Kick Start 2020 - Round B

Analysis: Robot Path Decoding

To facilitate parsing of the program, let us define *ClosingBracket*(i) as the index of the closing bracket corresponding to the opening bracket at index i. We can find *ClosingBracket*(i) for each opening bracket using a stack in linear time, which is similar to checking whether a string is a correct bracket sequence or not, see this article for more details.

Test Set 1

The total number of moves is limited by 10^4 per test. Consider the expanded version of a program P to be the string consisting of characters N,S,W,E only and having the same moves as P. For example, the program 2(3(N)EW) would expand to NNNEWNNNEW. Since the number of moves is small, we can generate the expanded program, and calculate the position of the robot easily by taking one step at a time.

For an original subprogram between indices L and R, the equivalent expanded version *Expanded*(L, R) can be constructed recursively as follows. We start with an empty string *Result*, iterate the subprogram from the left index L to the right index R, and consider two cases:

- If the i-th symbol is in {'N','S','E','W'}, append it to *Result*.
- If the i-th symbol is a digit D then
  - Call *Expanded*(i+2, *ClosingBracket*(i+1)-1) to construct the expanded version P' of the next subprogram recursively,
  - Append P' to *Result* D times, and
  - Advance the current position i to *ClosingBracket*(i+1)+1.

The first case takes constant time. In the second case, it takes O(D × |P'|) time to append the subprogram P' to the result D times. Let LEN be the length of the expanded program. The total expanded length of the subprograms at the second nesting level is at most LEN/2. The total expanded length of subprograms at the third nesting level is at most LEN/4, and so on. Thus the time complexity would be bounded by LEN + LEN / 2 + LEN / 4 + LEN / 8 + .. which is equal to 2 × LEN as this is a geometric progression. Hence, the time complexity to generate the expanded version of the original program would be O(LEN).

Test Set 2

Now it is possible that the number of moves is exponential in the length of the original program. Thus it would be impossible to execute the moves one by one in the given time.

For the ease of explanation, let us assume that the rows and columns are numbered from 0 (inclusive) to 10^9 (exclusive). Suppose that the robot is at position (a, b) and now we come across instruction X(Y) in the program. Let us say subprogram Y changes the current position of the robot by dx, dy (because of the torus shape of Mars, we can assume that $0 \le dx < 10^9$ and $0 \le dy < 10^9$). Then the position of the robot after following the instruction X(Y) would be $((a + X * dx) \bmod 10^9, (b + X * dy) \bmod 10^9)$ as the subprogram Y is repeated X times. Hence, we just need to find the relative displacement of the robot by each subprogram.

For a subprogram between indices L and R, the relative displacement of the robot can be calculated using *Evaluate*(L, R) recursively as follows. Consider that we are currently at the square (a, b), which is initially the square (0, 0). Iterate the subprogram from the left index L to the right index R, and consider two cases:

- If the i-th symbol is in {'N','S','E','W'}, change the current position of the robot accordingly.
- If the i-th symbol is a digit D then
    - Call *Evaluate*(i+2, *ClosingBracket*(i+1)-1)to get the relative displacement (dx, dy) of the robot by the next subprogram recursively,
    - Change the current position to ((a + D * dx) mod 10^9, (b + D * dy) mod 10^9), and
    - Advance the current position i to *ClosingBracket*(i+1)+1.

Clearly, we visit each character in the program exactly once. Hence, the time complexity of the solution is O(N), where N is the length of the program.