**Motivation-Why dictionary?**

Data structures and algorithms are widely used in almost every modern application that is being made. As a project, we have to choose application on which we can implement both data structure and algorithm and to solve the real world problem and try to make it efficient. So, we choose “Dictionary” as ours project. Today, pocket dictionaries are widely as an application in mobile phone which makes it easy to have a look at anytime anywhere. Making a dictionary efficient is a great task to achieve and be proud of as a student of computer science. So, we started our work on project without wasting any time.

**Problem Statement:**

There are a lot of dictionary apps but are slow to search words. Our target is to make efficient dictionary which can run in constant time complexity. We want to add one more that is if user do not find word that is being searched, then our dictionary will allow him to add word and its meaning and sort it at its place.

**Solution:**

* Working:

Dictionary is like a collection of words in one place so that you can access it at any time. Our file(dictionary.csv) stores words and its meanings in English language. When user will enter any word to search its meaning, our algorithm will first convert the word and generate the code to will the exact location of the word in the file and then it will read its meaning from file and give it to the user. If the word is not found then our algorithm will allow user to enter word in the dictionary and also its meaning.

* Data Structure:

At first, we wanted to find the constant time complexity so the user can search in real time. But we realized that if we will achieve constant time complexity, then we will have to use more memory space and it will be not much efficient we have to care about both, time and memory. So, we decided to reach almost the constant time complexity. That’s why, we used **HASH TABLES**.

When the user will enter any word, algorithm will use **hash function** present in the string library ( String.,hash() ). The array of pointers is used having size of 32 initially. Array points to the linked list which has the location of the word stored in the file. So, the hash function is used to generate the “almost unique” code. Then algorithm will take modulus with the size of array which is initially 32.

Example:

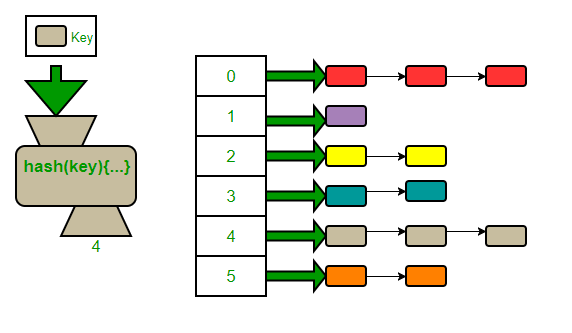
🡪Word=power

🡪The generated code is = 28393893

🡪28393893(hash code) **%**  32(array size) = **25 = key**.

🡪Now we will go to the array[25] which will point to the location of word in file.

🡪In last, just retrieve the word and read its meaning and show to the user.

If the hash code collides with the other hash code, then it will be stored in another node which is pointed by node that is already present in the array at that key.

Example is illustrated in the figure.

* Time complexity:

Our target was to achieve the constant time complexity but due to memory space consumption, we have to compromise a little to balance both of the factors. So, the time complexity at average case is almost **constant**, but we have to look at the worst case.

Our array has initial size of 32 and it can store 32 nodes in one index pointing to the array. If the collision of keys increases and the node links exceeds 32, then our array will be resized to double its size (i-e: 64). Hence the max number of nodes will be used in one array index is 32.

Therefore the worst time complexity is O(1) for the array by using hash table and the O(n) for using linked lists.

Big-Oh = O(1) + O(n)

But, as the maximum size on n is 32 and will not exceeds the number 32. Hence our time complexity is near constant in worst case also because for processor, it is not take even a millisecond to search between 32 nodes.

* Conclusion:

The problem is solved by using hash tables to make an efficient dictionary. We were not able to achieve the exact target of getting the constant time complexity due to taking care of memory consumption. But we have reached almost near to our target and the efficient dictionary is ready to in every one’s pocket. The outputs and source code and its explanation are also given below.