Assignment 1 - Theory questions.

Please refer to the quiz solutions on mycourses for the exact question wording, question weighting and correct answers.

Detailed explanations in blue (by Roman SG)

1) What is the number of collisions expected when inserting a key in a hash table of size m using Open Addressing when alpha -> 1?

Acceptable answers : m, M

When inserting a key into a hash table with linear probing that is full, we will be forced to try all values of i until we reach the number of slots, since we will be trying to insert on every single slot without success. The only acceptable answer that could fit in fewer than 2 characters was m, especially given that it was specified in the question.

0 is not an acceptable answer because the concept of collision for an open addressing hash table was clearly defined in the lecture and in the assignment.

The number you obtained in the programming part of the assignment is also not acceptable answer because the number of collisions in this situation for a table of size m obviously depends on m.

2) Assuming alpha=1, which of the following statement(s) offer the best explanation for your answer to question 1 ? (Select all that apply)

1) The algorithm cannot terminate until it reaches i=m
2) Because open addressing tends to cause clustering of keys, creating a lot of collisions when alpha approaches 1 because the keys are not evenly distributed.
3) Because by the time alpha reaches 1, the remaining empty slots in the array are not reachable with the hash function.
4) The array is full
5) The table is empty

1 - must be true since, as described in the previous question, the algorithm for inserting into an open addressing hash table with linear probing was clearly defined in class: you start at i=0 and increment i until you find an empty slot, or until your reach i=m. In this case, we assume alpha=1, thus the table must be full, and the algorithm cannot terminate until i=m.

2 - this statement can be true in a vacuum, but in the context of this question, it does not contribute to the explanation at all. Indeed, at alpha=1, the table is full. If the table is full, the keys must be evenly distributed, and they cannot be forming clusters. This does not contribute at all to explaining why the number of collisions goes to m when alpha=1.

3 - This is simply not true. Incrementing i means that all slots are reachable with any remotely reasonable hash function.

4 - This is true. When alpha=1, the array is full, and the array being full is the main reason for the number of collisions being m; it's because the key does not fit in the array because the array is full.

5 - this statement is absolutely false. At alpha = 1, the table is full.

3) Assume that using a hash table based on open addressing, you implemented a method that, given a key, finds it in the table and deletes it by marking the slot as "empty", or -1. Which of the following statements is (are) true ? Select all that apply

1) The key is always deleted correctly while visiting the minimum number of slots.
2) If, when searching through the hash table, your method terminates when reaching an empty slot, the key might not be removed correctly.
3) If the method never misses the key when deleting, then it will always output m when the key is not in the table.
4) If the method never misses the key when deleting, then it will always output m-1 when the key is not in the table.
5) The method will visit every slot in the table at each execution.

First, there are multiple keys to interpreting this question correctly. First, it starts with the word "Assume", which means that it does not necessarily match the method you used in the assignment. This assumes that the slot the key is found in is marked as empty, or -1, i.e. for the rest of the question, assume you did the assignment without choosing to use another notation for deleted slots.

Second, let's establish the method that visits the fewest slots to delete the key. The idea is to, given a key to remove, get the hash function, enter the table at that hash function and visit every slot one by one until you reach an EMPTY slot (a slot that has never had a key in it). Since this slot has never had a key in it, and you have already passed the hash function of the key you are looking for, then the key would have been inserted here in priority before being inserted further into the table, so you can stop the search here. This method stops working if you don't use a different notation for EMPTY and REMOVED though, since it is very possible that you try to insert a key, you jump over a couple keys, and then when of those keys are removed later on. When you call removeKey on your initial key after that, the algorithm will terminate when reaching the EMPTY slot. In order to always remove the key correctly when using only one notation for both EMPTY and REMOVED, you have to keep increasing i until i=m when the key is not found to make sure it is not in the array.

With this in mind:
1- This is not true because if you use the algorithm that visits the minimum number of slots without having 2 different notations, you will fail to remove some keys. If you remove all keys correctly, you will often visit more slots than you need.

2- That is true assuming you mark deleted key slots as EMPTY, which is the main reason you need to use another notation for REMOVED.

3- That is true; with only one notation, you have to visit the whole table every time to make sure you did not miss the key.

4- Here, it is not the number of collisions we are talking about but the number of slots visited, which will be m.

5- That is never true because both algorithms terminate upon finding the key.

5) Let w=10, r=5, m=32. For which values of \alpha does chaining become strictly more efficient than open addressing? Give an approximate answer

Answer : about 0.45

Everyone should have the same graph. We are looking for the approximate value after which chaining strictly outperforms open addressing, which is about 0.45.

D) Question suggestion: Consider your plot on the pdf file you submitted. Which of the following behavior could be observed between 0 and 1 if you increased n ?
1) the two lines would separate further
2) the two lines would be closer
3) it would not change, because the plot only depends on w
4) The lines could come closer or separate further depending on the W values selected
5) increasing n without changing w is not possible

answer : a

This is something that is very easy to observe if you plot different w values at a constant n, and you try again for a different n.

2- This should never happen. If you got full marks for your code and you are observing this, please let us know, I would be very curious to see this.

3- The number of collisions obviously depends on n as well as w.

4- If you choose reasonable values of w, this would never happen. I think you can observe it if you go high enough to get overflow but obviously those cannot be considered reasonable values.

5- w does not depend on n.

6) Consider your plot for task 1 of the main method in Assignment 1. Why is it that the number of collisions between the two methods is so close for small values of alpha. Out of the following statements, highlight those that contribute to answering that question.

1) For very low alpha values, any reasonable hashing methods will perform well
2) Open Addressing + Linear Probing is at its best on low alphas

3) When the slot corresponding to the hash value of the insert key is empty, Open addressing and chaining generate the same number of collisions.
4) Open Addressing and Linear Probing always generate the same number of collisions
5) Collisions are impossible for alpha<0.05, so both methods will report zero collisions, hence will have the same performance on the alphas in that range.

1 - Always true. By definition, a very low alpha means that the hash table is almost empty, which makes collisions rare.

2- This is true for any method. The emptier the table, the fewer collisions.

3- Always true. When the slot corresponding to a hash value is empty, then both methods will insert with no collision. You can notice that this does not depend on i at all, as this holds for any value of i. Any output of some hash function h(x) or g(x) that maps to an empty slot will cause zero collision.

4- This is false, as observed in the homework.

5- At low alphas, collisions are rare, but definitely not impossible.