682. Baseball Game

10 April 2022 10:07 AM

You are keeping score for a baseball game with strange rules. The game consists of several rounds, where the scores of past rounds may affect future rounds' scores.

At the beginning of the game, you start with an empty record. You are given a list of strings `ops`, where `ops[i]` is the `i`th operation you must apply to the record and is one of the following:

1. An integer `x` - Record a new score of `x`.
2. `"+"` - Record a new score that is the sum of the previous two scores. It is guaranteed there will always be two previous scores.
3. `"D"` - Record a new score that is double the previous score. It is guaranteed there will always be a previous score.
4. `"C"` - Invalidate the previous score, removing it from the record. It is guaranteed there will always be a previous score.

Return *the sum of all the scores on the record.*

Example 1:

```
Input: ops = ["5","2","C","D","+"]
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.
```

1) [5, 2, C, D, +]

5, ~~2~~ 10, 15

5 + 10 + 15 = 30 Ans

2) 5, -2, 4, C, D, 9, +, +

5, -2, ~~4~~, -4, 9, 5, 14

5 + (-2) + (-4) + 9 + 5 + 14 = 27

We can do Stack or vector

if we do by vector

```
vector<int> score;
for (auto it : ops) {
    if (it == "D") {
        score.push_back (score.back * 2)
    else if (it == 'C') {
        score.pop_back();
    else if (it == '+') {
        size = score.size();
        score.push_back (score[size-1] + score[size-2])
    }
    else {
        //string will be given convert to integer
        // for that we use stoi() function.
        score.push_back (stoi(it));
    }
}
for (int &ans : score) ans += score;
return ans
```

T.C ⇒ O(n * k)

S.C ⇒ O(N)

C++ • Autocomplete

```cpp
class Solution {
public:
    int calPoints(vector<string>& ops) {
        vector<int> score;

        for(auto &it: ops){
            if(it == "D"){
                score.push_back(score.back() * 2);
            }
            else if(it == "C"){
                score.pop_back();
            }
            else if(it == "+"){
                // we need last two element so we find size
                int size = score.size();
                score.push_back(score[size - 1] + score[size - 2]);
            }
            else {
                // convert string to integer
                score.push_back(stoi(it));
            }
        }
        int sum = 0;
        for(auto &it: score){
            sum += it;
        }

        return sum;
    }
};
```

Approach #1: Stack [Accepted]

Intuition and Algorithm

Let's maintain the value of each valid round on a stack as we process the data. A stack is ideal since we only deal with operations involving the last or second-last valid round.

```java
class Solution {
    public int calPoints(String[] ops) {
        Stack<Integer> stack = new Stack();

        for(String op : ops) {
            if (op.equals("+")) {
                int top = stack.pop();
                int newtop = top + stack.peek();
                stack.push(top);
                stack.push(newtop);
            } else if (op.equals("C")) {
                stack.pop();
            } else if (op.equals("D")) {
                stack.push(2 * stack.peek());
            } else {
                stack.push(Integer.valueOf(op));
            }
        }

        int ans = 0;
        for(int score : stack) ans += score;
        return ans;
    }
}
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the length of `ops`. We parse through every element in the given array once, and do $O(1)$ work for each element.
- Space Complexity: $O(N)$, the space used to store our `stack`.