Assignment 4 Solutions

COT 6405 - Introduction to Theory of Algorithms

1. Exercise 11.2-1: Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l$ and $h(k) = h(l)\}$?

   Answer:
   For each pair of keys k, l, where $k \neq l$. When $h(k) = h(l)$, we define $X_{kl} = I\{h(k) = h(l)\}$, which is the indicator of collision. Because we assume simple uniform hashing, the probability $Pr\{X_{kl} = 1\} = Pr\{h(k) = h(l)\} = 1/m$. So $E[X_{kl}] = 1/m$.
   Let Y as the total number of collisions. So $Y = \sum_{k \neq l} X_{kl}$. So The expected number of collisions is :

$$E[Y] = E\left[\sum_{k\neq l} X_{kl}\right]$$
$$= \sum_{k\neq l} E[X_{kl}]$$
$$= \binom{n}{2}\frac{1}{m}$$
$$= \frac{n(n-1)}{2} \times \frac{1}{m}$$
$$= \frac{n(n-1)}{2m}$$

2. Exercise 11.2-3: Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

   Answer: For successful and unsuccessful searches, we still may need to search the length of an entire chain. Thus, the complexity of both remains unchanged at $\Theta(1 + \alpha)$.
   For deletions, we still need to operate a search followed by a constant-time deletion, so it remains $\Theta(1 + \alpha)$.

For insertions, we cannot simply prepend elements to a chain. We must walk them to their proper location. This might involve a full walk of a chain in the worst case, increasing the complexity for insertions from $\Theta(1)$ to $\Theta(1+\alpha)$.

Thus, the Professor's hypothesis is incorrect; sorting chains would generally worsen performance.

3. Exercise 11.4-1: Consider inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m-1)$.

   Answer:
   linear probing: Insert x h(x) = x mod 11, if collision, use linear probing.
   h(10) = 10 mod 11 = 10;h(22) = 0; h(31) = 9; h(4) = 4;
   h(15) = 4 collision, probe down h(15) = h(15,1) = (15+1) mod 11 = 5;
   h(28) = 6; h(17) = 6, so h(17) = h(17,1) = (17+1) mod 11 = 7;
   h(88) = 0, collision h(88) = h(88,1) = 1;
   h(59) = 59 mod 11 = 4, collision, we could see 5, 6 ,7 have collisions. So h(59) = h(59,4) = (59+4) mod 11 = 8 The slots are

0	22
1	88
2	
3	
4	4
5	15
6	28
7	17
8	59
9	31
10	10

   quadratic probing: $h(k,i) = (h(k) + c_1 i + c_2 i^2) mod 11$
   h(10) = 10 mod 11 = 10;h(22) = 0; h(31) = 9; h(4) = 4
   h(15)=4, collision so = h(15,1) = (15+1 + 3) mod 11 = 8.
   h(28) = 6;
   h(17)= 6 collision so = h(17,1) = 10, also collision = h(17,2)= (17+ 2 + 3*4) = 9, continue h(17,3) = (17+3+3*9) mod 11 = 3
   h(88) = 0 collision so h(88,1) = 4, h(88,2) = 3, h(88,3) = 8, h(88,4) = 8, h(88,5) = 3, h(88,6) = 4, h(88,7)=0, h(88,8) = (88+8+3*64) mod 11 = 2
   h(59) = 4, collision, h(59,1) = 8, h(59,2) = (59+2+3*4)mod 11 = 7. The slots are

0	22
1	
2	88
3	17
4	4
5	
6	28
7	59
8	15
9	31
10	10

Double hashing probing.
$h_1(k) = k and h_2(k) = 1 + (k\ mod\ (m-1)):$
h(10) = 10 mod 11 = 10;h(22) = 0; h(31) = 9; h(4) = 4;
h(15) = 4, collision, so h(15,1) = $(15 + 1 * (1 + 15\ mod\ 10))\ mod\ 11 = 10, h(15, 2) = (15 + 2 * (1 + 15\ mod\ 10))\ mod\ 11 = 5$;
h(28) = 6;
h(17) = 6; collision, so h(17,1) = $(17 + 1 * (1 + 17\ mod\ 10))\ mod\ 11 = 3.h(88) = 0$; collision, so $h(88, 1) = 9; h(88, 2) = (88 + 2 * 9)\ mod\ 11 = 7$.
h(59) = 4; collision, so h(59,1) = 3; h(59,2) = $(59 + 2 * 10)\ mod\ 11 = 2$.
The slots are

0	22
1	
2	59
3	17
4	4
5	15
6	28
7	88
8	
9	31
10	10

4. Exercise 11.4-3: Consider an open-address hash table with uniform hashing. Give upper bounds on the expected number of probes in an unsuccessful search and on the expected number of probes in a successful search when the load factor is 3/4 and when it is 7/8.

   Answer: The upper bound for the number of probes in an unsuccessful search is $1/(1 - \alpha)$, where α is the load factor.

$$\text{For } \alpha = 3/4 \longrightarrow \frac{1}{1-\frac{3}{4}} = \frac{1}{\frac{1}{4}} = 4$$

$$\text{For } \alpha = 7/8 \longrightarrow \frac{1}{1-\frac{7}{8}} = \frac{1}{\frac{1}{8}} = 8$$

The upper bound for the number of probes in a successful search is $(1/\alpha)ln(1/(1-\alpha))$.

$$\text{For } \alpha = 3/4 \longrightarrow \frac{1}{\frac{3}{4}}ln\left(\frac{1}{1-\frac{3}{4}}\right) = \frac{4}{3}ln(4) \approx 1.85$$

$$\text{For } \alpha = 7/8 \longrightarrow \frac{1}{\frac{7}{8}}ln\left(\frac{1}{1-\frac{7}{8}}\right) = \frac{8}{7}ln(8) \approx 2.38$$

5. Exercise 12.1-2: What is the difference between the binary-search-tree property and the min-heap property (see page 153)? Can the min-heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

   Answer: The BST property is that for all parent nodes, the left child is less than the parent, and the right child is greater than the parent. The min-heap property is that all parent nodes are larger than their children. This cannot be used for $O(n)$ printing because the relation between the two children of a parent is not known. The heap must be sorted, and heapsort (and other comparison sort algorithms) are $O(nlgn)$ in the worst case.

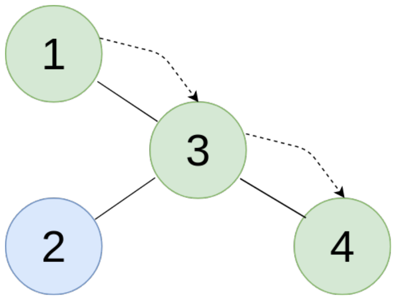
6. Exercise 12.2-3: Write the Tree-Predecessor procedure.

Algorithm 1 Tree-Predecessor(x)

```
1: if x.left ≠ NIL then
2:     return Tree-Maximum(x.left)
3: y = x.parent
4: while (y ≠ NIL and x == y.left)
5:     x = y
6:     y = y.parent
7: return y
```

7. Exercise 12.2-4: Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professors claim.

Answer: In this counterexample, we search this binary tree for key $k = 4$. The search path, B, is in green. Elements to the left of the search path, A, are in blue; we see that it is only one element, 2. However, because $2 > 1$, and 1 is in the search path, $a \geq b$, disproving the Professor's claim.



8. Exercise 12.2-5: Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

    Answer: If a node has a left child, its predecessor is the rightmost node in the left subtree; the rightmost node in this subtree will have no right child by definition, as having a right child would not make it rightmost, and thus not the predecessor. Similarly, the successor of a node, should it have a right child, is the leftmost node in the right subtree. By definition, the leftmost node of that subtree will have no child, following similar logic.

9. Exercise 12.3-2: Suppose that we construct a binary search tree by repeatedly inserting distinct values into the tree. Argue that the number of nodes examined in searching for a value in the tree is one plus the number of nodes examined when the value was first inserted into the tree.

    Answer: The Tree-Insert and Tree-Search algorithms share the same traversal logic, save that Tree-Insert places a new element at the final location, while Tree-Search traverses to that final location. Thus, Tree-Search examines one additional node while traversing the tree.

10. Exercise 12.3-3: We can sort a given set of n numbers by first building a binary search tree containing these numbers (using Tree-Insert repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

    Answer: Tree-Insert has a running time of $\Theta(h)$, where h is the height of the tree. For n elements, inserting them all takes $\Theta(nh)$.

    In the worst case of a maximally unbalanced tree that approximates a linked list, $h = n$, and so Tree-Insert takes $O(n)$. The tree construction time becomes $O(n * n) = O(n^2)$. Printing them all via a tree walk adds n operations for $O(n^2 + n) = O(n^2)$.

In the best case of a perfectly balanced tree, $h = lgn$, resulting in a running time for Tree-Insert of $\Theta(lgn)$. The tree construction running time becomes $\Omega(nlgn)$. Printing them all leads to a total running time of $\Omega(nlgn + n) = \Omega(nlgn)$.