15-451/651 Algorithm Design & Analysis, Fall 2024

Recitation #4

Objectives

- Practice defining potential functions and performing amortized analysis
- Practice using union-find data structure

Recitation Problems

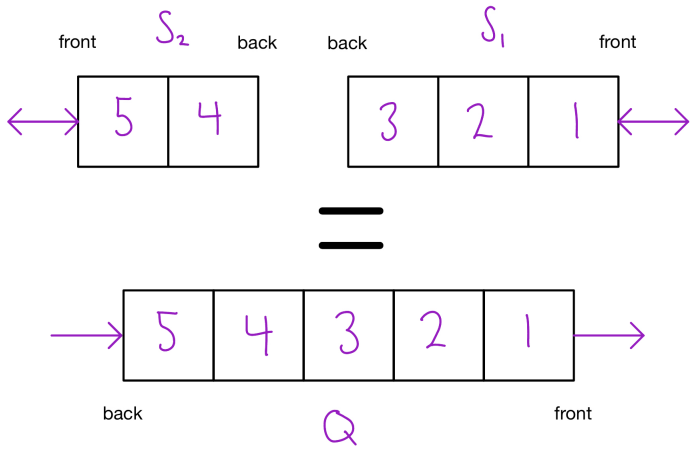
1. **(Deque)** There is a classic method to construct a first-in-first-out queue with $O(1)$ amortized cost operations `pushBack` and `popFront` from two last-in-first-out stacks with cost 1 operations `pushFront`, `popFront`, and `size` as follows:

   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()
}
```

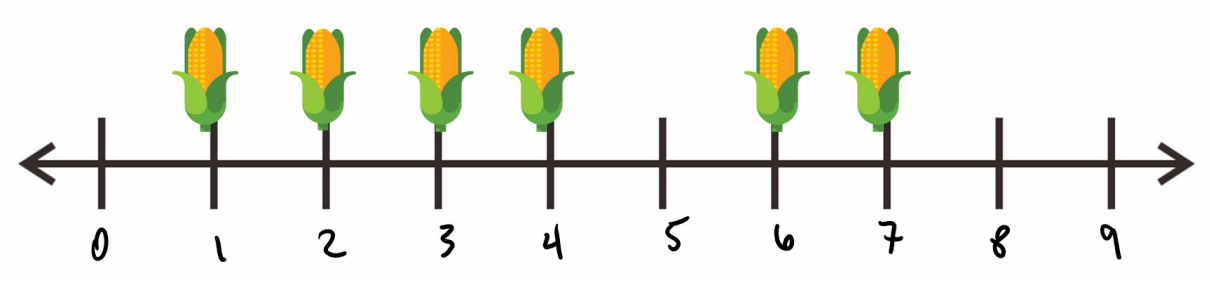
Using the aggregate method, give a sequence of n operations under which each operation has $\Omega(n)$ amortized cost.

(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. **(Corn Intervals)** You are a serf and you have a singular strip of land shaped like a number line. Your capitalist overlords have given you a list of n integer locations to plant cornstalks at in the order they are given. However, you are also lazy and only want to harvest good intervals of corn. A good interval $[i, j]$ is defined as a interval containing at least k consecutive cornstalks such that there are no cornstalks at $i-1$ and $j+1$.



In this picture, if $k = 3$, the interval $[1, 4]$ is the only good interval.

Write an algorithm keeping track of where you have planted corn such that at any time, you can determine the number of good intervals in $O(1)$. You're allowed $O(\log n)$ time to process each time you plant a cornstalk.

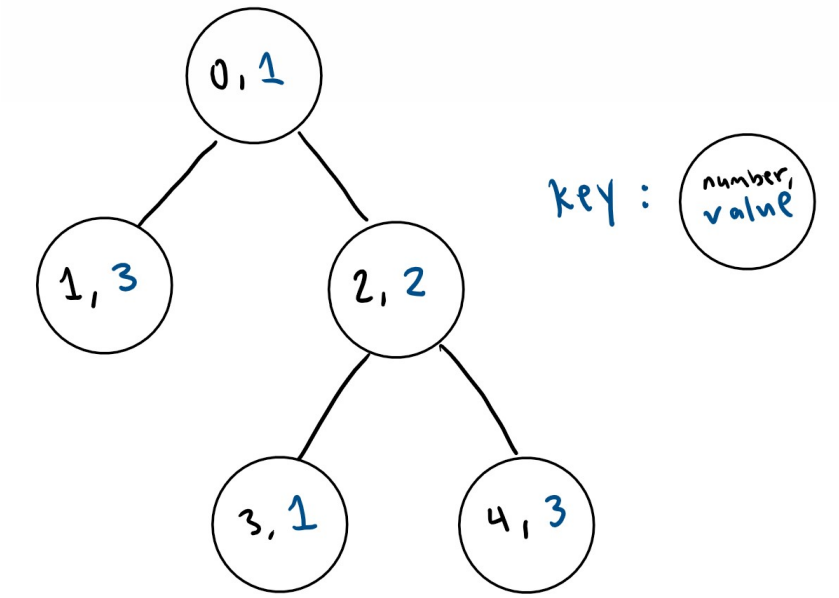
3. **(Tree Path)** There is a tree consisting of n nodes numbered from 0 to $n-1$ and exactly $n-1$ edges. Each node also has an integer value.

   You are also given a set E of undirected edges where $(i, j) \in E$ means that there is an undirected edge connecting nodes i and j.

   A good path is a simple path that satisfies the following conditions:

   The starting node and the ending node are distinct nodes with the same value. All nodes between the starting node and the ending node have values less than or equal to the starting node (and the ending node).

Note that a path and its reverse are counted as the same path.



In the picture above, there is 1 good path: 1–>0–>2–>4.

(a) Let's assume you know the maximum value in the tree, v_{max}. Find the number of good paths where the starting/ending node have value v_{max}.

(b) Now let's broaden the problem- find the total number of good paths in the tree in $O(n \log n)$.