CST 370 – Homework 4

Due: **2:00pm on April 9, 2019**

Final Submission Deadline: **2:00pm on April 16, 2019**

Changelog:

- 3/28/19: Added starter code links to Huffman Codes problem.
- 3/28/19: Clarified language in the constraints section of Huffman Codes.

About

For this 2-week homework assignment, you be implementing tries, huffman codes, and radix sorts.

Submissions

https://www.hackerrank.com/contests/cst370-s19-hw4/challenges

The problems for this homework will be hosted on HackerRank where automated tests will run. You may submit code on HackerRank as many times as you'd like. Once you are comfortable with your code, you can copy the submission link for that problem and include it in the ilearn assignment for me to grade. **I will only grade hackerrank links submitted on iLearn.** iLearn has videos describing how to get a submission link.

After the due date, you may continue working on problems on HackerRank. If you submitted something at the initial deadline, you may submit HackerRank links again on the iLearn assignment for resubmissions. Resubmissions will be assessed a 10% deduction.

Scoring

Your score on HackerRank or Kattis will give you *an approximation* of what your grade will be; however, I will read your code and determine your final grade myself, as detailed on the syllabus.

Problems

1	Tries	15 points
2	Huffman codes	20 points
3	Radix Sort	15 points

This page is intentionally left blank.

1. Tries (15 points)

https://www.hackerrank.com/contests/cst370-s19-hw4/challenges/hw4-tries

For this problem you will implement a trie (prefix tree) and use it to autocomplete words. In C++, the trie node and class definitions might look like:

```
struct TrieNode {
    bool isLeaf;
    vector<TrieNode*> letters;
};
class Trie {
    TrieNode* root;
};
```

You will need to implement the following operations:

`lookup(string)` which returns a boolean indicating whether given word was found.

`insert(string)` which inserts the given word into the trie.

`remove(string)` which removes the given word from the trie (really removes it, not just changing the flag).

`info()` which prints out the number of nodes and number of words in the trie.

`alphabetical()` which prints out the words in the trie in alphabetical order.

`autocomplete(prefix, k)` which prints alphabetically all possible words that can be formed by the given prefix and k additional characters.

For example, in C++, the function headers would be the following:

```
class Trie {
    TrieNode *root;

  public:
    Trie() {
        // ...
    }
    bool lookup(string str) {
        // ...
    }
    void insert(string str) {
        // ...
    }
    void remove(string str) {
        // ...
    }
    int info() {
        // ...
    }
    void alphabetical() {
        // ...
    }
    void autocomplete(string prefix, int k) {
        // ...
    }
};
```

Input

- The first line of the input will be an integer n indicating how many commands follow it.
- The next n lines will consist of one of 5 commands ("insert", "remove", "lookup", "info", and "autocomplete").
- The commands "insert", "remove", and "lookup" will be followed by a space, then a string representing the word to perform the command on.
- The commands "info" and "alphabetical" have nothing following it.
- The command "autocomplete" will be followed by a space, then a string to perform the command on, then a space, then finally an integer representing the number of additional characters to consider. *The integer may be a special value of "-1" which means consider all words without a length limitation.*

At the end of the input there will be a blank line. Your program should initialize an empty trie and perform the commands given by the input, in sequence, on it.

Constraints

You can assume there will be no invalid input. You can also assume the trie will consist of only lowercase words consisting of characters a-z with no duplicate words.

Output

- For the "insert" and "remove" commands, print nothing.
- For the "lookup" command, print the line "0" if the string is not found and the line "1" if it is found.
- For the "info" command, print a line containing two space separated integers. The first integer is the number of nodes in the trie, the second is the number of words.
- For the "alphabetical" command, print all the words contained in the trie in alphabetical order. One word per line.
- For the "autocomplete" command, print the possible autocompleted words **on a single space-separated line** in alphabetical order (the line will end in a space).

See the sample output section and hackerrank for concrete examples

Sample Input 1

```
17
insert james
insert jackie
lookup ja
insert ja
lookup ja
insert jay
insert jane
lookup jackie
alphabetical
info
autocomplete ja 2
autocomplete ja -1
remove jackie
lookup jackie
lookup ja
alphabetical
info
```

Sample Output 1

```
0
1
1
ja
jackie
james
jane
jay
13 5
ja jane jay
ja jackie james jane jay
0
1
ja
james
jane
jay
9 4
```

2. Huffman codes (20 points)

https://www.hackerrank.com/contests/cst370-s19-hw4/challenges/hw4-huffman

For this problem you will implement a Huffman tree to encode and decode strings to and from variable-width encodings.

Input

- The input will consist of two lines.
- The first line will be the string to derive your codes from and encode. Build your Huffman tree from this string.
- The second line will consist of a test string of 0s and 1s to decode.

At the end of the input there will be a blank line. Your program should build a huffman tree encoding from the first line and use that tree to decode the second line.

In order to build a Huffman Tree, you will need a priority queue. You may use the following priority queues as starter code:

C++ https://repl.it/@dsyang/HuffmanQueue-Cpp

Java https://repl.it/@dsyang/HuffmanQueue-java

Python https://repl.it/@dsyang/HuffmanQueue-py

Constraints

The order in which you insert nodes into your priority queue and the implementation of the priority queue itself has an impact to the created huffman tree. To achieve the same encoding as the test cases, build your huffman tree as such:

- Use a MinHeap in the implementation of the priority queue. You can use one of the MinHeaps provided above or modify your heap from homework 1.
- When initially adding characters into your priority queue, insert the characters in alphabetical order. For example, if your frequency map is: `b:2, c:1, x:1, a:4, [space]:3` then you would add them into the priority queue in this order: `[space], a, b, c, x`.

Output

- The output will consist of two lines.
- The first line will be the encoding of the first input string according to your tree.
- The second line will be the decoding of the second input string according to your tree.

See the sample output section and hackerrank for concrete examples

Sample Input 1

```
i love computer science
1100101110101100110111111011100011010101101100100111
```

Sample Output 1

```
1100101110100001101111110110000001110001100100 (line wrap)
   11111110100101010110011001110110100111
i live in nice
```

3. Radix Sort (15 points)

https://www.hackerrank.com/contests/cst370-s19-hw4/challenges/hw4-radix

For this problem you will be provided an array of integers and asked to sort them using radix sort. You will be printing the partially sorted array after sorting by each digit.

Input

The input will consist of one line containing space integers. At the end of the input there will be a blank line.

Constraints

You can asssume the input is valid and falls within the constraints under which it is appropriate to use radix sort.

Output

The output will consist of k lines, where k is the number of calls to counting sort (i.e., the number of digits in the largest element in the input). Each line will contain a list of space integers in their partially sorted order after that call (the line will end with " ")

See the sample output section and hackerrank for concrete examples

Sample Input 1

```
12
9 87 199 15 3 214 19 26 58 2 102 23
```

Sample Output 1

```
2 102 3 23 214 15 26 87 58 9 199 19
2 102 3 9 214 15 19 23 26 58 87 199
2 3 9 15 19 23 26 58 87 102 199 214
```