M23CS1.304 Data Structures and Algorithms for Problem Solving

Assignment 1

Deadline: 11:59 pm. August 22, 2023

Important Points:

1. **Only C++ is allowed**.
2. **Directory Structure:**

   2023201001_A1

   |_____2023201001_A1_Q1.cpp

   |_____2023201001_A1_Q2.cpp

   Replace your roll number in place of 2023201001

3. **Submission Format:** Follow the above mentioned directory structure and zip the RollNo_A1 folder and submit RollNo_A1.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. 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:

1. Addition(+), subtraction(-), multiplication(x, lowercase "X"), division(/)
   E.g.
   Input: 32789123+99893271223x9203232392-4874223
   Output: 919340989462382970316

   Input: 3423542525+6773442x5345345-213213197786/45647
   Output: 36209803199102
2. Exponentiation
   - Base will be a big int and exponent will be $< 2^{63}$
3. GCD of two numbers
4. Factorial

Constraints: For all the operations, input will be such that the number of digits in output won't exceed 3000 digits.

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. For 1st and 4th type of operation, there will be one string and for the 2nd & 3rd type of operation, there will be 2 space separated Strings.

Evaluation Parameters: Accuracy of operations and performance

Sample Cases:

- Sample input:
  1
  1+2x6+13/5-2
  Sample output:
  13
- Sample input:
  2
  2 10
  Sample output:
  1024
- Sample input:
  3
  9 15
  Sample output: 3
- Sample input:
  4
  12
  Sample output: 362880

Note :

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

2. Deque

Problem Statement: Implement Deque.

What is deque?

- Deque is the same as dynamic arrays with the ability to resize itself automatically when an element is inserted, with their storage being handled automatically by the container.
- They support insertion and deletion from both ends in amortized constant time.
- Inserting and erasing in the middle is linear in time.

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of deque must satisfy the following performance requirements: (consider the data type as T)

1. **deque()** - initialize an empty deque. Time complexity: O(1)
2. **deque(n)** - initialize a deque of length n with all values as default value of T. Time complexity: O(n)
3. **deque(n, x)** - Initialize a deque of length n with all values as x. Time complexity: O(n)
4. **bool push_back(x)** - append data x at the end. Return true if operation is performed successfully, else return false. Time complexity: O(1)
5. **bool pop_back()** - erase data at the end. Return true if operation is performed successfully, else return false. Time complexity: O(1)
6. **bool push_front(x)** - append data x at the beginning. Return true if operation is performed successfully, else return false. Time complexity: O(1)
7. **bool pop_front()** - erase an element from the beginning. Return true if operation is performed successfully, else return false. Time complexity: O(1)
8. **T front()** - returns the first element(value) in the deque. If the first element is not present, return the default value of T. Time complexity: O(1)
9. **T back()** - returns the last element(value) in the deque. If the last element is not present, return the default value. Time complexity: O(1)

10. **T D[n]** - returns the nth element of the deque. You need to overload the [ ] operator. If nth element is not present return default value of T. Time complexity: O(1)
11. **bool empty()** - returns true if deque is empty else returns false. Time complexity: O(1)
12. **int size()** - returns the current size of deque. Time complexity: O(1)
13. **void resize(n)** - change the size dynamically to new size n. Time complexity: O(n)
    - If the new size n is greater than the current size of the deque, then insert new elements with the default value of T at the end of the queue.
    - If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.
14. **void resize(n, d)** - change the size dynamically to new size n. Time complexity: O(n)
    - If the new size n is greater than the current size of the deque, then insert new elements with value d at the end of the queue.
    - If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.
15. **void reserve(n)** : change the capacity of deque to n, if n > current capacity; otherwise do nothing. Time complexity: O(n)
16. **void shrink_to_fit()** - reduce the capacity of the deque to current size. Time Complexity: O(size())
17. **void clear()** - remove all elements of deque. Time complexity: O(1)
18. **int capacity()** - return the current capacity of deque. Time complexity: O(1)

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.

Evaluation parameters: Accuracy of operations and performance.

Note :

1. Your deque should be generic type i.e. it should be datatype independent and can support primitive data types like integer, float, string, etc. **Hint:** Use template in C++ (link)
2. For 1, 2 & 3 You can either define a constructor for the class or initialize the class object using void return type functions.
3. C++ STL is **not allowed** (including vectors, design your own if required)
4. D[0] - element at index 0 (i.e. first element from the front),
   D[1] - element at index 1 (i.e. second element from the front),
   D[-1] - element at last index (i.e. first element from the back),
   D[-2] - element at second last index (i.e. second element from the back)
5. Size of the deque is the number of elements currently present in your deque.
6. Capacity of the deque is the number of elements your deque can accommodate with currently held memory.
7. During Operation 1 both size and capacity of the deque should be set to zero.
8. If size is equal to capacity and a new element is inserted, then the capacity is doubled, unless capacity is zero, then it will become one.
9. If you have doubts about deciding the new capacity in any of the operations, refer to the behavior of the member functions of STL vector containers.

QUESTION 3 -
COMING SOON !!