C - Parentheses Inversion 解説

原案：monkukui
解説：Slephy

問題概要

問題概要

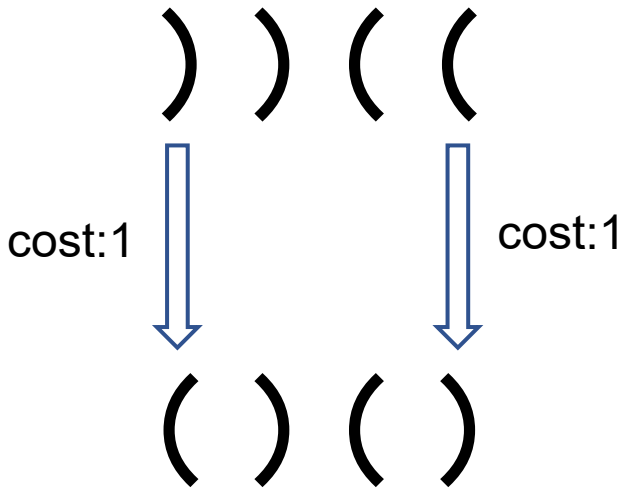
- 長さ N の括弧列 S が与えられます。（Nは偶数）
- 与えられた括弧列に含まれる括弧を好きなだけ反転させることで、これを”正しい括弧列”にします。
- ただし、前から i 番目の括弧を反転するためにはコストが A_i だけかかります。正しい括弧列にするために必要なコストの総和の最小値を求めてください。

制約

- $1 \leq N \leq 2 \times 10^5$
- S は ’(’ と ’)’ のみからなる、長さ N の文字列
- $1 \leq A_i \leq 10^9$
- N は偶整数
- A_i はすべて整数

サンプルケース１

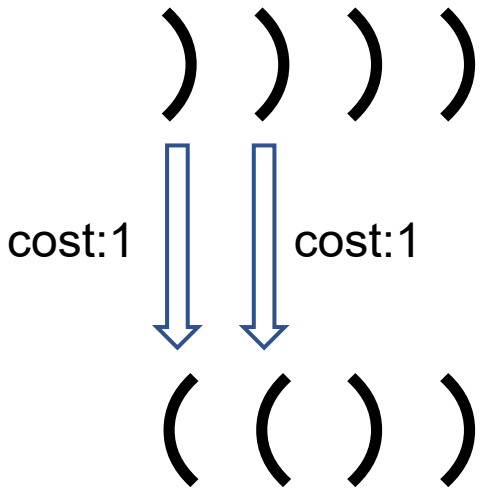
- $N = 4,\ S =$ "））（（", $A = \{1, 1, 1, 1\}$



最小コストは２

サンプルケース２

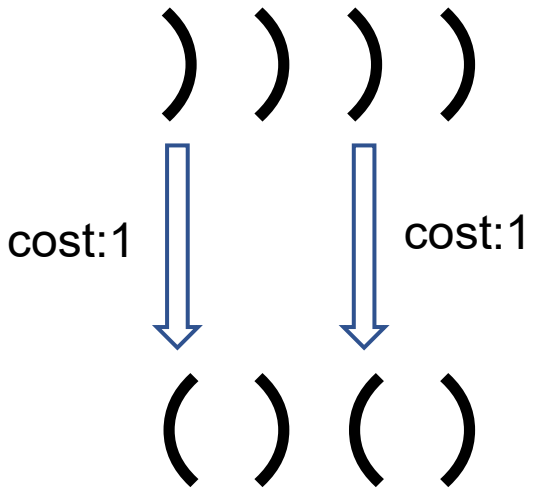
- $N = 4,\ S =$ "））））" , $A = \{1, 1, 2, 2\}$



最小コストは２

サンプルケース３

- $N = 4,\ S =$ "））））" ,　$A = \{1, 2, 1, 2\}$



最小コストは２

考察

コストを気にせず
反転しなければならない括弧が存在することがある

- 端の例

)( )(( )) ･･･　　先頭の閉じ括弧

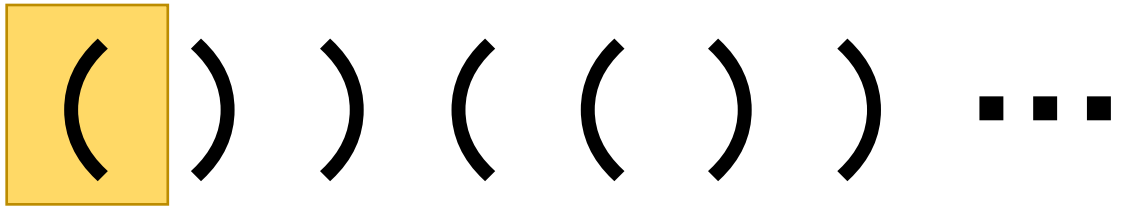
･･･( ))) ( ) (　　末尾の開き括弧

- その他の例

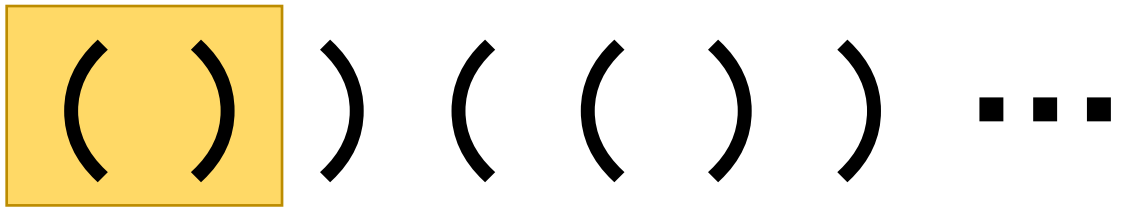
( )) ( ))) ･･･　　左から2, 3番目の括弧のうち
少なくとも一つを反転させる必要がある

反転しなければいけない括弧である条件

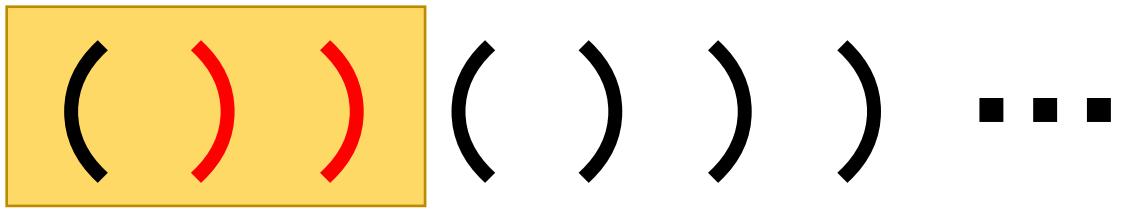
- その他の例を先頭から観察



ここまでOK



ここまでOK



NG！
区間内の開き括弧の個数よりも
閉じ括弧の個数の方が多い！

個数の累積和に着目すればよさそう

開き括弧を 1、閉じ括弧を -1として累積和を取る

( )) ( ))) …

s	(	)	)	(	)	)	)	…
累積和	1	0	-1	0	-1	-2	-3	…

個数の累積和で考える

S	(	)	)	(	)	)	)	…
累積和	1	0	-1					…

区間 [0,2]において
累積和がはじめて
負になった

区間 [0,2]には
閉じ括弧が多すぎる

区間内の閉じ括弧を
少なくとも１つ反転させる
必要がある

正しい括弧列であるための２つの必要条件

以上の考察から、「先頭からの累積和に負数があらわれるならば、まだ反転する必要がある。すなわちこれは、正しい括弧列ではない」ということが分かった。

この対偶を考えると、正しい括弧列であるための必要条件が導ける。

- **先頭から始まる任意の区間での総和がゼロ以上であることは、それが正しい括弧列であるための必要条件である。**

これは末尾から考えたときにも同様のことが成り立つ。

- **末尾で終わる任意の区間での総和がゼロ以下であることは、それが正しい括弧列であるための必要条件である。**

想定解

想定解

- **実は正しい括弧列であるための必要十分条件は、考察で導いた２つの必要条件を共に満たすことである。（証明略）**
- ゆえに「考察」で説明した手順で先頭から反転していき、その後同様の手順で末尾から反転していけばよい。
- ２方向からの探索が終わったら、それは正しい括弧列になっている。
- 「区間内で最もコストの小さい閉じ括弧」を調べるために優先度付きキューなどを用いれば、全体の時間計算量は $O(NlogN)$ となる。

手順のイメージ

S	(	)	)	(	)	)	)	…
累積和	1	0	-1					…

区間 [0,2]において
累積和がはじめて
負になった

区間 [0,2]には
閉じ括弧が多すぎる

区間内の閉じ括弧を
少なくとも１つ反転させる
必要がある

手順のイメージ

３番目の括弧を反転させて、累積和を更新した

S	(	)	(	(	)	)	)	…
累積和	1	0	1	2	1	0	-1	…

区間 [0,6]において
累積和が再び
負になった

区間 [0,6]には
閉じ括弧が多すぎる

区間内の閉じ括弧を
少なくとも１つ反転させる
必要がある

反転させる括弧の選び方

- 区間内に複数閉じ括弧がある場合、その中で**もっともコストの小さいものを反転させればよい**。なぜなら、どの括弧を反転させてもこれ以降の累積和への影響は同じだから。

回答例

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
template <class T> using pqg = priority_queue<T, vector<T>, greater<T>>;
#define ALL(x) x.begin(),x.end()

int main(){
    // input
    int n; cin >> n;
    string _s; cin >> _s;
    vector<int> s(n); // '(' -> 1,  ')' -> -1
    for(int i = 0; i < n; i++) s[i] = (_s[i] == '(' ? 1 : -1);
    vector<int> a(n);
    for(int i = 0; i < n; i++) cin >> a[i];

    // calculate
    ll ans = 0;
    auto traverse = [&](vector<int> &s) -> void{
        int count = 0;
        pqg<pair<int, int>> pq;
        for(int i = 0; i < n; i++){
            count += s[i];
            if(s[i] == -1) pq.emplace(a[i], i);

            if(count < 0){
                auto [cost, index] = pq.top(); pq.pop();
                ans += cost;
                s[index] *= -1;
                count += 2;
            }
        }
    };
    traverse(s);
    // reverse
    reverse(ALL(s));
    for(int i = 0; i < n; i++) s[i] *= -1;
    reverse(ALL(a));
    traverse(s);

    // output
    cout << ans << endl;
    return 0;
}
```

想定解が正しいことの証明

想定解について残る疑問

一見正しそうな想定解だが、次のような疑問が残る。

- 「先頭から順に反転させた後、末尾から順に反転させる」とあるが、末尾から反転させていったことによって、先頭から見たときにまた反転させるべき括弧が現れてしまわないのか。

- 言い換えると、「先頭からの累積和をすべてゼロ以上にした後、末尾からの累積和をすべてゼロ以下にしても**先頭からの累積和はゼロ以上であることが保たれているのか**」

想定解について残る疑問

- この疑問は妥当なものであり、厳密には解法の正当性を確認するために証明が必要である。
- ここからは２方向からの操作後に、先頭からの累積和がゼロ以上であることが保たれていることを証明し、想定解の正当性を示す。

証明のための準備

以下 0-indexed で考え、次のように定義する。

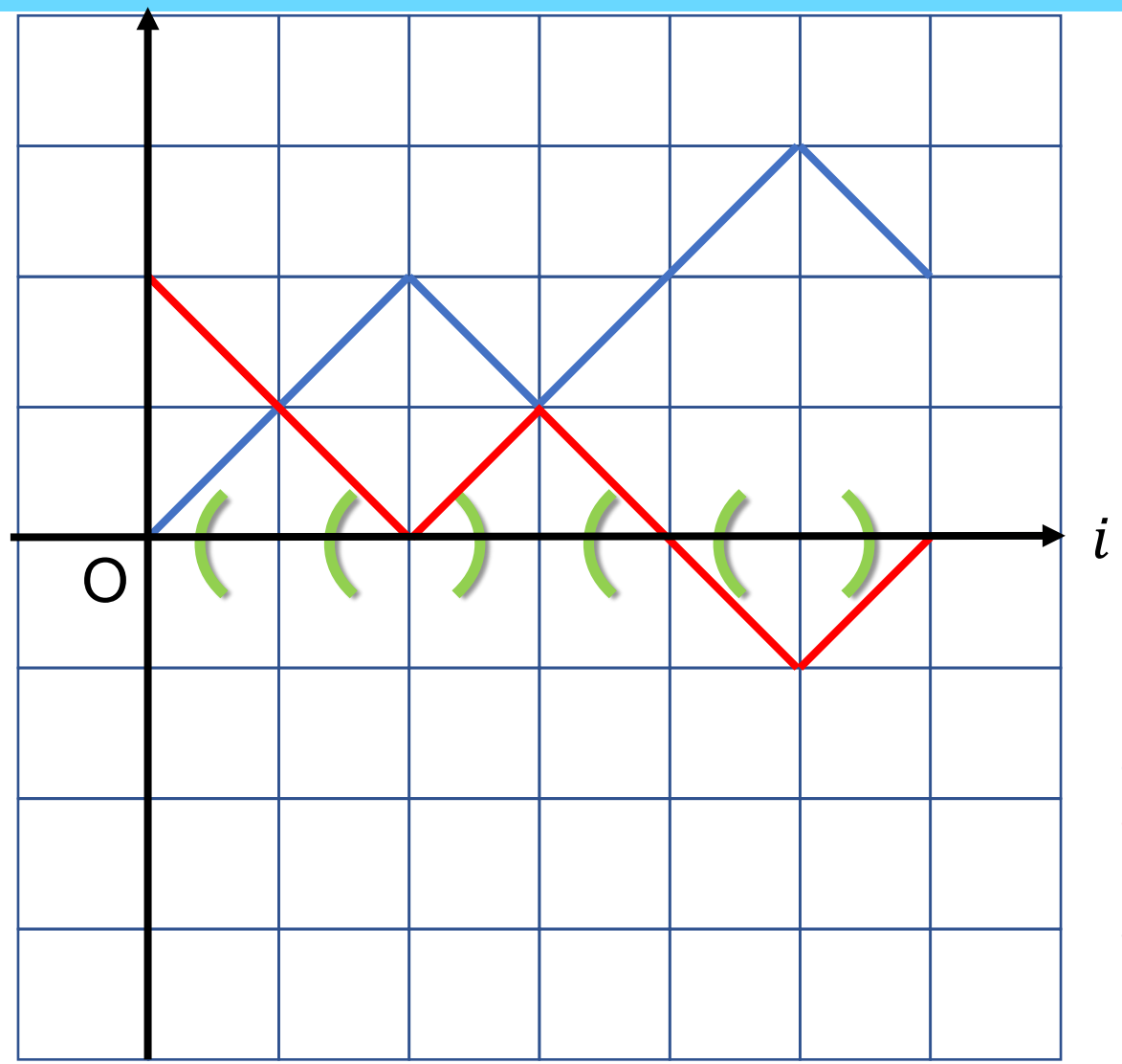
- $left[i] := Sum[0, i)$
- $right[i] := Sum[i, n)$
- $SUM := Sum[0, n) = left[n] = right[0]$

すると、必要十分条件は次のように表せる。

$$\forall i \in [0, n],\ left[i] \geq 0 \ \text{かつ}\ right[i] \leq 0$$

【参考】先頭から順に反転した時点の例

$left[i]$
$right[i]$



- $left$ は常にゼロ以上
- $right$ はまだゼロ以下でない箇所がある
- $SUM = 2$

証明①

まず、末尾からの反転が終わった後に

必ず $SUM = 0$ となることを示す。

先頭からの反転が終わった時点で、SUM は 0 以上の偶数である。

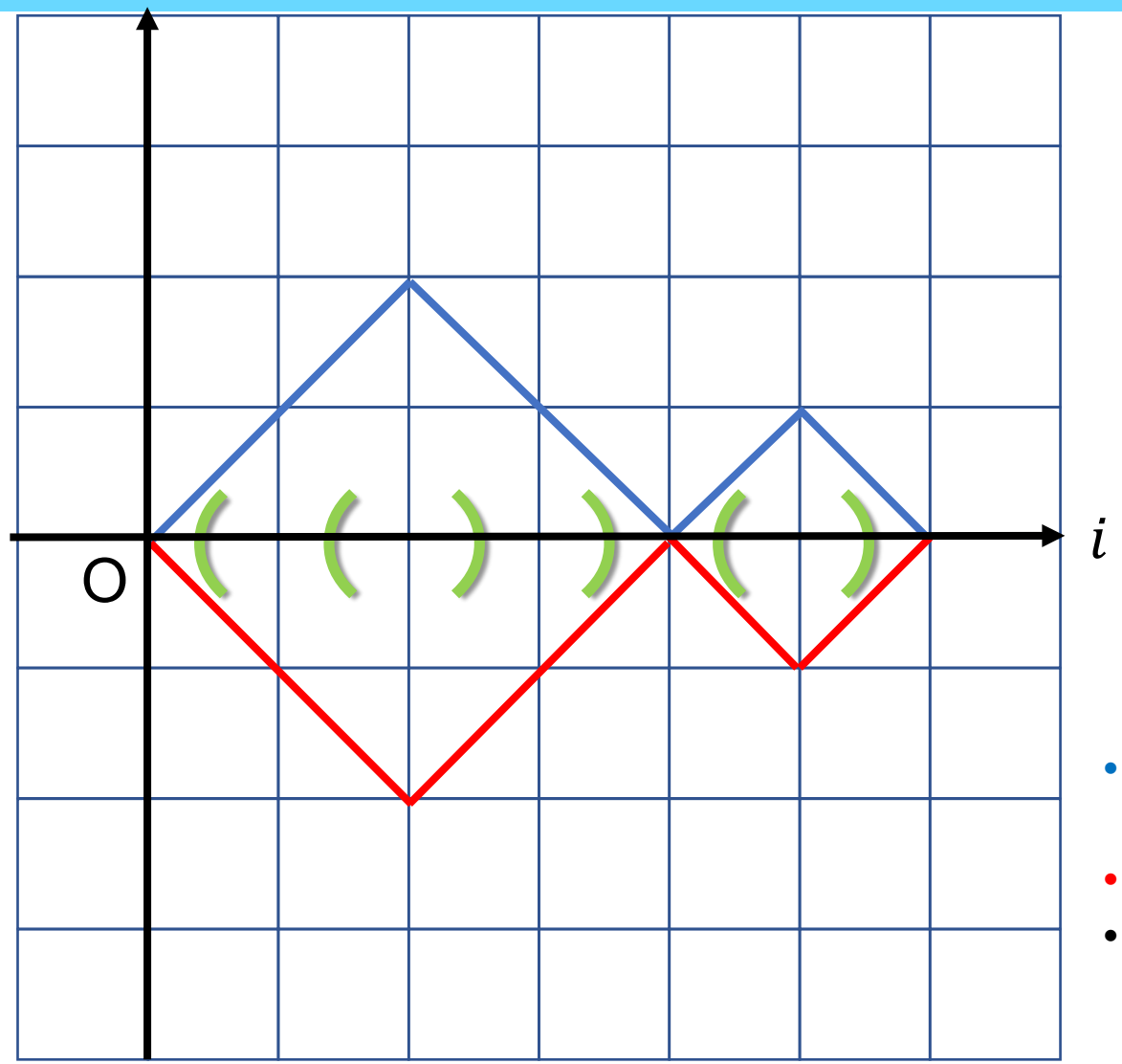
また、常に $max(right[i]) = right[0] = SUM$ が成り立っている。

末尾からの反転では、すべての $right[i]$ が 0 以下になるまで反転を繰り返す。これは $right[0]$ すなわち SUM が 0 以下になるまで反転することと同値である。

一度の反転で SUM は 2 だけ小さくなり、かつ操作の開始時にSUMは偶数であるから、操作の終了時には $SUM = 0$ となる。

さらに末尾から順に反転した後

$left[i]$
$right[i]$



- $left$ が常にゼロ以上であることは保証されていない
- $right$ は常にゼロ以下
- $SUM = 0$

証明②

次に、末尾からの反転が終わった後で

$$\forall i \in [0, n],\ left[i] \geq 0$$ **が成り立つことを示す。**

定義より、$left[i] + right[i] = SUM$ が成り立つ。

先ほどの議論により $SUM = 0$ であるから、$left[i] = -right[i]$ である。

さらに末尾からの反転によって $\forall i \in [0, n],\ right[i] \leq 0$ が成立している。よって、$\forall i \in [0, n],\ -right[i] \geq 0$

すなわち、 $\forall i \in [0, n],\ left[i] \geq 0$ が成り立つ。

これをもって、末尾からの反転後も先頭からの累積和がゼロ以上であることは保たれていることが示せた。

証明③

以上より、２方向からの反転後に正しい括弧列であるための必要十分条件を満たしていることが分かった。

手順からわかる通り、一連の操作で反転させた括弧は「必ず反転させなければならない括弧」から選んだものであり、その中からコストが小さいものを貪欲に選んだのだから、これが最小値を取るのは明らかである。

よって、この想定解は適切な解法である。

最後に

- 長いスライドになってしまいました。最後までお読みいただいた方に感謝いたします。
- 証明を書きながら、もう少し丁寧に示す必要があるか...？と自問自答しておりました。不十分な点がありましたら、ご指摘ください。今後の参考にさせていただきます。
- 別解としては、先頭からの累積和が最小値を取る箇所まで先頭から反転し、それ以降は末尾から反転する。という解法もあります。（そちらの方が、正当性が自明かも）
- より簡潔な証明や自分なりの別解を思いついた方は、インターネットの皆さんに共有していただけると幸いです。