CSCE 221 - Homework Set 6
Due 3/8/2015, 11:59 PM

Problem 1. R-9.1 (10 points)

In separate chaining, the load factor of the hash table is below 1. With open addressing, the load factor of the hash table is at most 1.

Problem 2. R-9.7 (10 points)

0	1	2	3	4	5	6	7	8	9	10
13	94	-	-	-	44	-	-	12	16	20
-	39	-	-	-	88	-	-	23	5	-
-	-	-	-	-	11	-	-	-	-	-

Problem 3. R-9.8 (10 points)

0	1	2	3	4	5	6	7	8	9	10
13	39	94	16	5	44	88	11	12	23	20

Problem 4. R-9.10 (10 points)

0	1	2	3	4	5	6	7	8	9	10
13	94	23	88	39	44	11	-	12	16	20

For the key (5), there is no empty bucket available.

Problem 5. R-9.13 (10 points)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-	-	13	18	41	-	36	25	-	54	-	-	38	10	-	90	28	-	-

Problem 6. R-9.15 (10 points)

The worst case is O(n), since every entry's hash function of its key has to be calculated, and then inserted into the bucket of the hash table.
In the best case, it can be implemented to run in O(1) expected time.

Problem 7. Why do we use the special object 'AVAILABLE' when removing an element from an hash table using open addressing? (10 points)

In its simplest form, AVAILABLE lets the program know that upon its next iteration, there is an empty entry that was not previously empty.

Problem 8. C-9.7 (20 points)

First, we will search for the desired elements and delete it. Move to the next bucket, and if that one is also empty, we are done. Otherwise, if the bucket is full, then delete the bucket and re-add the element to the table using normal means. To avoid adding this back to its original spot, the item must be deleted before re-adding it.

Problem 9. R-9.16 (10 points)

Drawn on additional problems.

Problem 10. R-9.17 (15 points)

For removing a key from the skip list, search the key by using the SkipSearch(k) function, which returns a position p in the bottom list that contains the entry with the largest key value that is $\leq$ k. If the entry at position p contains the entry with key different from k, return null. If else, remove p and all positions above p using the above function.
Algorithm: erase(k)
// find a position p having the largest key less than or equal to k.
p $\leftarrow$ SkipSearch(k)
// entry at p is not equal to k
if(key(p) != k) then
return null
// entry at p is equal to k remove the position p and positions above p.
while(above(p)) $\leftarrow$ null) do
p $\leftarrow$ above(p)
remove(p)