Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-7017 Advanced Computing for Electrical Engineers

Assignment 1

Submission Deadline: August 25, 2013 (Sunday), 11:59 pm (IST)

Note: You can use any language from a set of languages (C, C++, Java, Pascal). You must follow academic ethics. Plagiarism is punishable act. Any academic unethical act may result into failure of the course.

Question 1: Write a routine to implement two stacks using one array. Your stack should not declare overflow unless every slot in the array is used.

Question 2: A *deque* is a data structure consisting of a list of items, on which the following operations are possible:
Push (X, D): Insert item X on the front end of *deque D*.
Pop (D): Remove the front item from *deque D* and return it.
Inject (X, D): Insert item X on the rear end of *deque D*
Eject (D): Remove the rear item from *deque D* and return it.
Write routines to support *deque* that take *O(1)* time per operation

Question 3: *Skip list* is an efficient linked list based data structure that attempts to implement binary search on linked list. Develop programs to implement search, insert, and delete operations in skip lists.

Question 4: A *queue Q* contains the items a_1, a_2, . . . , a_{n-1}, in the ordered with a_1 at front and a_{n-1} at the back. It is required to transfer these items on to a stack *S* (initially empty) so that a_1 is at the top of the stack and the order of all the items is preserved. Using *enqueue* and *dequeue* operations for the queue and *push* and *pop* operations for stack, design and implement an efficient *O(n)* algorithm to accomplish the above task, using only constant amount of additional storage.

Question 5: An alternative to the deletion strategy we have given is to use *lazy deletion*. To delete an element, we merely mark it deleted (using an extra bit field). The number of deleted or non deleted elements in the list is kept as part of the data structure. If there are many deleted elements as nondeleted elements, we traverse the entire list, performing the standard deletion algorithm on all marked nodes. Write a program to implement the standard linked list operations using lazy deletion.

Question 6: Write a program that will reverse a linked list while traversing it only once. At the conclusion, each node should point should point to the node which was previously its predecessor: the head should point to a node which was formerly at the end, and the node which was formerly first should have a null link.

Question 7: Write an array implementation of *self-adjusting* lists. A *self-adjusting* list is like a regular list is like a regular list, except that all insertions are performed at the front, and when the element is accessed by a *get()*, it is moved to the front of the list without changing the relative order of the other item.