**Task 1**

1. (a) Table 1 shows a list of items that are to be stored in the hash table, Table 2. The hashing algorithm uses the remainder method:

address = value mod 11

In the case of a collision, the item is stored in the next free space.

Start storing items in the hash table (Table 2). Use a different colour for storing collisions.

How many collisions have occurred after storing 7 items?

When all 11 items have been stored, how many items are in the correct place, without having caused a collision?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 28 | 15 | 47 | 70 | 33 | 27 | 55 | 81 | 66 | 9 | 38 |

*Table 1*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Slots** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |

*Table 2*

(b) Now store the first 10 items from Table 1 in the second hash table, Table 3, using a mod 10 hashing method. (This table has only 10 spaces.) This time, in the event of a collision, use a “skip” factor of 2. e.g. If an item cannot be placed in the correct location, say 5, look in locations 7, 9, 1 etc until a free space is found.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Slots** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  |  |  |  |  |  |  |  |  |  |  |

*Table 3*

What happens when you try to store **81** in the hash table? Why does this occur? Can you suggest how this situation could be avoided?

How can collisions be minimised?

**Task 2**

2. (a) The mid-square hashing method is to be used to find addresses to store items.

Using this method, the number to be stored is squared and the middle 2 digits are then divided by the table size (11 in this case) to find an address. In the case of an odd number of digits, the digit before the middle item, and the middle item are used. Collisions are handled by storing the item in the next available free space.

Show where the following items will be stored in the hash table, Table 5.

456, 37, 3, 12, 5875

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Slots** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 3 | 12 | 5875 | 37 |  |  |  | 456 |  |  |  |

*Table 5*

(b) A folding method of hashing divides the numeric key into a number of 2-digit integers (there may be an odd digit at the end).

These 2-digit numbers, and the final digit if there is one, are then added together to give a number x. The address in the hash table (of size 5) is calculated as x mod 5.

Use the folding method to store the following telephone numbers in a table of size 5.

01473 664987, 07989 342126

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Slots** | 0 | 1 | 2 | 3 | 4 |
|  |  |  |  | 01473 664987 | 07989 342126 |

*Table 6*

(c) Part of an ASCII table is shown below. Show the addresses at which the words “CAN”, “FIND” and “CALM” would be stored in a table of size 5, using the method of adding the ordinal values for each letter and taking the remainder when divided by 5. Collisions are stored in the next free space.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** |
| 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| CAN | CALM |  |  | FIND |

(d) Describe how the word CALM is located in the table when searching.

The algorithm would add the ascii values of each letter in the word then MOD 5. Since CALM Mod 5 is 0 it checks index 0. Since CALM is not at index 0 it will then check the next index along. Since Calm is in the next index, the algorithm has found CALM.

(e) Suppose CAN is deleted from the table. How is CALM located?

CAN will be replaced by the phrase “deleted” or similar and if the algorithm sees that, it will know to visit the next index along

(f) Suggest a solution to this problem.

CAN will be replaced by the phrase “deleted” or similar and if the algorithm sees that, it will know to visit the next index along

**Task 3 Mowing task**

3. The nursery rhyme ‘One man went to mow’ has been analysed and a dictionary named ‘**mow’** created to show the frequency of each word.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Key** | one | man | went | to | mow | a | meadow | and | his | dog | two | three | four | five | men |
| **Value** | 6 | 6 | 16 | 17 | 17 | 12 | 12 | 5 | 5 | 5 | 5 | 4 | 3 | 2 | 14 |

(a) Why are the words in the dictionary not in alphabetical key order?

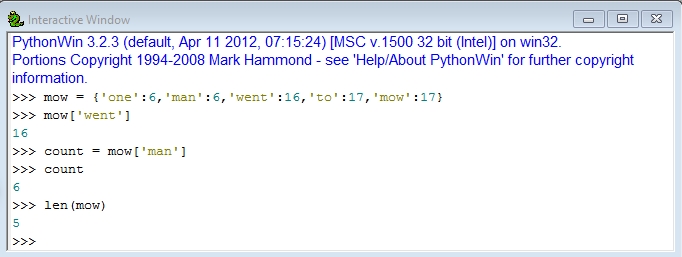
(b) Describe briefly how this dictionary may have been created

(c) Describe how to find the number of times the word ‘dog’ appears in the rhyme.

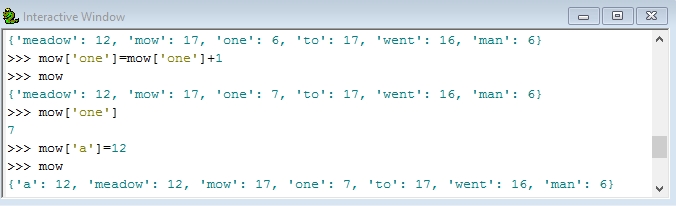
(d) Provide the outputs for the following operations as performed on the value of the dictionary ‘mow’ as shown above.

|  |  |
| --- | --- |
| **Operation** | **Output** |
| mow.length() |  |
| ‘meadow’ in mow |  |
| mow [‘five’] |  |

The following screenshots show an interactive Python session to illustrate how the dictionary works:



You can add a new element to the dictionary, or update the value associated with the key.



To check whether an element is in a dictionary, use the ‘in’ operator:

