CS2710 - Programming and Data Structures Lab

Lab Midsem

Sep. 11, 2024

Instructions

- You are expected to solve ALL three problems in the lab, using the local computer, C++ language, and g++ compiler.
- This being a lab midsem exam:
  - Do not talk with your classmate, do not take code snippets from your other lab assignments, and do not resort to any forms of malpractice.
  - You are only allowed one A4-sized cheat sheet and cppreference.com access.
  - Ask your TA if you need assistance with the question statement or which data structures should not be used for a question; TAs won't discuss your solution to the question.
  - As always, internet access and mobile devices are prohibited in the lab.
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm, strictly following the same file name conventions as before. For example, for this Lab MidSem (MS) exam consisting of 3 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_MIDSEM_Q1.cpp**, **CS23B000_MIDSEM_Q2.cpp** and **CS23B000_MIDSEM_Q3.cpp**
  - Put these three .cpp files in a directory named **CS23B000_MIDSEM**
  - Zip this directory into a file named **CS23B000_MIDSEM.zip**
  - Submit only this single .zip file to moodle.

Extra Instructions on Questions:

- **Allowed/Disallowed Libraries:** You are allowed to use inbuilt std::string, std::stack, std::vector/C-style-array, and std::sort, if required. But **for question 2 on linked list, you should NOT use std::vector, C-style-array, and std::list**.
- **Expected complexity for full/partial credit:** The expected time complexity is $O(n)$ for all three questions to get full credit (where n is the input size). However, if a linear-time solution escapes you, you are encouraged to code up any other solution you can think of to get partial credit.

---

1. [WHERE DID I END UP?] To help implement syntax highlighting for a code editor, you are tasked to find the closing paranthesis that matches a given opening paranthesis in an expression of well-formed/balanced parantheses.

   Given a string representing a valid (well-formed) expression of parantheses of different types (`'('`, `'['`, `'{'`), if the start index of the open paranthesis is given, output the index of the matching closing paranthesis.

   Constraints:

- $1 \leq$ The length of the string ≤ 100000

   Data Format :

   - **Input Format** : The index of the opening bracket , followed by bracketed sequence.
   - **Output Format :** A single integer denoting the (0-indexed) index of the corresponding closing bracket

   Examples:
   Input1:
   1
   `{[()]({})}`
   Output1:
   4
   Explanation1:
   The opening bracket `'['` at index 1 corresponds to the closing bracket `']'` at index 4.

   Input2:
   0
   `{[()]({})}()`
   Output2:
   9
   Explanation2:
   The opening bracket `'{'` at index 0 corresponds to the closing bracket `'}'` at index 9.

2. [SMALLEST AND LARGEST DISTANCE] Given a linked list of integers, a node in the list is called a "local minima" if the current node has a value strictly smaller than the previous node and the next node. Note that a node can only be a local minima if there exists both a previous node and a next node.

   Create the SLL from the input sequence of integers pointed to by `head`, output the smallest distance between any two distinct local minima and the largest distance between any two distinct local minima. If there are fewer than two local minima, return -1 and -1.

   Do not use array/vector for this question (not even for taking inputs).

   Constraints:

   - $1 \leq$ `# of Nodes in the Linked List` $\leq 10^3$
   - $1 \leq$ `Node.val` $\leq 10^5$

   Data Format :

   - **Input Format :** N (size of linked list) followed by
     Space separated N integers , each being the element of the linked list.
   - **Output Format :** 2 space separated integers , minimum and maximum distance.

   ## Examples:

   Example 1:
   Input:
   ```
   2
   3 1
   ```
   Output:
   ```
   -1 -1
   ```
   Explanation:
   There are no critical points in `3->1`.

   Example 2:
   Input:
   ```
   8
   5 1 3 2 0 2 1 30
   ```
   Output:
   ```
   2 5
   ```
   Explanation:
   There are three local minima:

   - The 2nd node because 1 is less than 5 and 3.
   - The 5th node because 0 is less than 2 and 2.
   - The 7th node because 1 is less than 2 and 30.

   The smallest distance is between the 5th and the 7th node: $(7 - 5 = 2)$.

   The largest distance is between the 2nd and the 7th node: $(7 - 2 = 5)$.

3. [IT'S WAY TOO HOT! OR IS IT?] Given an array of integers `temperatures` representing the daily temperatures, return an array `answer` such that `answer[i]` is the number of days you have to wait after the *i*-th day to get a strictly warmer temperature. If there is no future day for which this is possible, keep `answer[i] == 0` instead.

   Constraints:

   - $1 \leq$ `temperatures.length` $\leq 10^5$
   - $1 \leq$ `temperatures[i]` ≤ 1000

   Data Format:

   - **Input Format :** N (size of the array) followed by N space separated integers , representing the elements.
   - **Output Format :** N space separated integers each being `answer[i]` , representing number of days to wait for a warmer day, for each day.

   Examples:

   Example1:
   Input1:
   `8`
   `73 74 75 71 69 72 76 73`
   Output1:
   `1 1 4 2 1 1 0 0`
   Explanation1:
   Explanations for some of the given temperatures is given below:

   - On day 0 (temperature 73), the next strictly warmer temperature is on day 1 (temperature 74), which is 1 day later.
   - On day 2 (temperature 75), the next strictly warmer temperature is on day 6 (temperature 76), which is 4 days later.
   - On day 6 (temperature 76), there is no strictly warmer temperature on future days, so the answer is 0.

   Example2:
   Input2:
   `4`
   `30 40 50 50`
   Output2:
   `1 1 0 0`