CSC201 Data Structures
Fall 2020
Frank Liu
Program 3 Write Up

Test Case 1	
file name (with pixel dimensions)	Ollie200x200.raw
initial table size	1001
# collisions	36691(do not count rehash) 51513 (count rehash)
# rehashes	5
final table size	32117
runtime	1103974600 nanoseconds 1103 milliseconds

Test Case 2	
file name (with pixel dimensions)	Ollie200x200.raw
initial table size	101
# collisions	36962 (do not count rehash) 49883 (count rehash)
# rehashes	8
final table size	27803
runtime	1396439900 nanoseconds 1396 milliseconds

Test Case 3		
file name (with pixel dimensions)	Hibiscus400x300	LadyBugs800x600
initial table size	120	480

# collisions	119438 (do not count rehash) 180877 (count rehash)	478512 (do not count rehash) 975561 (count rehash)
# rehashes	10	11
final table size	126487	1002017
runtime	3002680200 nanoseconds 3002 milliseconds	17234144000 nanoseconds 17234 milliseconds

The method I am implementing is quadratic probing. Quadratic probing deal with collision by searching positions after the original position using step size of i^2. This will effectively deal with collision since if we promise that the hash table is less than half full, we are guaranteed to probe to an empty space. When run my program with Ollie200x200.raw, I have at most 40000 different kind of pixels. But we can be confident that we will get a table size (actual size) less than 40000 since there will be identical pixels. But the final table size of the hash table is larger than actual size since there will be empty place in the hash table. When the initial table size is 1001 rather than 101, I have less collision. That make sense since if you have a larger table size to start with, you will have more empty spots, and the position output from hash function will falls into a bigger range (0 – table size – 1). The number of rehashes is fewer in 1001 initial size than in 101 initial size. That is because we have fixed number of different pixels. You will rehash in the 1001 case when your actual size is greater than 500, but rehash in the 101 case when your actual size is greater than 50, and even doubling 101 and find the next prime, the rehashed table still have a size less than 1001. Therefore, it should be clear that 1001 initial size will have fewer number of rehashes. The final table size for 1001 case is greater than 101 case because if you start at a bigger size, even with fewer rehashes, you are doubling your table size from an already-big-enough size, which ends up larger than the final size of 101 case. Since 1001 table size has fewer collisions and fewer number of rehashes, it will have a faster running time (1103 ms) comparing to 101 case (1369 ms). Follow this trend, when I move on to more complicated graphs such as Hibiscus400x300.raw and LadyBugs800x600.raw, we see the relationship that with increase of image's dimension, which means an increase of numbers of different pixels, there's an increase of number of collisions, number of rehashes, final table size, and runtime. That is because we have more pixels to process, and starting from a relatively small size, we need to do more rehash, probing, and expanding, which accumulates and results in a long runtime.

With a careful analysis of my own collision resolution's behavior, I am going to compare my result with my teammate's collision resolution. Since we are in a group of two, I will focus on compare and contrast separate chaining and quadratic probing. Overall, when keeping our hash function, image, dimension, initial table size to be exactly same, separate chaining performs extremely better than quadratic probing in terms of runtime and number of collisions. For separate chaining, the run time of all four different situations not yet reached 1 ms. It measure at a nanosecond scale while quadratic probing's run time in all four cases reach from 1000 ms to 20000 ms. That is because when encounter collision, separate chaining will go into that specific position arrayList and search in depth of that arrayList. It will directly append at the end if not find same pixels. While the probing empty spot in separate chaining cost similar to that of quadratic probing, the insertion of separate chaining cost is always O(1). However, for quadratic probing, whenever the hash table is more than half full, we need to rehash the entire table to a bigger size. This rehash process is costly and will produce more collisions. As we discussed, the number of collisions for quadratic probing is significantly bigger than that of separate chaining.

Separate chaining have an advantage of not requiring rehash. This saves a significant amount of runtime and number of collisions. Separate chaining method never fills up the table. We can add more elements to the chain with fixed initial table size. However quadratic probing holds a dynamic table size. That means separate chaining is less sensitive to the choice of hash function and spread of data key. Quadratic probing can outperform separate chaining with the knowledge of spread of data and hash function if carefully planned. Table size will expand as more elements being hashed to the table. That doesn't mean quadratic probing do not have advantage. Quadratic probing has the advantage of storing elements in the same table rather than a complicated 2-D table. Quadratic probing utilize memory more efficiently. Even if nothing hash to a position, there's still possibilities that that spot will be used after probing. However, with separate chaining, specific spot will never be utilized if nothing is going to hash there.