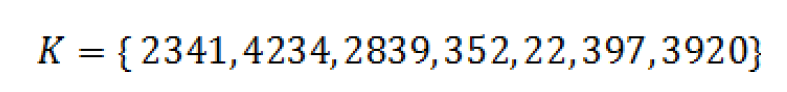
Question 1.



Hash function:

The hash function is used to compute the index based on the key.

Index of 2341 = 2341 % 7 = 3

Index of 4234 = 4234 % 7 = 6

Index of 2839 = 2839 % 7 = 4

Index of 352 = 352 % 7 = 2

Index of 22 = 22 % 7 = 1

Index of 397 = 397 % 7 = 5

Index of 3920 = 3920 % 7 = 0

|  |  |
| --- | --- |
| **Index** | **Key** |
| 0 | 3920 |
| 1 | 22 |
| 2 | 352 |
| 3 | 2341 |
| 4 | 2839 |
| 5 | 397 |
| 6 | 4234 |

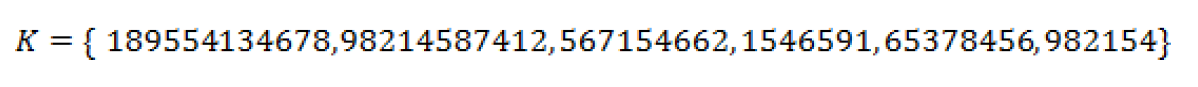
Question 2:



Middle Square method: Square the key, and get the middle 3 digits

|  |  |  |
| --- | --- | --- |
| **Index** | **Key** | **Key Squared** |
| 908 | 1221 | 1490841 |
| 539 | 2134 | 4553956 |
| 805 | 2254 | 5080516 |
| 123 | 2452 | 6012304 |
| 494 | 2941 | 8649481 |
| 0 | 1000 | 1000000 |
| 118 | 1874 | 3511876 |

Question 3:



1. The hash table ranges from 0 to 999, meaning we need to guarantee that there are at least 3 digits in our fold. This means each group should have **3 digits**.
2. Compute the addresses using the shift folding:

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **Key** | **Groups** | **Groups Sum** |
| 555 | 189554134678 | 189, 554, 134, 678 | 1555 |
| 013 | 98214587412 | 982, 145, 874, 12 | 2013 |
| 383 | 567154662 | 567, 154, 662 | 1383 |
| 814 | 1546591 | 154, 659, 1 | 814 |
| 493 | 65378456 | 653, 784, 56 | 1493 |
| 136 | 982154 | 982, 154 | 1136 |

1. Folding at the boundaries

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **Key** | **Groups** | **Groups Sum** |
| 654 | 189554134678 | 189, **455**, 134, **876** | 1654 |
| 418 | 98214587412 | 982, **541**, 874, **21** | 2418 |
| 680 | 567154662 | 567, **451**, 662 | 1680 |
| 111 | 1546591 | 154, **956**, 1 | 1111 |
| 196 | 65378456 | 653, **487**, 56 | 1196 |
| 433 | 982154 | 982, **451** | 1433 |

1. Selecting 6th,4th and 1st digits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Key** | **6th Digit** | **4th Digit** | **1st Digit** |
| 451 | 189554134678 | 4 | 5 | 1 |
| 519 | 98214587412 | 5 | 1 | 9 |
| 415 | 567154662 | 4 | 1 | 5 |
| 961 | 1546591 | 9 | 6 | 1 |
| 476 | 65378456 | 4 | 7 | 6 |
| 419 | 982154 | 4 | 1 | 9 |

Question 4:

1. Behaviour of hash table T

|  |  |
| --- | --- |
| **Index** | **Key** |
| 0 | 161 |
| 1 | 911 |
| 2 | 421 |
| 3 | 374 |
| 4 | 227 |
| 5 | 83 |
| 6 | 1091 |

Solve collisions using linear probing. When a collision happens, move on to the next position, until an empty space is found.

Apply hash function: Find the modulus of each number by 7

374 % 7 = 3. T[3] is empty, so T[3] = 374.

1091 % 7 = 6. T[6] is empty, so T[6] = 1091.

911 % 7 = 1. T[1] is empty, so T[1] = 911.

227 % 7 = 3. T[3] not empty T[4] is empty, so T[4] = 227.

421 % 7 = 1. T[1] not empty T[2] is empty, so T[2] = 421.

161 % 7 = 0. T[0] is empty, T[0] = 161.

83 % 7 = 6. T[6] not empty T[0] not empty T[1] T[2] T[4] not empty T[5] is empty, so T[5] = 83.

We see that the Big-O class is O(n) – worst case. However, we see that in optimal case, inserting can be done in constant time, or Big class of .

Look up:

1091 % 7 = 6. T[6] == 1091, so return 6.

83 % 7 = 6. T[6] != 83 T[0] != 83 T[1] T[2] T[4] != 83 T[5] == 83, so return 5.

1092 % 7 = 0. T[0] != 1092 T[1] != 1092 T[2] T[5] T[6], since all 7 elements have been visited, so return -1 (not found).

Delete:

911 % 7 = 1. T[1] == 911, so valid delete, and set T[1] = “DEL”

|  |  |
| --- | --- |
| **Index** | **Key** |
| 0 | 161 |
| 1 | DEL |
| 2 | 421 |
| 3 | 374 |
| 4 | 227 |
| 5 | 83 |
| 6 | 1091 |

Look up 83.

83 % 7 = 6. T[6] != 83 T[0] != 83 T[1] T[2] T[4] != 83 T[5] == 83, so return 5.

1. Find the following:
2. The load factor: , so overflows.
3. Average number of probes to find valid value:

Calculate the number of probes to lookup each element.

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **Key** | **Probes** | **Number of Probes** |
| 0 | 161 | 161 % 7 = 0. T[0] == 161 | 1 |
| 1 | DEL |  |  |
| 2 | 421 | 421 % 7 = 1. T[1] != 421 T[2] == 421 | 2 |
| 3 | 374 | 374 % 7 = 3. T[3] == 374 | 1 |
| 4 | 227 | 227 % 7 = 3. T[3] != 227 T[4] == 227 | 2 |
| 5 | 83 | 83 % 7 = 6. T[6] != 83 T[0] T[1] T[2] T[3] T[4] T[5] == 83 | 7 |
| 6 | 1091 | 1091 % 7 = 6. T[6] == 1091 | 1 |

Total: 1 + 2 + 1 + 2 + 7 + 1 = 14 probes.

Number of keys: 6 keys (since T[1] is deleted, it doesn’t count).

Average

1. Average number of probes to find invalid value: Since to know that a value does not exist, we need to iterate through the whole table, so the number of probes is always the same as the table size 7 probes.
2. Separate chaining. Each key is a linked list.

|  |  |
| --- | --- |
| **Index** | **Key** |
| 0 | 161 |
| 1 | 911 | 421 | |
| 2 |  |
| 3 | 374 | 227 |
| 4 |  |
| 5 |  |
| 6 | 1091 | 83 | |

Inserting…

374 % 7 = 3. T[3] is empty, so T[3] = 374.

1091 % 7 = 6. T[6] is empty, so T[6] = 1091.

911 % 7 = 1. T[1] is empty, so T[1] = 911.

227 % 7 = 3. T[3] not empty, so T[3].Next = 227

421 % 7 = 1. T[1] not empty, so T[1].Next = 421

161 % 7 = 0. T[0] is empty, T[0] = 161.

83 % 7 = 6. T[6] not empty, so T[6].Next = 83

Look up…

1091 % 7 = 6. T[6] == 1091, so return 6.

83 % 7 = 6. T[6] != 83, so move on to T[6].Next == 83, so return 6.

1092 % 7 = 0. T[0] != 1092, so move on to T[0].Next == null, so return -1.

Delete 911

911 % 7 = 1. T[1] == 911, so T[1] = T[1].Next.

Question 5.

b. Quadratic probing.

If a collision happens at index , move on to , then , then until find an empty space.

|  |  |
| --- | --- |
| Index | Key |
| 0 | 55 |
| 1 | 12 |
| 2 | 23 |
| 3 | 42 |
| 4 |  |
| 5 | 38 |
| 6 | 50 |
| 7 | 29 |
| 8 | 16 |
| 9 | 31 |
| 10 | 17 |

38 % 11 = 5

31 % 11 = 9

29 % 11 = 7

50 % 11 = 6

42 % 11 = 5. T[5] occupied T[5 + 1] = T[6] occupied T[5 + 22] = T[9] occupied T[5 + 32] = T[14], wrap to T[3] empty. T[3] = 42

12 % 11 = 1.

17 % 11 = 6. T[6] T[7] T[6 + 22] = T[10]. T[10] = 17

23 % 11 = 1. T[1] T[2]. T[2] = 23

16 % 11 = 5. T[5] T[6] T[9] T[14] wrap to T[3] T[21] wrap to T[10] T[30] wrap to T[8]. T[8] = 16

55 % 11 = 0. T[0] = 55