M24CS1.304 Data Structures and Algorithms for Problem Solving

Assignment 3

Deadline: 17/11/2024 - 11:55pm

Important Points:

1. **Only C++ is allowed.** (Unless specified otherwise)
2. **Directory Structure:**
    i. 20242010XX_A3
    |___20242010XX_A3_Q1.cpp
    |___20242010XX_A3_Q2.cpp
3. Replace your roll number in place of 20242010XX
4. **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.*
5. **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.
6. **You can ask queries by posting on Moodle.**

NOTE: In case of plagiarism in any of the questions, the students involved will be awarded -10 as the final marks of Assignment 1.

1.Big Integer Library

Problem Statement: Create a big integer library, similar to the one available in Java. The library should provide functionalities to store arbitrarily large integers and perform basic math operations.

Operations:

Arithmetic Operations and Constraints:

1. Basic Arithmetic Operations:
    - Addition (+)
    - Subtraction (-)
    - Multiplication (x, lowercase "X")
    - Division (/)
2. Examples:
    - **Input:** `32789123+99893271223x9203232392-4874223`
    - **Output:** `919340989462382970316`
    - **Input:** `3423542525+6773442x5345345-213213197786/45647`
    - **Output:** `36209803199102`
3. Exponentiation:
    - Base will be a big integer and the exponent will be less than 2632^{63}263.
4. GCD of Two Numbers
5. Factorial

Constraints:

- For all operations, the number of digits in input, output, and intermediate results won't exceed 3000 digits.

Expected Time Complexity:

- **Addition (s1+s2):** O(n+m)
- **Subtraction (s1-s2):** O(n+m)
- **Multiplication (s1*s2):** O(n*m)
- **Division (s1/s2):** O(n*m)
- **Exponentiation (s1^x):** $O(n2 \cdot \log(x))$
- **GCD (s1, s2):** O(max(n,m))

- **Factorial (s1):** O(n3)

Input Format:

- **First line** will contain an integer value which denotes the type of operation. The integer value and operation mapping is as follows:
    1. **Addition, Subtraction, Multiplication, & Division**
    2. **Exponentiation**
    3. **GCD**
    4. **Factorial**
- **The following line** will contain input according to the type of operation:
    1. For the **1st and 4th types** of operation, there will be one string.
    2. For the **2nd and 3rd types** of operation, there will be 2 space-separated strings.

Evaluation Parameters:

- **Accuracy of operations**
- **Performance**

Sample Cases:

- **Sample Input:**

Sample Input 0-	1 1+2x6+13/5-2
Sample Output 0-	13
Sample Input 1-	2 2 10
Sample Output 1-	1024
Sample Input 2-	3 9 15

Sample Output 2-	3
Sample Input 3-	4 12
Sample Output 3-	479001600

Note:

1. **Negative Numbers:** Negative numbers won't be present in the intermediate or final output (i.e. no need to consider cases like 2-3).
2. **Brackets:** There are NO brackets in the input.
3. **Integer Division:** Perform integer division operation between two big integers, disregarding the remainder.
4. **Operation Precedence:** Addition, Subtraction, Multiplication, and Division follow the same precedence and associativity rules as in Java/C++.
5. **Special Cases:** Ignore Division by zero, gcd(0, x), gcd(x, 0).
6. **C++ STL:** C++ STL is not allowed (including vectors & stack, design your own if required).
7. **Regex Library:** You are not allowed to use the regex library.
8. **String Manipulation:** string, to_string and string manipulation methods are allowed.
9. **Main Function Design:** Design your main function according to sample input/output given.

2. Skip List

The skip list is an example of a probabilistic data structure because it makes some of its decisions at random. While the skip list is not guaranteed to provide good performance, it will provide good performance with extremely high probability.
References: Skip list

Problem Statement : Implement Skip List with following Operations.

Sr no.	Operation	Expected Time Complexity
1	Insertion	O(log N)
2	Deletion	O(log N)
3	Search	O(log N)
4	Count element occurrences	O(log N)
5	lower_bound	O(log N)
6	upper_bound	O(log N)
7	Closest element to a value	O(log N)

Important Points:

- **Implement it with a class or struct.** It should be generic, capable of handling primitive data structures (integer, float, string, etc.), as well as class objects.
- **Duplicates are allowed.**
- **Comparison details:**
    - For strings, you can simply compare them.
    - For Class data type, you must pass a comparator object to compare two objects.

- **Exceptions for operations:**
    - For strings and custom Class objects, you do not need to implement the Closest Element operation.

- **Instructions for Class data type:**
    - Name your custom comparator function as “comp”. Strictly follow this convention as we will also be evaluating the program on our custom class and its custom comparator function (which will be named "comp”).

➔ In your driver code, you need to print the skip list (bottom level) after every operation.

Operations: Consider T e, where T is the type of element e

1. **void insert(e):** Inserts e into the skip list.
2. **delete(e):** Deletes all the occurrences of the element e, if it is present in the skip list.
3. **bool search(e):** Returns true if e is present in the skip list, otherwise returns false.
4. **int count_occurrence(e):** Returns the count of occurrences of the element e.
    - Example: If the skip list has the elements: 1, 1, 2, 2, 2, 3
        - `count_occurrence(2)` will return 3.
        - `count_occurrence(734)` will return 0.
5. **T lower_bound(e):** Return the first element that is greater than or equal to e. If no such element exists, return the default value for type T.
    - Example: If the skip list has the elements: 1, 1, 2, 2, 2, 3
        - `lower_bound(2)` will return an element corresponding to 2, i.e., the 3rd element from the left.
    - If the skip list has the elements 1, 1, 2, 5, 6, 6, 7
        - `lower_bound(3)` will return an element corresponding to 5, i.e., the 4th element from the left.
6. **T upper_bound(e):** Return the first element that is greater than e. If no such element exists, return the default value for type T.
    - Example: If the skip list has the elements: 1, 1, 2, 2, 2, 3
        - `upper_bound(2)` will return an element corresponding to 3, i.e., the last element from the right.
    - If the skip list has the elements: 1, 1, 2, 5, 6, 6, 7
        - `upper_bound(7)` will return 0, i.e., the default value for the type of element e (integer in this case).

7. **T closest_element(e):** Returns the element closest to e. If no such element exists, return the default value for type T.
    - Example: If the tree has the elements: 1, 1, 2, 2, 2, 5, 7.
        - `closest_element(2)` will return 2.
        - `closest_element(3)` will return 2.
        - `closest_element(4)` will return 5.
        - `closest_element(-1472)` will return 1.
        - `closest_element(6)` will return 5 (If there are multiple closest elements present, return the minimum).

NOTE: In case of plagiarism in any of the questions, the students involved will be awarded -10 as the final marks of Assignment 1.
