Mujin Programming Challenge 2017 Editorial

writer : sugim48

2017 年 2 月 25 日

For International Readers: English editorial starts on page 8.

A: Robot Racing

まず，部分点解法を説明します．$N \leq 8$ なので，$N!$ 通りの順列をすべて試すことができます．各順列について，その順番通りにロボットがゴールできるかを判定すればよいです．

次のことが示せます : ロボット k が最初にゴールできるための必要十分条件は，各 $i < k$ について $x_i \geq 2i-1$ が成り立つことである．条件 $x_i \geq 2i-1$ が成り立っている場合，各 $i < k$ についてロボット i を座標 $2i-1$ へ動かした後，ロボット k をゴールさせることができます．成り立っていない場合，$x_j < 2j-1$ なる j $(< k)$ が存在します．ロボット k が各 $i < k$ を飛び越えてゴールできたと仮定します．ロボット k がロボット i を飛び越えた時点での，ロボット i の座標を y_i とおきます．すると，$0 < y_1$, $y_2 - y_1 \geq 2$, $\ldots$, $y_j - y_{j-1} \geq 2$, $y_j < 2j-1$ が示せますが，これは矛盾です．

よって，順列 $p_1, \ldots, p_N$ の順番通りにロボットがゴールできるかを判定するには，次のようにすればよいです :

- まず，ロボット p_1 について上述の条件が成り立っているかをチェックする．
- 次に，ロボット p_1 を取り除き，ロボット p_2 について上述の条件が成り立っているかをチェックする．
- 以下同様．

満点解法は次の通りです．まず，最初にゴールできるロボットの個数を数えます．これらのロボットから 1 体選んで最初にゴールさせる必要があります．このとき，どのロボットを選んで最初にゴールさせても，残りのロボットがゴールする順番の通り数は変わらないことに注意してください．よって，適当なロボットを選んで最初にゴールさせ，そのロボットを取り除くことができます．元の問題の答えは，残り $N-1$ 体のロボットがゴールする順番の通り数に，最初にゴールさせるロボットの通り数を掛けたものになります．

このアイデアを愚直に実装すると，計算量が $\mathrm{O}(N^2)$ となってしまいます．計算量が $\mathrm{O}(N)$ の実装例を以下に示します．スタックを用意して，ロボットを左から右へと 1 体ずつ push していくことにします．スタックに新しくロボットを push したとき，もしそのロボットがスタックのうち k 番目の要素であり，その座標が $2k-1$ より小さいならば，そのロボットを pop し，答えを k 倍します．なぜならば，これら k 体のロボットのうちどれか 1 体を取り除かないといけないからです．すべてのロボットをスタックに push した後，スタックに入っているロボットは任意の順番でゴールさせられます．よって最後に，残ったロボットの個数の階乗を答えに掛ければよいです．

B: Row to Column

初期状態において，白いマスを 1 個でも含む列の本数を c とします．これら c 本の各列について，操作を行って配色を変える必要があります．

黒いマスだけからなる行を少なくとも 1 本作るために，必要な操作回数の最小値を k とします．最初の k 回の操作中に，白いマスを 1 個でも含む列の本数が減ることはありません．(行 i から列 j へ操作を行ったとすると，列 i には白いマスが含まれていたはずなので，列 j にも白いマスが含まれてしまいます．) また，真っ黒な列へ対して無駄な操作を行わない限り，白いマスを 1 個でも含む列の本数が増えることもありません．よって，k 回の操作の後，追加でちょうど c 回の操作を行えば，マス目全体を真っ黒にすることができます．今の目標は，k を最小化することです．

行 r をひとつ固定します．この行を真っ黒にするために必要な操作回数の最小値を求めましょう．

- もしマス目全体が真っ白ならば，明らかに不可能である．
- 初期状態で行 r に含まれる白いマスの個数を x とおく．
  - もし列 r に黒いマスが存在するならば (このマスを s とおく)，マス s を含む行を用いて x 回操作を行うことで，行 r を真っ黒にすることができるので，答えは x．
  - 存在しないならば，列 r に黒いマスを作るために事前にもう 1 回操作が必要なので，答えは $x+1$．

解法をまとめると次のようになります．マス目全体が真っ白の場合，答えは -1 です．そうでなければ，N 行を全探索し，各行についてその行を真っ黒にするために，必要な操作回数の最小値を求めます．これら N 個の値の最小値に c を足せば，それが答えです．

C : Robot and String

文字列 s の l 文字目から $r-1$ 文字目まで (ともに 0-indexed) の連続した部分文字列を，$s[l,r)$ と書くことにします．例えば，“abcde”$[1,4)$ = “bcd” です．また，文字列 t をロボットに処理させて最終的に得られる文字列を，$f(t)$ と書くことにします．

各 l $(0 \leq l \leq |s|)$ について，$f(s[l,r))$ が空文字列となる最小の r $(> l)$ を，$next(l)$ と定義します．ただし，そのような r が存在しない場合，$next(l) = \infty$ と定義します．また，$next(\infty) = \infty$ と定義します．すべての l について $next(l)$ が求まれば，ダブリングを用いることで，各質問クエリに高速に答えられます (ダブリングの方法については後述)．そのため，目標はすべての l について $next(l)$ を求めることです．

各 l $(0 \leq l \leq |s|)$ と各英小文字 c について，$f(s[l,r))$ が 1 文字の c となる最小の r $(> l)$ を，$next_c(l)$ と定義します．そのような r が存在しない場合，$next_c(l) = \infty$ と定義します．また，$next_c(\infty) = \infty$ と定義します．すると，例えば s の l 文字目 (0-indexed) が $\mathtt{x}$ の場合，各 $next_c(l)$ や $next(l)$ は次のように順に計算できます．

- $next_{\mathtt{x}}(l) = l + 1$
- $next_{\mathtt{y}}(l) = next_{\mathtt{x}}(next_{\mathtt{x}}(l))$
- $next_{\mathtt{z}}(l) = next_{\mathtt{y}}(next_{\mathtt{y}}(l))$
- $next(l) = next_{\mathtt{z}}(next_{\mathtt{z}}(l))$

- $next_{\mathtt{a}}(l) = next_{\mathtt{a}}(next(l))$
- $next_{\mathtt{b}}(l) = next_{\mathtt{a}}(next_{\mathtt{a}}(l))$
- $\vdots$
- $next_{\mathtt{w}}(l) = next_{\mathtt{v}}(next_{\mathtt{v}}(l))$

よって，動的計画法を用いて l が大きい方から順に計算すると，すべての l について $next_c(l)$ や $next(l)$ が求まります．この動的計画法の計算量は $\mathrm{O}(|s|A)$ です (A はアルファベットサイズ)．

最後に，ダブリングを用いて各質問クエリに高速に答える方法を説明します．各 l $(0 \leq l \leq |s|)$ について，

- $next^1(l) = next(l)$
- $next^2(l) = next(next(l))$
- $next^3(l) = next(next(next(l)))$
- $\vdots$

と定義します．ここで，$next^{2t}(l) = next^t(next^t(l))$ に注目し，各 l について $next^1(l)$, $next^2(l)$, $next^4(l)$, $next^8(l)$, ... を前計算しておくことを考えます．これは，動的計画法を用いて l が大きい方から順に計算できます．この動的計画法の計算量は $\mathrm{O}(|s| \log |s|)$ です．

各 l について $next^1(l)$, $next^2(l)$, $next^4(l)$, $next^8(l)$, ... が求まれば，各質問クエリに高速に答えることができます．質問クエリの内容は，「$next^t(l) = r$ となるような t が存在するか？」と言い換えることができます．これは，次のような手順で高速に判定できます．まず，$x \leftarrow l$ と初期化し，$2^k \geq |s|$ なる最小の k を選びます．その後，

- $next^{2^k}(x)$ が r を超えないならば $x \leftarrow next^{2^k}(x)$ とする．
- $next^{2^{(k-1)}}(x)$ が r を超えないならば $x \leftarrow next^{2^{(k-1)}}(x)$ とする．
- $\vdots$
- $next^2(x)$ が r を超えないならば $x \leftarrow next^2(x)$ とする．
- $next^1(x)$ が r を超えないならば $x \leftarrow next^1(x)$ とする．

という処理を順に行います．最後に $x = r$ であるか判定すればよいです．この手順を用いてすべてのクエリに答える計算量は $\mathrm{O}(Q \log |s|)$ です．

全体の計算量は $\mathrm{O}(|s|A + (|s| + Q) \log |s|)$ であり，十分高速です．

D : Oriented Tree

まず，D の最小値を求めましょう．パス (s,t) の長さを $dist(s,t)$ と書くことにします．すると，$d(s,t) + d(t,s) = dist(s,t)$ が成り立ちます．よって，T の直径を $diam$ と書くことにすると，$D \geq \lceil diam/2 \rceil$ です．一方，T の辺を「交互に」向き付けると，実際に $D = \lceil diam/2 \rceil$ となります．よって，D の最小値は $\lceil diam/2 \rceil$ です．

では，D が $\lceil diam/2 \rceil$ となるような T の向き付けを数えましょう．T の向き付けをそのまま数えるのは難しいので，T の向き付けと一対一に対応する数列 h を代わりに数えましょう．T の向き付けをひとつ固定したとき，長さ N の数列 h を次のように定義します．

- ある基準となる頂点 v_0 について，$h(v_0) = 0$
- T の各有向辺 $(u \to v)$ について，$h(u) - h(v) = 1$

すると，$d(s,t) = (dist(s,t) - h(s) + h(t))/2$ が成り立つことが分かります．よって，

$$\max\{(d(s,t), d(t,s)\} = \frac{dist(s,t) + |h(s) - h(t)|}{2}$$

です．よって，D が $\lceil diam/2 \rceil$ となるための h の必要十分条件は

$$\frac{dist(s,t) + |h(s) - h(t)|}{2} \leq \left\lceil \frac{diam}{2} \right\rceil$$

$$\Leftrightarrow \quad |h(s) - h(t)| \leq 2 \left\lceil \frac{diam}{2} \right\rceil - dist(s,t) \tag{1}$$

となります．条件 (1) を満たす h を数えるのが目標です．

まずは，$diam$ が偶数の場合を考えましょう．この場合，T の中心は 1 個だけ存在するので，その頂点を r とします．$diam$ が偶数の場合，条件 (1) は

$$|h(s) - h(t)| \leq diam - dist(s,t) \tag{2}$$

となります．まず分かるのは，直径の両端となりうる頂点たち，つまり r からの距離が $diam/2$ の頂点たちの間では，h の値は同一でなければならないということです．簡単のため，この値を 0 とします．さらに，各頂点 v について，$|h(v)| \leq diam/2 - dist(r,v)$ でなければならないということも分かります．逆に，すべての頂点についてこの条件が成り立っていれば，条件 (2) は成り立つことが示せます．よって，この条件と，「隣り合う頂点どうしの h の値の差が 1」という条件を満たすような h を数えればよいです．これは，r を根とした木 DP によって，$O(N^2)$ 時間で計算できます．

$diam$ が奇数の場合もほぼ同様です．この場合，T の中心は 2 個存在するので，その頂点を s, t とします．各頂点 v について，$dist(s,v) < dist(t,v)$ ならば v は黒，そうでなければ v 白とします．

すると，数列 h は次の少なくとも一方を満たさなければなりません (証明は $diam$ が偶数の場合と同様です)．

- 隣り合う頂点の各組 (s,t) について, $|h(s) - h(t)| = 1$.
- 白い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v)$.
- 黒い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v) + 1$.

または

- 隣り合う頂点の各組 (s,t) について, $|h(s) - h(t)| = 1$.
- 白い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v) + 1$.
- 黒い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v)$.

もちろん，これら 2 つの場合の共通部分を差し引く必要があります．数列 h 全体に定数だけずらしたものは区別しないため，共通部分を差し引く際には注意する必要があります．共通部分は，

- 隣り合う頂点の各組 (s,t) について, $|h(s) - h(t)| = 1$.
- 白い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v)$.

- 黒い各頂点 v について, $|h(v)+1| \leq diam/2 - dist(r,v)$.

かつ

- 隣り合う頂点の各組 (s,t) について, $|h(s)-h(t)| = 1$.
- 白い各頂点 v について, $|h(v)| \leq diam/2 - dist(r,v)$.
- 黒い各頂点 v について, $|h(v)-1| \leq diam/2 - dist(r,v)$.

Mujin Programming Challenge 2017 Editorial

sugim48

February 25th, 2017

A: Robot Racing

First we'll describe a solution for partial score. Since $N \leq 8$, we can try all $N!$ permutations, and for each permutation we want to check whether the frogs can finish the race in this order.

We can prove the following: The frog k can finish the race first if and only if for each $i < k$, $x_i \geq 2i - 1$. When the condition $x_i \geq 2i - 1$ is satisfied, we can first move the frog i to $2i - 1$ and then move the frog k to the goal (i.e., positions with non-positive coordinates). Otherwise, there exists j such that $x_j < 2j - 1$. Suppose that the frog k jumped over the frog i when the frog i is at y_i. We can show that $0 < y_1, y_2 - y_1 \geq 2, \ldots, y_j - y_{j-1} \geq 2, y_j < 2j - 1$. This is a contradiction.

Thus, in order to check whether the frogs can finish the race in the order $p_1, \ldots, p_N$, we should check the following:

- First, for the frog p_1, we check whether the conditions above is satisfied.
- Then, remove the frog p_1 and check the conditions for the frog p_2.
- And so on.

The solution for the full score is as follows. First, count the number of frogs that can finish the race first. We should choose one of them as the first frog that finish the race. Notice that the number of valid orderings of the remaining frogs doesn't depend on the choice of this first frog. Thus, we can fix the first frog (the choice is arbitrary) and remove it. The answer for the original problem is the product of the answer for the remaining $N - 1$ frogs and the number of valid choices of the first frog.

The straightforward implementation of this idea is $O(N^2)$. An example of $O(N)$ implementation is as follows. Create a stack, and we push the frogs from the left to the right one by one. When we push a frog into the stack, if the frog is the k-th element in the stack and its coordinate is smaller than $2k - 1$, we pop the frog from the stack and multiply the answer by k (since one of these k frogs must be the next frog to be removed). After we push all the frogs,

the remaining frogs in the stack can finish the race in arbitrary order. Thus, we multiply the answer by the factorial of the number of remaining frogs.

B: Row to Column

Let c be the number of columns that contain at least one white square in the initial state. For each of these c columns, we need to change the colors by performing operations.

Let k be the smallest integer such that after k operations, there exists at least one row that consists only of black squares. In the first k operations, the number of columns that contain at least one white square doesn't decrease. (When an operation is performed on the row i and the column j, after the operation the column j contains at least one white square because at least one square in the row i is white). Also, unless we make meaningless operations on entirely black columns, the number of such columns doesn't increase. Thus, after the k operations we need c more operations to achieve the goal, and our objective is to minimize the value of k.

Fix a row r. We want to compute the minimum number of operations required to make this row black.

- If the entire grid is white, obviously we can't do anything.
- Let x be the number of white squares in the row r in the initial grid.
  - If there exists a black square in column r (let's call this square s), we can make the row r black in x steps by performing operations using the row that contains the square s.
  - Otherwise, we need extra one step to create a black square in column r. Thus the answer is $x + 1$.

The solution of the problem will look as follows. In case all the squares are white, print -1. Otherwise, we check all N rows, and for each row we compute the minimum number of operations required to make this row black. The answer is the minimum of these N numbers plus c.

C: Robot and String

In this editorial, we denote the string we get by performing the operation on the substring from the l-th character (inclusive) to the r-th character (exclusive) as $f[l, r)$.

We can simulate the operations using a stack. When we want to compute $f[l, r)$, we push the characters from the l-th to the r-th one by one into the stack. Whenever the top two characters in the stack are the same, it changes (according to the rule in the statement).

For each position l, we define $next(l)$ as the smallest r such that $f[l, r)$ is an empty string. Also, for each pair of a position l and a character c, we define $next_c(l)$ as the smallest r such that $f[l, r) = c$. (If such r doesn't exist, it is defined as ∞.)

How can we compute these values? For example, suppose that $s[l] = \texttt{x}$. We create a stack, and we push characters in s from the l-th position one by one. Consider the character at the bottom of the stack. Since only the topmost character in the stack changes, when the bottommost character changes, it means that the stack only consists of this character. Also, by the transition rule, x only changes to y, y only changes to z, and when z changes it disappears. Thus, $next_{\texttt{x}}(l) < next_{\texttt{y}}(l) < next_{\texttt{z}}(l) < next(l)$, and other characters don't appear at the bottom of the stack until the stack becomes empty.

$next_{\texttt{x}}(l)$ is obviously $l + 1$.

How do we compute $next_{\texttt{y}}(l)$? Create two empty stacks. First push the l-th character (i.e., an `x`) into the first stack, and then push the characters from the position $l + 1$ one by one into both stacks. The character `x` only interacts with another `x`. Thus, until the bottom of the second stack becomes `x`, both stacks behave exactly the same except for an extra `x` at the bottom of the first stack. Then, when the `x` appears at the bottom of the second stack, the first stack becomes a single `y`. Thus, $next_{\texttt{y}}(l) = next_{\texttt{x}}(next_{\texttt{x}}(l))$. Similarly, we can compute $next_{\texttt{z}}(l)$ and $next(l)$.

How do we compute $next_{\texttt{a}}(l), \ldots, next_{\texttt{w}}(l)$? These characters won't appear until the stack becomes empty, and after the stack becomes empty we can forget everything before it, so for example $next_{\texttt{a}}(l) = next_{\texttt{a}}(next(l))$.

Here is the summary:

- $next_{\texttt{x}}(l) = l + 1$
- $next_{\texttt{y}}(l) = next_{\texttt{x}}(next_{\texttt{x}}(l))$
- $next_{\texttt{z}}(l) = next_{\texttt{y}}(next_{\texttt{y}}(l))$
- $next(l) = next_{\texttt{z}}(next_{\texttt{z}}(l))$
- $next_{\texttt{a}}(l) = next_{\texttt{a}}(next(l))$
- $next_{\texttt{b}}(l) = next_{\texttt{b}}(next(l))$

- $\vdots$
- $next_{\mathtt{w}}(l) = next_{\mathtt{w}}(next(l))$

Thus, we can compute these values in the decreasing order of l.

In order to answer a query $[l, r]$, we need to check if r is in the sequence $l-1, next(l-1), next(next(l-1)), \ldots$. This can be done in $O(logN)$ per query with $O(NlogN)$ pre-computation using doubling (i.e., pre-compute $next^{2^i}(j)$ for each (i, j)).

The complexity of the solution is $\mathrm{O}(NA+(N+Q)logN)$, where N is the length of the string and A is the size of the alphabet.

D: Oriented Tree

Let $diam$ be the diameter of the tree. Obviously, $D \geq \lceil diam/2 \rceil$. On the other hand, if we direct the edges "alternately", we can achive this D. Thus, the minimum D is $\lceil diam/2 \rceil$.

Choose an arbitrary vertex and call it v_0. For each vertex v, we assign a label $h(v)$ as follows:

- $h(v_0) = 0$
- For each directed edge $(u \to v)$, $h(u) - h(v) = 1$

This way, the labeling correspond to the orientation bijectively.

We can show that $d(s,t) = (dist(s,t) - h(s) + h(t))/2$. Thus,

$$\max\{(d(s,t), d(t,s)\} = \frac{dist(s,t) + |h(s) - h(t)|}{2}$$

and the condition $D = \lceil diam/2 \rceil$ is equivalent to

$$\begin{aligned} & \frac{dist(s,t) + |h(s) - h(t)|}{2} \leq \left\lceil \frac{diam}{2} \right\rceil \\ \Leftrightarrow \quad & |h(s) - h(t)| \leq 2 \left\lceil \frac{diam}{2} \right\rceil - dist(s,t) \end{aligned} \tag{1}$$

Let's count the number of labelings that satisfy (1).

Consider the case where $diam$ is even. Let r be the center (i.e., the midpoint of the diameter) of the tree. The condition (1) is equivalent to

$$|h(s) - h(t)| \leq diam - dist(s,t) \tag{2}$$

We call a vertex v "deep leaf" if $dist(v,r) = diam/2$. We can show that for each deep leaf, the labeling must be the same. For simplicity assume that this label is zero. Then, we can show that for each v, $|h(v)| \leq diam/2 - dist(r,v)$. On the other hand, this is sufficient: when this condition is satisfied for all vertices, (1) is satisfied.

Thus, our task is to count the number of labelings that satisfies the following:

- For each adjacent pair of vertices (s,t), $|h(s) - h(t)| = 1$.
- For each vertex v, $|h(v)| \leq diam/2 - dist(r,v)$.

This can be done easily in $O(N^2)$ by DP on the tree.

The case where $diam$ is odd is quite similar. Let s, t be the two centers of the tree. We call a vertex v black if $dist(s,v) < dist(t,v)$, and otherwise call it white.

Then, a valid labeling must satisfy one of the following (the proof is similar to the case with even diameter):

- For each adjacent pair of vertices (s,t), $|h(s) - h(t)| = 1$.
- For each black vertex v, $|h(v)| \leq diam/2 - dist(r,v)$.
- For each white vertex v, $|h(v)| \leq diam/2 - dist(r,v) + 1$.

or

- For each adjacent pair of vertices (s,t), $|h(s) - h(t)| = 1$.
- For each white vertex v, $|h(v)| \leq diam/2 - dist(r,v) + 1$.
- For each black vertex v, $|h(v)| \leq diam/2 - dist(r,v)$.

Of course, we should subtract the intersection between these two cases. Note that, since we don't distinguish two labelings that are shifted by a constant, we should be careful when we take the intersection of these two cases. The intersection is,

- For each adjacent pair of vertices (s,t), $|h(s) - h(t)| = 1$.
- For each white vertex v, $|h(v)| \leq diam/2 - dist(r,v)$.
- For each black vertex v, $|h(v) + 1| \leq diam/2 - dist(r,v)$.

and

- For each adjacent pair of vertices (s,t), $|h(s) - h(t)| = 1$.
- For each white vertex v, $|h(v)| \leq diam/2 - dist(r,v)$.
- For each black vertex v, $|h(v) - 1| \leq diam/2 - dist(r,v)$.