Dictionary

Dictionary is an abstract data type (ADT). It is a general purpose data structure used for storing a group of objects. It is a key value store. Every entry has a key and a value attached to it. The key is used to find the entry in the dictionary and the value is displayed attached to the key. However, the key in the dictionary must be unique otherwise it will lead to an ambiguity of data that will often produce false results or in some languages it overwrite the duplicate existing value for that key.

It is basically the abstract model of a database. Take into consideration the bank accounts. The key here can be the account number as it always be the unique thing and no two entity will have the same account number. The value associated with this account number will be the name of the account holder, the current balance, the address, the other contact details and the type of account they hold. However these values can be same for two or more persons but the key, in this case the account number can never be same.

There is a difference between key not existing in a dictionary and a key having a null value. A person may or may not have a telephone number but that doesn't mean that his key, the account number is not existing in the dictionary or the bank database.

Items in dictionary are unordered so iteration over them will result in an arbitrary order return of the values. And yes most of the programming languages support the iteration over the key or value in a dictionary.

The basic operations done on a dictionary are:-

1. Search item -> O(1)
2. Add item -> O(1)
3. Delete item -> O(1)
4. List items -> O(n)
5. Copy all the items -> O(n)

However, The worst case for all of the above operations will have time complexities of O(n).

Space complexity of a dictionary is O(N).

Implementing dictionary in python:

1. Initializing a dictionary:

```
>>>dict = {}
```

2. Inserting item in dictionary:

```
>>>dict[‘a’] = 1
>>>dict[‘b’]=10
>>>dict[‘c’ ]=20
```

3.Searching an item in dictionary:

```
>>> dict[‘a’]
1
```

4. Listing the dictionary:

```
>>>dict
{'a': 1, 'b': 10, 'c': 20}
```

5. Deleting an item from dictionary:

```
>>>del dict[‘b’]
>>>dict
{'a': 1, 'c': 20}
```

6. Copying the dictionary

```
>>>dict1=dict.copy()
>>>dict1
 {'a': 1, 'c': 20}
```

Python dictionaries are implemented as resizable hash tables. Compared to B-trees, this gives better performance for lookup (the most common operation by far) under most circumstances, and the implementation is simpler.

Dictionaries work by computing a hash code for each key stored in the dictionary using the **hash()** built-in function. The hash code varies widely depending on the key; for example, “Python” hashes to -539294296 while “python”, a string that differs by a single bit, hashes to 1142331976. The hash code is then used to calculate a location in an internal array where the value will be stored. Assuming that you’re storing keys that all have different hash values, this means that dictionaries take constant time – O(1), in computer science notation – to retrieve a key. It also means that no sorted order of the keys is maintained, and traversing the array as the .keys() and .items() do will output the dictionary’s content in some arbitrary jumbled order. The hash computation is complicated by the possibility that some members of the list may be unhashable and also by the possibility of arithmetic overflow.