Daniel D'Alessandro
Tamer Wahba
Henry Dinhofer

Project Proposal:

We will test two hashing methods, and three input types. We will evaluate insert time, search time, and delete time. We will also evaluate the space complexity required to store the hash table based on the input types. First we will implement open addressing hashing and perfect hashing. For the open addressing method we will use three different probing methods: linear, quadratic, and double hashing. Github will be used for collaboration.

Project Management:

We divided the work as follows:
-- Collaboratively create relevant method signatures for all classes and functions.
-- Henry will create the main, which will be the driver for all the tests. There will be minimal input checking since we will be the ones generating the input.
-- Tamer will implement the perfect hashing method.
-- Daniel will implement the open addressing method and its linear probing function.
-- Henry will implement the quadratic method.
-- Tamer will implement the double hashing method.
-- Daniel will create a tester that ensures the functions are implemented correctly and function as expected.
-- Tamer will create the test data.

Project Expectations:
We expect the running time for these methods to be O(n) in the worst case. For perfect hashing we will use dynamically select hashing functions from the set of universal hashing functions. After ensuring that the implementations function correctly, we will create four input sets. The first will be densely packed with no keys that hash into the same slot. This set is expected to have the best case scenario for all functions being tested. It will serve as our 'control.' The second set will be composed of keys that hash into the same values. The duplicate keys will be 25% of the set. The third input type will be composed of sparse keys with large gaps in their values. We will do each test twice, one time with some deletions after insertions, and another with only insertions and searches. This is to show that perfect hashing performs in O(1) worst case time when no deletions occur. We will fix the possible key range to 2^16 values.

Method	Expected Search Running Time	
	Worst	Average
Linear	O(n)	O(1)
Quadratic	O(n)	O(1)
Double	O(n)	O(1)
Perfect	O(1)	O(1)

Project Input Example:

Our input file consisted of various keys and values along with the operation to perform on them. For example INSERT 3 means that 3 keys will be inserted into the hash table. The first line is the number of lines in the file. If an invalid instruction is encountered, the program prints an error message and exits.

Test Data Generations:

We used a python script to create the different datasets. Four sets are generated; one with 25% repeated keys, one with 50% repeated keys, one with no repeated keys, and one with spread out keys. We chose these data sets to show diversity in input and to identify the strength of the different collision avoidance techniques.

Project Results:

We found that the perfect hashing method was the most effective algorithm however double hashing also works really well. It was expected that linear hashing was not going to be the most effective because of its sheer simplicity. A Hash Tables runtime cost is directly associated with its effectiveness at dealing with collisions.