AtCoder Grand Contest 011 解説

writer: semiexp

2017 年 3 月 12 日

For International Readers: English editorial starts on page 6.

A: Airport Bus

バス B に b 人，バス B' に b' 人の乗客が乗っているとする．バス B' に，バス B のある乗客より到着時刻の遅い乗客が乗っているとき，バス B' の乗客は全員，バス B のどの乗客よりも到着時刻が遅いとしてよい．さもなくば，バス B, B' のいずれかの乗客のうち，到着時刻が早い方から b 人を新たにバス B に，残った b' 人を新たにバス B' に乗せることができる．

よって，到着時刻が早い乗客から順に,「まずその人がバスに乗っていないなら，その人をバスに乗せ，さらに乗せられるだけ到着時刻が早い人から乗せていく」ことを繰り返せばよいことがわかる．これは，T_i をソートした後,「今残っている乗客のうち，先頭の乗客と同じバスに乗る乗客」を乗せられるだけ先頭から選んでいくことを繰り返せば解くことができる．これは $O(N \log N)$ で可能である．

B: Colorful Creatures

生き物を大きさでソートすることで，一般性を失わず $A_1 \leq A_2 \leq \cdots \leq A_N$ としてよい．$2\sum_{i=1}^{k} A_i < A_{k+1}$ であるような最大の k を t とおく．(このような k が存在しないときは $t = 0$ とする) $1, 2, \ldots, t$ 番目の生き物の色が最後まで残るためには，大きさが $\frac{A_{t+1}}{2}$ より大きくなってから $t+1$ 番目の生き物を吸収する必要があるが，$1, 2, \ldots, t$ 番目の生き物すべてが合体しても不可能であることから，このようなことはできない．(もちろん，$t+1$ 番目より大きい生き物を先に吸収することもできない) 一方，$t+1, t+2, \ldots, N$ 番目の生き物は，小さい生き物から順に吸収していくことで，最終的に残ることができる．

よって，上で示した t を計算し (左辺の総和は，$k = 1, 2, \ldots, N$ の順番で試していけば全部で $O(N)$ で求められる)，$N - t$ を出力すればよい．

C: Squared Graph

問題を一般化して，次のような問題を考える：

> 無向グラフ $A = (V_A, E_A), B = (V_B, E_B)$ が与えられる．グラフ C を，頂点集合は $V_A \times V_B$ であり，辺集合は「$(a, b), (a', b')$ 間に辺があることと，A において a, a' 間に辺がありかつ B において b, b' 間に辺があることは同値」として定めるとき，C の連結成分の個数を求めよ．

この問題は，上の問題において $A = B$ の場合に対応するから，上の問題が解ければ元の問題も解けることがわかる．

まず，A, B がいずれも連結である場合を考える．C において $(a, b), (a', b')$ 間に辺があることと，ある整数 l が存在し，A において a, a' 間に，B において b, b' 間に，それぞれ長さ l のパスが存在することは同値であることに注意する．

A, B いずれも頂点数が 2 以上の場合，任意のパスの長さを 2 伸ばすのは容易であるため，この条件は「A において a, a' 間に，B において b, b' 間に，長さの偶奇の等しいパスが存在する」と言い換えられる．

- A, B の少なくとも一方の頂点数が 1 のとき
  C には辺は存在しないため，連結成分の個数は $|A| \cdot |B|$ である．

- A, B の両方の頂点数が 2 以上であり，A, B の少なくとも一方が二部グラフではないとき
  このときは，C の連結成分の個数は 1 であることを示す．そのためには，A において a, a' 間に辺があり，B において b, b' 間に辺があるとき，C において $(a, b), (a', b)$ 間，$(a, b), (a, b')$ 間の両方に辺があることを示せばよい．

  B が二部グラフではないと仮定する．(A が二部グラフではない場合も同様) すると，B には長さが奇数のサイクル $v_1 v_2 \dots v_k v_1$ (k は奇数，この列において隣り合う頂点間には辺がある) が存在する．よって，b から v_1 を経由して，この奇数長のサイクルを通り v_1 に戻り，最初と同じパスを経由して b を戻ることで，B において b から b への奇数長のパスを得る．a から a' の間には奇数長のパスがあるのは明らかであるから，$(a, b), (a', b)$ 間 に辺がある．これと $(a', b), (a, b')$ 間に辺があることと合わせ，$(a, b), (a, b')$ 間にも辺がある．

- A, B の両方の頂点数が 2 以上であり，A, B の両方が二部グラフであるとき A の頂点集合 V_A を 2 つの部分集合 S_A, T_A に分け，すべての辺が S_A, T_A の間にまたがっているようにできる．B についても同様に V_B を S_B, T_B に分ける．

  先程と同様に，$a \in S_A, b \in S_B$ であるような (a, b) および，$a' \in T_A, b' \in T_B$ であるような (a', b') はすべて C で同じ連結成分に属することがわかる．$a \in S_A, b \in T_B$ であるような (a, b) および，$a' \in T_A, b' \in S_B$ であるような (a', b') についても同様である．一方，$a \in S_A, b \in S_B$ であるような (a, b) および，$a' \in S_A, b' \in T_B$ であるような (a', b') については，S_A 中の 2 頂点間には偶数長のパスしかなく，S_B, T_B にまたがる 2 頂点間には奇数長のパスしかないことから，この 2 つは同じ連結成分に属さない．よって，この場合は連結成分の個数は 2 である．

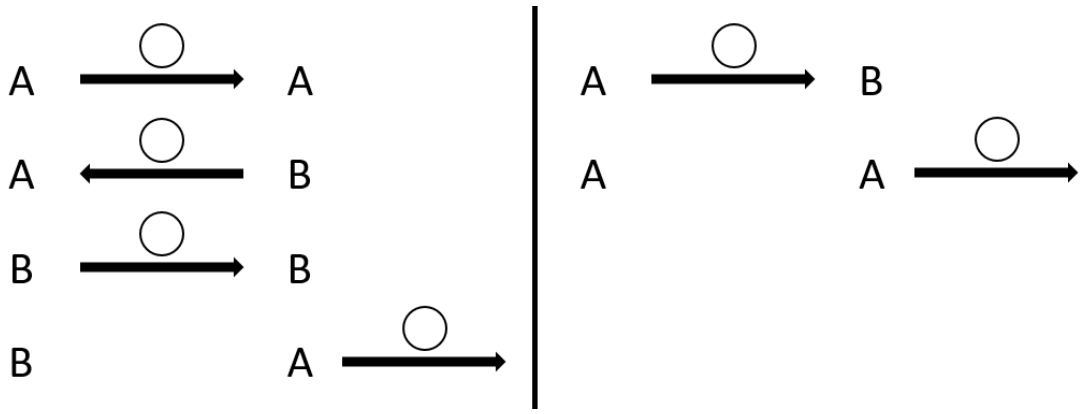
A, B が一般の場合には，連結成分のペアごとに問題を解いて，それぞれの連結成分の個数の合計を求めればよい．A, B の全頂点数を N_A, N_B，次数 0 の頂点数を i_A, i_B，二部グラフでない連結成分の個数を p_A, p_B，二部グラフである連結成分の個数を q_A, q_B とおく．このとき，連結成分のペアごとに考えることで，全体の連結成分の個数は

$$(i_A i_B + i_A(N_B - i_B) + i_B(N_A - i_A)) + p_A p_B + p_A q_B + q_A p_B + 2 q_A q_B$$

であることがわかる．これを計算するのは，グラフ連結成分を求め，それぞれの連結成分の頂点数が 0 か，また二部グラフかを判定するのが $O(V + E)$ で行えることから，$O(V + E)$ で行うことができる．

D: Half Reflector

1 回ボールを入れたときの装置の状態の変化を考える．まず，一番左の装置の状態が `A` であれば，この装置の状態が `B` になり，左端からボールが飛び出してくるだけである．一番左の装置の状態が `B` である場合を考える．ボールはこの装置を通過し，一番左の装置の状態が `A` になった状態で，次はこの装置の右端から飛び出してくる．さて，この「状態 `A` の装置の右端からボールが飛び出してくる」状況から，どのような状態変化が起こるかを考えると，下図のようになる．



この図から,「状態 A の装置 E の右端からボールが飛び出してくる」状況においては，E の状態は E の右の装置の元々の状態と反対の状態になり，E の右の装置が状態 A になって，その装置の右端からボールが飛び出してくる状態に変化することがわかる．さらに，装置 N は最終的には必ず状態 A になることもわかる．

以上をまとめると，1 回ボールを入れたときの装置の状態変化は，次のようになる．

- 一番左の装置の状態が A のときは，この装置の状態が B に変化する．それ以外の装置の状態は変化しない．
- 一番左の装置の状態が B のときは，$1, 2, \ldots, N-1$ 番目の装置の状態は，それぞれボールを入れる前における $2, 3, \ldots, N$ 番目の装置の状態と逆の状態になる．N 番目の装置の状態は，必ず A になる．

一番左の装置の状態が B のときの状態変化は，状態列を文字列をみなしたとき,「1 文字目を取り除き，他の文字については A, B を入れ替えた上で，末尾に A を加える」ことに対応する．ここで，文字列の偶数文字目の文字をあらかじめ反転させておくと，これは「1 文字目を取り除き，末尾に (N の偶奇に応じて) A または B を加える」ことと同じになる．この操作は deque を用いると $O(K)$ で行える．また，K が大きい場合も，$2N$ 回のシミュレートを行った後は状態の周期が 1 もしくは 2 になることを利用すると，$2N + O(1)$ 回で全体のシミュレートを完了できる．このアルゴリズムは $O(N)$ で動作し，十分高速に問題を解くことができる．

E: Increasing Numbers

10 進法において各桁がすべて 1 であるような数を **レピュニット** と呼ぶ．(i 桁からなる) i 番目のレピュニットを R_i で表すことにする．たとえば，$R_1 = 1, R_2 = 11, R_3 = 111$ である．便宜上 $R_0 = 0$ としておく．

整数 M が増加的であることと，M が高々 9 個のレピュニットの和で書けることは同値になる．実際,

- M が増加的なとき，i 以上の数が現れる桁のうち最も大きい箇所を D_i とすると (1 の位の位置を 1 とする．そのような箇所が存在しないときは $D_i = 0$ とする) $M = R_{D_1} + R_{D_2} + \cdots + R_{D_9}$ と表される．
- M が高々 9 個のレピュニットの和で書けるとき，$M = R_{D_1} + R_{D_2} + \cdots + R_{D_9}$ かつ $D_1 \geq D_2 \geq \cdots \geq D_9 \geq 0$ であるような整数 $D_1, \ldots, D_9$ が存在する．

よって，N が高々 k 個の増加的数の和で書けることと，N が高々 $9k$ 個のレピュニットの和で書けることは同値である．

任意のレピュニットは，非負整数 r に対して $\dfrac{10^r - 1}{9}$ と表される．この形で表される整数には 0 も含まれる．よって，N が高々 k 個の増加的数の和で書けるとき,

$$9N = \sum_{i=1}^{9k} (10^{r_i} - 1) = \sum_{i=1}^{9k} 10^{r_i} - 9k$$

を満たすような整数 $r_1, r_2, \ldots, r_{9k}$ が存在する．このような $9k$ 個の整数が存在するかどうかを知るには，$9N + 9k$ の各桁の数の和が $9k$ 以下であるかどうかを確認すればよい．

任意の整数は，$99\ldots9, 199\ldots9, \ldots, 999\ldots9$ のような形の増加的数を引くことで 1 桁小さい整数にできることに注意すると，L 桁の整数に対するこの問題の答えは高々 L である.「N が高々 k 個の増加的数の和で書けるか？」の判定は，上で示した方法を用いると $O(L)$ でできるので，k を二分探索で決定することによりこの問題は $O(L \log L)$ で解くことができる.

なお，この問題は $O(L)$ で解くこともできる．そのためには，$k = 1, 2, \ldots$ と小さい方から上の判定を順次行う．k が 1 増えたとき，$9N + 9k$ の値は 9 増える．多倍長整数に 9 増やす操作を最大 L 回行う必要があるが，繰り上がりが起きなくなった地点で加算処理を終了することで，これらの処理は全体で $O(L)$ で行うことができる．9 増やす際に，各桁の数の和も適切に更新することで，各 k における判定を効率よく行うことができ，$O(L)$ で問題が解ける.

いずれの解法においても，多倍長整数の実装が必要であるが，必要とされるものは「整数を 9 倍する処理」「各桁の数の和を求める処理」および，$O(L \log L)$ 解法では「int 型に収まる程度の正の整数を足す処理」，$O(L)$ 解法では「各桁の数の和を更新しながら 9 を足す処理」のみであり，比較的容易である.

F: Train Service Planning

どの列車も，反対方向の列車とすれ違いを行う位置は同じであることに注意する．駅 $0, N$ 発の列車をそれぞれ適当に選んで T, T' で表すことにする.

所要時間が $\frac{K}{2}$ より大きい単線区間が 1 個でも存在する場合，条件を満たすように列車を走らせることは不可能である．なぜならば，そのような区間では K 分の間に駅 $0, N$ 発の列車をそれぞれ，同時にこの区間にいることはないように走らせる必要があるためである．一方，そのような区間が存在しない場合は，条件を満たす列車の走らせ方が存在する．実際，駅 0 発の列車は駅 $0, 1, \ldots, N-1$ のすべてにおいて出発時刻を K の倍数とし，駅 N 発の列車は駅 $N-1, N-2, \ldots, 0$ のすべてにおいて到着時刻を K の倍数とするように列車を走らせればよい.

全区間が単線の場合

まず，全区間が単線の場合を考える．このとき，両方向の列車がすれ違うことができるのは駅においてのみである．列車 T がすれ違いを行う駅 (便宜上，駅 $0, N$ も含めて考える) を固定して考える．列車 T がすれ違いを行わない駅ですれ違いを行う列車は存在しないから，すれ違いを行う駅同士の間はまとめて 1 つの単線区間 (所要時間は，それらの駅の間に元々あった区間の所要時間の合計) とみなしてよい．すると，列車 T はすべての駅ですれ違いを行うとみなすことができる．このとき，列車 T' もすべての駅ですれ違いを行う.

所要時間の和を最小にする列車の走らせ方の中に，列車 T の停車時間がどの駅でも 0 であるようなものが存在することを示す．列車 T のある駅 i の停車時間が T_i $(T_i > 0)$ であるとき，次の操作を行うことができる.

- 駅 0 から駅 N へのすべての列車の，駅 $i, i+1, \ldots, N-1$ の出発時刻および駅 $i+1, i+2, \ldots, N$ の到着時刻を T_i 早くする.
- 駅 N から駅 0 へのすべての列車の，駅 $N, N-1, \ldots, i+1$ の出発時刻および駅 $N-1, N-2, \ldots, i$ の到着時刻を T_i 早くする.

条件を満たす列車の走らせ方に対してこの操作を行っても，相変わらず条件は満たされており，また所要時間の和は変化しない．一方，列車 T の駅 i での停車時間は 0 になり，この列車の他の駅での停車時間は変化しない．よって，所要時間の和を最小にする適当な運転計画から始めて，この操作を列車 T の停車時間が 0 より大きい駅すべてに対して行うことで，所要時間の和が最小かつ列車 T の停車時間がどの駅でも 0 であるような列車の走らせ方を得ることができる.

次に，列車 T の停車時間が 0 であるような列車の走らせ方において，列車 T' の所要時間を考える．先程の操作と同様にして，列車 T' は駅 s $(s = N-1, N-2, \ldots, 1)$ において，すれ違いを行う駅 0 発の列車が発車するのと同時に発車するとしてよいことがわかる．さらに，列車 T' の駅 $N-1$ への到着は，駅 $N-1$ の出発時刻と同時としてよいこと

もわかる．よって，区間が端から $A_1, A_2, \ldots, A_N$ であり，列車 T が各駅ですれ違いを行うときの列車の所要時間の和は，次で表される：

- $N = 1$ のときは，$2A_1$
- $N \geq 2$ のときは，$2A_1 + K(N-2) + 2A_N$

すれ違い駅を決めたときの所要時間の和がわかったので，次にすれ違い駅を決める方法を考える．駅 0 から駅 i までの区間でうまくすれ違い駅を決めたときの，「(最初の区間の所要時間の和) $\times 2 +$ (区間の数 $- 1$) $\times K$」の最小値を X_i とする．各 i について X_i が計算できれば，駅 i から駅 N まですれ違いなしで行くときの所要時間の和の最小値が計算でき，結果として答えを求めることができる．

まず，駅 0 から駅 i までの所要時間の和が $\frac{K}{2}$ 以下であれば，X_i はこの所要時間の和の 2 倍に等しい．さもなくば，駅 j から駅 i までの所要時間の和が $\frac{K}{2}$ 以下であるような最小の j に対して，$X_i = \min_{j \leq i' < i}(K + X_{i'})$ となる．このような j は尺取法などにより容易に計算できるので，X_i を順次 Segment Tree を用いた動的計画法で計算することができる．ところで，X_i は広義単調増加である．これは，駅 0 から駅 i までの途中のすれ違い駅の決め方は，そのまま駅 $i-1$ までのすれ違い駅の決め方として使うことができることからわかる．よって，$X_i = \min_{j \leq i' < i}(K + X_{i'})$ とするかわりに，$X_i = K + X_j$ としてしまってかまわないことがわかる．

以上より，$i = 0, 1, \ldots, N$ に対して X_i を $O(N)$ で決定できることがわかったので，全区間単線の場合を $O(N)$ で解けることがわかった．

複線区間が存在する場合

複線区間では，区間の途中でも自由にすれ違いを行うことができる．これは，途中に勝手に駅を設置することができる (ただし，区間の所要時間の合計は保たれなければならない) ような単線区間と同等と思うことができる．

このもとで，先程の X_i を計算することを考える．X_i に相当する値を，仮想的に複線区間の途中でも考えることにすると，この値は駅 0 からの所要時間の合計に対して広義単調増加である．つまり，直前のすれ違い地点を決めるときには，相変わらずできるだけ現地点から遠い地点を選ぶべきである．

X_i を計算するときに，駅 i から所要時間の和が $\frac{K}{2}$ であるような手前の地点が単線区間の途中であった場合には，前のすれ違い地点は (先程と同様に) その単線区間の終わりとする．一方，この地点が複線区間の途中であった場合には，その地点をすれ違い地点として，さらに $\frac{K}{2}$ だけ遡って，すれ違い地点が駅となる地点を探す．つまり，所要時間の和を $\frac{K}{2}$ ずつ遡っていったとき，初めてぶつかる単線区間の途中はどこであるか，を求めたい．これは，「最後に出てきた単線区間の位置」を $\bmod \frac{K}{2}$ で順次更新していけば求められる (実際には，所要時間のほうをすべて 2 倍したほうが簡便である)．この操作は，C++ の set などを使えばならし $O(\log N)$ で実装できる．

複線区間が存在する場合，最も駅 N に近いすれ違い地点が複線区間の途中になる可能性があるため，X_i を計算しただけでは答えを求めることができない．しかし，両方向の列車の所要時間の和を最小にしつつ，すれ違い地点が駅に一致するような箇所は存在するとしてよい．これは，駅 0 から駅 N への列車の時刻すべてを同様に，連続的に大きくしていくと，すべてのすれ違い地点が少しずつ駅 0 側に移動し，時刻をちょうど K ずつ増やしたときにはすれ違い地点は最初と同じになっていることからわかる．(この間に，すれ違い地点が単線区間の途中になって条件を満たさなくなる可能性があるが，最初に条件を満たさなくなる瞬間の直前には，そのすれ違い地点はちょうど駅に一致していたはずである．) よって，駅 i から駅 N までの区間でうまくすれ違い地点を決めたときの，X_i と同様の値の最小値を Y_i としたとき，答えは $\min_{i=0,1,\ldots,N}(X_i + Y_i)$ として求められる．この Y_i の計算は X_i とまったく同様に行うことができる．

A: Airport Bus

Suppose that b people take bus B and b' people take bus B'. If some passenger on bus B' arrives at the airport after some passenger on bus B, we can assume that all passengers on bus B' arrive at the airport after all passengers on bus B. (Otherwise, among those $b + b'$ people, we can assign the first b people to bus B and the last b' people to bus B').

Therefore, we should repeat the following: "Choose the first person who haven't taken bus yet, create a new bus, put the person to the new bus, and assign the largest possible number of people to the new bus (in the increasing order of arriving time)".

First, sort the people by T_i, and assign people to buses from left to right. Let i be the smallest index that is not assigned to a bus yet. Then, we should find the maximum j such that $T_j - T_i \leq K$ and $j - i + 1 \leq C$, and assign them to the bus. (The next i will be $j + 1$). We can get the answer by repeating this process. This solution works in $O(NlogN)$.

B: Colorful Creatures

Suppose that $A_1 \leq A_2 \leq \cdots \leq A_N$. Let t be the largest integer k such that $2\sum_{i=1}^{k} A_i < A_{k+1}$. (If such k doesn't exist, $t = 0$.) We claim that the last color will be one of $t+1, \ldots, N$.

If one of $1, 2, \ldots, t$ wants to survive, its size must be at least $\frac{A_{t+1}}{2}$ (and the eats $t+1$). However, even if all of $1, 2, \ldots, t$ merges, its size doesn't reach $\frac{A_{t+1}}{2}$. On the other hand, each of $t+1, t+2, \ldots, N$ can survive by eating in the increasing order of size.

Thus, we should compute such t and the answer is $N - t$.

C: Squared Graph

In general, consider the following problem:

> You are given two graphs $A = (V_A, E_A), B = (V_B, E_B)$. Define a graph C: the set of vertices are $V_A \times V_B$ and there is an edge between $(a, b), (a', b')$ iff there is an edge between a, a' in A and there is an edge between b, b' in B. Compute the number of connected components in C.

(The original problem corresponds to the case $A = B$).

First, suppose that both A and B are connected.

If $|V_A| = 1$, the answer is obviously $|V_B|$. Similarly, if $|V_B| = 1$, the answer is obviously $|V_A|$.

Otherwise, each graph contains at least two vertices. There is an edge between $(a, b), (a', b')$ in C iff there is a path of legnth l between a, a' in A and there is a path of legnth l between b, b' in B for some integer l. Since we can easily extend the length of paths by two, we are only interested in parities.

If both A and B are bipartite, there will be two connected components in C. (If V_A are divided into S_A, T_A and V_B are divided into S_B, T_B in bipartite coloring, the two connected components are $S_A \times S_B \cup T_A \times T_B$ and $S_A \times T_B \cup T_A \times S_B$).

Otherwise, C will be connected because we can always find a path with given parity between two vertices of non-bipartite graph.

When A, B are not connected, we can solve the problem for each pair of connected components and compute the sum. Let N_A, N_B be the number of vertices, i_A, i_B be the number of isolated vertices, p_A, p_B be the number of non-bipartite components, q_A, q_B be the number of bipartite components (except for isolated vertices). Then, the answer is:

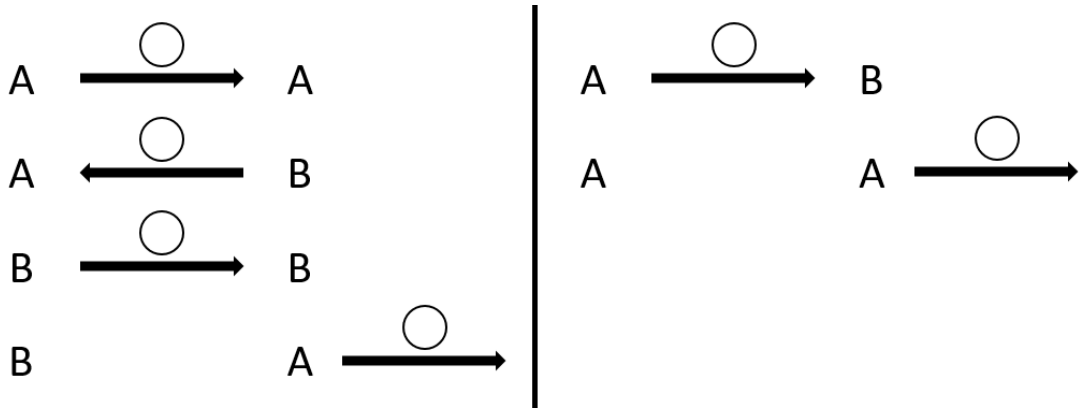
$$(i_A i_B + i_A(N_B - i_B) + i_B(N_A - i_A)) + p_A p_B + p_A q_B + q_A p_B + 2 q_A q_B$$

This solution works in $O(M)$.

D: Half Reflector

First, let's see what happens when the operation is performed only once. If the leftmost device is in state `A`, the state of this device will be `B` after the operation and nothing else happens. If the leftmost device is in state `B`, the ball first changes the state of the leftmost device to `A` and then enters the second device from the left side.

See the following picture (the left picture shows the case where the second device is in `A`, the left picture shows the case where the second device is in `B`):



From this, you see that the state of the first device after the operation is the negation of the state of the second device before the operation. Then, the ball will enter the third device and similar changes will happen. Finally, the ball leaves the contraption from the right side of the device N, and the state of this device always end with `A`.

Here is the summary:

- If the leftmost device is in state `A`, it will change into `B` and the other devices won't change.
- If the leftmost device is in state `B`, the states of devices $1, 2, \ldots, N-1$ after the operation is the negation of the states of devices $2, 3, \ldots, N$ after the operation, respectively. The state of the device N is always `A`.

In the latter case, we can see the operation as "delete the first character, negate all characters, and append `A` at the end". If we negate all characters at even indices beforehand, the only operations we need to do is "delete the first character and append a character `A` or `B` (depending on the parity of N)". This can be done in $O(1)$ using a deque.

This way the problem can be solved in $O(K)$. Also, notice that after $2N$ operations, the strings we get after each operation will be periodic, and the period is at most 2. Thus, we need only $O(N)$ simulations and this algorithm works in $O(N)$.

E: Increasing Numbers

An integer is called *Repunit* if it only consists of ones in its decimal representation. For example, $1, 11, 111, ...$ are Repunit.

It is easy to see that an integer is increasing if and only if the integer can be represented as the sum of at most 9 Repunit numbers. Thus, an integer N can be written as the sum of at most k increasing numbers if and only if N can be written as the sum of at most $9k$ Repunit numbers.

A Repunit number can be written of the form $(10^r - 1)/9$ using a non-negative integer r. Suppose that

$$N = \sum_{i=1}^{9k} (10^{r_i} - 1)/9$$

This is equivalent to

$$9N + 9k = \sum_{i=1}^{9k} 10^{r_i}$$

In order to check if such $r_1, \ldots, r_{9k}$ exist, we need to check if the sum of digits of the decimal representation of $9N + 9k$ is at most $9k$.

Let L be the number of digits of N. We can prove that the answer of the problem is at most L (always choose the greatest integer of the form $99\ldots9, 199\ldots9, \ldots, 999\ldots9$ greedily and subtract it from N). Thus, by using binary search, this problem can be solved in $O(L \log L)$.

It is also possible to solve this problem in $O(L)$. Instead of using binary search, compute the decimal representations of $9N$, $9N + 9$, $9N + 18$, ... one by one. The time complexity of one addition can be $O(L)$, but since the big carries will not happen after a big carry, its amortized complexity is $O(L)$.

We need big integers, but the required operations are extremely simple: we only need to increment big integers.

F: Train Service Planning

Consider a train that goes from station 0 to station N. For this train, define $p_0, \ldots, p_{N-1}$ as follows:

- The train departs station 0 at time p_0.
- After A_0 minutes, it arrives at station 1.
- The train waits for p_1 minutes there.
- After A_1 minutes, it arrives at statopn 2.
- The train waits for p_2 minutes there, and so on.

Of course, $p_1, \ldots, p_{N-1}$ must be non-negative.

Then, consider a train that goes from station N to station 0 in a strange way. We assume that this train goes from station 0 to station N, but spends negative amount of time. Define $q_0, \ldots, q_{N-1}$ as follows:

- The train departs station 0 at time $-q_0$.
- After $-A_0$ minutes, it arrives at station 1.
- The train waits for $-q_1$ minutes there.
- After $-A_1$ minutes, it arrives at statopn 2.
- The train waits for $-q_2$ minutes there, and so on.

Again, $q_1, \ldots, q_{N-1}$ must be non-negative.

What happens if the section s is single-tracked?

- One train leaves station $s-1$ at time $A_0 + \ldots + A_{s-1} + p_0 + \ldots + p_{s-1}$ and arrives at station s at time $A_0 + \ldots + A_s + p_0 + \ldots + p_{s-1}$.
- The other train "leaves" station $s-1$ at time $-(A_0 + \ldots + A_{s-1}) - (q_0 + \ldots + q_{s-1})$ and "arrives" at station s at time $-(A_0 + \ldots + A_s) - (q_0 + \ldots + q_{s-1})$.

The constraint on this single-tracked section is as follows: The value $p_0 + \ldots + p_{s-1} + q_0 + \ldots + q_{s-1}$ is not in the (open) interval $(-2(A_0 + \ldots + A_s), -2(A_0 + \ldots + A_{s-1})$, modulo K. Formally, no number in this open interval is equivalent to $p_0 + \ldots + p_{s-1} + q_0 + \ldots + q_{s-1}$ modulo M. Note that when $2A_s$ is greater than K, this constraint is unsatisfiable and you need to output -1.

Intuitively, the constraint only depends on the total waiting time in the stations $[0, s)$ (of two trains). Let $x_i = p_0 + \ldots + p_{s-1} + q_0 + \ldots + q_{s-1}$ (i.e., the total waiting time). The problem asks to find the minimum possible value of $x_{N-1} - x_0$ of non-decreasing sequence $x_0, \ldots, x_{N-1}$ under the constraints above.

Now, we can state a simpler general version of this problem: (Note that after the reduction this N corresponds to the number of single-tracked sections.)

You are given a modulo M and N intervals $[L_i, R_i]$. You are asked to find a non-decreasing sequence $x_0 \leq x_1 \leq \ldots \leq x_{N-1}$ that minimizes the value $x_{N-1} - x_0$, under the constraint that for each i, x_i must be in the interval $[L_i, R_i]$, modulo M. (Formally, there must be $L_i \leq y_i \leq R_i$ such that $y_i \equiv x_i \pmod{M}$.

The easier understanding of this problem is as follows. Suppose that someone is standing on a number line. He can move to the right at arbitrary speed (or he can stay at the same place), but he is not allowed to go left. At time i, he must be in the interval $[L_i, R_i]$, modulo M. What is the minimum distance he must travel?

Now, it is easy to prove that we are only interested in "important coordinates" $L_0, \ldots, L_{N-1}, R_0, \ldots, R_{N-1}$. We can assume that in one of optimal solutions all x_i are one of these values. Any reasonable DP that uses this fact will run in polynomial time and get 500 points.

How can we solve this problem faster? First we define values $dpL[i], dpR[i]$. $dpL[i]$ means the following: "Suppose that you are at L_i at time i. What is the minimum distance you have to travel in the future (to satisfy the constraints)? Define $dpR[i]$ similarly.

Unless you are forced to do so, it makes no sense to move. Instead, you can move to the right later when you are forced. Thus, we can compute the value of $dpL[i]$ in the following way. First, find minimum j such that $j > i$ and $L[i]$ is not in the interval $[L[j], R[j]]$ modulo M. (If such j doesn't exist, $dpL[i] = 0$. Then, $dpL[i] =$ (the distance you have to move from $L[i]$ to $L[j]$) + $dpL[j]$. The computation of dpR is similar. If you compute these values in the decreasing order of i, and if you use RMQ (Range Minimum Query) to find such j, it works in $O(NlogN)$.

How to get the answer? The optimal solution will look like one of the following:

- Stay at L_i until time i, and then move $dpL[i]$ in total. This is valid only when L_i is in the interval $[L_j, R_j$ for each $j < i$.
- Stay at R_i until time i, and then move $dpR[i]$ in total. This is valid only when R_i is in the interval $[L_j, R_j$ for each $j < i$.

Thus, there are only $2N$ possibilities. We can check the validity of each of these possibilty in $O(NlogN)$, again using RMQ. Therefore, this solution works in $O(NlogN)$.