15-451/651 Algorithm Design & Analysis
Fall 2022, Recitation #2

Objectives

- Practice deriving data structures with good amortized bounds
- Practice using potential functions to determine amortized bounds
- Understand splay trees and how to implement standard operations with them
- Understand the main tool used to analyze Splay Trees: the Access Lemma

Recitation Problems

1. **(Set with choice)** Your favorite programming language has an API for maintaining a stack that supports (among others) the following operations:

   - top(): Returns the top element of the stack. The stack must be non-empty.
   - pop(): Removes the top element of the stack. The stack must be non-empty.
   - push(x): Pushes x onto the top of the stack.

   It also has the following API for a set:

   - member(x): Returns `true` if x is in the set, `false` otherwise.
   - insert(x): Inserts x into the set (does nothing if x is already there).
   - delete(x): Deletes x from the set (does nothing if x is not there).

   For the purposes of this problem, all of the above API functions on the set and the stack take 1 unit of time.

   As you're writing a program, you realize that you also need the following operation on non-empty sets:

   - getone(): This returns some element of the set (which must be non-empty). It doesn't matter which element is returned – just something in the set.

   You cleverly realize that by using a set and a stack, and replacing insert(x) with your own version called myInsert(x), then you can then implement getone(). (member() and delete(x) will stay the same.) In your implementation all of the operations will have constant amortized cost. Here the cost measure is the number of calls to the set and stack API operations listed above.

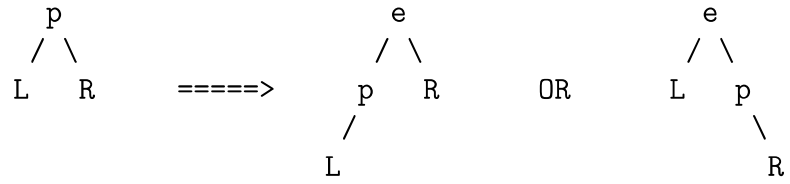
   (a) Assume that the stack and the set are initially empty. And assume that the program calling this API never calls getone() on an empty set. Write pseudo-code implementations of the operations.

(b) Prove that the operations take amortized constant time, and state specifically what these constants are.

2. **(Inserting into a splay tree)** Consider inserting a new element into a splay tree

   (a) What is not ideal about using the ordinary BST (without self-balancing) insertion algorithm?

   (b) Consider the following alternative algorithm for inserting a new element e: Start by searching for the parent that e would have if it were inserted using the ordinary BST insertion algorithm, call it p, then splay p to the root. The tree now looks like (L, p, R), where p is the root node that was just splayed, L is the left subtree, and R is the right subtree. If $e < p$, then join the trees (L, e) and (p, R), otherwise join the trees (L, p) and (e, R), as per the following diagram.



   Prove that the amortized cost of this insertion algorithm is $O(\log n)$.

   (c) Give an $O(\log n)$ amortized algorithm for deleting a node from a splay tree

3. **(Splaying red and blue)** Consider a splay tree with $n + m$ nodes where n are red and m are blue. Choose weights and use the Access Lemma to prove the following

   (a) Amortized number of splay steps done when a red node is accessed is $4 + 3 \log n$

   (b) Amortized number of splay steps done when a blue node is accessed is $4 + 3 \log m$

   Extra thing to think about: if there are very few red items, then we can give a very tight amortized bound on accessing them, even though the algorithm has no idea what is and what isn't a red node. Can you explain why this is true?

Further review

1. **(Short answer / multiple choice)**

   (i) In the game of Fizzbin there are two operations: fizz and bin. Fizz involves putting a card on a stack, and costs 1. Bin involves taking all cards off the stack and has cost equal to the size of the stack plus 1, which is the same as the number of fizz operations since the most recent bin, plus 1.

   For example, the sequence of operations: fizz, bin, fizz, fizz, bin, bin, fizz would cost $1, 2, 1, 1, 3, 1, 1$. Which of the following is the tightest correct upper bound on the amortized cost per operation in a worst-case sequence of n operations?

   (a) 1
   (b) 2
   (c) 3
   (d) 4
   (e) $\log n$
   (f) n

   (ii) Consider a sequence of operations, where the i^{th} operation costs $\lceil \lg i \rceil$. Mark the best upper bound in terms of n on the amortized cost per operation of a sequence of n operations from the choices below:

   (a) $O(1)$
   (b) $O(\lg \lg n)$
   (c) $O(\lg n)$
   (d) $O((\lg n)^2)$
   (e) $O(n)$

   (iii) In lecture we considered the amortized cost of array resizing, where the cost to resize an array from size S to size S' was S. (We assume we start from an empty array.) We considered the strategy where $S' = 2S$ and showed that the amortized cost of a sequence of n appends was at most 3. Now suppose the cost of resize becomes kS for some $k > 0$, all other costs remain the same as in lecture. Choose the best upper bound on the amortized cost per operation for a sequence of n appends from the options below.

   (a) 3
   (b) $k + 2$
   (c) $2k + 1$
   (d) $3k$
   (e) $3n$

   (iv) Now, consider the strategy where you triple the size of the array once it is full, i.e., $S' = 3S$. Choose the best upper bound on the amortized cost per operation for a sequence of n pushes from the options below.

   (a) 2.5

(b) 3
(c) 4.5
(d) 5.5
(e) 8.5
(f) 9

(v) Which of the splay steps distinguishes splaying from just rotating the accessed element to the top:

(a) zig
(b) zig-zag
(c) zig-zig
(d) none of the above

(vi) If each node in a splay tree with n nodes has unit weight ($w(x) = 1$), then the maximum size of any node is

(a) $\Theta(1)$
(b) $\Theta(\log^* n)$
(c) $\Theta(\log \log n)$
(d) $\Theta(\log n)$
(e) $\Theta(n)$

(vii) Consider a complete balanced binary search tree T of $n = 255$ nodes with height 8. In splay tree terminology, a node x that is at depth d from the root (the root node has depth 0) is assigned weight $w(x) = 2^{7-d}$. What is the maximum rank of a node in T?

(viii) Let x be a node in a splay tree, with y as parent and z as grandparent. Let r_x be the rank of x and r_z be the rank of z. Suppose we perform a single splay step at x. Let r'_x be the new rank of x and r'_z be the new rank of z. Then it must be the case that $r_x = r'_z$.

(a) True
(b) False

(ix) Let x be a node in a splay tree, with y as parent and z as grandparent. Let r_x be the rank of x and r_z be the rank of z. Suppose we perform a single splay step at x. Let r'_x be the new rank of x and r'_z be the new rank of z. Then it must be the case that $r'_x = r_z$.

(a) True
(b) False

2. **(Binary Counter Revisited)** Suppose we are incrementing a binary counter, but instead of each bit flip costing 1, suppose flipping the i^{th} bit costs us 2^i. (Flipping the lowest order bit `A[0]` costs $2^0 = 1$, the next higher order bit `A[1]` costs $2^1 = 2$, the next costs $2^2 = 4$, etc.) What is the amortized cost per operation for a sequence of n increments, starting from zero?

3. **(Ternary Counter)** Suppose that instead of a binary counter, we maintain a ternary counter such that incrementing each digit still costs one.

   (a) What is the total cost of a sequence of n increments starting from zero?

   (b) Define a suitable potential function for analyzing the cost of each increment

   (c) Use your potential function to determine the amortized cost per increment and show that this matches the cost you got in Part 3a.

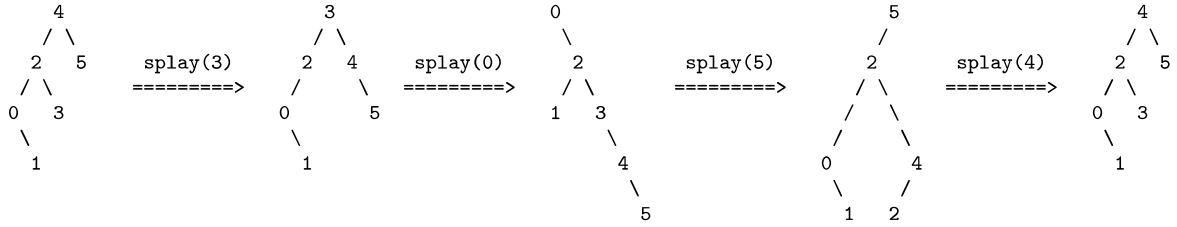
4. **(Simulating a queue)** A stack supports "push x" and "pop" and each operation has a cost of 1. Show how to implement a queue that supports enqueue and dequeue using two stacks. Prove using a potential function that the amortized cost of enqueue is 3 and the amortized cost of dequeue is 1.

5. **(Balls in bins)** There are n balls and an infinite number of bins. A bin can have 0 or more balls in it. A move consists taking all the balls of some bin and putting them into distinct bins. The cost of a move is the number of balls moved. Define the potential of a state of this system as the sum of the potentials of all the bins. The potential of a bin with k balls in it is:

$$\Phi(k) = \max(0, k - z)$$

   Where for convenience $z = \lfloor \sqrt{n} \rfloor$. Prove that the amortized cost of a move is at most $2z$.

6. **(Cyclic Splaying)** Starting from a tree T_0 of n nodes a sequence of $\ell \geq 1$ `splay` operations is done. It turns out that the initial tree T_0 and the final tree T_ℓ are the same. Let k be the number of *distinct* nodes splayed in this sequence. (Clearly $k \leq \ell$.) Below is an example where $k = \ell = 4$ and $n = 6$.



   Use some setting of node weights to show that the average number of splaying steps in this cycle (i.e., the average per `splay` operation) is at most $1 + 3\log_2 n$. Make use of the Access Lemma for splay trees covered in lecture yesterday.

7. **(Cyclic splaying even faster)** Recall the cyclic splaying problem from above. Use a different setting of node weights to show that the average number of splaying steps in this cycle (per `splay` operation) is at also most $1 + 3\log_2 k$.

8. **(Spray paint)** At the FTW Motor Company, there's an infinite line of cars, each of which are initially colored white. Associate the cars with the integers, both positive and negative. Management sends the paint crew a sequence of `Spray` commands: each command is of the form `Spray`(x, y, c) (where $x \leq y$), which requires spray-painting

all the cars/integers in the interval $[x, y]$ with the color c. The cost of each operation `Spray`(x, y, c) is the number of distinct colors in the range $[x, y]$ before the operation is performed.

(a) Show using a potential function that the cost of N paint operations is at most $3N$.

(b) Show that any constant less than 3 would not work.