Hash tables

Nariman Haidar

October 2022

1 Introduction:

In this assignment we are going to learn how to organize a setup data that should be accessible with a given key. We are going to do that by reading a zip file and inserting each item in an array of entries by by using the hash map.To achieve the purpose of the task, we will apply a number of different methods, such as line search, binary search search with a given key and by applying a hash function.

2 Method:

A table of zip codes:

In this section of assignment we are going to use the hash table to read the file and insert each item in an array of entries. Each entry consists of the zip code, the name of the area and the population. To implement this code first I write a class zip that have an array of data and int max then write constructor to this class, then we are going to make a node class with the properties: String code, String name and Integer pop, then write constructor to this class. Then we are going to start with insert method that are going to insert all codes that we have with there keys to an array of data and it takes integer code and node entry like a parameter then to insert keys we are going to take the mod of the code per max size then we set the node nxt the the array of data then while nxt is not null and if the code equals to nxt code then set the nxt to the next and then break then set the previous is the nxt then the set the next to the next and if the previous is not null then set the previous nxt to the entry and set the entry next to the nxt . Then I write the method binary search to make a search through all zip codes looking for a specific entry and first I start with binary method that takes a string zip like a parameter and then set m to the max and while true compare and then return the string that I want to search. Then I write the linear search to make a search through all zip codes looking for a specific entry that takes a string zip like a parameter return the string that I want to search. Then I write a benchmark that searches for ”111 15” and ”994 99” and the results is when I measure the time I realized that the linear search is better when it is in the start in the list that we have but when it is in the mid or in the . Then I convert all zip codes to Integers before creating the entries. Then I make the binary method and linear method take the integer code like a parameter instead the string. Then I write a benchmark that searches for "111 15" and "994 99". We ca see all the result down and it is the time measurement when searching i two different way the first way by using the binary search and the other way by using the linear search by using postcodes 11115 and 98499. When we did this search with string we realized that in linear search it takes short time when it is at the beginning in first of list of zip code and longer for binary search because the binary search is so much better just when it is so long list and we are going to search in a deep in the list, but when it is searching in the last of the list of zip code in binary search takes time shorter than linear search. Because it the binary search is so much better when it is so long list of codes because in the binary search like we learned previously that will split the list into two parts and compare them all search step, while before it lines search so it will immediately find it postcode because you search through each index. Then the other table is to measure time when we changed the data type from String to Integer and it is so much better than the previous and that is because its easier to deal with integers than stringer because it takes time to compare the strings.

table

code	String Linear	String Binary
11115	470	933
98499	10994	1196

Table 1: This table to measure time integer linear and integer binary.

table

code	Integer Linear	Integer Binary
11115	396	211
98499	1938	157

Table 2: This table to measure time string linear and string binary.

Use key as index:

In this section we are going to search by using the key of integers because in our zip every code has a key and we are going to lookup the code by using this key. So I started to write the lookup method, so first set all the data in an array by they keys with using mod of the max of size. Then I write the method compare by key to compare all keys with the the key that I want to lookup it until to find it. The result in the table down when we want to lookup 11115 and 98499. If we compare these results with binary search so it takes a less time to lookup and that is because the lookup method using the postcode as index but in binary search it is going to divide the list into two parts.

table

code	Lookup	Binary
11115	170	97
98499	53	95

Table 3: This table to measure time lookup and binary.

size matters:

In the previously lookup it save time but not the memory so to solve this problem we are going to use the hash function. The hash function is going to transforms a key to an index. To defining a hash function is to simply take the key modulo some value m in hope that the indexes should be fairly unique. If we have two keys that maps to the same index then we have collision that is something that we need to handle (and will be able to handle) but the fewer collisions the better. But the hash function has a disadvantage when it comes to the collision and that because increased the number of the collision when placed in them differently calculated indices with modulo operation and that when two postcodes get the same index and we can see that in the table down and we can see its better we we use the prime numbers compared with the other odd numbers or the even numbers. To solve the collections problem there is one simple solution is to have an array of buckets, each bucket holds a small set of elements that all have the same hash value and to implement this method, it is going to create a pointer to points to the head of a linked list of those index that are not empty, and if a several data comes in the same index so that we are going to use the hash method in the given zip code to find the list when that index points to to see if the zip code is there.

table

slightly better?:

To do this better of version of the bucket implementation we are going to use the array it self without indirection to separate buckets and this is going to

mode	**Number of collections**
11111	2633
14112	2157
18911	1749
19531	1563

Table 4: This table to measure number of collections.

start with the hashed index and then move forward in the array to find the right entry. We can see the results in the table down that shows the number of steps that one had to move to find the given post code. We can see that when the mod is small the number of steps to find the given post code is bigger than when the mod is larger then the steps is fewer.

table

mod	**11115**	**98499**
11111	11	279
14112	3	5
18911	9	13
19531	1	2

Table 5: This table shows the number of steps to find the given post codes.