CS2800 Design and Analysis of Algorithms
Problem Set #8 March 31, 2023

1. Show how to implement a queue using two stacks so that the amortized cost of enqueue and dequeue is $O(1)$. You can only perform push and pop on the stacks.

2. Let us consider a data structure to store a set of n elements that supports fast search and insertion. For a number n with binary representation $(b_k, b_{k-1}, \ldots, b_1, b_0)$, we have k arrays $A_0, A_1, \ldots, A_k$ such that if $b_i = 0$, then A_i is empty and otherwise A_i has 2^i elements. All the elements of the array A_i are sorted for each i, but there is no relation among the elements of different arrays.

   For instance, 11 elements will be stored using this data structure in sorted arrays A_3, A_1 and A_0 of lengths 8, 2 and 1 respectively since the binary representation of 11 is 1011.

   (a) Describe how you will perform a search operation in this data structure. What is the worst-case running time of your algorithm?

   (b) Describe how you will insert a new element in this data structure. What is the worst-case and amortized running times of your algorithm?

   (c) How will you implement a delete operation in this data structure? What is the worst-case running time of your algorithm?

3. Recall the analysis of path compression that we did in class. We had divided the nodes into buckets based on the rank where all the nodes with rank between k and 2^k were put in a single bucket.

   Suppose that we were not so clever, and we decided to create buckets where each bucket has nodes with ranks between $k+1$ and $2k$ starting from $k = 2$ together with nodes of rank 0, 1 and 2. If we did the analysis exactly like we did in class, what is the amortized running time for each Find operation that we will obtain?

4. Recall the interval scheduling problem that we saw when we discussed greedy algorithms. We had briefly mentioned a recursive way of thinking about finding the optimal set of non-overlapping intervals. Design a dynamic programming based algorithm for the problem.

   (The running time will be worse than the greedy algorithm we saw in class).

5. In the text segmentation problem that we saw in class, you are given a sequence of letters in an array $A[1, 2, \ldots, n]$, and you want to divide them into valid words. You have access to a subroutine IsWord that takes two indices i, j and returns **true** iff $A[i, i+1, \ldots, j]$ is a valid word.

   (a) Given an array $A[1, 2, \ldots, n]$ of characters, compute the number of partitions of A into valid words. For instance ARTISTOIL can be split in two ways: ARTIST-OIL or ART-IS-TOIL.

(b) Given two arrays $A[1, 2, \ldots, n]$ and $B[1, 2, \ldots, n]$ of characters, decide if A and B can be partitioned into valid words at the same indices. For instance, BOTHEARTHANDSAT and PINSTARTRAPSAND can be partitioned at the same indices as BOT-HEART-HAND-SAT and PIN-START-RAPS-AND.