

AtCoder Grand Contest 020 Editorial (Japanese)

writer: tourist

2018 年 1 月 14 日

For International Readers: English editorial starts on page 9.

A: Move and Win

まず、プレイヤーが負けになるのは、動かしたい駒がもう一つの駒と紙切れの端に挟まれた場合のみであることに注意します。

二つの駒の間の距離を $D = B - A$ とします。毎ターンに D が $+1$ か -1 変動することに注意すると、 D の偶奇は毎ターン変化することがわかります。

仮に、アリスの最初のターンの前の D の初期値が偶数であるとしてします。すると、アリスにターンが回ってきたとき D は必ず偶数で、ボリスにターンが回ってきたとき D は必ず奇数となります。

ここで、二つの駒が隣接するとき、必ず $D = 1$ であることに注意します。したがって、 D が偶数であれば、二つの駒は隣接しておらず、アリスは必ず駒を動かすことができます。

視点を変えると、アリスは必ず駒を動かすことができるため、いずれ駒をマス $N - 1$ に動かすことができます。このとき、ボリスの駒はマス N に閉じ込められ、アリスの勝ちになります。

同様に、 D の初期値が奇数である場合、ボリスが駒をマス 2 に動かして勝ちます。

よって、答えは $B - A$ の偶奇に依存し、この値が偶数ならアリスの勝ち、奇数ならボリスの勝ちとなります。

B: Ice Rink Game

ラウンド i の直前の時点で脱落せず残っている子供の人数を X とします。ラウンド i の直後には何人残るでしょうか？作られるグループの個数は明らかに $\lfloor \frac{X}{A_i} \rfloor$ であるため、 $\lfloor \frac{X}{A_i} \rfloor \cdot A_i = X - X \bmod A_i$ 人が残ることになります。以後、 $f_p(X) = X - X \bmod p$ と書きます。

この問題では、次のような N の値をすべて求めることが要求されています。

$$g(N) = f_{A_K}(f_{A_{K-1}}(\dots f_{A_1}(N)\dots)) = 2$$

主に二つの方針が考えられます。

線形走査

最終的に 2 人が生き残るような、ラウンド i の直後での最小の生存人数を L_i とし、同様に最大の生存人数を R_i とします。 $L_K = R_K = 2$ であることが分かっています、求めたいものは L_0, R_0 です。

L_i, R_i から L_{i-1}, R_{i-1} を復元することができるでしょうか？答えは Yes です。

まず、ラウンド i の直後での生存人数は A_i で割り切れなければなりません。もし L_i と R_i の間に A_i の倍数が存在しない場合、答えは -1 となります。

そうでない場合、区間 $[L_i, R_i]$ 内の A_i の倍数のうち最小のものと最大のものをそれぞれ X_i, Y_i とすると、

$$\begin{aligned} L_{i-1} &= X_i, \\ R_{i-1} &= Y_i + A_i - 1. \end{aligned}$$

となります。

ここで、 $X_i = \lceil \frac{L_i}{A_i} \rceil \cdot A_i, Y_i = \lfloor \frac{R_i}{A_i} \rfloor \cdot A_i$ であるため、結局、

$$\begin{aligned} L_{i-1} &= \lceil \frac{L_i}{A_i} \rceil \cdot A_i, \\ R_{i-1} &= \lfloor \frac{R_i}{A_i} \rfloor \cdot A_i + A_i - 1. \end{aligned}$$

となります。

この解法の計算量は $O(K)$ です。

二分探索

f には、広義単調増加であるという便利な性質があります。正式には、 $X \leq Y$ のとき $f_p(X) \leq f_p(Y)$ となります。

広義単調増加同士の合成関数もまた広義単調増加であることに注意します。今回は、 $X \leq Y$ のとき $g(X) \leq g(Y)$ となります。

二分探索を用いて、 $g(N) \geq 2$ であるような最小の N を求め、また別に二分探索を用いて、 $g(N) \leq 2$ であるような最大の N を求めることができます。明らかに、これらの値が問題の答えです（前者が後者を上回る場合は -1 ）。

答えが $2 + K \cdot \max(A_i)$ 以下であることは簡単にわかります。なぜなら、毎ラウンドの脱落者の数はたかだか $\max(A_i)$ 人であるためです。したがって、この解法の計算量は $O(K \log(K \cdot \max(A_i)))$ となります。

C: Median Sum

$S_0 = 0$ とします。これは A の空の部分列に対応します。 S_{2^N-1} が A のすべての要素の和であることに注意します。

A のすべての部分列を二つずつペアにします。このとき、すべてのペアについて、 A のすべての要素がペアのちょうど一方に含まれるようにします（したがって、それぞれの部分列は自分自身と相補的な部分列と組むこととなります）。明らかに、これらのペアの数は 2^{N-1} です。

任意の部分列のペア (P_i, Q_i) を考え、 P_i, Q_i の要素の和をそれぞれ $\Sigma_{P_i}, \Sigma_{Q_i}$ とします。一般性を失うことなく、 $\Sigma_{P_i} \leq \Sigma_{Q_i}$ と仮定します。ペアの組み方より $\Sigma_{P_i} + \Sigma_{Q_i} = S_{2^N-1}$ であるため、 $\Sigma_{P_i} \leq \frac{1}{2}S_{2^N-1}, \Sigma_{Q_i} \geq \frac{1}{2}S_{2^N-1}$ となります。

すべての P_i は、 $\frac{1}{2}S_{2^N-1}$ という値を隔てて、すべての Q_j から分離されていることがわかります。したがって、 Σ_{P_i} は S の前半 $S_0, S_1, \dots, S_{2^{N-1}-1}$ に属し、 Σ_{Q_i} は S の後半 $S_{2^{N-1}}, \dots, S_{2^N-1}$ に属するとみなして構いません。

よって、 S_{2^N-1} の値を求めるには、 A の部分列であって、和が $\frac{1}{2}S_{2^N-1}$ 以上で最小であるものを求める必要があります。

これには動的計画法を用いることができます。 A の最初の i 項の部分列であって、和が j であるものが存在するか否かを $f(i, j)$ で表します。 $f(i, j)$ の値はブール値であり、遷移は bitwise OR で行うことができるため、この動的計画法は bitset（例えば C++ の `std::bitset`、Java の `java.util.BitSet` があります）が、手書きの bitset でも十分で、実装も容易です）を用いて高速化することができます。

この解法の計算量は $O(\frac{N^2 \max(A_i)}{64})$ となります。

D: Min Max Repetition

クエリを一つ一つ処理しましょう。 $f(A, B)_{C..D}$ を求めます。

まず、 $f(A, B)$ の最長の同じ文字からなる部分文字列の長さ、 K を決定しましょう。 $A < B$ とします。文字 A が A 個あり、 B 個の文字 B を間に挟む必要があります。文字 B を入れる場所は $A + 1$ 箇所存在するため、鳩の巣原理より、これらの場所のうち一箇所は文字 B を $\lceil \frac{B}{A+1} \rceil$ 個以上含みます。この数は、文字 B を文字 A の間に均等に入れることで、容易に達成することができます。 $A \geq B$ の場合も同様です。結局、

$$K = \lceil \frac{\max(A, B)}{\min(A, B) + 1} \rceil.$$

となります。

文字列を左から右に一文字ずつ構成していきましょう。辞書順最小の文字列を求めたいため、明らかに、以下のような状況では次の場所に A を置くべきです。

- 置くことが可能である（置いても A が $K + 1$ 連続で現れない）。
- 置いても、最長の同じ文字からなる部分文字列の長さが K 以下となるように文字列を完成させることが可能であることがわかっている。

これらの条件を用いることで、部分点向けの解法を容易に実装することができます。

分析を進めましょう。しばらくは、第二の条件は無関係です。もし第二の条件が存在しなければ、文字列の先頭部分は $AA\dots ABAA\dots ABAA\dots AB\dots$ (A は各グループで K 個ずつ) のようになります。

では、現在の場所に A を置くと第二の条件に反するような状況に初めて遭遇した時を考えます。 A をあと A 文字、 B をあと B 文字使わなければならないとします。もし A を置くと、あと $A - 1$ 文字残ることになります。文字列を適切に完成させることができるための条件は何でしょうか？ある程度考えると、 $B \leq A \cdot K$ がこのための必要十分条件であることがわかります (B のグループを A 組作ることができ、各グループの大きさはたかだか K です)。

この条件が成立しなくなると、 $B > A \cdot K$ となります。この条件が再び成り立つまで、 B のみを置いていくこととなります。

したがって、 B を $B - A \cdot K$ 個置くこととなります。このあと、 $B = A \cdot K$ となります。すると、文字列の残りの部分は $\dots ABB\dots BABB\dots BABB\dots B$ のようになります (B は各グループで K 個ずつ)。これは、 A を置いたときに直ちに B を K 個置いて第二の条件を再び成り立たせる必要があるためです。

これらの分析から、 $f(A, B)$ は次の二つのパーツを連結したような形式であることがわかります。

- $AA\dots ABAA\dots ABAA\dots AB\dots$ (A は各グループで K 個ずつ) の接頭辞
- $\dots ABB\dots BABB\dots BABB\dots B$ (B は各グループで K 個ずつ) の接尾辞

最後に残る問題は、この二つをどこで連結するべきかという点です。

これは、二分探索で容易に求められます。 $f(A, B)$ の第一のパーツに含まれる A の個数、 N_A を求めましょう。二分探索では、 N_A の値を仮定し、まず第一のパーツに含まれる B の個数、 N_B を $N_B = \max(0, \lfloor \frac{N_A - 1}{K} \rfloor)$ から求めます。すると、第二パーツには $A - N_A$ 個の A と $B - N_B$ 個の B が残されることとなります。

もし $B - N_B \leq (A - N_A + 1) \cdot K$ であれば、第二のパーツを作ることは可能で、実際の N_A の値は現在の値より小さくはありません。そうでない場合、第二のパーツを作ることは不可能で、実際の N_A の値は現在

の値より小さいです。

N_A の値が判明すれば、 $f(A, B)_{C..D}$ のそれぞれの文字は $O(1)$ で容易に求められます。この解法の計算量はクエリごとに $O(\log(A + B) + (D - C + 1))$ となります。

E: Encoding Subsets

S のすべてのサブセットについての合計を計算する代わりに、デコードすると S のサブセットになるようなエンコード済み文字列の個数を一齐に数えましょう。文字列 S に対するこの問題の答えを $f(S)$ とします。これを求めるには、エンコード済み文字列の最初の文字に注目します。次の二つの可能性があります。

- 最初の文字は桁 0 または 1 である。この場合、この文字は文字列の残りの部分のエンコーディングとは無関係で、残りの部分をエンコードする方法は $f(S_{2..|S|})$ 通り存在します。 $S_1 = 1$ の場合はこの値を 2 倍するべき（エンコード済み文字列の最初の文字は 0 でも 1 でもよい）で、 $S_1 = 0$ の場合はこの値を 1 倍するべき（エンコード済み文字列の最初の文字は 0 でなければならない）です。
- 最初の文字は開きカッコ $($ である。この場合、このエンコード済み文字列の先頭部分は (PxK) (P はある文字列のエンコード結果) となります。 $K \cdot |A| \leq |S|$ のもとで、正の整数 K と $|A|$ の組み合わせをすべて試みましょう。 $AA \dots A$ (A を K 個連結したもの) が $S_{1..K|A|}$ のサブセットでなければならず、これは、 A が $S_{1..|A|}$ のサブセットであり、同時に $S_{|A|+1..2|A|}$ のサブセットであり、 \dots 、同時に $S_{(K-1)|A|+1..K|A|}$ のサブセットであることと同値です。さらに言えば、これは A がこれらすべての文字列の logical AND (\wedge) であることと同値です。そして、この文字列の残りの部分をエンコードする方法が $f(S_{K|A|+1..|S|})$ 通り存在します。

$f(S)$ の漸化式の全体は次のようになります。

$$f(S) = (1 + S_1)f(S_{2..|S|}) + \sum_{|A|=1}^{|S|} \sum_{K=1}^{\lfloor \frac{|S|}{|A|} \rfloor} f(w(S, K, |A|)) \cdot f(S_{K|A|+1..|S|}),$$

ここで、 $w(S, K, |A|) = S_{1..|A|} \wedge S_{|A|+1..2|A|} \wedge \dots \wedge S_{(K-1)|A|+1..K|A|}$ です。

メモ化することにより、 $f(S)$ を直接計算しましょう。 f は何通りの引数で呼ばれるでしょうか？明らかに、上界は $O(2^{|S|})$ です。しかし、この解法は十分高速であり、正解できることがわかります。

これを確かめる方法は少なくとも二つあります。

- プログラマー流。 $f(S)$ を呼ぶ代わりに、 V を長さ $N = |S|$ のビットマスクのベクトルとして $f(V)$ を呼ぶことにしましょう。このベクトルは f の引数になりうる文字列に対応します。 V_i は、文字 i が元の文字列で対応する文字たちの logical AND であることを表します。このとき、 $N = 100$ に対して $f(\{\{1\}, \{2\}, \dots, \{N\}\})$ を呼んでみましょう。1 から N までのすべての長さのベクトル V や f の引数の個数を数えます。すると、このようなベクトルで長さが 12 を超えるものは 41703 個であることがわかります。また、このようなベクトルで長さが 12 以下のものは 8190 個であることもわかります。したがって、状態数は合計でたかだか 49893 通りです。公式テストデータには、45000 通り以上の状態数のデータが含まれていました。

- 数学者流。 f が T を呼ぶとすると、 T は入力される文字列に以下の二つの操作を繰り返して得ることができます。

- 部分文字列をとる。
- " K -fold" する ($K \geq 2$)。 $S_{1..K|A|}$ を $S_{1..|A|}$, $S_{|A|+1..2|A|}$, \dots , $S_{(K-1)|A|+1..K|A|}$ の AND に変換する。

Fold の操作をすると必ず文字列の長さが半分以下になるので、 $|T| > N/8$ のときは Fold を高々二回しかしていないことが分かり、また、二回する方法は以下の三通りのみです。

- 部分文字列をとる, 2-fold, 部分文字列をとる, 2-fold, 部分文字列をとる
- 部分文字列をとる, 2-fold, 部分文字列をとる, 3-fold, 部分文字列をとる
- 部分文字列をとる, 3-fold, 部分文字列をとる, 2-fold, 部分文字列をとる

例えば最初の場合では、得られる文字列はパラメータ i, j, k を用いて四つの文字列 $S_{i..i+k-1}$, $S_{i+k..i+2k-1}$, $S_{j..j+k-1}$, $S_{j+k..j+2k-1}$ の AND として表せることがわかります。同様に、すべてのケースで、得られる文字列の個数は $O(N^3)$ であることがわかります。

F: Arcs on a Circle

一般性を失うことなく、最長の円弧の長さを L_N とします。この円弧の位置を適当に決めてしまいましょう (対称性より、問題ありません)。円周上に座標系を導入し、上記の N 番目の円弧の両端の座標が $0, L_N$ となるようにし、円周上のその他の点は区間 $[0, C)$ 内の座標値を持つようにします。

円弧 i の左端の座標を X_i とすると、右端の座標は $(X_i + L_i) \bmod C$ となります。問題文より、 X_i の値は $[0; C)$ から一様ランダムに選ばれます。

$P_i = \lfloor X_i \rfloor, F_i = X_i - P_i$ を用いて、 $X_i = P_i + F_i$ と表します。すなわち、 P_i, F_i はそれぞれ X_i の整数部分と小数部分です。 X_i を $[0; C)$ から選ぶ代わりに、整数 P_i を $[0, C - 1]$ から選び、実数 F_i を $[0, 1)$ から選ぶことにしても同等です。また、 $X_N = P_N = F_N = 0$ となります。

$i \neq j$ に対し $F_i = F_j$ となる確率は 0 であるため、 F_i はすべて異なるとみなせます。

F_i を昇順に並べた際の順序を表す順列 A_1, A_2, \dots, A_{N-1} を固定しましょう。すなわち、 $F_{A_1} < F_{A_2} < \dots < F_{A_{N-1}}$ とします (F_N はすでに 0 に固定されています)。このような順列は $(N - 1)!$ 通り存在し、対称性よりこれらの事象の発生確率はすべて等しいです。

これで、 $O((N - 1)! \cdot C^{N-1} \cdot N \log N)$ 時間の解法が得られます。 F_i を昇順に並べた際の順序 ($(N - 1)!$ 通り) と P_i の値 (C^{N-1} 通り) を固定し、円弧が演習全体を覆うか確認する ($O(N \log N)$) というものです。 F_i の値を知る必要はなく、これらの大小関係のみが必要である点に注意してください。

この解法は遅すぎますが、正解への大きなヒントとなります。 F_i の大小関係を固定したあと、 P_i のすべての組み合わせを探索するのではなく、動的計画法を用いましょう。

円弧の端になりうるすべての座標を昇順に列挙してみましょう。

$$\begin{aligned} &0; 0 + F_{A_1}; 0 + F_{A_2}; \dots; 0 + F_{A_{N-1}}; \\ &1; 1 + F_{A_1}; 1 + F_{A_2}; \dots; 1 + F_{A_{N-1}}; \\ &\dots \\ &C - 1; C - 1 + F_{A_1}; C - 1 + F_{A_2}; \dots; C - 1 + F_{A_{N-1}}. \end{aligned}$$

このような座標は CN 個存在します。これらを $x_0, x_1, \dots, x_{CN-1}$ と書き換えましょう。

$f(i, s, j)$ を、集合 s に含まれる円弧の左端として x_i 未満の座標値を割り当て、 x_j 以下のすべての点が一本以上の円弧に覆われるようにする方法の個数とします。

起点となるケースは $f(0, \{N\}, L_N \cdot N) = 1$ で、求めたい値は $f(CN, \{1, 2, \dots, N\}, CN)$ です。

$f(i, s, j)$ からの一つの遷移は、 x_i を左端とする円弧がない場合で、 $f(i + 1, s, j)$ へと遷移します。もう一つの遷移については、 x_i が左端となりうる円弧は $A_{i \bmod n}$ のみであり、さらに $i \bmod n \neq 0, A_{i \bmod n} \notin s$ である場合に限られることに注意します。すると、 $f(i, s, j)$ からのもう一つの遷移は $f(i + 1, s \cup \{A_{i \bmod n}\}, \min(CN, \max(j, i + L_{A_{i \bmod n}} \cdot N)))$ への遷移であることがわかります。

この解法の全体の計算量は $O((N - 1)! \cdot (CN)^2 \cdot 2^{N-1})$ となります。

AtCoder Grand Contest 020 Editorial

writer: tourist

January 14, 2018

A: Move and Win

First, note that the only way to lose is when the token to be moved has the other token on one side and the outside of the strip on the other side.

Let the distance between tokens be $D = B - A$. Note that every turn D is changed by $+1$ or -1 . Thus, after every turn the parity of D changes.

Suppose that initially, before Alice moves, D is even. Then D will always be even when Alice moves, and D will always be odd when Borys moves.

Note that whenever the tokens of Alice and Borys are in neighboring cells, $D = 1$. Thus, if D is even, the tokens are not in neighboring cells, and Alice can always make a move.

On the other hand, since Alice can always make a move, she can eventually move her token to cell $N - 1$. In this case, Borys's token will always be blocked in cell N , and Alice will win.

Similarly, if initially D is odd, Borys wins by moving his token to cell 2.

Thus, the answer depends on the parity of $B - A$: if it's even, Alice wins, otherwise Borys wins.

B: Ice Rink Game

Suppose there were X children left in the game before round i . How many children will be left after round i ? Clearly, the number of groups they will form is $\lfloor \frac{X}{A_i} \rfloor$, therefore, $\lfloor \frac{X}{A_i} \rfloor \cdot A_i = X - X \bmod A_i$ children will be left. Let's denote $f_p(X) = X - X \bmod p$ from now on.

The problem asks us to find all N such that

$$g(N) = f_{A_K}(f_{A_{K-1}}(\dots f_{A_1}(N)\dots)) = 2.$$

There are two main directions which lead to a solution of the problem.

Linear scan

Let L_i and R_i be the smallest and the largest number of children that could be in the game after round i which lead to exactly 2 children in the end. It's known that $L_K = R_K = 2$, and we need to find L_0 and R_0 .

Can we recover L_{i-1} and R_{i-1} from L_i and R_i ? Yes, we can!

First, the number of children after round i must be divisible by A_i . If no integer between L_i and R_i is divisible by A_i , the answer is -1 .

Otherwise, let X_i and Y_i be the smallest and the largest integers in $[L_i; R_i]$ divisible by A_i . Then,

$$\begin{aligned} L_{i-1} &= X_i, \\ R_{i-1} &= Y_i + A_i - 1. \end{aligned}$$

But $X_i = \lceil \frac{L_i}{A_i} \rceil \cdot A_i$ and $Y_i = \lfloor \frac{R_i}{A_i} \rfloor \cdot A_i$. Thus, overall,

$$\begin{aligned} L_{i-1} &= \lceil \frac{L_i}{A_i} \rceil \cdot A_i, \\ R_{i-1} &= \lfloor \frac{R_i}{A_i} \rfloor \cdot A_i + A_i - 1. \end{aligned}$$

The complexity of this solution is $O(K)$.

Binary search

A useful property of function f is that it's monotonically non-decreasing. Formally, if $X \leq Y$, then $f_p(X) \leq f_p(Y)$.

Note that a composition of monotonically non-decreasing functions is also monotonically non-decreasing. In our case, if $X \leq Y$, then $g(X) \leq g(Y)$.

We can use binary search to find the smallest N such that $g(N) \geq 2$, and another binary search to find the largest N such that $g(N) \leq 2$. Clearly, these two values are the answer to the problem (or -1 if the first value exceeds the second one).

It's easy to see that the answer doesn't exceed $2 + K \cdot \max(A_i)$ — in every round, at most $\max(A_i)$ children leave the game. Thus, the complexity of this solution is $O(K \log(K \cdot \max(A_i)))$.

C: Median Sum

Let $S_0 = 0$, which corresponds to an empty subsequence of A . Observe that S_{2^N-1} is equal to the sum of all elements of A .

Divide all subsequences of A in pairs such that every element of A belongs to exactly one subsequence in every pair (thus, every subsequence is paired with its complementary subsequence). Obviously, there are 2^{N-1} pairs.

Consider an arbitrary pair of subsequences (P_i, Q_i) , and let Σ_{P_i} and Σ_{Q_i} be the sum of elements in P_i and Q_i , respectively. Without loss of generality, let $\Sigma_{P_i} \leq \Sigma_{Q_i}$. By the way the pairs are formed, $\Sigma_{P_i} + \Sigma_{Q_i} = S_{2^N-1}$. Thus, $\Sigma_{P_i} \leq \frac{1}{2}S_{2^N-1}$ and $\Sigma_{Q_i} \geq \frac{1}{2}S_{2^N-1}$.

We see that every P_i is separated from every Q_j by the value of $\frac{1}{2}S_{2^N-1}$. Therefore, we can safely assume that Σ_{P_i} belongs to the first half of S , which is $S_0, S_1, \dots, S_{2^{N-1}-1}$, while Σ_{Q_i} belongs to the second half of S , which is $S_{2^{N-1}}, \dots, S_{2^N-1}$.

Thus, to find the value of S_{2^N-1} , we need to find the subsequence of A with the smallest sum greater than or equal to $\frac{1}{2}S_{2^N-1}$.

This can be done using dynamic programming: $f(i, j)$ denotes if there exists a subsequence of the first i elements of A with sum j . Since the values of $f(i, j)$ are boolean and the transitions can be performed using bitwise OR, this DP can be optimized using bitsets (for example, `std::bitset` in C++ or `java.util.BitSet` in Java, but even hand-written bitsets are good enough and easy to implement).

The complexity of this solution is $O(\frac{N^2 \max(A_i)}{64})$.

D: Min Max Repetition

Consider queries one by one — we need to find $f(A, B)_{C..D}$.

First, let's determine K — the length of the longest substring of $f(A, B)$ consisting of equal letters. Let $A < B$. There are A letters A, and B letters B have to be placed in between. Since there are $A + 1$ places to put letters B, by pigeonhole principle, one of these places will have at least $\lceil \frac{B}{A+1} \rceil$ letters B. But this number is also easy to achieve if letters B are placed uniformly between A's. The situation with $A \geq B$ is similar. Overall,

$$K = \lceil \frac{\max(A, B)}{\min(A, B) + 1} \rceil.$$

Let's build the string letter-by-letter from left to right. Clearly, since we want to get the lexicographically smallest string, we should place A on the next place whenever:

- it's possible (we won't form $K + 1$ letters A in a row);
- we know we will be able to finish the string so that the longest substring of equal letters will still have length at most K .

It's easy to use these conditions for implementing a solution for partial points.

Let's analyze further. Until some moment, the second condition will not bother us. If there were no second condition, the string would start like

AA...ABAA...ABAA...AB..., where A is repeated K times in each group.

Then, for the first time we encounter a situation that the second condition is false if we place A on the current position. Suppose we had A remaining letters A and B remaining letters B. If we place letter A, we'll have $A - 1$ letters A remaining. What is the condition that we can finish the string appropriately? It's not hard to see that we can do that if and only if $B \leq A \cdot K$ (we'll be able to form A groups of letters B, and each group will have size at most K).

Once this condition is false, we have $B > A \cdot K$. We will have to place only letters B until this condition is true again.

Thus, we will have to place $B - A \cdot K$ letters B. After this, we'll have $B = A \cdot K$. Now, the rest of the string will look like ...ABB...BABB...BABB...B, where B is repeated K times in each group. That's true because now every time we place letter A, we'll have to place K letters B immediately to make the second condition true again.

Overall, this analysis tells us the form of $f(A, B)$ — it will consist of two parts merged together:

- a prefix of AA...ABAA...ABAA...AB..., where A is repeated K times in each group;
- a suffix of ...ABB...BABB...BABB...B, where B is repeated K times in each group.

The only question left is: where is the merging place?

The merging place is easy to find using a single binary search. Indeed, let's find N_A — the number of letters A belonging to the first part of $f(A, B)$. In binary search, for some supposed value of N_A , first we find $N_B = \max(0, \lfloor \frac{N_A - 1}{K} \rfloor)$, which is the number of letters B in the first part of $f(A, B)$. Then, we'll have $A - N_A$ letters A and $B - N_B$ letters B left for the second part.

If $B - N_B \leq (A - N_A + 1) \cdot K$, the second part is possible to obtain, and the actual value of N_A is not lower than the current one. Otherwise, the second part is impossible to obtain, and the actual value of N_A is strictly lower than the current one.

After we know N_A , it's easy to find every character of $f(A, B)_{C..D}$ in $O(1)$. The complexity of this solution is $O(\log(A + B) + (D - C + 1))$ per query.

E: Encoding Subsets

Instead of doing summation over all subsets of S , let's count encodings which are decoded into a subset of S all together. Let $f(S)$ denote the answer to the problem for string S .

How to calculate it? Consider the first character of the encoding. There are two options:

- It is a digit, 0 or 1. Then this character is independent from encoding of the rest of the string, and there are $f(S_{2..|S|})$ ways to encode it. This should be multiplied by 2 if $S_1 = 1$ (the first character of the encoding may be 0 or 1) or multiplied by 1 if $S_1 = 0$ (the first character of the encoding has to be 0).
- It is an opening bracket, (. Then this encoding starts with $(P\mathbf{x}K)$, where P is an encoding of some string A . Let's loop over positive integers K and $|A|$ subject to $K \cdot |A| \leq |S|$. We need $AA \dots A$ (A repeated K times) to be a subset of $S_{1..K|A|}$. This is equivalent to A being a subset of $S_{1..|A|}$, a subset of $S_{|A|+1..2|A|}$, ..., and a subset of $S_{(K-1)|A|+1..K|A|}$ at the same time. This, in turn, is equivalent to A being a subset of logical AND (\wedge) of all these strings. And there are $f(S_{K|A|+1..|S|})$ ways to encode the rest of the string.

The overall formula for $f(S)$ is:

$$f(S) = (1 + S_1)f(S_{2..|S|}) + \sum_{|A|=1}^{|S|} \sum_{K=1}^{\lfloor \frac{|S|}{|A|} \rfloor} f(w(S, K, |A|)) \cdot f(S_{K|A|+1..|S|}),$$

where $w(S, K, |A|) = S_{1..|A|} \wedge S_{|A|+1..2|A|} \wedge \dots \wedge S_{(K-1)|A|+1..K|A|}$.

Let's calculate $f(S)$ directly with memoization. How many different arguments will f be called with? Obviously, the upper bound is $O(2^{|S|})$. But it turns out that the solution is fast enough to get accepted.

There are at least three ways to see this:

- Programmer's way. Instead of calling $f(S)$, let's call $f(V)$ where V is a vector of bitmasks of length $N = |S|$. This vector corresponds to a string which is a potential argument of f . V_i means that character i is logical AND of the corresponding characters in the original string. Now, call $f(\{\{1\}, \{2\}, \dots, \{N\}\})$ for $N = 100$. Count the number of vectors V , arguments of f , of every length from 1 to N . It turns out that the number of these vectors of length more than 12 is 41703. At the same time, the number of strings of length up to 12 is 8190. Thus, the total number of states is at most 49893. The official test data had cases with more than 45 thousand states.
- Mathematician's (handwaving?) way. Again, there are only $2^{13} - 2$ strings of length at most 12. It can be shown that for each length more than 12, there are only $O(N^2)$ ways to get a string of this length using the

operations in f . In fact, the number of states can be bounded by $O(N^3 + 2^{N/8})$, with a fairly small constant in front of N^3 .

- Believer's way. Run your solution on cases of maximum length with different number of ones and see that it is very fast.

Appendix: Detail of Mathematician's way.

If we get a string T as an argument of f , T must be obtained from the input string by repeating the following two types of operations:

- Take a substring.
- "K-fold" a string ($K \geq 2$). That is, from $S_{1..K|A|}$, get logical AND (\wedge) of the strings $S_{1..|A|}$, $S_{|A|+1..2|A|}$, ..., $S_{(K-1)|A|+1..K|A|}$.

If $|T| > N/8$, we can perform "fold" operations at most twice (because each fold operation will halve the length). There are three ways to perform two "fold" operations:

- Take a substring, 2-fold it, take a substring, 2-fold it, and take a substring.
- Take a substring, 2-fold it, take a substring, 3-fold it, and take a substring.
- Take a substring, 3-fold it, take a substring, 2-fold it, and take a substring.

For example, in the first case, the string we get will be the logical AND (\wedge) of four strings $S_{i..i+k-1}$, $S_{i+k..i+2k-1}$, $S_{j..j+k-1}$, $S_{j+k..j+2k-1}$ for some parameters i, j, k . Thus, there are at most $O(N^3)$ such strings. The second and the third cases are similar.

F: Arcs on a Circle

Without loss of generality, suppose the longest arc has length L_N . Let's fix the position of this arc on the circle arbitrarily (it doesn't matter due to symmetry). Introduce a coordinate system on the circle in such a way that the ends of the N -th arc have coordinates 0 and L_N , while all other points on the circle have coordinates from $[0; C)$.

Let X_i be the coordinate of the left end of arc i . Then, its right end has coordinate $(X_i + L_i) \bmod C$. By the problem statement, X_i is chosen uniformly from $[0; C)$ at random.

Let's represent $X_i = P_i + F_i$, where $P_i = \lfloor X_i \rfloor$ and $F_i = X_i - P_i$. That is, P_i and F_i are the integer and the fractional parts of X_i , respectively. Instead of choosing X_i from $[0; C)$, let's choose integer P_i from $[0; C - 1]$ and real F_i from $[0; 1)$ independently, which is equivalent. Also, $X_N = P_N = F_N = 0$.

We can assume that all F_i are distinct, since the probability that $F_i = F_j$ for $i \neq j$ is 0.

Let's fix the permutation A_1, A_2, \dots, A_{N-1} which determines the order of F_i : $F_{A_1} < F_{A_2} < \dots < F_{A_{N-1}}$ (F_N is already fixed at 0). There are $(N - 1)!$ permutations, and all such orders are equally likely due to symmetry.

Now we can build a solution which works in $O((N - 1)! \cdot C^{N-1} \cdot N \log N)$. Fix the order of F_i ($(N - 1)!$ ways), the values of P_i (C^{N-1} ways), and check if the arcs cover the whole circle ($O(N \log N)$). Note that we don't need to know the values of F_i , we only need to know how they compare to each other.

This is too slow, but gives a big hint to the full solution. After fixing the order of F_i , instead of brute forcing all combinations of P_i , let's use dynamic programming.

Consider all coordinates which can potentially turn out to be the ends of arcs in increasing order:

$$\begin{aligned} &0; 0 + F_{A_1}; 0 + F_{A_2}; \dots; 0 + F_{A_{N-1}}; \\ &1; 1 + F_{A_1}; 1 + F_{A_2}; \dots; 1 + F_{A_{N-1}}; \\ &\dots \\ &C - 1; C - 1 + F_{A_1}; C - 1 + F_{A_2}; \dots; C - 1 + F_{A_{N-1}}. \end{aligned}$$

There are CN such coordinates. Let's encode them by $x_0, x_1, \dots, x_{CN-1}$.

Let $f(i, s, j)$ denote the number of ways to assign left ends' coordinates less than x_i to arcs in set s so that all points with coordinates up to x_j are covered by at least one arc.

The base case is $f(0, \{N\}, L_N \cdot N) = 1$, and the value we want to know is $f(CN, \{1, 2, \dots, N\}, CN)$.

One transition from $f(i, s, j)$ is to $f(i + 1, s, j)$ if no arc's left end is at x_i . Otherwise, note that the only arc which can start at x_i is $A_{i \bmod n}$, and only if $i \bmod n \neq 0$ and $A_{i \bmod n} \notin s$. Then, the other transition from $f(i, s, j)$ is to $f(i + 1, s \cup \{A_{i \bmod n}\}, \min(CN, \max(j, i + L_{A_{i \bmod n}} \cdot N)))$.

The overall complexity of this solution is $O((N - 1)! \cdot (CN)^2 \cdot 2^{N-1})$.