Hash Tables with Open Addressing

Description In this assignment you are requested to implement insert, search, and delete operations for an **open-addressing hash table with double hashing**. Create an empty hash table of size $m = 13$. Each integer of the input will be a key that you should insert into the hash table. Use the double hashing function $h(k, i) = (h_1(k) + ih_2(k)) \mod 13$ where $h_1(k) = k \mod 13$ and $h_2(k) = 1 + (k \mod 11)$. The input terminates when the key -1 is read (such a key must not be inserted in the hash table). At that point, print the content of the hash table to the screen (see sample input/output below for the printing format). Then, read integers from the input until the number -2 is found (do not process that number). For each number inputted, print the index of the element in the hash table. If the number is not inside the hash table, print `NOT_FOUND`. Finally, read integers from the input until the number -3 is found (do not process that number). For each number inputted, delete it from the hash table (note that the integer might not be in the table). Once the integer -3 is found, print the hash table.

Testcase number to use with `grade_me`: 46.

Bonus questions: sorting in linear time

1. Implement the **Counting Sort** algorithm. You can assume that the input will be comprised of integers only and will not have any negative values. Your algorithm should dynamically figure out the maximum integer in the sequence (variable k in the algorithm in the book). This is the same implementation as presented in section 8.2 of the book.
   Testcase number to use with `grade_me`: 40.

2. Implement the **Radix Sort** algorithm. You can only assume that the input will consist of nonnegative integers, each of which is stored in binary format in the computer with word size w (e.g. $w = 32$). As a first attempt, consider radix sorting on each binary digit. You can then have your program have a user argument r indicating the number of bits that each digit will have. For example, the number 123 (in decimal) can be considered as 1111011 in binary ($r = 1$: `1-1-1-1-0-1-1`), 0173 in octal ($r = 3$: `1-111-011`), `0x7B` in hexadecimal ($r = 4$: `111-1011`), etc.. *Hint*: modify **Counting Sort** to use the window of r bits in the input integer that corresponds to the dth digit of r bits.
   Testcase number to use with `grade_me`: 50.

In both cases, the input may contain multiple cases, each starting with an integer number which indicates the number of elements to be sorted, followed by the elements (one per line). A sequence of 0 elements indicates the end of the input, and produces no output. Output the sorted sequence one element per line. Do not insert spaces at the beginning or at the end of any element.

Examples of input and output

Input for Test Case 46

```
79
69
98
72
14
50
0
-1
69
70
32
50
-2
5
69
98
1
-3
```

Output for Test Case 46

```
0
79


69
14

98
```

```
72


50

4
NOT_FOUND
NOT_FOUND
11
0
79



14


72


50
```

Input for Test Case 40

```
6
1
5
2
39
21
17
0
```

Output for Test Case 40

```
1
2
5
17
21
39
```

Input for Test Case 50

```
6
100
5
2121
39
201
11117
0
```

Output for Test Case 50

```
5
39
100
201
2121
11117
```

Your solutions Before leaving the lab, submit a zipped tar archive of your program through the assignments page of UCMCROPS. Please use your UCMNetID as the filename for the zipped tar archive. Be careful since UCMCROPS strictly enforces the assignment deadlines (deadlines will be every lab date at either 4:20pm or 7:20pm depending on your lab session.).