A 問題

答え
$f(20) = 50$
$f(50) = 313$
$f(313) = 12246$
答えは 12246 です。

B 問題

文字列の各インデックス i に対して、区間 $[0, i]$ の‘(’の個数が‘)’の個数を下回らず、全体で‘(’と‘)’の個数が等しい場合、文字列は‘(’と‘)’の対応の取れた文字列であるといえます。

回数が最も少なくなるような操作は、カーソルを右に動かす操作と文字を変更する操作のみで行われます。通り過ぎた場所にもう一度戻るような操作は無駄となるためです。

また、操作の最後がカーソルの移動になるような操作も意味がありません。

つまり最適に操作した場合のの操作回数は、

“最後に文字を変更したカーソルの位置”+“文字を変更した回数”

となります。

カーソルを左に動かす必要がないので、与えられた文字列 S を一文字ずつ走査していき、文字を変更しない場合と、文字を変更する場合で分岐する探索を行うことができます。ですが、このままだと状態数が多く間に合いません。

考察から“最後に文字を変更したカーソルの位置”と“文字を変更した回数”と“ここまでで‘(’が‘)’よりどれだけ多いか”が同じならば、同じ状態とみなす事ができます。

同じ状態を 2 回計算しないように工夫して探索すると、時間内に計算することができます。

C 問題

順位を求めるのに誰に勝ったかは関係なく、必要なのは勝った数だけであり、

それぞれの人が勝った数は独立に考えることができます。

まず、アンドウくんの記録で s 勝である人が、ハシモトくんの記録でが t 勝である確率を考えます。

これは、(s 試合のうち u 試合をハシモトくんが正しく記録する確率) * ($N-1-s$ 試合のうち $t-u$ 試合をハシモトくんが間違って記録する確率) を各 u に対して求めて和をとることで求まります。

そして、(m 試合のうち n 試合をハシモトくんが正しく記録する確率) は、$P^n * (1-P)^{(m-n)} * mCn$ によって求まります。

次に、「アンドウくんの順位での p 位から N 位の人のみを考えアンドウくんとハシモトくんの記録から順位を求めると一致し、アンドウくんの順位で p 位の人がハシモトくんの記録で q 勝以下である確率」を考え、それを $f(p, q)$ とします。

こう定義すると、アンドウくんが付けた対戦記録から求められる順位とハシモトくんが付けた対戦記録から求められる順位が一致する確率は、 $f(1, N-1)$ となり、これが答えとなります。

i) $p = N$ の場合

$f(N, q)$ は、単に、アンドウくんの順位で N 位の人がハシモトくんの記録で q 勝以下になる確率です。

ii) $p < N$ の場合

アンドウくんの順位での p 位の人を X さん、アンドウくんの順位での $p+1$ 位の人を Y さんとし、

ハシモトくんの記録で、X さんの勝数は m、Y さんの勝数は n とします。

アンドウくんの順位で $p+1$ 位から N 位の人に対して順位が一致するとすると、

X さんの予選順位が Y さんの予選順位より高い場合 $m \geq n$ 、そうでない場合 $m > n$ ならば
p 位から N 位に対して順位が一致することになります。

よって、$0 \leq k \leq q$ に対して、(ハシモトくんの記録で X さんが勝った数が k である確率) * $f(p+1, r)$ を求めて和をとることで $f(p, q)$ が求まります。ただし、X さんの予選順位が Y さんの予選順位より高い場合 $r = k$ 、そうでない場合 $r = k - 1$ とします。

D 問題 天下一数列にクエリを投げます

まず、全ての時間についての配列の値を全部持つとどうなるか考えます。たとえばサンプル 1 ならば、以下の様な二次元配列を持ちます。

0	1	2
10	11	2
-10	-9	-18
-10	11	2

この表において加算クエリと調査クエリはどのような操作になっているのかを考えます。

加算クエリは、長方形について一様に同じ数を足す操作であることがわかります。例えば 1 つめの加算クエリは、下図の長方形の区間に一様に+10 を足す、という動作に対応します。

0	1	2
10	11	2
-10	-9	-18
-10	11	2

調査クエリは、幅が 1 の細長い長方形について、中の値の最小を求める操作であることがわかります。例えば 2 つめの調査クエリは、下図の長方形の区間の最小を求める、という動作に対応します

0	1	2
10	11	2
-10	-9	-18
-10	11	2

今回の問題だと、この二次元配列を直接持つことすら出来ないので、効率よく計

算する必要があります。

これは実は、長方形を 90 度回転させると見通しが良くなります。
つまり、時間 0, 1, 2,..と考えていくのではなく、$a_1, a_2, \ldots, a_N$と順に考えていきます。
今a_iの列を見ていて、次にa_{i+1}の列を考える時、配列はどのように変化するでしょうか。

まず、数列の初期値の差の分だけ、数列全体が増減します。
a_{i+1}の列を左端に持つ加算クエリが存在すると、数列のうちどこかから後ろ全てが増減します。
a_iの列を右端に持つ加算クエリが存在すると、数列のうちどこかから後ろ全てが増減します。
よって、以下の操作が可能なデータ構造を用意すればよいです

- 区間に一様に数を足す
- 区間の中での最小値を取得

これらは、SegmentTree と呼ばれるデータ構造を用意することで、どちらも O(log(要素数))で処理することが出来ます。

よって、合計で$O((A + B + N)\log A)$で答えを求めることが出来て、間に合います。

E 天下一合体

まず、2 つの数 a, b について、 $|a|+|b| \geq |a+b|$ が成立します。これは、マージすることにより必ず総和は減る、ということを表しています。よって、最終的に値の数が M 個になる場合のみを考えればよいです。

累積和として、$s_i = a_1 + a_2 + \cdots + a_{i-1}$という数列を用意しておきます。

仮に、$a_l, a_{l+1}, \ldots, a_{r-1}$が最終的に 1 つの要素になったとすると、その要素の値は$s_r - s_l$となります。仮に$(a_1, a_2, a_3), (a_4), (a_5, a_6)$の 3 つに分けたとすると、要素の値はそれぞれ$s_4 - s_1, s_5 - s_4, s_7 - s_5$となります。よって求める値は$|s_1 - s_4| + |s_4 - s_5| + |s_5 - s_7|$となります。よって、この問題は結局、$s_2, s_3, \ldots, s_N$ から M-1 個選び、$|s_1 - s_x| + |s_x - s_y| + \cdots |s_z - s_w| + |s_w - s_{N+1}|$を最小化する問題と考えられます。

例えば入力例 2 だと、$\{s_i\}$は$[0, 0, 3, -2, -3, 2, 2, -1, 4, 0]$となります。
なので$s_2 = 0$ と$s_8 = -1$を選び、$|0-0| + |0-(-1)| + |(-1)-0| = 2$が最小となります。

では、これはどのように解けばよいでしょうか。

$dp[p][c] =$ $s_2, \ldots, s_{p-1}$から$c-1$ 個とs_pを選び、min $(|s_1 - s_x| + \cdots + |s_y - s_p|)$

と定義します。すると dp[N+1][M]が答えになります。

この dp は、

$$dp[p][c] = \min_{i \leq p}(dp[i][c-1] + abs(s_p - s_i))$$

という漸化式で計算できます。よって、$O(N^2 M)$で答えを求めることが出来ます。ですが、これでは間に合わないので、高速化する必要があります。

まず、式を以下のように変形します。

$$dp[p][c] = \min_{i \le p}\left(dp[i][c-1] + abs(s_p - s_i)\right)$$

$$= \min\left\{\min_{i \le p \text{ かつ} s_i \le s_p}\left(dp[i][c-1] - s_i + s_p\right), \min_{i \le p \text{ かつ} s_p \le s_i}\left(dp[i][c-1] + s_i - s_p\right)\right\}$$

ここで、$pos_i = s_i$は$s_1, .., s_{N+1}$を$sort$すると何番目に来るか という配列を事前に計算しておくと、更に以下のように変形できます。

$$\min\left\{\min_{i \le p \text{ かつ } pos_i \le pos_p}\left(dp[i][c-1] - s_i + s_p\right), \min_{i \le p \text{ かつ } pos_p \le pos_i}\left(dp[i][c-1] + s_i - s_p\right)\right\}$$

ここで、$dp[1][c], dp[1][c], \ldots, dp[N+1][c]$と順に計算することを考えると、RMQ(range min query)で高速化することが出来ます。

具体的にどうするかというと、rmq を 2 個用意しておいて、$dp[i][c]$の計算が終わったら rmq1 のpos_iに$dp[i][c-1] - s_i$を、rmq2 のpos_iに$dp[i][c-1] + s_i$をセットすると、数式の 2 つの min はどちらも、この rmq に対する区間 min クエリとなるからです。
1 つめの min は、rmq1 の$[0, pos_p]$の最小にs_pを足せば求められ、
2 つめの min は、rmq2 の$[pos_p, N]$の最小にs_pを引けば求められます。

これにより、O(NMlogN)で計算することが出来て、間に合います