CS 580 Algorithm Problem Set 2
Instructor: Prof. Yi Wu
Wei-Yen Day

Collaborators

Problem 1

1. Implement the stack operations: Push, Pop, and Minimum in O(1) time.
   Ans:

   We can use two stacks to implement these three operations in O(1) time. The first stack, S, is a normal stack for push and pop, and the second stack, $minS$, is for storing the minimum value.

   For Push routine, we push the given item into S, and check if the given item is less than the top of $minS$. If it is smaller, then we update the minimum value for the stack and push the smaller item to $minS$. If not, we keep pushing the top of $minS$ to $minS$. In this way, we can keep putting the minimum item on the top of $minS$.

   For Pop routine, we pop item from the normal stack S, as well as from the minimum stack $minS$.

   For Minimum routine, we just need to return the top item of $minS$ since the top item of $minS$ is always the current minimum value in the stack.

   The pseudo code looks like the following (given two stacks S and $minS$):

```
procedure PUSH(i)
    S.push(i)                                   ▷ O(1)
    if minS is empty then
        minS.push(i)                            ▷ O(1)
    else
        if i < minS.top() then
            minS.push(i)                        ▷ O(1)
        else
            minS.push(minS.top())               ▷ O(1)
        end if
    end if
end procedure
procedure POP(i)
    S.pop()                                     ▷ O(1)
    minS.pop()                                  ▷ O(1)
end procedure
procedure MINIMUM(i)
    return minS.top()                           ▷ O(1)
end procedure
```

   All the operations are bounded by O(1) since Push, Pop, and Top in stack operation are all O(1).

2. Implement the queue operations: Enqueue, Dequeue, and Minimum in amortized cost O(1).
Ans:

Suppose we already have a stack data structure, called min-stack, supporting Push, Pop, and Minimum operations in O(1) (implemented by the answer in previous question). Then following the idea in Prof. Wu's lecture, we only need to use two min-stacks to implement a queue with operations Enqueue, Dequeue, and Minimum in amortized O(1) time.

Now we have two min-stacks, called inbox and outbox respectively. For Enqueue, we push an item into the inbox min-stack. For dequeue, we pop one item from outbox if it is not empty, otherwise we pop all items from inbox and push them into outbox to make the order in inbox reversed, and pop one from outbox. For Minimum, we check the minimum of these two min-stacks and return the smaller one from both.

The pseudocode looks like follows:

```
procedure ENQUEUE(i)
    inbox.push(i)                                          ▷ O(1)
end procedure
procedure DEQUEUE(i)
    if outbox is not empty then
        outbox.pop()                                       ▷ O(1)
    else
        while inbox is not empty do
            outbox.push(inbox.pop())                       ▷ O(k), k is the number of items in inbox
        end while
        outbox.pop()                                       ▷ O(1)
    end if
end procedure
procedure MINIMUM(i)
    if inbox is not empty and outbox is not empty then
        return min(inbox.Minimum(), outbox.Minimum())      ▷ O(1)
    else if inbox is not empty then
        return inbox.Minimum()                             ▷ O(1)
    else if outbox is not empty then
        return outbox.Minimum()                            ▷ O(1)
    end if
end procedure
```

To do amortized analysis, we first define a potential function $\phi(s_i)$, which denotes the number of items in inbox and s_i denotes the state i. Each operation in stack costs 1. For Enqueue, the amortized cost would be $a_i = \phi(s_{i+1}) - \phi(s_i) + 1 = 2$ since the difference of potential function before and after push $\phi(s_{i+1}) - \phi(s_i)$ is 1. For Minimum, it is O(1) since all the operations are bounded by the running time of Minimum in min-stack. For Dequeue, we define two different scenarios: outbox empty or outbox not empty. If outbox is not empty, then it is just the running time of pop operation, which is O(1). If outbox is empty, then the difference of potential function would be $\phi(i+1) - \phi(i) = 0 - k = -k$ (assume k elements in inbox). We also need to push all the items to outbox and pop one from outbox, and each operation costs 1. Thus, the total amortized cost is $a_i = -k + (k+1) = 1$. Therefore, the

amortized cost of Dequeue is O(1).

Problem 2

Problem 3

Show the amortized cost of insertion for Splay Tree is $O(\log n)$.

Ans:

First, we have the insertion function like this:

procedure INSERT(T, x) ▷ T_0
  BST_insert(T, x) ▷ T_1, c
  Splay(T, x) ▷ T_2, c
end procedure

That is, we do a BST insertion first for x, and perform Splay on x. Assume before the BST insertion, after BST insertion, and after Splay operation, the tree is T_0, T_1, and T_2, respectively. Note that we have a constant cost c for doing BST insertion and Splay operation. Also, suppose we have a potential function $\Phi(T) = 2 \cdot \sum_{a \in T} r(a)$, where $r(a) = \log |T(a)|$. For the BST insertion, the largest change of potential function occurs at insert a node on its top; that is,

$$\Phi(T) \leq 2 \cdot \sum_{a \in T} r'(a) = 2(\sum_{a \in T} r(a) + r(i)) = 2 \sum_{a \in T} r(a) + 2 \log(n+1)$$

Where $r'(a)$ is the rank after insertion, and i is the inserted node. After insertion on the top, the potential function would increase $\log(n+1)$, where $n+1$ is the size of the new tree.

Therefore, we know that

$$\Phi(T_1) - \Phi(T_0) + c \leq 2 \log n$$

and we already know after Splay operation, the change of potential function is bounded by $3 \log n$:

$$\Phi(T_2) - \Phi(T_1) + c \leq 3 \log n$$

Add these two equations, we get

$$\Phi(T_2) - \Phi(T_0) + 2c \leq 5 \log n = O(\log n)$$

which is saying that the change of potential function before insertion and after insertion is bounded by $O(\log n)$. Thus, the amortized cost of insertion in Splay Tree is $O(\log n)$.