PROJECT-3 REPORT

Analysis of Algorithms

Emre KÖSE – 150130037
01.12.2017

BLG 335E - Analysis of Algorithms I, Fall 2017
Project 3 – Report

A) The running time tables of the procedures with the data set that contains **Book_Character** objects these are in size M = 131,070 elements.

Data Procedure/ Procedure	Block insertion
Dictionary	0.649 s
List	0.021 s

Data Structure/ Procedure	Block look-up
Dictionary	0.712 s
List	99.032 s

B) If we examine the tables which are given in above, we see the dictionary method is a superior procedure to list in the look-up case. This is because the array structure is used in dictionary method and if we look into the implementation, firstly the possible indexes are searched with the help of key values and this facilitates to find the wanted element and look-up times are resulting in very short times. In the other hand, list structure has a little more speed in the insertion case because there is no any different behavior in terms of adding, this just inserts to back of list, but in dictionary adding should be in control with using the key values and hashing function.

C)

- Average number of Collisions (first 1000 items) | 0.002s
- Average number of Collisions (first 10000 items) | 0.0337s
- Average number of Collisions (first 100000 items) | 0.80516s
- Average number of Collisions (first 131070 items) | 56.6761s

These results were taken from project's average number of collisions values for inserting method of dictionary. Therefore, we cannot say the rate is increasing linearly or exponentially because this is a little related with hashing and probing methods. If the indexes which are found with using of keys point out the similar probed results, the collision number will increase because the system cannot find the empty region for inserting and the rate of collision will increase, too. In addition, with the increasing of the inserted items, there will be less place in hash table and the hashing and probing methods will work hard to find appropriate index and maybe they cannot find after the limited number of tries. This is known as a load factor which equal to n/k (n: the number of entries, k: the number of buckets) and with the increasing of this factor the hash table becomes slower, and it may even fail to find appropriate index.

D) The worst case for looking up a key for my dictionary is O(n) because, in look-up method my probing function works for maximally M(the size of table) times. If the searching of key fails M times, the worst case occurs. In the project, we want M objects to put into the hash table if this worst case occurs in all cases, at last the time complexity will be the sum of Ms and corresponds to M^2 which corresponds to $O(n^2)$ time complexity. This situation can realize with completely irrational hash function. Therefore, probing will not help to rally the situation.