Debug Competition (Draft)

Topic: Segment Tree

— UM-JI (Fall 2019)

1 Background

Chelsea, an intern in Lemonion Co. Ltd, is very good at writing buggy code. You can never imagine how many bugs can be found in a piece of her code.

Recently, Lemonion developed an Artificial Intelligence Debugging Tool, AIDT, which is able to generate a rough list about every bug in the code and the range of lines it may occur. Chelsea was asked to use this tool on her intern project, and not surprising, a long list of potential bugs was shown on her screen, with desperation on her face.

Chelsea needs a reference letter from the Lemonion Company for her graduate application, so she has to fix all of these bugs before the submission of the letter, or being fired by the company. Once she started the attempt to fix them, Reapor, the mentor of the interns, came up to her screen with a weird smile and told her that she also need to verify whether the bugs found meet the list generated by AIDT.

Being capable at everything except writing code, chelsea found a good method to verify the bugs she found. Given a range of her code, it's possible to find the maximum and minimum number of bugs in it according to the result of AIDT; and if the number of bugs she found doesn't fall into this invertal, she must make something wrong. However, the list is extremely long, and for the correctness, many ranges should be verifed. With a simple algorithm based on linear search, it's not possible to acheive these requirements in limited time.

The only thing Chelsea can do now is to find some help from her friends in FOCS (Fans of Computer Science). After some discussion, Salut, a talented freshman programmer in FOCS, proposed that Segment Tree can be introduced to solve this problem. She said,

Segment Tree

A Segment Tree[1] is a data structure that allows answering range queries over an array effectively, while still being flexible enough to allow modifying the array.

This includes finding the sum of consecutive array elements $a[l \dots r]$, or finding the minimum element in a such a range in $O(\log n)$ time. Between answering such queries the Segment Tree allows modifying the array by replacing one element, or even change the elements of a whole subsegment (e.g. assigning all elements $a[l \dots r]$ to any value, or adding a value to all element in the subsegment).

Chelsea thought that this idea should work, but she was still wondering about how to write an algorithm based on Segment Tree, and how to fix all the bugs on time. At this moment, Jane, her another friend in FOCS asked, "Why not holding a Debugging Competition in the Joint institute about the verification algorithm and finding an expert in debugging through it to help you? You can attract them with bonus in the courses!" Jane is correct and all of you are coming to the competition.

2 Problem

In the competition, each team should implement all of the functions in the header file segment_tree.h, and also a main program to solve Chelsea's problem. You are provided with a sample C implentation of a segment tree, written by one of Chelsea's good friend, Nanthan, in FOCS.

2.1 Segment Tree (Data Structure)

Copy something from the website?

2.2 Main Program

The main program should help Chelsea solveing her debugging problem. The bug report of AIDT will have the following format:

- First line: two positive integer *n* and *m*, *n* bug reports in Chelsea's code, and *m* verifications Chelsea wants to try.
- The next *n* lines: a tuple <start end>, which means there the bug is in the interval [start, end].
- The next *m* lines: a tuple <start end>, which means your program should output the expectation and maximum number of bugs in the interval [start, end] on *m* lines.
- Line numbers start from 1. The maximum line number must be in the range of 32 bit unsigned int.

Sample Input:

```
3 3
1 10
6 15
11 20
1 20
6 15
1 10
```

The expectation of bugs should have three digits after the decimal point.
Sample Output:

```
3.000 3
2.000 3
1.500 2
```

Hint:
To calculate the expectation of bugs on a interval, update the expectation on each line, and accumulate them on the segment tree. To calculate the maximum of bugs on a interval, update the maximum on each line and take the range maximum on the segment tree.

2.3 Test Cases

Both of your data structure and main program will be tested by some seperated test cases. Make sure that your implementation of the segment tree provides the same output with the sample code.

3 Competition Rules

3.1 Orgnization of the Competition

In order to keep a fairer competition environment, all participants are split into two brackets, the lower bracket for freshmans and sophoremores, and the upper bracket for juniors and seniors. The competition will be consisted of two stages, the coding stage and the debugging stage.

3.1.1 Coding Stage

In the coding stage, each team should write two versions of code according to the requirements and submit them on JOJ. A team can enter the debugging stage only if its program pass all test cases.

3.1.2 Debugging Stage

In the debugging stage, each team will be provided with all the buggy programs in their section. The goal will be fix as many bugs as possible using GDB.

3.2 Requirements

All of the code should be written in pure C/C++, and be able to be compiled and run on JOJ. All of the features in the standard library are allowed, except the multi-threading library. Each team should submit two versions of code, the correct version and the buggy version.

3.2.1 The correct version

In the correct version, any memory leak, invalid read/write in memory or undefined behavior is not allowed. The program will be tested by clang for memory and undefined behavior checks.

3.2.2 The buggy version

In the buggy version,

- Each team should introduce a maximum of five bugs such that the program crashes.
- Each bug should be rated with a number of credits to be won when it is fixed. The total credits for all the bugs of a team should be 10.
- Each bug should be roughly in 10 continuous lines (in the same function), and the characters need to be altered should not exceed a limit of about 20.
- A list of bugs should also be submited to Chelsea.

The buggy version should fail at least one test on JOJ. Wrong answer, runtime errors, memory leaks, invalid reads/writes in memory and undefined behaviors are all allowed.

3.2.3 Allowed behaviors

The following behaviors are allowed and recommended:

- Obscure the code by adding useless computation.
- Express simple things in a complex manner.
- Use a different implementation of the Data Stricture and the Algorithm.

3.2.4 Forbbiden behaviors

Any of the following behaviors can be reported by other teams and judged by Chelsea. If verified, the guilty team will be kicked out and the reporter will get a bonus.

- Write code or add bugs on them not related to the problem.
- The source code isn't easily readable (e.g. writing the whole program on a single line, use meaningless function and variable names, code mixture).
- Use non-standard library, or the multi-threading libary (intentionally).
- Write Asemmbling code.
- Cheating by providing other teams with an incomplete code (e.g. removing a function)
- The buggy version doen't meet all requirements. (eg. too many bugs, no bugs, can not be compiled, changing too many lines in a bug)

3.3 Scores

The final score is calculated as follows:

- Initial score is 0 for all the team.
- When a team solves a bug it gets its corresponding credits.
- The team who created the bug loses a number of credits equal to the credits corresponding to the fixed bug, times the number of teams who fixed it.
- At the end all the scores are calculated and the teams are ordered in increasing order with respect to their final score. In case of a tie, groups are ordered with respect to the number of bugs they fixed and if this is still a tie the number of their bugs fixed by other teams is considered.

Example: team A wrote 3 bugs, a_1 , a_2 and a_3 , worth 1, 3 and 6 credits respectively. Team B wrote 5 bugs, b_i , $1 \leqslant i \leqslant 5$, worth $\frac{1}{2}$, $\frac{1}{2}$, 1, 3 and 5, respectively. Team C wrote 1 bug worth 10 credits. Assume team A fixes b_1, b_2 and b_5 , team B fixes a_1 and a_3, and team C fixes b_5 and a_2 . The final scores are then:

- Team A: $\frac{1}{2} + \frac{1}{2} + 5 - 1 - 6 - 3 = -4$
- Team B: $1 + 6 - \frac{1}{2} - \frac{1}{2} - 5 \times 2 = -4$
- Team C: $5 + 6 = 11$

Team C wins and since team A fixed one more bug than team B the order is then: 1st team C , 2nd team A, 3rd team B.

References

[1] Segment Tree: https://cp-algorithms.com/data_structures/segment_tree.html