Problem Set No. 3

1. If a MULTIPUSH operation were included in the set of stack operations, would the $O(1)$ bound on the amortized cost of stack operations continue to hold? (CLR p. 360)

2. Show that if a DECREMENT operation were included in the k-bit counter example, n operations could cost as much as $\Theta(nk)$ time. (CLR p. 360)

3. Consider a redundant binary numbering system, in which each position denotes a power of two (as in ordinary binary numbering), but in which a digit of 2 is allowed, as well as 0 and 1. Note that, in such a system, a given number will in general have many representations, not just one. Suppose we start with the number zero and do an arbitrary sequence of intermixed increments and decrements. Show that the amortized cost per operation is O(1). (Compare to problem 2.)

4. Show that any sequence of m MAKE-SET, FIND-SET, and UNION operations, where all the UNION operations appear before any of the FIND-SET operations, takes only $O(m)$ time if both path compression and union by rank are used. What happens in the same situation if only the path-compression heuristic is used? (CLR p. 450)

5. *Off-line minimum*

   The *off-line minimum problem* asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations INSERT and EXTRACT-MIN. We are given a sequence S of n INSERT and m EXTRACT-MIN calls, where each key in $\{1, 2, \ldots, n\}$ is inserted exactly once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array *extracted* $[1..m]$, where for $i = 1, 2, \ldots, m$, *extracted* $[i]$ is the key returned by the ith EXTRACT-MIN call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys. (CLR p. 458)

   a. In the following instance of the off-line minimum problem, each INSERT is represented by a number and each EXTRACT-MIN is represented by the letter E:

      4,8,E,3,E,9,2,6,E,E,E,1,7,E,5.

      To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by

      I_1,E,I_2,E,$I_3, \ldots, I_m$,E,I_{m+1},

      where each E represents a single EXTRACT-MIN call and each I_j represents a (possibly empty) sequence of INSERT calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following,

```
OFF-LINE-MINIMUM(m, n)
for i ← 1 to n
    do determine j such that i ∈ K_j
       if j ≠ m + 1
          then extracted[j] ← i
               let l be the smallest value greater than j
                   for which set K_l exists
                 K_l ← K_j ∪ K_l, destroying K_j
return extracted
```

b. Argue that the array *extracted* returned by OFF-LINE-MINIMUM is correct.

c. Describe how to use a disjoint-set data structure to implement OFF-LINE-MINIMUM efficiently. Give a tight bound on the worst-case running time of your implementation.

6. Suppose that a node x is inserted into a red-black tree with RB-INSERT and then immediately deleted with RB-DELETE. Is the resulting red-black tree the same as the initial red-black tree? Justify your answer. (CLR p. 277)

7. *Join operation on red-black trees*

   The *join* operation takes two dynamic sets S_1 and S_2 and an element x such that for any $x_1 \in S_1$ and $x_2 \in S_2$, we have $key[x_1] \leq key[x_1] \leq key[x] \leq key[x_2]$. It returns a set $S = S_1 \cup \{x\} \cup S_2$. In this problem, we investigate how to implement the join operation on red-black trees.

   a. Given a red-black tree T, we store its black-height as the field $bh[T]$. Argue that this field can be maintained by RB-INSERT and RB-DELETE without requiring extra storage in the tree and without increasing the asymptotic running times. Show that while descending through T, we can determine the black-height of each node we visit in $O(1)$ time per node visited.

   We wish to implement the operation RB-JOIN(T_1, x, T_2), which destroys T_1 and T_2 and returns a red-black tree $T = T_1 \cup \{x\} \cup T_2$. Let n be the total number of nodes in T_1 and T_2.

b. Assume without loss of generality that $bh[T_1] \geq bh[T_2]$. Describe an $O(\lg n)$-time algorithm that finds a black node y in T_1 with the largest key from among those nodes whose black-height is $bh[T_2]$.

c. Let T_y be the subtree rooted at y. Describe how T_y can be replaced by $T_y \cup \{x\} \cup T_2$ in $O(1)$ time without destroying the binary-search-tree property.

d. What color should we make x so that red-black properties 1,2, and 4 are maintained? Describe how property 3 can be enforced in $O(\lg n)$ time.

d. Argue that the running time of RB-JOIN is $O(\ln n)$. (CLR p. 278)