AGC 022 Editorial (Japanese)

zscoder

2018 年 4 月 1 日

For International Readers: English editorial starts on page 10.

A. Diverse Word

S の長さが 26 未満の場合、辞書順で次の多彩な文字列は、S にまだ現れていない中で最も小さい文字を S の後ろに付け足したものです。

そうでない場合、S の長さは 26 です。$S = p_1p_2p_3...p_{26}$ として、$p_ip_{i+1}...p_{26}$ を S の接尾辞であって文字が左から右に降順に並んでいる（すなわち、$i \leq j \leq 25$ に対して $p_j > p_{j+1}$ である）ようなもののうち最長のものとします。$i = 1$ のとき、S は辞書順で最も大きい多彩な単語であるため、答えは -1 です。そうでなければ、$p_i, p_{i+1}, ..., p_{26}$ のうち p_{i-1} より大きいものの中で最も小さい文字を q として、辞書順で次の多彩な文字列は $p_1p_2...p_{i-2}q$ となります。（このような文字は必ず存在します。なぜなら、そうでないと仮定すると $p_{i-1} > p_i$ となって接尾辞の最長性に矛盾するからです。）

B. GCD Sequence

まず、各要素 a_i と残りの要素の和の最大公約数（以下 gcd）を考える代わりに、各要素 a_i とすべての要素の和 S の gcd を考えても構いません。$\gcd(a_i, S - a_i) = \gcd(a_i, S)$ であるためです。

実は、制約が解法のヒントを多少含んでいました。この厳しい制約は、1 から N までの数のうち 2/3 程度を選ぶ必要があるを意味します。

アイデアの重要部分は、S を何らかの小さい素数の倍数とすれば、a_i たちがその小さい素数で割り切れ、かつそれらの gcd が 1 であればよくなる、というものです。ここから、k を何らかの整数として $S = 6k$ とする案が浮かびます。すると、a_i がいずれも 2 か 3 の倍数であって、それらすべての gcd が 1 となるように選べばよいことになります。

偶然にも、1 から N までの数のうち 2/3 程度が 2 か 3 で割り切れます（つまり、$6k+2, 6k+3, 6k+4, 6k$ のいずれかの形式の数です）。さらに、2 と 3 を両方選べば、選んだ要素の gcd は必ず 1 となります。

これでほぼ解けました。あとは、N が小さいケースを処理して、S が確実に 6 の倍数になるようにすればよいです。$N \leq 5$ のときは、それぞれ $\{2, 5, 63\}, \{2, 5, 20, 63\}, \{2, 5, 20, 30, 63\}$ を選べばよいです（他にも解はあります。また、このうち最初の二つは都合よくサンプルにありました）。以降、$N \geq 6$ と仮定します。

$6k+2, 6k+3, 6k+4, 6k+6$ の形式の数を小さい順に選んでいきましょう。唯一の問題は、和が 6 にならないかもしれないことです。和を 6 で割った余りは $0, 2, 3, 5$ のいずれかになることが容易にわかるので、これらのケースを個別に処理しましょう。

- S が 6 で割り切れるとき: 何もする必要はありません。
- $S \equiv 2 \pmod 6$ のとき: 8 を集合から削除して、次の 6 の倍数を追加すればよいです。
- $S \equiv 3 \pmod 6$ のとき: 9 を集合から削除して、次の 6 の倍数を追加すればよいです。
- $S \equiv 5 \pmod 6$ のとき: 9 を集合から削除して、次の $6k+4$ の形式の数を追加すればよいです。

より単純な解もあります。$N = 3$ のときはサンプルにある解を使います。N が偶数のときは $2, 10, 3, 9, 4, 8, 6, 12, ...$（次の 8 個はこれらにそれぞれ 12 を足したもの、以下同様）の順に追加すればよく、N が奇数のときは $6, 2, 10, 3, 9, 4, 8, 12, ...$（次の 8 個はこれらにそれぞれ 12 を足したもの、以下同様）の順に追加すればよいです。

C. Remainder Game

まず、それぞれの数 v に対して、その数に x で割る操作を適用したら、それ以降その数を $y \geq x$ で割る操作を適用しても値が変わらないため無意味です。したがって、操作は k の (strict な) 降順に行われると仮定して構いません。特に、それぞれの値はたかだか一度しか使われません。

$2^x > 2^{x-1} + 2^{x-2} + ... + 2^0$ より、実質的には a_i を b_i に変える辞書順最小の操作列を求めることになります。

明らかに、50 より大きい数を使うことはありません。次の問いに答える必要があります。

操作で使う数の集合 S を固定したとする。S の要素のみを使って目標を達成できるか？

この問いに答えることができれば、まず 50 から始めて、$\{1, 2, ..., 49\}$ を操作で使って目標を達成できるか判定します。可能なら 99 に進み、$\{1, 2, ..., 48\}$ を操作で使って目標を達成できるか判定します。使わざるを得ない整数が出てきたら、それを S に恒久的に追加します。この過程を続けることで、目標を達成するのに必要な辞書順最小の数の集合が求まります。目標が達成不可能であるかどうかは、$S = \{1, 2, ..., 50\}$ に対して同じ問いに答えることでわかります（ある単純な条件の成立を確認してもわかりますが)。

では、部分問題の解き方に移りましょう。次のような有向グラフを考えます。頂点には 0 から 50 の番号がついていて、各整数 v に対して、すべての $t \in S$ について v から $v \bmod t$ への有向辺があります。目標が達成可能なのは、各 b_i が a_i から到達可能なときで、そしてそのときに限ります。(必要性はグラフの定義から明らかで、十分性も、各ペアが到達可能なときは同じ値による操作を一度の操作でまとめて行えるため、明らかです。）これは BFS か DFS で確認できます。

最悪ケースでは、部分問題を $O(N)$ 回解くことになります（便宜上 $N = 50$ とします)。それぞれの部分問題を解くために、関係する頂点で BFS か DFS を行うか、Floyd-Warshall のアルゴリズムを実行することができます。辺が $O(N^2)$ 本存在しうるため、これらのアルゴリズムはどちらも最悪ケースで $O(N^3)$ 時間を要します。したがって、アルゴリズム全体の実行時間は $O(N^4)$ となります（かかる定数は小さいですが)。

部分問題ごとにグラフを再構築したり BFS/DFS をやり直したりしないことで、問題を全体で $O(N^3)$ 時間で解くことも可能ですが、これは読者への練習問題とします。

D. Shopping

本質的には、すべての駅を一度以上訪れて帰れるまでの電車の最小の往復回数を求めていることに注意します。電車が路線を一方向に走りきる回数を求めて L 倍することにします（とはいえこの回数は偶数なので答えは $2L$ の倍数です)。

まず、二乗時間の解法を説明します。座標 0 にダミーの駅を置きます。各駅について、電車がその駅に右から入って右から出ていくなら値 $+1$ を（最初、電車は座標 0 に右から入るものとします)、電車がその駅に左から入って左から出ていくなら値 -1 を、その他の場合（左から入って右から出るかその逆）は値 0 を割り当てます。電車がとりうる経路は、すべての駅への値 $1, 0, -1$ の割り当てであって、次の条件を満たすようなものとして表せます：駅の列のどの prefix（全体を除く）についても値の和が正であり、すべての値の和が 0 である。それぞれの値を各駅に割り当てるコストは値 x_i, t_i を用いて $O(1)$ 時間で求められます。ここで、$dp[i][j]$ を「最初の i 駅に値を割り当てて、それらの和を j とするのに必要な最小のコスト」とすると、$O(N^2)$ 時間の解法が得られ、部分点を取る上では十分です。

では、線形時間の解法に移ります。

まず、すべての t_i を $2L$ で割って余りをとります。ただし、t_i が $2L$ で割り切れる場合、その駅を取り除くわけにはいかない（一度は通るようにしなければならない）ので注意が必要です。以下、$0 \leq t_i < 2L$ とします。

各駅は組 (x, y) で特徴付けられます。ここで、x は、電車がその駅に左から来たときに降りて買い物をした場合に、次の電車が右から来るなら、つまり $t_i \leq 2(L - x_i)$ であれば 0 であり、次の電車が（次の往復に入ってから）左から来るなら x は 1 です。y は、電車がその駅に右から来たときに降りて買い物をした場合に、次の電車が左から来るなら、つまり $t_i \leq 2x_i$ であれば 0 であり、そうでなければ x は 1 です。($2L$ で割ったあと）$t_i = 0$ であるような駅は $(1, 1)$ の駅として、あとで答えを適切に減らすことにしましょう。

電車の行程は、左端（座標 0）から出発してこれらの駅を何らかの順番に訪れるような動く点とみなせます。$x = 0$ であるような駅に左から入ると、進行方向が反転して右から左に動くようになり、$x = 1$ であるような駅に左から入ると、そのまま通り抜けてコストが 1 増加します。駅に右から入る場合も同様です。点が左端や右端（座標 $0, L$）に達すると、やはり進行方向が反転してコストが 1 増加します。我々の目標は、行程の総コストを最小化することです（最後に答えに適切な定数を足し、L 倍します)。

$(1, 1)$ の駅はほぼ無視できることに注意します。単に通り抜けてコストが 1 増えるだけだからです。ただし、経路がすべての駅を一度は必ず通るようにあとで考慮する必要があります。当面の間、$(1, 1)$ の駅は存在しないと仮定します。

また、定義より $(1, 0)$ の駅は $(0, 1)$ の駅より前に現れることがない点にも注意します。

駅の並びが $(0,1),(0,1),...,(0,1),(0,0),(1,0),...,(1,0)$ という形であると仮定します。ここで、$(0,0)$ の駅の数は 0 か 1 であるとします。S を駅の数とします。ここでのアイデアは、それぞれの駅について、その駅はコスト 1 で外側に通り抜けるか、その駅でコスト 0 で外側方向に進行方向が反転するかのどちらかである、というものです。ところが、このような形で駅が配置されているため、次の駅に追加コストを発生させずに入ることはできません（次はより外側にある駅をコスト 1 で通り抜けるか、座標 0 か L で反転するかです)。駅が追加コストを発生させないのは、それが最後に訪れる駅である場合のみです。しかし、$(1,0)$ の駅が一つ以上存在する場合、少なくとも一度は右端で反転しなければならないことがいえ、少なくとも $S+1$ のコストが発生することが証明できます。$(1,0)$ の駅が存在しない場合は、以下の場合を除き最小コストは $S-1$ です。その場合というのは、最後の駅（$(0,1)$ か $(0,0)$）より遠くに駅が存在し、その駅に一度は寄らなければならない場合で（$(1,1)$ の駅か、後述の「削除済み」駅のために起こる)、この場合の答えは変わらず $S+1$ となります。

一般には、駅の並びは $(0,0),(0,1)$ や $(0,0),(0,0)$ や $(1,0),(0,0)$ といった形の部分列を含むことがあります。駅の並びからこれらのペアを最も多く組む方法であって、最後のペアの一番目の駅の添え字が最小のものを求めましょう。これは線形時間で貪欲に行うことができます。ここでのアイディアは、これらのペアはコストに影響を与えずに駅の並びから取り除くことができる、というものです。なぜなら、ペアの二番目の駅に先に行き、反転して一番目の駅に行ってそこからまた反転することができるからです。よって、これらの駅は無視することができ、残りの経路が一番目の駅を一度は通ることさえ保証すればよいです。ペアを最も多く組むのが最適なのは、これらのペアを取り除くと答えは $S+1$ か $S-1$ になるため、常に S を最小化するべきだからです。

これらのペアを取り除いたあと、$(0,1),(0,1),...,(0,1),(0,0),(1,0),...,(1,0)$ という形の駅の並びが残り、ここで $(0,0)$ の駅の数は 0 か 1 です。ここでの最小コストが $S-1$ か $S+1$ のどちらになるかを計算すればよいです。この解法の所要時間は $O(N)$ です。

E. Median Replace

まず、どのようなビット列が美しいか分析しましょう。

筆者の解法より説明しやすいため、ここでは admin の解法を説明します。

文字列を、最初の 1 の前の 0 の数、隣り合う 1 と 1 の間の 0 の数、最後の 1 の後の 0 の数を並べることで、配列として表しましょう。例えば、文字列 0010001011 に対応する配列は $[2,3,1,0,0]$ です。

文字列に対する操作を、配列に対する操作に「翻訳」することができます。行えるのは、配列中の $x \geq 3$ を $x-2$ に変えること、連続する二つの要素 x, y を $x+y-1$ に置き換えること、両端以外で二つ連続する 0 を削除すること、です。

2 より大きい奇数の要素をすべて 1 まで減らし、2 より大きい偶数の要素をすべて 2 まで減らしても操作に本質的な影響はないため、以下、配列の全要素は必ず区間 $[0,2]$ 内にあるとします。

すると、行える操作は以下のように簡略化されます：二つの連続した偶数であって「両方とも 0」ではないものを 1 で置き換える、1 を配列から削除する、両端以外で二つ連続する 0 を削除する。目標は、すべてが 0 であるような配列を得ることです。

1 は即座に取り除いてよいことに注意します。したがって、以下では配列は 0 と 2 のみを含むと仮定します。

実は、以上のように縮小された配列 $[a_1, a_2, ..., a_k]$ は、二つの添え字 $p < q$ であって $a_p = a_q = 0$ であり、p が奇数、q が偶数であるようなものが存在するとき、そしてそのときに限って目標を達成できます。

実際、この条件を満たさない配列にどのように操作を行っても、得られる配列はやはり条件を満たさず、また、条件を満たさないような小さい配列は美しくないことがわかります。さらに、この条件が満たされる場合、a_p と a_q の間の要素をすべて削除し、a_p より前の要素をすべて削除し（ただし先頭に 0 を二つ残しても可)、最後に後ろから要素を削除していく（ただし末尾に 0 を二つ残しても可）ことで配列のすべての要素を 0 にすることができます。

まとめると、はじめに最初の 1 の前の 0 の個数、1 と 1 に挟まれた 0 の個数、最後の 1 の後の 0 の個数を求め、配列に書き込みます。次に、1 より大きい偶数をすべて 2 に減らし、1 より大きい奇数をすべて 1 に減らし、1 をすべて削除します。残った配列に $a_p = a_q = 0, p < q, p$ が奇数で q が偶数であるような p, q があれば、その文字列は美しく、なければ美しくありません。

この判定条件を用いて、単純な DP を行うことで美しい文字列の数を $O(N)$ 時間で数えることができます。

筆者はこの問題に別の方向からアプローチしました。結果は以下の通りです。

・文字列が悪い（美しくない）のは以下の条件が満たされる場合で、またその場合に限られる。

- 文字列中で 1 が連続するのは最大で 3 個までである。
- 文字列が 3 個の連続する 1 を含むなら、それより左側の部分と右側の部分の長さは偶数でなければならず、またそれらは 00-bad （$b_i \in \{00, 01, 10\}$ として $0b_1b_2...b_k0$ という形の文字列）でなければならない。
- 文字列が 2 個の連続する 1 を含むなら、偶数長の部分は 00-bad でなければならず、奇数長の部分は悪くなければならない。
- 文字列が 2 個の連続する 1 を含まないなら、1 で始まって 1 で終わってはならない。

この性質を用いれば、$O(N)$ 時間で動作する DP の解法を素直に書くことができます。

F. Checkers

駒の位置は、係数（桁の値）が負になることを許した X 進法で考えることができます。すなわち、はじめ i 番目の駒の位置は $(0, 0, ..., 0, 1, 0, ..., 0)$ で表されます。ここで、1 となっているのはタプルの i 番目の要素です。

位置 a の駒が位置 b の駒を飛び越すとき、新たな駒が位置 $2b - a$ に生まれ、元々の二つの駒は取り除かれます。$N - 1$ 回の操作後の最後のタプルがどのようなものになるか分析しましょう。

重要なのは、タプルに現れる係数からなる多重集合だけである点に注意します。ある係数の多重集合が達成可能であると分かれば、その係数を並び替える方法の数をかけて答えに足せばよいです。したがって、以降は係数の多重集合がどのようになるかのみを考えます。

はじめ、$\{2^0\}$ が N 個あります。各ステップでは、二つの集合 A, B を新たな集合 $2B \cup -A$（重複した要素はそれぞれ加える）に併合します。最初にわかることは、係数は必ず 2 の累乗であることです。次にわかることは、どの集合の係数の和も常に 1 であることです。

さらに、どの集合に対しても、もし 2^i か -2^i がその集合に存在するなら、2^{i-1} か -2^{i-1} も必ずその集合に存在することにも注意します。最後に、集合には $\pm 2^0$ が必ずちょうど一つ存在します。これらはすべて必要条件であり、そのことは容易に示せます。

では、上記の条件を満たす集合はすべて最後に残りうる集合なのでしょうか？答えは No です。最小の反例は $\{-2^0, -2^1, -2^1, -2^1, -2^1, -2^1, 2^2, 2^3\}$ で、どのように操作してもこの係数の集合を得ることはできません。というわけで、必要条件をさらに見つけなければなりません。

この問題の難所は、以下の必要条件を見つけるところです。

すべての整数 i に対し、集合中の要素 $\pm 2^i$ の符号を変えることで、$0 \leq j \leq i$ であるような要素 $\pm 2^j$ の和を 1 にすることができる。

この条件を見つけたら、これが必要であることを証明するのは難しくありません（この条件を満たす二つの集合を併合するとどうなるか考えてみてください）。

そして実は、この条件は十分でもあります。数学的帰納法によりこれを示します。ここでのアイデアは、現時点での集合を $2B, -A$ という形の二つのよい部分集合に分けることが常に可能である、というものです。S を現時点での集合とします。$-2^0 \in S$ である場合のみを考えます（もう一つの場合も同様であるため）。k を、$+2^k$ が集合に存在するような最小の整数とします。もし、$1 \leq i < k$ のすべてに対し、S に -2^i が二度以上現れる場合は、$B = \{-2^0, -2^1, ..., -2^{k-2}, +2^{k-1}\}$ として、残りの要素を（符号を反転させて） A とします。もし、-2^i が S にちょうど一度現れるような i が存在する場合は、そのよう

な i のうち最小のものを考えます。$i = 1$ なら、$B = \{-2^0\}$ を選んで続けます。一方、$i > 1$ のときは $-2^0 - 3(2^1 + ... + 2^{i-1}) = -2^i + 1 - 2(2^1 + ... + 2^{i-1}) < -2^i - 1$ であるため、-2^i の符号を変えて和を 1 にすることは不可能であり、条件に矛盾します。

よい係数の集合の判定条件がわかったので、ありうるタプルの数を数える段階に進めます。小さい方から大きい方へと 2 の累乗を加えていくことで、タプルを構築します。また、必要が生じるたびに二項係数を掛けます。$dp[i][j]$ を、長さ i のありうるタプルであって、含まれる 2 の累乗たちの和が $1 + j \cdot V$ であるようなものの数とします。ここで、直前にタプルに含まれていた最大の 2 の累乗を $\frac{V}{2}$ としています（DP の遷移を行う上で、V の値を知る必要はないことに注意してください）。DP の遷移には、使われる $+V$ と $-V$ の数を列挙し、$\pm V$ の符号を付け替えて和を 1 とすることができるかを確認すればよいです。各状態から $O(N^2)$ 個の遷移があり、状態の数は $O(N^2)$ 個であるため、時間計算量は $O(N^4)$ となります（$O(N^5)$ の解法も前計算をせずとも問題なく通ります）。

AGC 022 Editorial

zscoder

30 March 2018

A. Diverse Word

If the length of the string is strictly less than 26, then the next diverse string in lexicographical order is the string appended by the smallest letter that has not yet appeared in the string.

Otherwise, the string has length 26. Let $S = p_1p_2p_3...p_{26}$ and suppose $p_ip_{i+1}...p_{26}$ is the longest suffix such that the letters are in descending order from left to right, i.e. $p_j > p_{j+1}$ for $i \leq j \leq 25$. If $i = 1$, the answer is -1, because S is the largest possible diverse string. Otherwise, the next smallest diverse string is $p_1p_2...p_{i-2}q$, where q is the smallest letter among $p_i, p_{i+1}, ..., p_{26}$ that is strictly larger than p_{i-1}. (this letter exist because otherwise $p_{i-1} > p_i$, contradicting the maximality of the suffix).

B. GCD Sequence

Firstly, instead of considering the gcd of each element a_i and the sum of the remaining elements, we can consider the gcd of each element a_i with the sum of all elements, S, because $\gcd(a_i, S - a_i) = \gcd(a_i, S)$. The constraints actually contain a small hint to the solution. The tight constraints suggests that we have to take around two-thirds of the numbers between 1 to N.

The main idea is that if we pick S to be a multiple of some small primes, then we just have to ensure a_i are divisible by the small primes and the gcd of everything is 1. This inspires the choice of $S = 6k$, for some integer k. Then, all we need to do is to pick a_i such that they are all multiples of 2 or 3, and make sure the gcd of every element is 1.

Coincidentally, around two-thirds of the numbers from 1 to N are divisible by 2 or 3, i.e. the numbers of the form $6k + 2, 6k + 3, 6k + 4$ and $6k$. Furthermore, if we pick both 2 and 3, then the gcd of the chosen elements will always be 1.

We're almost done. We just need to take care of some small cases and ensure that S is a multiple of 6. For $N \leq 5$, we can take $\{2, 5, 63\}, \{2, 5, 20, 63\}, \{2, 5, 20, 30, 63\}$ respectively (there are also other constructions, and note that the first 2 are conveniently given in the samples). Thus, from now we assume $N \geq 6$.

Let's take the numbers of the form $6k + 2, 6k + 3, 6k + 4, 6k + 6$ from small to large. The only issue is that the sum might not be divisible by 6. It can be easily seen that the sum is either congruent to $0, 2, 3, 5$ modulo 6 respectively. Let's consider each of them separately :

- If S is divisible by 6, we're done.
- If $S \equiv 2 \pmod 6$, then we remove 8 from our set and insert the next multiple of 6.
- If $S \equiv 3 \pmod 6$, then we remove 9 from our set and insert the next multiple of 6.
- If $S \equiv 5 \pmod 6$, then we remove 9 from our set and insert the next number of the form $6k + 4$.

Here's another simpler construction :

For $N = 3$, use the construction in the samples. If N is even, we can add the numbers in the order $2, 10, 3, 9, 4, 8, 6, 12, ...$ (the next 8 terms are the first 8 terms added by 12 and so on). If N is odd, we can add the numbers in the order $6, 2, 10, 3, 9, 4, 8, 12, ...$ (the next 8 terms are the first 8 terms added by 12 and so on).

C. Remainder Game

First, note that for each number v, if we have applied an operation on it with x, then it doesn't make sense to apply an operation on it with any $y \geq x$, since it will not change the value anymore. Thus, we may assume that the operations are done in strictly decreasing value of k. In particular, each value should be used at most once.

Since $2^x > 2^{x-1} + 2^{x-2} + ... + 2^0$, essentially we need to find the lexicographically smallest sequence of operations to convert a_i into b_i.

Clearly, we will not use any numbers larger than 50. We need to answer the question :

Suppose we fixed a set of values S to use. Is it possible to achieve the goal using only the elements in S?

If we can answer this question, then we can start from 50 and determine if we can finish the task using the set $\{1, 2, ..., 49\}$ in our operations. If it is possible, we move on to 49 and determine if we can finish the task using $\{1, 2, ..., 48\}$ in our operations. Whenever we reach an integer that we must use, we add it to the set S permanently. Continuing this process, we get the lexicographically smallest set of values needed to achieve the goal. We can also check whether the goal is impossible by answering the same question with $S = \{1, 2, ..., 50\}$ (though there is also a simple condition to check it).

Now, we focus on answering the subproblem. Construct a directed graph where the vertices are the numbers 1 to 50 and for each integer v, there is a directed edge from v to $v \bmod t$ for all $t \in S$. The task is possible if and only if each integer b_i is reachable from a_i (one direction follows from the definition of the graph, the other direction is also obvious since if each pair can be reached individually, we can combine the operations with the same value into one operation). This can be checked with BFS or DFS.

In the worst case, we'll answer the question $O(N)$ times (we let $N = 50$ for convenience), and to answer each question, we can do a bfs or dfs on each relevant vertex or just run Floyd-Warshall algorithm. Both of these algorithms take $O(N^3)$ time in worst case, since there can be $O(N^2)$ edges. Thus, the entire algorithm runs in $O(N^4)$ time (albeit with a small constant).

It is also possible to solve the problem in $O(N^3)$ time by not reconstructing the graph and redoing bfs/df for each question. This is left as an exercise for the reader.

D. Shopping

Note that we are essentially just calculating the minimum number of round trips the train should make before we can visit every station at least once and go back home. We'll calculate the number of trips in one direction (call this a round) made by the train and multiply the answer by L. (though note that the number of trips must be even, so the answer is always divisible by $2L$)

First, we describe the quadratic-time solution. Add a dummy station at coordinate 0. For each station, we will assign a value $+1$ if in our trip, the train enters the station from the right and departs from the right from this station after shopping (assume that the train enters the station at coordinate 0 from the right at the start), -1 if the train enters the station from the left and departs from the left, and 0 otherwise (enters from left, departs from right or vice versa). A valid path of the train can be described as an assignment of values $1, 0, -1$ to the stations such that every proper prefix of stations have positive sum and the sum of all values are 0. The cost of assignment of each value to each station can be computed in $O(1)$ time using the values of x_i and t_i. Now, we can let $dp[i][j]$ be the minimum cost to assign values for the first i stations so that the sum is j. This solution works in $O(N^2)$ time. This is enough to get the partial points.

Now, we move on to the linear-time solution.

Firstly, we can take all t_i modulo $2L$. However, care must be taken when t_i is divisible by $2L$, since we can't just remove the station (we will have to make sure we pass through it at least once). Anyway, from here we will assume $0 \leq t_i < 2L$.

Each station can be characterized by a pair (x, y), where x is 0 if when we stop at the station when the train is travelling from the left, and after we finish shopping the next train will come from the right, i.e. $t_i \leq 2(L - x_i)$, and 1 if the next train will come from the left (after going through one more round). y is 0 if when we stop at the station when the train is travelling from the right, and after we finish shopping the next train will come from the left, i.e. $t_i \leq 2x_i$, and 1 otherwise. Let's treat the stations with $t_i = 0$ (after reduction mod $2L$) as $(1, 1)$ stations, and subtract the answer appropriately later.

We can consider the train trip as a moving point, where we start from the left side (coordinate 0), and enter these stations in some order. Whenever we enter a station with $x = 0$ from the left, we rebound in the opposite direction (so now it moves from right to left), while if we enter a station with $x = 1$ from the left, we pass through it and increase our cost by 1. The move pattern is similar when entering a station from the right. When our point hit the left and rightmost point (coordinates 0 and L), it rebounds and the cost is also increased by 1. Our goal is to minimize the total cost of the trip (in the end we add the answer

with an appropriate constant and multiply the answer by L).

Note that we can almost ignore $(1, 1)$ stations, as we just pass through them and increase the cost by 1. However, we have to make sure our path passes through all these stations at least once later. For now, suppose there are no $(1, 1)$ stations.

Note that by definition, $(1, 0)$ can never appear before $(0, 1)$ stations.

Suppose our sequence of stations is of the form $(0, 1), (0, 1), ..., (0, 1), (0, 0), (1, 0), ..., (1, 0)$, where there can be 0 or 1 $(0, 0)$ stations. Let S denote the number of stations. The idea is that for each station, either we pass through it with the side with cost 1, or we make a rebound on the side with cost 0. However, the arrangement of stations of this form means that it is impossible to enter the next station without additional cost (either you pass through the new station on the side with cost 1, in which case we can ignore it, or you rebound from coordinate 0 or L). The only way a station will not incur an additional cost is if it's the last station visited. However, if at least one $(1, 0)$ station exist, we can that we're forced to rebound from the right end at least once, so we can prove the cost is at least $S + 1$. If no $(1, 0)$ stations exist, then the minimum cost is $S - 1$, unless there exist a station beyond the last station $(0, 1)$ or $(0, 0)$ which we must touch at least once (caused by either a $(1, 1)$ station or removed stations that we will see later), in which case the answer is still $S + 1$.

In general, our sequence of stations can have subsequences of the form $(0, 0), (0, 1)$, $(0, 0), (0, 0)$ and $(1, 0), (0, 0)$. Let's find the maximum pairing of these subsequences in our sequence of stations, where the index of the first station of the last matched pair is minimal. This can be done in linear time greedily. The idea is that we can remove these subsequences without affecting our cost, because we can go to the 2nd station of the pair first, rebound to the first station of the pair, and rebound from there again. So, we can ignore these stations and only have to make sure our remaining path passes through the first station at least once. The maximum matching is optimal because once we remove these pairs, our answer is either $S + 1$ or $S - 1$, so it is always best to minimize S.

After removing these pairs, we're left with a sequence of stations of the form $(0, 1), (0, 1), ..., (0, 1), (0, 0), (1, 0), ..., (1, 0)$, where there can be 0 or 1 $(0, 0)$ stations, and we can compute whether the minimum cost is $S - 1$ and $S + 1$. This solution takes $O(N)$ time.

E. Median Replace

Firstly, we'll analyze which binary strings are beautiful.

Below, I will describe the admin's solution since it's simpler to describe.

Let's represent our string as an array, denoting the number of zeroes before the first occurrence of 1, between two consecutive 1s and after the last occurrence of 1. For example, the array for the string 0010001011 is $[2, 3, 1, 0, 0]$.

The operations on the string can be translated to operations on the array. We can either change $x \geq 3$ in the array to $x-2$, change replace consecutive elements x, y with $x + y - 1$, or delete two consecutive 0s that are in the middle of the array.

Below we will assume that all elements of the array are always in the range $[0, 2]$, where all odd elements larger than 2 has been reduced to 1 and all even elements larger than 2 has been reduced to 2, since it doesn't really affect our operations.

Then, the operations are simplified to replace two consecutive even numbers which are not both 0 with 1, remove 1 from our array and remove 2 consecutive 0s that are in the middle of the array. Our goal is to reach an array with all zeroes.

Note that we can remove all the 1s in the array immediately. Thus, from now we will assume our array only consists of 0s and 2s.

We claim that the reduced array $[a_1, a_2, ..., a_k]$ works if and only if there exist two indices $p < q$ such that $a_p = a_q = 0$, p is odd, q is even.

Indeed, note that any operation on an array that doesn't satisfy the condition will also give an array that doesn't satisfy the condition, and we can see that the small arrays not satisfying these conditions are bad. Also, if this condition holds, we can remove all the elements between a_p and a_q, remove all the elements before a_p (except potentially a pair of leading 0s), and finally remove elements from the back (except potentially a pair of suffix 0s) until all elements of the array becomes 0.

To summarize, first we find the number of 0s before the first 1, between each pair of 1s and after the last 1 and write it into an array. Then, we reduce all even numbers larger than 1 to 2 and reduce all odd numbers larger than 1 to 1. Remove all the 1s in the array. If in the remaining array we can find p, q such that $a_p = a_q = 0$, $p < q$, p odd and q even, then the string is beautiful. Otherwise, it is not.

With the criteria above, we can do a simple dp to count the number of beautiful strings in $O(N)$ time.

The author approached the problem from a different direction and here's the result :

A string is bad (not beautiful) if and only if it satisfies the following :

- It contains at most 3 consecutive ones.
- If it contains 3 consecutive ones, the parts to the left and right must have even length, and must be 00-bad (which is of the form $0b_1b_2...b_k0$, where $b_i \in \{00, 01, 10\}$).
- If it contains 2 consecutive ones, the even-length part must be 00-bad while the odd-length part must be bad.
- If it does not contain 2 consecutive ones, it must not both start and end with 1.

With this property, it is straightforward to write a dp solution that works in $O(N)$ time.

F. Checkers

We can consider the positions of the checkers in base-X, possibly with negative coefficients. Thus, initially the i-th checker can be represented by $(0, 0, ..., 0, 1, 0, ..., 0)$, where the i-th element of the tuple is 1.

When a checker at position a hops over a checker at position b, a checker is created at position $2b - a$ and the two previous checkers are removed. Let's analyze how the final tuple looks like after $N - 1$ operations.

Note that the only important thing is the multiset of coefficients that appear in the tuple. Once we know that a multiset of coefficients is achievable, we can multiply it by the number of ways to arrange the coefficients and add it to the answer. Thus, from now on we'll only consider how the multiset of coefficients look like.

Initially, we have N copies of $\{2^0\}$. Every step, we can merge two sets A, B into a new set $2B \cup -A$ (elements are counted with multiplicity). The first observation is that the coefficients are always powers of 2. The next observation is that the sum of coefficients of every set is always 1. Also, note that for any set, if 2^i or -2^i exist in the set, then 2^{i-1} or -2^{i-1} must exist in the set. Finally, $\pm 2^0$ must appear exactly once in the set. All these are necessary conditions and are very easy to prove.

However, are all sets satisfying the above conditions valid final sets? The answer is no. The smallest counterexample is the set $\{-2^0, -2^1, -2^1, -2^1, -2^1, -2^1, 2^2, 2^3\}$. No matter how you try, you can never form this set of coefficients by the operations. Thus, we still need to find more necessary conditions.

The hard part of the problem is to find the following necessary condition :

For all integers i, it is possible to change the signs of the elements $\pm 2^i$ in the set such that the sum of the elements $\pm 2^j$ with $0 \le j \le i$ is equal to 1.

Once you find this condition, it is not hard to prove that it is necessary (just consider what happens when we merge two sets satisfying this condition).

In fact, this condition is also sufficient. We can use induction to prove this fact. The idea is that we can always find a partition of the current set into two good subsets of the form $2B$ and $-A$. Let S be our current set. We'll only look into the case where $-2^0 \in S$ since the other case is similar. Let k be the smallest integer such that $+2^k$ exist in the set. If for all $1 \le i < k$, -2^i appears at least twice in S, then we can choose $B = \{-2^0, -2^1, ..., -2^{k-2}, +2^{k-1}\}$ and the remaining elements (negated) goes to A. If some i exist so that -2^i appears exactly once in S, then consider the smallest such i. If $i = 1$, we can just pick

$B = \{-2^0\}$ and proceed. However, if $i > 1$, then $-2^0 - 3(2^1 + ... + 2^{i-1}) = -2^i + 1 - 2(2^1 + ... + 2^{i-1}) < -2^i - 1$, so it is impossible to change the sign of -2^i so that the sum is 1, which contradicts the condition.

Once we have the criterion of good sets of coefficients, we can proceed to the count the number of valid tuples. We build the tuples starting from the smallest power to the largest power. We will multiply binomial coefficients whenever necessary. Let $dp[i][j]$ be the number of valid tuples of length i such that the sum of the powers are $1+j \cdot V$, where the previous largest power of 2 is $\frac{V}{2}$. (Note that it is not necessary for us to know the value of V to do the dp transitions). For the dp transitions, we just iterate through the number of $+V$ and $-V$ taken and check whether it is possible to reassign the signs for the $\pm V$s so that the sum is 1. The dp transitions are $O(N^2)$ and there are $O(N^2)$ states, so the time complexity is $O(N^4)$. ($O(N^5)$ solutions can also pass without problems even without precomputation)