CS2710 - Programming and Data Structures Lab

Lab 4 (graded)

Aug 21, 2024

Instructions

- You are expected to solve ALL the problems in the lab, using the local computer, C++ language, and g++ compiler. (Bonus problems can fetch 5% capped bonus, and Optional problems are just for fun and won't be graded.)
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm (so that TAs can subsequently grade your submissions for graded lab assignments using the private test cases).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Session 4 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB4_Q1.cpp** and **CS23B000_LAB4_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB3**
  - Zip this directory into a file named **CS23B000_LAB4.zip**
  - Submit only this single .zip file to moodle.
- The questions are based on your training in programming in CS1111. So no additional inputs are needed, except the changes needed to switch from C to C++.
- If you need assistance, ask your TA, not your classmate.
- Internet access and mobile devices are prohibited in the lab.
- Check course Moodle for the public test cases and the evaluation script, which you can use to test your programs.
- Solve the problems using array/vector/strings only (do not use any other data structure). You can use std::sort as needed.

Problems to be solved in lab

1. [SMART SEARCH ON DIGITAL LIBRARY] You are tasked with designing a smart search suggestion feature for the digital library at IIT Madras. The library has a vast collection of digital resources, and users often search for specific titles. To enhance the user experience, your system should suggest up to four titles as the user types each search query character (if there are more than four matching titles, pick the four lexicographically smallest titles, and report them in lexicographic order also).

   Using vector of strings would be convenient (you could also arrays if you prefer) and expected time complexity is $O(nk \log n)$, where k is the maximum length of a book title.

   Input Format:

   - N number of books
   - N book titles separated by space
   - A *word* query

   Output Format:

   - Print four (lexicographically smallest) titles whose prefix matches the query. Matching titles when considering length-1 prefix is printed in the first line, length-2 prefix is printed in the second line, and so on (with titles in each line printed in lexicographic ordering).

   Constraints:
   $1 \leq N \leq 1000$
   $1 \leq sum(title[i].length) \leq 10^4$
   $1 \leq query.length \leq 100$
   Book titles do not contain space within the title.

   Examples:
   Sample Input-1:
   ```
   8
   artificial algorithm analysis apple architecture application archive array
   arr
   ```
   Sample Output-1:
   ```
   algorithm analysis apple application
   architecture archive array artificial
   array
   ```
   Explanation:
   ```
   After typing "a":  algorithm analysis apple application
   After typing "ar":  architecture archive array artificial
   After typing "arr":  array
   ```

   Sample Input-2:
   ```
   10
   biology biography biomecha biometrics biosphere biotic bitmap bitwise blockchain botany
   bio
   ```
   Sample Output-2:
   ```
   biography biology biomecha biometrics
   biography biology biomecha biometrics
   biography biology biomecha biometrics
   ```

2. [MATRIX MULTIPLICATION] Given input matrices M1 and M2 (both of dimensions N × N), implement Matrix Multiplication, using two different algorithms. The resultant matrix is stored in M3.

   - B: You have to implement the Basic Method that you would use to multiply matrices using pen and paper, which has a time complexity of O(N^3).
   - S: Strassen's multiplication algorithm. This is designed to be more efficient and has the time complexity of $O(N^{\log_2 7}) = O(N^{2.807})$. You will learn about this algorithm in CS2800 course.

   Compilation instructions and code segments for using Strassen.h file has been provided on the moodle page. You do not have to implement/modify any code for this faster algorithms; you have to simply run them to get insight into the improvements for comparison with the basic method that you will implement.

   Run each multiplication method 100 times and calculate the average time (in nanoseconds) each method takes for the same input.

   Input Format:
   Each testcase contains (2N + 1) lines.

   - The first line contains N. The dimension of both matrices is N×N.
   - From the second line, there are N lines each containing N integers per line separated by spaces. These are the inputs for M1.
   - From the next line, there are N lines each containing N integers per line separated by spaces. These are the inputs for M2.

   Output Format:

   - Product matrix (in stdout)
   - Two values T1 and T2 separated by spaces (in stderr)
   - T1: Average time of B (in nanosecs)
   - T1: Average time of S (in nanosecs)

   Constraints:

   - $0 \leq N \leq 1000$
   - $0 \leq M1[i], M2[i] \leq 10^9$

   Examples:
   Sample Input:
   2
   85 99
   11 32
   36 63
   70 36
   Sample Output: (in stdout using cout)
   9990 8919
   2636 1845
   Sample Error: (in stderr using cerr)
   83476 72462

3. [POLYNOMIAL ADT] We would like you to implement a Polynomial Abstract Data Type (ADT) using arrays in C++ (in future sessions, we may ask for a linked list implementation).

   The Polynomial ADT is provided as a class in the **Polynomial.hpp** file. Download this file from moodle, and implement the methods of this class (including methods to perform addition and multiplication of two polynomials). In the *.cpp file that you prepare for this question, there is no need to define any new class; instead, you simply have to implement the methods of the Polynomial class provided in the Polynomial.hpp file as shown below.

```
//this is the syntax to illustrate how to define the class method outside the scope of the class

[type] [CLASS_NAME]::[METHOD_NAME]([parameters]){
  // your code is here
}
```

   Input Format:

   - The first line will contain an integer N representing the numbers of literals for first polynomial.
   - Next N lines contain two integer first represent the coefficient and second refer to the degree of the variable.
   - The next line will contain an integer M representing the numbers of literals for second polynomial.
   - Next M lines contain two integer first represent the coefficient and second refer to the degree of the variable.

   Output Format:

   - The first line contains the sum of the two input polynomials.
   - The second line contains the multiplication of the two input polynomials.

   The polynomials should be printed with higher degree terms first, and literals with 0 coefficient should not be printed.

   Constraints: $0 \leq N, M \leq 100$

   Examples:
   Sample Input:
   1
   62 2
   7
   75 5
   40 5
   29 4
   53 2
   48 6
   30 6
   42 6

   Sample Output:

```
42x^6 + 40x^5 + 29x^4 + 115x^2
2604x^8 + 2480x^7 + 1798x^6 + 3286x^4
```

4. [**BONUS QUESTION** - ANOMALY SORT] In this challenge, you will design a new O(N)-algorithm for sorting integers (in non-descending order) called Anomaly Sort. Since the lower bound for sorting algorithms is O($N \log N$), the catch is that Anomaly Sort does not work on all inputs, but only on inputs which are almost sorted, which is defined below.

   For an array A consisting of N integers, a pair of indices i, j such that $0 \leq i, j \leq N-1$ form an anomaly if $A[i] > A[j]$. Further, an index $0 \leq i \leq N-1$ forms a local anomaly if $A[i] > A[i+1]$. Notice that every local anomaly is also an anomaly, but every anomaly need not be local.

   An array is called almost sorted if every anomaly in the array is a local anomaly. Write a program to determine if the input array is almost sorted. If so, you also need to sort the array.

   Expected Time Complexity: $O(N)$

   Constraints:
   $0 \leq N \leq 10^6$

   Input Format:

   - The first line contains N, the size of the array
   - The second line contains N space-separated integers

   Output Format:

   - If the input array is not almost sorted, then the output should be a single line containing NO.
   - If the input array is almost sorted, then the output should be a single line containing the sorted array (space-separated).

   Examples:
   Sample Input-1:
   3
   3 2 1
   Sample Output-1:
   NO

   Sample Input-2:
   12
   2 1 4 3 6 5 8 7 20 22 21 23
   Sample Output-2:
   1 2 3 4 5 6 7 8 20 21 22 23

5. [**OPTIONAL** - SNAKE GAME] Design and implement a text console version of the classic Snake game in C++.

   Game Rules:

   - Game Basics: The Snake game involves controlling a snake that moves around a grid, eating food, and growing in length. The game ends when the snake either collides with the walls or itself.
   - Movement and Controls: You can control the snake's direction (up, down, left, right) using user input. In environments where *conio.h* is not available, you can use *std* :: *getchar*() for capturing user input.
   - Timing: To control the snake's speed, you can use

     `this_thread::sleep_for(chrono::milliseconds(dfc))`
   - Grid Display: The game can be displayed on the text console using characters to represent the snake, food, and empty spaces.

   Optional Features:

   - Implement a scoring system.
   - Allow the snake's speed to increase as the game progresses.
   - Add obstacles within the grid.
   - Implement a pause and resume feature.

   Feel free to deviate from the above guidelines and explore your own approaches to enhance the game.

   Solution: You can find a basic guide and ideas for implementing the Snake game here. Feel free to deviate from this guide and explore your approaches to enhance the game.