CS2710 - Programming and Data Structures Lab

Lab 3 (graded)

Aug 14, 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 3 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB3_Q1.cpp** and **CS23B000_LAB3_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB3**
  - Zip this directory into a file named **CS23B000_LAB3.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).

Problems to be solved in lab

1. [DOCUMENT INDEXING] Textbooks generally have an index of words at the back, which is nothing but a sequence of important words used in the book arranged in an alphabetical order, also specifying the page number(s) at which each word appears. Such an index significantly simplifies the task of a reader who wants to search a particular word.

   For this task, the document will be inputted as a sequence of space-separated strings (with possible repetitions). You need to generate an index, which is an alphabetical ordering of all the strings present in the input document (without repetitions). In addition, you also need to specify for each string the sequence of indices (in increasing order) at which the string appears in the input document. Assume that the indexing begins from 0. If there is no such subarray, return 0 instead.

Write a program having O($N \log N$) time complexity.

Input Format:
The first line contains N, the number of words in the document.
The next line has N words separated by space

Output Format:

- If there are K distinct words in the document the o/p must have k lines
- Each line should have a word and the indices where it appears , all space separated.
- The words and indices are sorted

Constraints:

- $1 \leq N \leq 10^6$
- No letter in the document in capitalized (i.e. letter are only from a-z)

Examples:
Sample Input:
```
4
this is a document
```

Sample Output:
```
a 2
document 3
is 1
this 0
```

Sample Input:
```
10
flatness voluntary quote voluntary inseparable quests decimation textural quests solicitor
```

Sample Output:
```
decimation 6
flatness 0
inseparable 4
quests 5 8
quote 2
solicitor 9
textural 7
voluntary 1 3
```

2. [SORT() VS INSERTION SORT] Write a program to read a list of integers (of size N) from the standard input and store it in a vector of appropriate size.
Create a copy of this vector and implement *Insertion sort* as a separate function, to sort this copied list in ascending order. Measure the time taken to execute ONLY this sorting function.
Next, create a copy of the original vector and sort it using the $sort(sort(v.begin(), v.end()));$ function, where v is the vector. Measure the time taken to execute ONLY this sorting function.

For each algorithm, run the program (reading the list and sorting) for 30 times (for the same input) and measure the average time taken in nanoseconds (rounded to the closest integer), for sorting this list of numbers.
Output the average time taken by the two algorithms.

Input Format:

- Take N as input in the first line followed by N integers

Output Format:

- Output the average output time (as an integer in nanoseconds) for your *Selectionsort* algorithm and for the inbuilt *sort* algorithm, separated by a space

Constraints:

$1 \leq N \leq 10^6$

3. [SEARCH IN A ROW WISE COLUMN WISE SORTED MATRIX] Given a matrix that is sorted row wise and column wise, find a given element in the matrix $M \times N$.

   Write a program having O($N + M$) time complexity .

   Input Format:

   - Take Inputs M, N and *target*
   - Then take an array as input

   Output Format:

   - Print TRUE if target exist in array else print FALSE

   Constraints:
   $1 \leq M \leq 10000$
   $1 \leq N \leq 10000$
   $-10^9 \leq matrix[i][j] \leq 10^9$
   $-10^9 \leq target \leq 10^9$

   Examples:
   Sample Input

```
5 5 8
  1   4   7  11  15
  2   5   8  12  19
  3   6   9  16  22
 10  13  14  17  24
 18  21  23  26  30
```

Sample Output
TRUE

Sample Input
3 3 20
3 30 38 44 52 54 57 60 69

Sample Output
FALSE

4. [**BONUS QUESTION** - TEXT JUSTIFICATION] Given a paragraph as input to a string *Para* and a width *maxWidth*, format the text such that each line has exactly *maxWidth* characters and is fully (left and right) justified.

   You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ′ ′ when necessary so that each line has exactly *maxWidth* characters.

   Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

   For the last line of text, it should be left-justified, and no extra space is inserted between words.

   HINT: Convert the paragraph into a vector of strings, say *words*.

   Constraints:
   $0 \leq para.length() \leq 400$
   $1 \leq maxWidth \leq 100$
   $1 \leq word.lenght() \leq maxWidth$

   Input Format:
   paraLength or number of words (int)
   maxWidth (int)
   paragraph (string)

   Output Format: Print arrays of lines with proper spacing as output.

   Examples:
   Sample Input:
   7 16
   This is an example of text justification.

   Sample Output:

```
This    is    an|
example  of text|
justification.  |
```

5. **[OPTIONAL** - SUDOKU GENERATOR]Write a program to generate a 9 x 9 Sudoku grid that is valid for a player to fill the *grid*[ ][ ] by the below following set of rules

   A sudoku puzzle must satisfy all of the following rules:

   - Each of the digits 1-9 must occur exactly once in each row.
   - Each of the digits 1-9 must occur exactly once in each column.
   - Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

   Input Format:
   There is no I/P

   Output Format: Print a 9X9 *Array* as output

   Examples:
   Sample Input :

   Sample Output
   ```
   3 0 6 5 0 8 4 0 0
   5 2 0 0 0 0 0 0 0
   0 8 7 0 0 0 0 3 1
   0 0 3 0 1 0 0 8 0
   9 0 0 8 6 3 0 0 5
   0 5 0 0 9 0 6 0 0
   1 3 0 0 0 0 2 5 0
   0 0 0 0 0 0 0 7 4
   0 0 5 2 0 6 3 0 0
   ```

6. **[OPTIONAL** - SOLVE THE SUDOKU] Given an incomplete Sudoku configuration in terms of a 9 x 9 2-D square matrix *grid*[ ][ ], the task is to find a solved Sudoku if it exists, if not print FALSE. For simplicity, you may assume that there will be only one unique solution. Zeros in the grid indicates blanks, which are to be filled with some number between 1-9. You can not replace the element in the cell which is not blank.

   What is the best upper bound you can derive on the number of Sudoku solutions for a given input? What is the worst-case running time of your Sudoku solver? Give your answer in terms of an upper bound on the number of operations performed by the algorithm, with and without using the big-Oh notation.

   Input Format:
   Take a 9x9 *Grid*[ ][ ] as Input

   Output Format: Print a 9X9 array as output

   Examples:
   Sample Input
   ```
   3 0 6 5 0 8 4 0 0
   5 2 0 0 0 0 0 0 0
   0 8 7 0 0 0 0 3 1
   0 0 3 0 1 0 0 8 0
   9 0 0 8 6 3 0 0 5
   0 5 0 0 9 0 6 0 0
   1 3 0 0 0 0 2 5 0
   0 0 0 0 0 0 0 7 4
   0 0 5 2 0 6 3 0 0
   ```

Sample Output

```
True
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
```