Jerry's Expression

locked

by nikasvanidze

Statistics

Difficulty: Medium
Time Complexity: $O(n)$ Required
Knowledge: Stack, DSU
Publish Date: Jun 17 2017

The operators in the input can only be '+' or '-', so, we need to find which '?' act as positive numbers and which act as negative numbers. The final answer is to be zero, so, whichever of the two (positive numbers and negative numbers) is larger that will be evenly distributed among the '?' for each type. This will minimize the maximum positive integer.

Now, to get which '?' are positives and which are negatives, we can recursively find for each operator all the positives and negatives in the left operand and right operand. Merging operands from the left and right can be achieved using DSU.
For the given sample case `-A-BC`, the second operator will have `B` as positive and `C` as negative. The first operator will have `A` as positive and `-BC` as negative. Merging these two will yield `A, C` as positive and `B` as negative. This approach is provided in the code below.

The other approach can be to reverse the above process, by reversing the string and using a stack to maintain the operands. Whenever an operator is encountered, take top 2 elements from the stack as left and right operand. We can maintain the parent for each element and the final sign of the ? will be decided by the sign of the complete path to the root. This can also be done using DSU. This approach is given in the tester's code.

```
#include <bits/stdc++.h>

using namespace std;
#define vi vector<int>
#define pb emplace_back


const int MAX_N = (int)1e6 + 5;

int parent[MAX_N];
int size[MAX_N];
string expression;

int getParent(int node){
    if(parent[node] == node) return node;
    parent[node] = getParent(parent[node]);
    return parent[node];
}

int merge(int n1, int n2){
    int p1=getParent(n1), p2=getParent(n2);
    if(!p1) return p2;
    if(!p2) return p1;
    if(p1 == p2) return p1;
    if(size[p1] > size[p2]){
        parent[p2] = p1;
        size[p1] += size[p2];
        return p1;
    } else {
```

```
            parent[p1] = p2;
            size[p2] += size[p1];
            return p2;
        }
    }

    tuple<int, int, int> solve(int index){
        if (expression[index-1] == '?'){
            return {index+1, index, 0};
        }

        int next_index=index+1, lefPos, lefNeg, rigPos, rigNeg;
        tie(next_index, lefPos, lefNeg) = solve(next_index);
        tie(next_index, rigPos, rigNeg) = solve(next_index);
        int pos, neg;
        if(expression[index-1] == '+'){
            pos = merge(lefPos, rigPos);
            neg = merge(lefNeg, rigNeg);
        } else {
            pos = merge(lefPos, rigNeg);
            neg = merge(lefNeg, rigPos);
        }

        return {next_index, pos, neg};
    }

    vi complete_solve(){
        int n = expression.size();

        for(int i=0; i<=n; i++){
            parent[i] = i;
            size[i] = 1;
        }

        int dummy, pos, neg;
        tie(dummy, pos, neg) = solve(1);

        int posSize=size[pos], negSize=size[neg];
        int larger=max(posSize, negSize);
        int toFillPos=larger, toFillNeg=larger;
        int donePos=0, doneNeg=0;
        vi answer;
        for(int i=1; i<=n; i++){
            if(expression[i-1] == '?'){
                if(getParent(i) == pos){
                    int this_ans = (double)toFillPos/(posSize - donePos);
                    answer.pb(this_ans);
                    donePos++;
                    toFillPos -= this_ans;
                } else {
                    int this_ans = (double)toFillNeg/(negSize - doneNeg);
                    answer.pb(this_ans);
                    doneNeg++;
                    toFillNeg -= this_ans;
                }
            }
        }
        return answer;
    }

    int main(){
        getline(cin, expression);
        auto ans = complete_solve();
        for(auto x: ans) cout << x << '\n';
    }
```

 Tested by dansagunov

Problem Tester's code :

```
#include <bits/stdc++.h>
using namespace std;

#define sz(a) ((int)(a.size()))

const int N = int(1e6) + 5;
char s[N];

int p[N], sign[N];
int res[N];

stack <int> st;
vector <int> pos[2];

void go(int i) {
    if (p[i] == i)
        return;
    go(p[i]);
    sign[i] *= sign[p[i]];
    p[i] = i;
}

int main() {
    assert(scanf("%s", s) == 1);

    int n = (int)strlen(s);
    reverse(s, s + n);
    assert(1 <= n && n <= int(1e6));
    for (int i = 0; i < n; ++i) {
        if (s[i] == '?') {
            sign[i] = +1;
            p[i] = i;
            st.push(i);
            continue;
        }

        assert(sz(st) >= 2);
        assert(s[i] == '+' || s[i] == '-');
        int w = s[i] == '+' ? +1 : -1;

        int rhs = st.top(); st.pop();
        int lhs = st.top(); st.pop();

        st.push(rhs);
        p[lhs] = rhs;
        sign[lhs] *= w;
    }

    assert(sz(st) == 1);

    for (int i = 0; i < n; ++i) {
        if (s[i] == '?') {
            go(i);
            pos[sign[i] == +1].push_back(i);
        }
    }

    int t = 0;
    if (sz(pos[t]) < sz(pos[!t]))
        t = !t;

    assert(sz(pos[!t]) > 0);

    int div = sz(pos[t]) / sz(pos[!t]);
    int rem = sz(pos[t]) % sz(pos[!t]);

    for (int x: pos[t]) {
        res[x] = 1;
    }

    for (int x: pos[!t]) {
        res[x] = div;
        if (rem)
```

```
            res[x]++, rem--;
    }

    for (int i = n - 1; i >= 0; --i)
        if (res[i])
            printf("%d\n", res[i]);

    return 0;
}
```