15-451/651 Algorithm Design & Analysis
Spring 2023, Recitation #2

Objectives

- Practice deriving data structures with good amortized bounds
- Practice using potential functions to determine amortized bounds
- Understand how to use and analyze splay trees

Recitation Problems

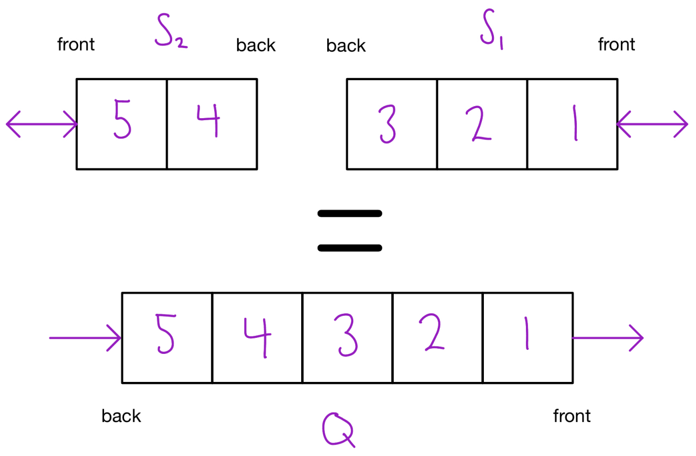
1. **(Deque)** There is a classic method to construct a first-in-first-out queue with $O(1)$ amortized cost operations `pushBack` and `popFront` from two last-in-first-out stacks with cost 1 operations `pushFront`, `popFront`, and `size` as follows:

   Let stacks S_1 and S_2 represent the front/head and back/tail of the queue Q respectively. Then implement the operations by moving all elements from S_2 to S_1 whenever we would need to remove but it is empty:

```
pushBack(x) {
   S2.pushFront(x)
}

popFront() {
   if (S1.size() == 0) {
      for i in range(S2.size()) {
         S1.pushFront(S2.popFront())
      }
   }
   return S1.popFront()
}
```

   Under this implementation the amortized bounds follow from setting $\Phi(S_1, S_2) = 2|S_2|$ and correctness follows from the invariant that the elements of Q are always equal to the elements of S_1 appended to the elements of S_2 in reverse.



One natural extension of this is to implement a deque, which in addition to the standard queue functionality, also provides `pushFront` and `popBack`, allowing users to insert and remove from either side as they please.

(a) Suppose you provide symmetric implementations of those methods as follows:

```
pushFront(x) {
   S1.pushFront(x)
}

popBack() {
   if (S2.size() == 0) {
      for i in range(S1.size()) {
         S2.pushFront(S1.popFront())
      }
   }
   return S2.popFront()
}
```

Using the aggregate method, give a sequence of n operations under which each operation has $\Omega(n)$ amortized cost.

(b) Now suppose you had access to a third stack S_3 to use for temporary processing. Come up with a way to implement the deque operations that maintains the invariant but avoids the expensive case above.

(c) Define a potential function $\Phi(S_1, S_2)$ and use it to prove that the operations you defined have $O(1)$ amortized cost.

(d) Suppose that you start with an empty deque and then perform n operations (push or pop from either the front of the back). Bound the total actual cost of these operations.

2. **(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:

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

3. **(Optional: Union Find)** One of the most classic powerful uses of amortized analysis is for the union find data structure, which keeps track of disjoint sets. Initially, all elements in the universe $\mathcal{U}$ are in singleton sets, and union find enables the following two operations:

   - `Find`(x): Returns the ID of the set containing $x \in \mathcal{U}$. Takes $O(1)$ time.
   - `Union`(x, y): Permanently joins the sets containing elements $x, y \in \mathcal{U}$ not already in the same set. Takes $O(\min(|S_x|, |S_y|))$ if S_x and S_y are the distinct sets containing x and y respectively.

   Suppose we perform a sequence of n `Union` and n `Find` operations. What are the amortized costs of the `Union` and `Find` operations in this sequence?

Further Review

1. **(Union Find With Potentials)** Come up with a potential function for the union find problem (that may potentially go negative) such that the amortized cost of each `Union` is constant. Then use that potential function to prove the same bound on the total cost as in the Union Find problem.

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. **(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 $2z$.

5. **(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 $n = 6$.

```
    4                        3                    0                              5                      4
   / \                      / \                    \                            /                      / \
  2   5     splay(3)       2   4     splay(0)       2         splay(5)         2       splay(4)       2   5
 / \       =========>     /     \   =========>     / \       =========>       / \     =========>    / \
0   3                    0       5                1   3                      /   \                 0   3
 \                        \                            \                    /     \                 \
  1                        1                            4                  0       4                 1
                                                         \                  \     /
                                                          5                  1   2
```

   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.

6. **(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 $1 + 3 \log_2 k$.

7. **(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.