Andy Kuo HW3
CS377C 5/1/01

HASH TABLE**

There are many computer applications that require storage of a number of OBJECTS. In addition, the following operations are required on this set of objects: INSERT, SEARCH, and DELETE. That is, it must be possible to add, find, and remove objects from this set. As always, EFFICIENCY is a key concern. One particularly efficient way of accomplishing this is described in the pattern below.

* * *

Many computer applications require efficient storage, insertion, search, and deletion of individual elements from a set.

A hash table is an efficient solution to this problem. Essentially, a hash table is a collection of key-value pairs. The *value* of the pair is the element or object you want to store; it can be anything: a number, a word, and even complex objects. The *key* of each pair is something uniquely associated with the corresponding value. That is, given any key, it is possible to tell exactly what value it corresponds to.

Even though this description sounds technical, you actually use a hash table in the real world quite often: the dictionary. The values, or objects being stored, are definitions. The keys, the accessors to those objects, are the words themselves. Thus, given any word, it is possible to locate its associated definition. A hash table is simply a generalization of this principle. As we continue to examine the properties of hash table, I will show how the principles apply in the dictionary example.

How do the keys allow access to the values? That is where “hashing” comes in. Given an input, a *hashing function* manipulates it somehow, and produces a unique output. In the case of computer programs, the key is hashed, and the resulting output is a unique index number that tells the program where to find the associated value. There is no direct corollary with the real-world dictionary, but we can make one up. Suppose along with every dictionary came a device. You can enter into the device the word you are looking for, and it will tell you exactly what page and line number to find it. That device is the equivalent of the hashing function.

All hash tables share the following properties:

1. Hash tables are most effective when the number of keys actually stored is small relative to the total number of possible keys, for various technical reasons.

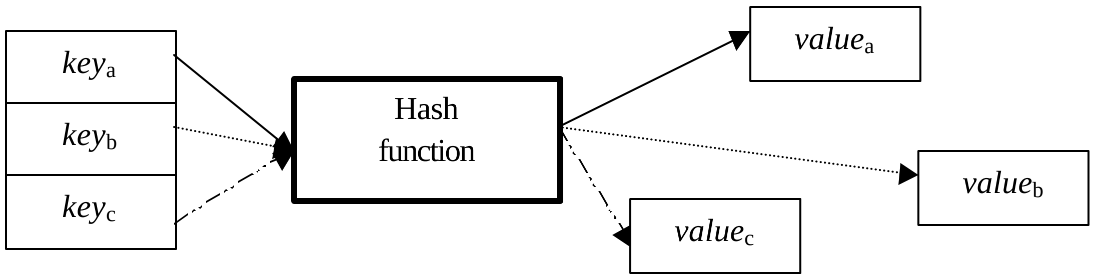
   The number of English words in the dictionary is much smaller than the total number of possible combinations of letters.

2.	Ideally, the hash function only associates one value with each key. However, it is possible to get away with associating a few values with a few keys.	*Many words in the English dictionary have only one definition, although there are also many words with multiple definitions.*
3.	Hashing functions should be fast, for obvious reasons.	*The hypothetical device we invented before should be fast.*

As stated above, the ideal case is when one exactly one value is associated with each key. In practice, this is fairly difficult to achieve. When more than one value is associated with each key, the result is a *collision*. There are a number of techniques, described in other patterns, of dealing with collisions.

Therefore:

Use hash tables to facilitate quick insertion, search, and deletion of values from a set. Values can be anything: numbers, words, or even complex objects. Each value should be uniquely identified by a key. The key in theory can be anything, but in practice is usually something fairly simple to manipulate, such as a number or a string. A hashing function, when applied to the key, will yield the location of the value. In addition, the function will do so quickly i.e. does not require a prohibitive amount of calculation. Finally, the hashing function must be deterministic. That is, given the same key, it should always yield the same location of the associated value.



* * *

The particulars of implementing hash tables are not described here. Choose good HASHING FUNCTIONS. To deal with collisions, consider using CHAINING…