Computational Thinking
Sample Solutions to Exercise 7 (Hashing)

1 Robin Hood Probing

a) Suppose a collision happens and the two objects have probe sequence lengths (psl) i and j. No matter which object we choose to keep in the bucket, the other object will travel the same length in its probing sequence from that point on, denoted k, since the next empty bucket is the same for both objects. Therefore, the sum of the probe sequence length will remain the same and equal to $i + j + k$. Thus, the expected value of the psl does not change with the Robin Hood modification.

b) Every time we have two colliding objects, we move the one with the shorter psl. Since linear probing suffers from primary clustering, both object from this point on will need the same number of probes to find an empty bucket. Thus, the longest psl cannot be longer since we started with the shorter psl.

c) Similarly to the previous question, it is easy to see that the shortest psl will be larger compared to the hashing with linear probing. Thus, the variance decreases. More formally, suppose we have two colliding objects k_1 and k_2 and their corresponding probing sequence length (in collision) are i and j, respectively, where $i < j$. Then we have three possibilities:

- $i < j < \mathbb{E}(psl)$: If we move k_1 then the variance will decrease more than in the case we moved k_2.
- $i < \mathbb{E}(psl) < j$: If we move k_1 we decrease the variance, while if we move k_2 we increase it.
- $\mathbb{E}(psl) < i < j$: If we move k_1 we increase the variance less than if we move k_2.

Thus, in any case, moving the object with the smallest probing sequence position is beneficial towards decreasing the variance of the psl.

2 Hashing with Probing: Deletion

a) Suppose we had a collision when inserting k_2 as the bucket $h_1(k_2)$ was taken by k_1, so k_2 was inserted in $h_2(k_2)$. If we do the naive removal of k_1 and later search for k_2, the search function finds the bucket $h_1(k_2)$ is empty and returns before finding k_2.

b) If we delete an object, we mark the bucket with a special flag. We can insert new keys in flagged buckets normally (like in empty buckets). The search operation stops when we find an empty bucket that is not flagged.

c) Although the table might be almost empty, we still have to go through the entire probing sequence we used to insert the objects. Thus, the search operation can be very inefficient. At the worst case we have to go through the entire table.

d) Instead of flagging buckets after removal, we do the following to delete a key: locate the key k you want to delete in some bucket b. Then do another linear probing through buckets $b+1, b+2, \ldots$ until you find either an empty bucket or a bucket c where some key k' was placed with the first probe: $c = h_1(k')$. Then, delete k and shift backwards by one bucket all the keys between $b+1$ and $c-1$. Mark $c-1$ as empty.

This works if we use Robin Hood insert from the previous exercise. Otherwise, if there is a key in bucket c, there might be some keys that "jumped over" c that we need to shift too. In this case, we probe until the first empty bucket and shift all keys except those placed with the first probe (but using Robin Hood insert should be simpler).

3 Not Quite Universal Hashing

Consider two keys $k_1 = (0, \ldots, 0)$ and $k_2 = (1, 1, 0, \ldots, 0)$. Then $h_a(k_1) = 0$ for all a, and $h_a(k_2) = 0 \Leftrightarrow a_1 + a_2 \equiv 0 \mod m$. Since we assume $a_1, a_2 > 0$, this means $a_2 = m - a_1$. There are $m-1$ possible values of a_1, each associated with a unique a_2 where $h_a(k_2) = 0$. Hence, $\Pr[h_a(k_1) = h_a(k_2)] = \frac{1}{m-1} > \frac{1}{m}$.