CPS 130 Homework 11 - Solutions

1. (CLRS 11.1-4) We wish to implement a dictionary by using direct addressing on a *huge* array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object should use $O(1)$ space; the operations SEARCH, INSERT and DELETE should take $O(1)$ time each; and the initialization of the data structure should take $O(1)$ time.

   (Hint: Use an additional stack, whose size is the number of keys actually stored in the dictionary, to help determine whether a given entry in the huge array is valid or not.)

   Solution: To implement a direct-address dictionary on a huge array, we initialize an additional stack which we can use to help determine whether a given entry in the huge array is valid or not. Call the arrays *huge* and *stack* and define a variable *head*, initially -1, to point to the top of the stack array. Then $head = -1$ corresponds to the dictionary being empty. Suppose we want to insert an element with key x. We increment *head*, set $huge[x] = head$ and $stack[head] = x$. Now suppose we want to SEARCH for an element in the huge array with key y. Then the following conditions must be satisfied:

   - $0 \leq huge[y] \leq head$,
   - $stack[huge[y]] = y$,

   Thus SEARCH and INSERT can be done in constant time. To DELETE an element with key x, we first SEARCH to make sure the element is in the huge array. In the stack array we have a position that will correspond to a deleted element, so we take the element at $stack[head]$ and copy it to $stack[huge[x]]$. We set then set $huge[stack[head]] = huge[x]$, $stack[head] = \text{NULL}$ and $huge[x] = \text{NULL}$ and decrement *head*. DELETE is also performed in constant time since it only involves a constant number of operations. The initialization of the data structure is done in constant time since the allocation of the stack takes constant time.

2. (CLRS 11.2-2) Demonstrate the insertion of the keys $5, 28, 19, 15, 20, 33, 12, 17, 10$ into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \ mod \ 9$.

   Solution: 5 hashes to 5; 28 hashes to 1; 19 hashes to 1 $\rightarrow$ collision. Put 19 at the head of the linked list for 1. 15 hashes to 6; 20 hashes to 2; 33 hashes to 6 $\rightarrow$ collision. Put 33 at the head of the linked list for 6. 12 hashes to 3; 17 hashes to 8; 10 hashes to 1 $\rightarrow$ collision. Put 10 at the head of the linked list for 1.

0	/					
1	$\rightarrow$	28	$\rightarrow$	19	$\rightarrow$	10
2	$\rightarrow$	38				
3	$\rightarrow$	12				
4	/					
5	$\rightarrow$	23				
6	$\rightarrow$	15	$\rightarrow$	33		
7	/					
8	$\rightarrow$	17				

3. (CLRS 11.2-3) Professor Marley hypothesizes that substantial performance gains can be obtained if we modify the chaining scheme so that each list is kept in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions and deletions?

   Solution: