an iterator?

Iterator

1. Empty Treap
2. K matches with add-ins
3. # of Items
4. Check per-order property?/ BST

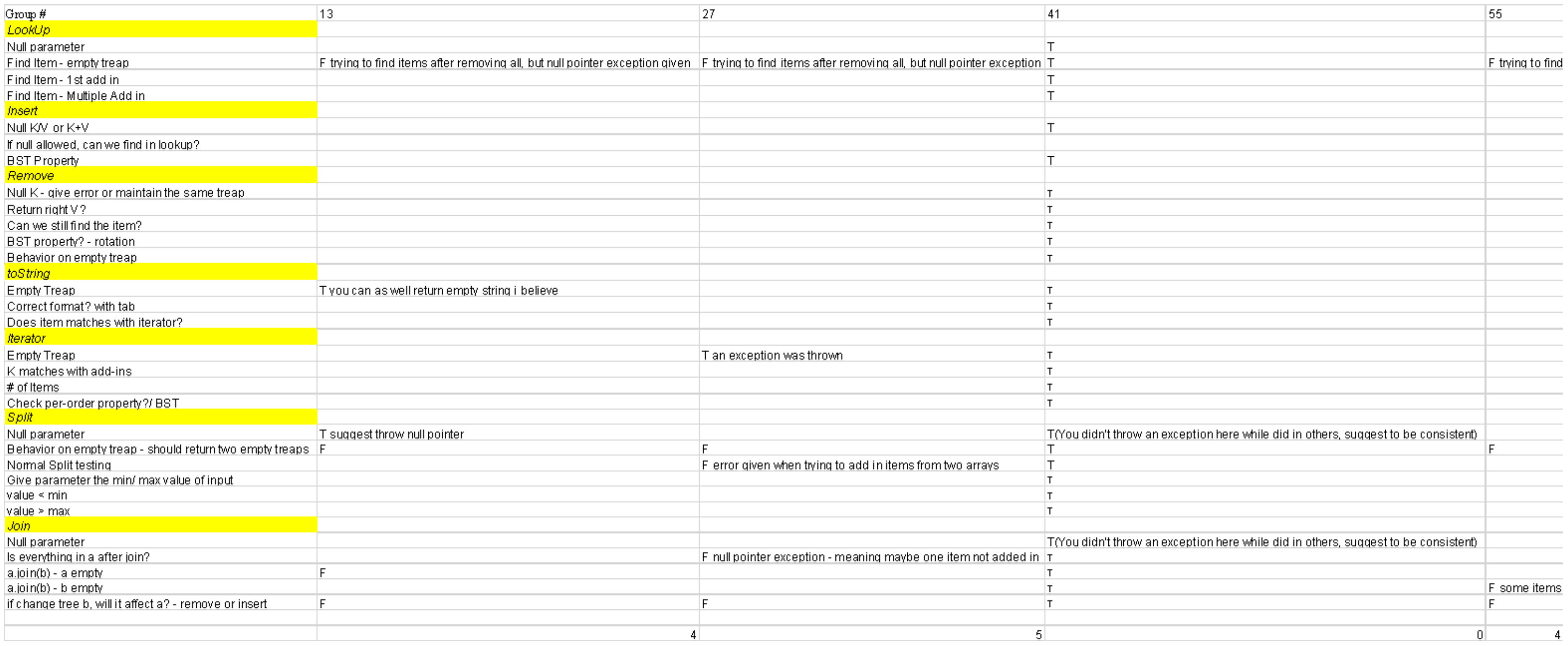
Split

1. Null parameter
2. Behavior on empty treap - should return two empty treaps
3. Normal Split testing
4. Give parameter the min/ max value of input
5. value < min
6. value > max

Join

1. Null parameter
2. Is everything in a after join?
3. a.join(b) - a empty
4. a.join(b) - b empty
5. if change tree b, will it affect a? - remove or insert

- Evaluation of Peer Review quality
  This is my ranking of peer reviews, from best to worst: 41-55-13-27. This ranking is based on the number of bugs I discovered for each team and how much I learned from them.



Group #	13	27	41	55
LookUp				
Null parameter			T	
Find Item - empty treap	F trying to find items after removing all, but null pointer exception given	F trying to find items after removing all, but null pointer exception	T	F trying to find
Find Item - 1st add in			T	
Find Item - Multiple Add in			T	
Insert				
Null K/V or K+V			T	
If null allowed, can we find in lookup?				
BST Property			T	
Remove				
Null K - give error or maintain the same treap			T	
Return right V ?			T	
Can we still find the item?			T	
BST property? - rotation			T	
Behavior on empty treap			T	
toString				
Empty Treap	T you can as well return empty string i believe		T	
Correct format? with tab			T	
Does item matches with iterator?			T	
Iterator				
Empty Treap		T an exception was thrown	T	
K matches with add-ins			T	
# of Items			T	
Check per-order property?/ BST			T	
Split				
Null parameter	T suggest throw null pointer		T(You didn't throw an exception here while did in others, suggest to be consistent)	
Behavior on empty treap - should return two empty treaps	F	F	T	F
Normal Split testing		F error given when trying to add in items from two arrays	T	
Give parameter the min/ max value of input			T	
value < min			T	
value > max			T	
Join				
Null parameter			T(You didn't throw an exception here while did in others, suggest to be consistent)	
Is everything in a after join?		F null pointer exception - meaning maybe one item not added in	T	
a.join(b) - a empty	F		T	
a.join(b) - b empty			T	F some items
if change tree b, will it affect a? - remove or insert	F	F	T	F
	4	5	0	4

Based on the items I came up above, my goal is clearly to see both by smaller tests and by a big integration test if each method functions correctly. To account for the result, I used a spreadsheet, which records whether a team past a certain item in the test, as shown below:

In the spreadsheet, I also wrote down things I noticed as tests are performed on each team's project. For example, when testing search tactics, team 27 fails when I am trying to retrieve all values from a joined treap. I noted this down on the spreadsheet so that I can give back more personalized feedbacks. This bug I found also helped us think more about edge cases that could happen during the split and join process. I Ire able to realize this when testing other people's codes.

I believe the testing methodology is decently robust since it accounts for all methods available in the interface. However, it is not perfect yet. For example, during peer testing, I assumed that if the iterator works correctly, then the entire Treap has BST property. Although to some degree this assumption is true, it's not a good assumption to ensure a comprehensive test. I then included this in our testing, while not able to address this in all four peer testing feedbacks.

- Description of revised solution
  I made some changes on our solution based on the comments presented below:
  1. "`should be [Priority] <key, value>; you have [priority "Key:" key]`"
     This comment points me to a simple mistake I made: I was using toString method for black box testing. However, I forgot to change it to the correct output format. With this comment, I then change my toString output to the format above with correct tabs for hierarchy.

2. “`Split test (include): Passes sometimes. When it fails, it fails on the splitting key`”
   This feedback helps me realize that I was using the old node to see where the “deleted” value should be plugged in to. I kept the value deleted during the split process but inserted into the wrong subtree because I used the old, rather than the new root to decide which side to plug in. Many thanks to their suggestion, I was able to find this bug and check my logic again.

3. “`when T1.join(T2) and either T1 or T2 is empty: NullPointerException`”
   This feedback helped me realize another edge case: one or both of the treaps can be empty. Thanks to this comment, I added two if statements to account for this case: if T1 is an empty Treap, then we let the root of T2 be the root of T1; if T2 is an empty treap, then we are done.