Algorithms and Data Structures (WS15/16)

Exercises for Units 14 and 15

Problem 1

Try to get some feel for the behavior of the splay tree data structure. Draw a binary search tree with 5-10 nodes, and work out a few splay operations on paper. You can also try some interactive demonstration of splay tree on the internet. Try to force splay to make costly operations. What happens?

Problem 2

Show that in a splay operation almost every node on the search path "roughly" halves its depth. What about the nodes outside the search path?

Problem 3

Data compression. Suppose we are processing a long stream of temperature measurements (for simplicity, integers between $-N$ and N). We would like to encode the sequence of values in binary (using 0s and 1s and no other symbols), such that the resulting binary string is as short as possible. We would also like the result to be *decodable*, meaning that we can recover the original sequence of integers from the binary string.

Here is one way to do it: Build a binary search tree with the integers $\{-N, \ldots, N\}$ at the leaves. To encode an arbitrary entry t in the stream, search from the root to t, and whenever you take an edge to the left, output 0, whenever you take an edge to the right, output 1.

Why is the resulting string decodable? (Assuming you have access to the same tree that was used for encoding). How would you choose a binary search tree for good compression?

Can you somehow use splay tree in the above scenario? What advantages would it give? Are there any disadvantages?

Problem 4

A *deque sequence* is an arbitrarily long sequence of insert, delete, and access operations, such that every delete or access refers either to the minimum or to the maximum of the currently stored keys, and every insert is for a key either smaller or larger than all currently stored keys.

Design a data structure based on a binary search tree (using no extra pointers or annotations) that (starting from an empty dictionary) serves every deque sequence with constant amortized cost per operation.

Problem 5

Consider a "simpler version" of splay, called move-to-root. In move-to-root, instead of the zig-zig and zig-zag operations, we simply rotate the accessed element up using normal rotations until it becomes the root.

Show that move-to-root can be very inefficient: construct an initial tree and an arbitrarily long search sequence that has a high cost per search if move-to-root is used instead of splay. Is the choice of the initial tree essential in your example?