

COMP3600/COMP6466 in 2007 – Assignment Two

Due: 5pm Wednesday, October 17
Late Penalty: 25% per day

This is a programming assignment. Submit your work in two parts:

- `splay.c` – a C program to be submitted electronically
- A typeset 2-page report to be handed in manually

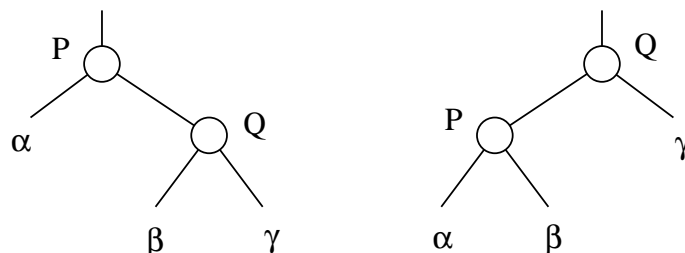
Make sure that both items have your name and student number on them.

Splay trees

Splay trees are binary search trees with more complex algorithms for insertion, lookup and deletion. These algorithms have the effect of keeping the structure of the tree reasonably balanced most of the time, leading to $O(\log n)$ time per operation in total over a long sequence of operations.

The structure of each node is the same as for ordinary binary search trees, though it is convenient to also include a pointer to the parent (**nil** for the root).

The key operation is called *splaying*. This takes a given node and uses a sequence of rotations to bring that node to the root of the tree. Rotations are the same as for red-black trees:

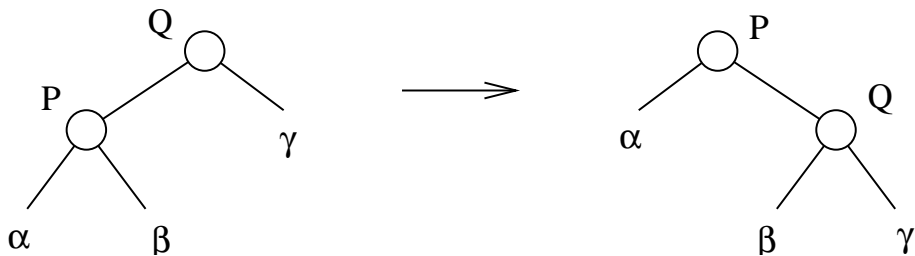


From left to right is a **left rotation at P**
From right to left is a **right rotation at Q**

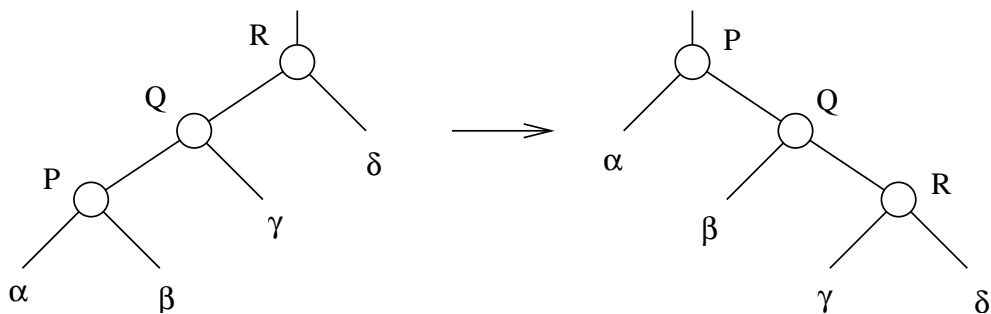
Given a node P, splaying P involves a sequence of rotations that depend on whether P is a left or right child, and whether its parent is a left or right child.

Splaying node P:

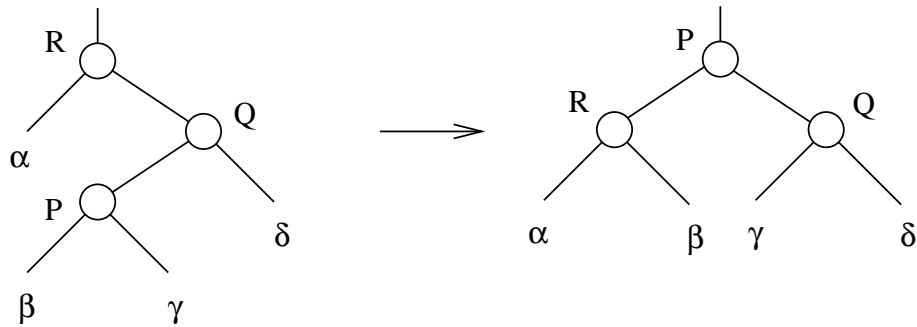
while P is not the root **do**
Case 1a: P is a left child of the root
 Do a right rotation at the parent of P
Case 1b: P is a right child of the root
 Do a left rotation at the parent of P
Case 2a: P is a left child of a left child
 Do a right rotation at the grandparent of P,
 then a right rotation at the parent of P
Case 2b: P is a right child of a right child
 Do a left rotation at the grandparent of P,
 then a left rotation at the parent of P
Case 3a: P is a left child of a right child
 Do a right rotation at the parent of P,
 then a left rotation at the new parent of P
Case 3b: P is a right child of a left child
 Do a left rotation at the parent of P,
 then a right rotation at the new parent of P
endwhile



Case 1a of splaying. Case 1b is the mirror-image.



Case 2a of splaying. Case 2b is the mirror-image.



Case 3a of splaying. Case 3b is the mirror-image.

The concept of splaying a *node* is applied via an operation of splaying a *key*:

SPLAYKEY(T : tree, K : key) :

if T is empty **then**

 Do nothing

else if K is present in T **then**

 Splay the node that contains K

else

 Splay a node that contains a key that either immediately precedes
 or immediately follows K in key order

endif

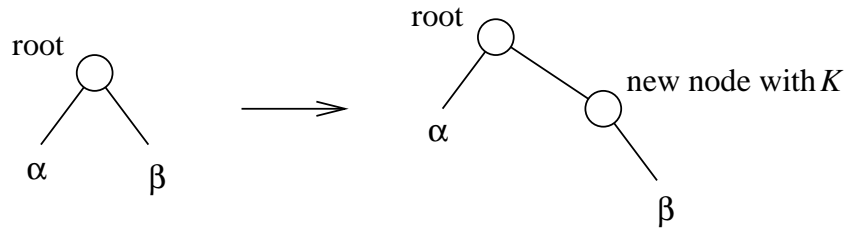
To implement the SPLAYKEY operation, just search for the key K in the usual fashion. If it is found, splay the node it is found in. If it is not found, splay the last node that is examined in the search.

Operations

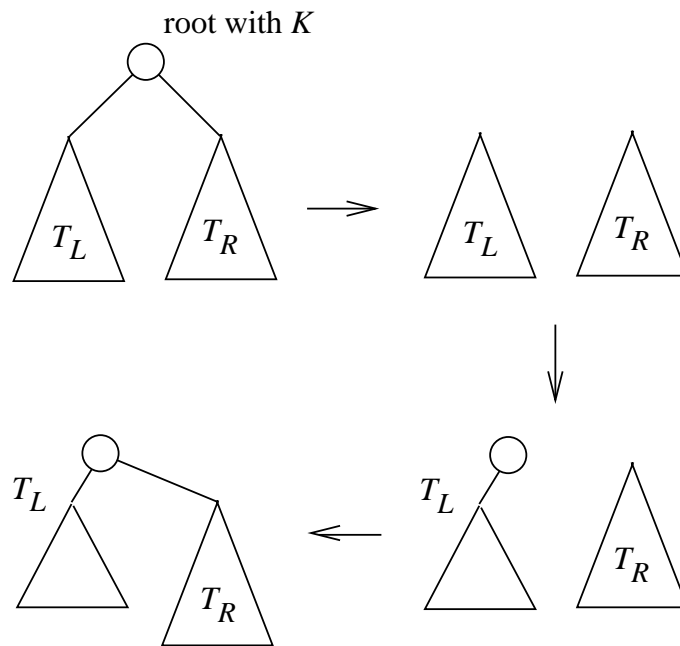
The standard search tree operations are implemented as follows.

- To **search** for key K , just splay K then check whether K appears in the root of the tree.
- To **insert** key K , first splay K . If the root contains K , then K is already present. If the key in the root is less than K , insert K in a new right child of the root (see figure). If the key in the root is greater than K , insert K in a new left child of the root (as the mirror image of the figure).
- To **delete** key K , first splay K . If the root does not contain K , then K is not present. Otherwise, remove the root to make two trees T_L and T_R from the left and right subtrees. Splay the key ∞ in T_L – this ensures that T_L has a root with

no right child. Now make the root of T_R into the right child of the root of T_L . See the figure. (The case where T_L is empty is special: just make T_R the new tree.)



One possibility for the final step of insertion.



Steps during deletion: after splaying K , after removing the root, after splaying ∞ in T_L , after attaching T_R .

Your task

1. Fetch the program <http://cs.anu.edu.au/student/comp3600/bstree.c> and familiarise yourself with its operation. You should also read it carefully until you understand every part of it.
2. Make a copy of `bstree.c` called `splay.c` and modify to use a splay tree. Note that you should start by adding a parent field to the node structure. Apart from changing the data structures, the behaviour of the program should be unchanged.
3. Experiment with different parameters in order to investigate whether splay trees are better than ordinary binary search trees. Present your findings in a report

to be handed in manually. The report should be typeset (not handwritten) in 12-point font and occupy at most 2 pages of normal size.

Notes:

4. Your program `splay.c` must take exactly the same parameter list as `bstree.c` has now. We plan to run your program using a script and expect no surprises.
5. Do not use recursion for any of your procedures (but you don't need to rewrite the existing recursive procedures `freetree()`, `printtree()` and `checknode()`). Don't add any arrays or other data structures except ordinary variables. To move upwards in the tree, use the parent field. Don't forget to maintain the parent fields in all of the operations you perform on the tree.
6. The cost of operations as implemented using the variable `counter` should include the cost of splaying.
7. The file `splay.c` includes procedures `printtree()` and `checktree()` that are not actually required. They are present so you can use them for debugging purposes. Uncomment the parts of `checktree()` and `checknode()` that check the parent field after you implement that field.
8. The call to `srandom()` near the start of the main program will cause the program to test different keys every time you run the program. If this inconsistency causes difficulty with debugging, temporarily comment out that call. However, don't forget to uncomment it again before you run tests and before you submit your program.
9. The key value ∞ used in the deletion algorithm can be simulated using the pre-defined constant `INT_MAX`.

Extra tasks for comp6466/PhD/hons students

Comp6466/PhD/hons students should complete the above tasks 1–3 and also the following. The report can extend up to 6 pages, with no more than 2 pages devoted to tasks 1–3.

10. The procedures `freetree()` and `printtree()` currently use recursion. After implementing parent fields, replace these procedures by non-recursive procedures. Don't use extra storage such as a stack but just use the left, right and parent fields to move around. In your report, prove that your methods use $O(n)$ time if the tree has n nodes.

11. A possible modification of the splay tree algorithm is to use the standard search procedure for the search operation, without any splaying. Does this have a significant effect on the efficiency?
12. Submit a program `splat.c` that implements tasks 2 and 10. Don't submit the code you used to answer question 11.

Approximate division of marks

- The program (correctness and implementation) is worth 70% and the report is worth 30%. There are no marks for the formatting of the report, but you might lose marks for unusually bad formatting or presentation, or for exceeding the page limit.
- For comp6466/PhD/hons students, the extra tasks will be worth an additional 30%, making 130% altogether.