Problem 10.1

a)

double hashing function is h(k, i) = (h1(k)+i\*h2(k)) mod m

In this sequence of <3, 10, 2, 4>, no collision has taken place, which is just h1(k) mod m.

Also, h1(k) = k mod 5 which is m, when collision happen, i is 0, hash function equals k mod 5.

白板上的文字

描述已自动生成

b)

hash\_table.cpp

h’ function = key % maxSize;

Idea of linear probing is to increment 1 if the current Hash-index is occupied. This means when h’ function = key % (a number smaller than maxSize), the number of collisions will not be minimal, because this function can only return a Hash-index at most to the number being moduled. Take h’ = function = key % 7 as an example, this function will only return up to index 6, and the index between 6 and maxSize will have to start at most from 6, which will eventually cost a lot of collision.

To prove the above statement, I have an int “Collison” in HashTable class which records how many collisions happen when a HashTable is full inserted. To be fair I used rand() with time as seed in the beginning for the keys. Below are collisions recorded when m=10.



This proves set h’ function as key % maxSize is well-suited.

Problem 10.2

a)

counterexample:

activity: A1 A2 A3 sort with shortest duration A2 A3 A1

start: 1 4 5 4 5 1

finish: 5 6 8 6 8 5

duration: 5 2 3 2 3 5

greedy choice of selecting the activity with shortest duration: **A2** one activity.

globally optimal solution: **A1->A3** two activities.

In this counterexample, if the shortest activity is in between and crosses the boundaries of the neighbor activities, it will fail at producing the globally optimal solution.