# Worksheet 3 Hash tables and dictionaries Answers

**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. Use a different colour for storing collisions.

 How many collisions have occurred after storing 7 items? 3

 When all 11 items have been stored, how many items are in the correct place, without having caused a collision? 5

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

*Table 1*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 33 | 55 | 66 | 47 | 15 | 70 | 28 | 27 | 81 | 9 | 38 |

**Slots**

*Table 2*

 (b) Now store the first eight items from Table 1 in the hash table, Table 3. 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.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 33 |  | 55 | 47 | 15 | 27 | 28 |  | 70 |  |

**Slots**

*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?

 81 cannot be stored because not all addresses can be generated with a skip factor of 2, and a table size of 10. This arises because 10 is divisible by 2, and it could be avoided by using a table size which is a prime number.

 How can collisions be minimised?

 By having a table size considerably larger than the number of items to be stored. Also, a different hashing algorithm may give a better spread.

**Task 2**

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

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

**Slots**

*Table 5*

 4562 = 207936 Mid-square = 79 Address = 79 mod 11 = 2

 372 = 1369 Mid-square = 36 Address = 36 mod 11 = 3

 32 = 9 Mid-square = 9 Address = 9 mod 11 = 9

 122 = 144 Midsquare = 14 Address = 14 mod 11 = 3

 58752 = 34515625 Midsquare = 15 Address = 15 mod 11 = 4

(b) 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*

 01 + 47 + 36 +64 + 98 + 7 = 253 Address = 253 mod 5 = 3

 07 +98 + 93 + 42 + 12 + 6 = 258 Address = 258 mod 5 = 3

(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 |

Table 6

CAN 🡪 (67 + 65 + 78) 210 mod 5 = 0 FIND 🡪 (70 + 73 + 78 + 68) 289 mod 5 = 4

CALM 🡪 (67 + 65 + 76 + 77) 285 mod 5 = 0 Collision occurs so store in next free space.

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

 The hashing algorithm is applied to CALM and the location is full. The next free space is located and CALM is stored there.

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

 The hashing algorithm is applied to CALM and the space is found to be empty so CALM will be reported as NOT FOUND.

(f) Suggest a solution to this problem.

 Items are not physically removed from the table. A placeholder indicating that the slot for CAN was filled and has now been freed up is written to the slot. When searching for CALM, address 0 will be treated as filled and the search will continue until a free space is found.

**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?

 Because the keys are used as input to a hashing algorithm which determines the address where they are stored. So, the {key:value} pairs are stored at the address generated by the hashing algorithm, which will not be alphabetical.

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

 The text needs to be stored as a list of words. Each word in the list is read. Search the dictionary to see if the word is already there. If not, add the word as key, value 1. If it is there, increment the value.

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

 Index the dictionary by the key term ‘dog’. For example, count 🡨 mow[‘dog’]

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

|  |  |
| --- | --- |
| **Operation** | **Output** |
| mow.length() | 15 |
| ‘meadow’ in mow | True |
| mow[‘five’] | 2 |

 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:

 