M23CS1.304 Data Structures and Algorithms for Problem Solving

Assignment 3

Deadline: 11:59 p.m. October 8th, 2023

Important Points:

1. **Only C++ is allowed**.
2. **Directory Structure:**
```
2023201001_A3
            |_____2023201001_A3_Q1.cpp
            |_____2023201001_A3_Q2a.cpp
            |_____2023201001_A3_Q2b.cpp
            |_____2023201001_A3_Q3a.cpp
            |_____2023201001_A3_Q3b.cpp
```
   Replace your roll number in place of 2023201001

3. **Submission Format:** Follow the above mentioned directory structure and zip the RollNo_A3 folder and submit RollNo_A3.zip on Moodle.
   Note: All submissions which are not in the specified format or submitted after the deadline will be awarded **0** in the assignment.
4. C++ STL (including vectors) is **not allowed** for any of the questions unless specified otherwise in the question. So "#include <bits/stdc++.h>" is **not** allowed.
5. You can ask queries by posting on Moodle.

Any case of plagiarism will lead to a 0 in the assignment or "F" in the course.

1. Spell Checker

Problem Statement: Design an efficient spell checker using trie data structure which supports the functionalities mentioned below.

Required Features:

1. **Spell Check**: Check if the input string is present in the dictionary.
2. **Autocomplete**: Find all the words in the dictionary which begin with the given input.
3. **Autocorrect**: Find all the words in the dictionary which are at an edit distance(Levenshtein distance) of at most 3 from the given input.

Input Format:

- First line will contain two space separated integers **n, q** which represents the number of words in the dictionary and the number of queries to be processed respectively.
- Next **n** lines will contain a single string s which represents a word in the dictionary.
- Next **q** lines will contain two space separated values, First one will be an integer $\mathbf{a_i}$ and second will be a string $\mathbf{t_i}$.
    - $\mathbf{a_i = 1}$ means Spell Check operation needs to be done on $\mathbf{t_i}$.
    - $\mathbf{a_i = 2}$ means Autcomplete operation needs to be done on $\mathbf{t_i}$.
    - $\mathbf{a_i = 3}$ means Autocorrect operation needs to be done on $\mathbf{t_i}$.
- Both strings s and t consists of lowercase english letters.

Output Format: For each query print the result in a new line.

- Spell check: Print ‘1’ if string is present in the dictionary, otherwise ‘0’.
- Autocomplete & Autocorrect: Print the number of words in the first line. The following lines will be the set of words in lexicographical order.

Constraints:

$1 <= n <= 1000$
$1 <= q <= 1000$
$1 <= len(s) <= 100$
$1 <= len(ti) <= 110$

Sample Input:

```
10 4
consider
filters
filers
entitled
tilers
litter
dames
filling
grasses
fitter
1 litter
1 dame
2 con
3 filter
```

Sample Output:

```
1
0
1
consider
5
filers
filters
fitter
litter
tilers
```

Note:

1. Only trie should be used for storing the words in the dictionary.
2. You are allowed to use vector for this problem

2. Unordered Map

a. Problem Statement: Implement unordered map

What is an unordered map?

- An unordered map is an associated container that stores elements formed by the combination of a key value and a mapped value.
- The key value is used to uniquely identify the element and the mapped value is the content associated with the key.

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of unordered map must satisfy the following performance requirements.

For **unordered_map<T1, T2>**, implement the following member functions.

1. **bool insert(T1 key, T2 value)**:
   - Inserts 'value' in the map with key = 'key' if 'key' was not present and returns whether insertion took place.
   - Expected Time Complexity: Amortized O(1)
2. **bool erase(T1 key)**:
   - Deletes 'key' and its value from the map if 'key' was present and returns true, otherwise false.
   - Expected Time Complexity: Amortized O(1)
3. **bool contains(T1 key)**:
   - Returns true if the key exists in map otherwise returns false.
   - Expected Time Complexity: Amortized O(1)
4. **T2& operator[ ](T1 key)**:
   - Inserts default value of T2 with key = 'key' if 'key' was not present in the map. Returns the T2 value by reference.
   - Expected Time Complexity: Amortized O(1)
5. **void clear()**:
   - Clears the map and releases the allocated memory
   - Expected Time Complexity: O(size())
6. **int size()**:
   - Returns the size of the map
   - Expected Time Complexity: O(1)

7. **bool empty():**
    - Returns whether the map is empty or not.
    - Expected Time Complexity: O(1)
8. **std::vector<T1> keys():**
    - Returns a vector of keys.
    - Expected Time Complexity: O(size())

Note:

1. You must use chaining to handle hash collisions.
2. **T1** and **T2** are data types of one of the following types {int, long, float, char, double, string}.
3. Use of **STL vector** library is allowed **only** for operation 8 of this question.

Input Format :

- First line will contain a single integer **q** denoting the number of queries.
- The next lines for each query will have the following pattern:
    - The first line will contain a single integer denoting the type of operation [1 - insert, 2 - erase, 3 - contains, 4 - map, 5 - clear, 6 - size, 7 - empty, 8 - keys]
    - The second line will contain space separated inputs based on the type of operation.

Output Format: For each query print the result in a new line.

- Type 1: true or false.
- Type 2: true or false.
- Type 3: true or false
- Type 4: Print the value associated with the key (use std::cout<<value).
- Type 5: No output.
- Type 6: Integer denoting size of map.
- Type 7: true or false.
- Type 8 : Print every key in the vector on a new line (use std::cout<<key).

b. Problem Statement: Given an array of integer numbers *nums* and an integer *K*, return the total number of subarrays whose sum equals to *K*. A subarray is a contiguous non-empty sequence of elements within an array.

Input Format :

- First line contains an integer that denotes the number of integers in the list (say N) and the value K.
- Next line contains N space separated integers.

Output Format : Print an integer denoting the total number of subarrays with sum equals to K.

Constraints :

$1 <= N <= 10^5$

$-10^5 <= nums[i] <= 10^5$

$-10^7 <= K <= 10^7$

Sample Input:

```
3 2
1 1 1
```

Sample Output:

```
2
```

Explanation:

There are two subarrays: from index [0,1] and index [1,2] where the sum of elements is equal to 2.

Note :

1. You are required to utilize the unordered_map that you implemented in part (a) of the same question.

3. Ordered Map

a. Problem Statement: Implement ordered map

What is an ordered map?

- Ordered map is a sorted associative container that contains key-value pairs with unique keys. Keys are sorted by using the comparison function std::less<key> i.e. operator< implemented inside class "key".

Underlying Implementation:

- The ordered map is implemented using **balanced BST** to maintain the sorted order of the keys. Use **AVL Tree** as the underlying data structure to implement the ordered map.
- AVL Tree: an AVL tree (named after inventors Adelson-Velsky and Landis) is a **self-balancing** binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take **O(log n) time** in both the **average** and **worst cases**, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be **rebalanced** by one or more tree rotations.

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of ordered map must satisfy the following performance requirements.

For **ordered_map<T1, T2>**, implement the following member functions. Also, print the values within main function using the format mentioned for each function:

1. **bool empty():**
    - Returns whether the map is empty or not.
    - Expected Time Complexity: O(1)
    - Print true or false on a new line.
2. **int size():**
    - Returns the size of the map
    - Expected Time Complexity: O(1)
    - Print size on a new line

3. **bool contains(T1 key):**
    - Returns whether ‘key’ is present in the map or not.
    - Expected Time Complexity: O(log size())
    - Print true or false on a new line.
4. **bool insert(T1 key, T2 value):**
    - Inserts ‘value’ in the map with key = ‘key’ if ‘key’ was not present and returns whether insertion took place.
    - Expected Time Complexity: O(log size())
    - Print true or false on a new line.
5. **bool erase(T1 key):**
    - Deletes ‘key’ and its value from the map if ‘key’ was present and returns whether deletion took place.
    - Expected Time Complexity: O(log size())
    - Print true or false on a new line.
6. **T2& operator[](T1 key):**
    - Inserts default value of T2 with key = ‘key’ if ‘key’ was not present in the map. Returns the T2 value by reference.
    - Expected Time Complexity: O(log size())
    - Print T2 on a new line (use std::cout<<value).
7. **void clear():**
    - Clears the map and releases the allocated memory
    - Expected Time Complexity: O(size())
    - Do not print anything.
8. **std::vector<T1> keys():**
    - Returns vector of keys in sorted order
    - Expected Time Complexity: O(size())
    - Print every key in the vector on a new line (use std::cout<<key).
9. **std::pair<bool, T1> lower_bound(T1 key):**
    - Returns a pair of true, first (when ordered using comparator) key ‘lb’ that fails the comparator std::less<T1>(‘key’, ‘lb’), if exists, otherwise a pair of false, default value of T1.
    - Expected Time Complexity: O(log size())
    - Print false on a new line if lower_bound does not exist. Else, print true and key on separate lines.

10. **std::pair<bool, T1> upper_bound(T1 key):**
    - Returns a pair of true, first (when ordered using comparator) key ‘ub’ that fails the comparator std::less<T1>(‘key’, ‘ub’) and std::equal_to<T1>(‘key’, ‘ub’), if exists, otherwise a pair of false, default value of T1.
    - Expected Time Complexity: O(log size())
    - Print false on a new line if upper_bound does not exist. Else, print true and key on separate lines.

Note :

1. **T1** and **T2** are custom classes. Implementation of both will be provided to you.
2. **T1** and **T2** will have **ostream& operator<<** , **istream& operator>>** and assignment operator **operator=** defined.
3. **T1** will have comparison operators **operator<** and **operator==** defined.
4. Use of **STL vector** library is allowed **only** for operation 8 of this question.
5. **std::less<>** is the same as **operator<** and **std::equal_to<>** is the same as **operator==** which are defined on **T1**, so you need not use the functional library.

Input Format: Design an infinitely running menu-driven main function. Each time the user inputs an integer corresponding to the serial number of the operation listed above. Then, take necessary arguments related to the selected operation and execute the respective method. Finally, the program must exit with status code 0, when 0 is provided as a choice.

Output Format: As mentioned in the operation descriptions. Handle printing in the driver code and not in the library implementation.

b. Problem Statement: You are given a list of tokens, where each token is represented by a pair of integers: [top_view, bottom_view]. Two tokens are considered equivalent if they either have the same representation or an inverted representation. In an inverted representation, the first token's top view matches the other token's bottom view, and also the first token's bottom view matches the other token's top view.

Your task is to write a C++ program that takes a list of token pairs as input and calculate the number of pairs (i, j) where 0 <= i < j < tokens.size(), and tokens[i] is equivalent to tokens[j].

Input Format :

- First line contains an integer that denotes the number of tokens in the list (say N).
- Next N lines contain two space separated integers denoting the top view and bottom view of the token.

Output Format : Print an integer denoting the number of equivalent pairs.

Constraints:

$1 <= N <= 10^6$

$-10^9 <=$ a_i , b_i $<= 10^9$ (where {a_i, b_i} represent token_i)

Sample Input:

```
5
12 9
9 12
12 9
3 4
4 3
```

Sample Output:

```
4
```

Explanation:

There are four pairs of tokens that are equivalent (0 - indexed) => {0, 1}, {0, 2}, {1, 2} and {3, 4}.

Note :

1. You are required to utilize the ordered_map that you implemented in part (a) of the same question.