Xmas Contest 2013

Solutions

Problem A: Random Trip

解法 1

確率が整数パーセントで与えられているので, スタート地点となる確率が 0 でない町は高々 100 個しかない. そのそれぞれに対し, 他の各町までの距離を (深さ優先探索などで) $O(N)$ 時間で求めることができる.

解法 2

確率の値の制約を用いない $O(N)$ 時間解法の概略を紹介する.

与えられる木の辺集合を E とし, 辺 $e \in E$ を通る確率を p_e, 辺 $e, e' \in E$ をともに通る確率を $q_{e,e'}$ とすると, 求める期待値は

$$\sum_{e \in E} p_e + \sum_{e,e' \in E,\, e \neq e'} q_{e,e'}$$

であることがわかる (期待値の線形性を用いよ).

p_e は, e で分けられる 2 個の部分木にスタート地点とゴール地点が 1 個ずつある確率である. 各部分木にスタート地点あるいはゴール地点がある確率を動的計画法によって求めておくことで計算できる.

e に対し, $\sum_{e' \in E,\, e' \neq e} q_{e,e'}$ を考える. e で分けられる 2 個の部分木を T, T' とし, e' としての T' に含まれるものを動かすとき, $q_{e,e'}$ は, T と, e' で分けられる 2 個の部分木のうち e を含まない方にスタート地点とゴール地点が 1 個ずつある確率である. T にスタート地点 (ゴール地点) がある確率は e' の選び方によらないので, T' 内の各辺 e' に対する, e' で分けられる 2 個の部分木のうち e を含まない方にスタート地点 (ゴール地点) がある確率の和を求めればよい. これもやはり動的計画法で, 今度は「各部分木にスタート地点あるいはゴール地点がある確率」の和を求めておけば全体で $O(N)$ 時間で計算できる.

Problem B: Rotating Coin

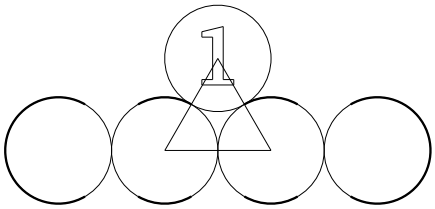
$N = 1$ の場合

一円玉を長さ $2\pi R_1$ の線分に沿って転がすと R_1 回転する. よって一円玉は「円周に対して」R_1 回転するが, 円周自体の向きが 1 回転しているため, 一円玉自身は $R_1 + 1$ 回転していることになる.

一般の場合

同様にして, 一円玉を半径 R のコインの円周のうち割合 t だけに沿って転がした場合, 一円玉自身は $t(R+1)$ 回転することがわかる. よって, 一円玉が各コインの円周のどれだけの割合に触れるかを求め, 回転数を合計すればよい.

一円玉より小さいコインがないことから, 一円玉はすべてのコインに接する. i 番目のコインと $i+1$ 番目のコインの両方に接する時点での興味ある角度は, 3 辺の長さが $R_i + R_{i+1}$, $R_i + 1$, $R_{i+1} + 1$ の三角形に余弦定理を用いて求めることができる.



Problem C: Really Long Sequences

長さ M, N の列の最長共通部分列を求める計算量 $O(MN)$ のアルゴリズムが知られているが, 本問では, $I = \{(i, i') \mid a_i = b_{i'}\}$ とするとき $O(M + N + |I| \log |I|)$ 時間で求める方法を用いる. 問題文で書かれた数列の生成法で $|I|$ が小さくなるかは後ほど検討するが, 値がランダムに選ばれていれば $|I|$ が MN よりずっと小さくなるのでよさそう, とおおよそ考えられる.

各値 x に対し $\{i' \mid b_{i'} = x\}$ を記録しておけば, I は $O(M + N + |I|)$ 時間で具体的に求めることができる. 長さ k の共通部分列は, I の k 個の元 $(i_1, i'_1), \ldots, (i_k, i'_k)$ で $i_1 < \cdots < i_k$, $i'_1 < \cdots < i'_k$ を満たすものに対応している. さらにこれは, I の元を第 1 成分の小さい順に, 第 1 成分が等しい場合は第 2 成分が大きい順に並べたものを考えたとき, その列から第 2 成分を取り出した数列の狭義単調増加部分列に対応している.

例えば, $a = (11, \underline{22}, \underline{11}, 22, \underline{33})$, $b = (22, \underline{22}, 33, \underline{11}, 11, \underline{33})$ に対し,

$$I = \{(1,5), (1,4), (2,2), (2,1), (3,5), (3,4), (4,2), (4,1), (5,6)\}$$

であり, 下線で示した共通部分列は, 数列 $5, 4, \underline{2}, 1, 5, \underline{4}, 2, 1, \underline{6}$ の下線で示した狭義単調増加部分列に対応している.

長さ $|I|$ の数列の最長増加部分列を $O(|I| \log |I|)$ 時間で求める方法はよく知られている (動的計画法の高速化) ので, これで数列 $(a_i)_i$ と $(b'_i)_{i'}$ の最長共通部分列が $O(M + N + |I| \log |I|)$ 時間で求まった.

残るは $|I|$ についての考察である. 数列 $(a_i)_i$ は $a_i \equiv 103a_{i-1} + R \pmod{1\,000\,003}$ のように定められている. $1\,000\,003$ が素数であるから, $\alpha \equiv 103\alpha + R$ を満たす α がとれるので, $a_i - R \equiv 103(a_{i-1} - \alpha)$, よって $a_i - R \equiv 103^i(P - \alpha)$ がわかる. $(103^i)_i$ は基本周期が $333\,334$ であることが確認できるので, $P - \alpha \neq 0$ であれば数列 $(a_i)_i$ も基本周期 $333\,334$ をもつ. このとき数列 $(a_i)_i$ は同じ値を高々 3 回しか含まないので, $|I| \leq 3N$ が得られる. 一方 $P - \alpha \equiv 0$ のとき数列 $(a_i)_i$ は定数である. 数列 $(b'_i)_{i'}$ に対しても同じ考察が成り立ち, $(b'_i)_{i'}$ が定数でなければ $|I| \leq 3M$ である.

以上をまとめると, 数列 $(a_i)_i$, $(b'_i)_{i'}$ のいずれかでも定数でないときは, $|I|$ は $O(M + N)$ である. 両方の数列が定数であるときは, 最長共通部分列は容易に求まる.

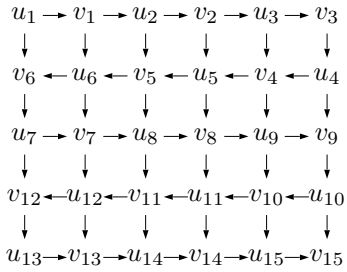
Problem D: Rabbit Pairs

グリッドグラフの各辺が赤か黒に塗られており, 赤い辺を高々 K 本用いた完全マッチングの個数を $\bmod p$ (p は素数) で求める問題を解けばよい. 平面的グラフの完全マッチングの個数を数えるアルゴリズムとして, FKT algorithm (Fisher, Kasteleyn, Temperley) が知られている. これを二部グラフに適用した形を示す：

> 平面的二部グラフ $G = (U \cup V, E)$, $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E \subset U \times V$ に対し, 各辺の向き付けであって, G の任意の閉路 (二部性より長さ偶数) が各向きの辺を奇数本もつようなものが存在する. そのような向き付けに対し, 行列 $A = (a_{i,j})_{1 \le i,j \le n}$ を
>
> $$a_{i,j} = \begin{cases} +1 & (u_i v_j \in E,\ u_i \text{ から } v_j \text{ への向き}) \\ -1 & (u_i v_j \in E,\ v_j \text{ から } u_i \text{ への向き}) \\ 0 & (u_i v_j \notin E) \end{cases}$$
>
> で定めると, G の完全マッチングの個数は $\det A$ の絶対値に等しい.

これは, $\{1, \ldots, n\}$ の置換 σ に対し $\prod_{i=1}^{n} a_{i,\sigma(i)} \neq 0$ であることが完全マッチング $\left(u_i v_{\sigma(i)}\right)_i$ に対応すること, どの完全マッチングに対してもその符号が一致することが向き付けの条件から従うことを考えると示される.

本問のグリッドグラフの場合には適切な向き付けを直接得られる. また, 完全マッチング $(u_i v_i)_i$ が存在し u_i から v_i への向きであるように頂点の番号付け・辺の向き付けを行うことで, $\det A \ge 0$ とすることができる. 例えば, N が偶数であるとして以下に示すようにすればよい.



この方法を応用する. 赤い辺 $u_i v_j$ に対し, $a_{i,j}$ を ± 1 の代わりに $\pm X$ と定めた行列 $A(X)$ を考える. 赤い辺を k 本含む完全マッチング $\left(u_i, v_{\sigma(i)}\right)_i$ に対し $\prod_{i=1}^{n} a_{i,\sigma(i)} = X^k$ となるから, 赤い辺をちょうど k 本含む完全マッチングの個数は $\det A(X)$ の X^k の係数である. $\det A(X)$ を効率よく計算するためには, これが高々 n 次の多項式であることを用い, $X = 0, 1, \ldots, n$ を代入したときの値を (体 $\mathbb{F}_p$ 上の計算で) 求め, 連立方程式を解いて (あるいは補間法を用いて) 多項式を復元すればよい. 時間計算量は $O(n^4)$ となる (本問では $n = \dfrac{MN}{2}$).

参考

グリッドグラフは bandwidth が小さいので, 行列の疎性を用いて行列式計算の計算量を落とせる.

また, $n+1$ 通りの値を代入する必要があるため, $n + 1 > p$ の場合を解くためには, 標数 p の有限体で十分大きいものをとって計算することになる.

Problem E: Range Composition

解法 1

segment tree を用いると, $O(N)$ 時間 $O(N)$ メモリの前計算により, a, b が与えられたとき合成関数 $F_b \circ F_{b-1} \circ \cdots \circ F_{a+1} \circ F_a$ を $O(\log N)$ 時間で計算できる.

具体的には, 区間 $[1,1], [2,2], [3,3], \ldots; [1,2], [3,4], [5,6], \ldots; [1,4], [5,8], [9,12], \ldots; \ldots$ 内の関数を合成したものを保持しておけばよい. 任意の区間はこれらの $O(\log N)$ 個に分割される.

解法 2

segment tree は区間 $[1, N]$ を 2 分割していくことで管理したが, 次の条件を満たすように 3 分割以上も許した構造を factorization forest は呼ばれる：区間 I を $I_1, \ldots, I_k$ に分割できる条件は,

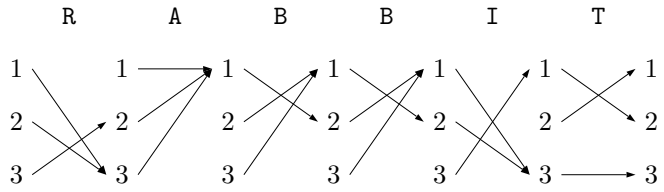
- $k = 2$
- $I_1, \ldots, I_k$ 内の関数を合成したものはいずれも等しい関数 e となり, さらに $e \circ e = e$ を満たす

のいずれかである.

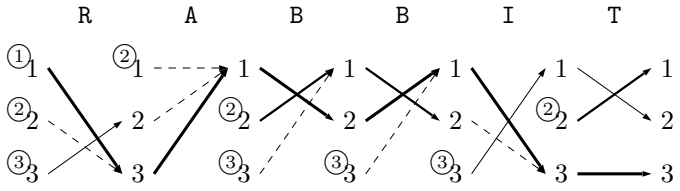
N によらない定数高さの factorization forest を構成できることが知られており, クエリごとに $O(1)$ 時間で答えることができる (ただし, 定数倍は大きい).

解法 3

下の図は関数適用の様子を矢印で表している.



これらのうちいくつかを取り出して, 交わらないパスの和集合として管理することを考える.



これらは次のように構成されている：$r = 1, 2, 3$ に対し, 次を行う. 左端で値 r を選んでから順に関数を適用していきパスを辿る. ただし途中で今まで訪れた点に到達する場合はパスを切り (図中破線), その関数の値域に含まれない値を選び新たなパスを開始する. これらのパスにはラベル r をつける.

すると, どの場所から矢印を辿っても, 破線は高々 2 本しか通らない. これは通るパスのラベルが減少するためである. よって, 各パスに対してその終点の位置, および, 場所とラベルからパスへのポインタを管理することで, 前処理にかかる計算量は $O(N)$ で, 矢印を辿る操作を $O(1)$ 時間で実現できる (演算回数は非常に少ないが, segment tree 解よりメモリアクセスは悪くなってしまう).

Problem F: Replacing Batteries

$N = 1$ の場合

型が a の電池が切れるまでの時間の確率密度関数は $t \mapsto \dfrac{d}{dt}(1 - a^{-t}) = a^{-t} \log a$ であるから, 型が a の電池と型が b の電池を同時に使い始め, 型 a の電池が先に切れる確率は,

$$\int_0^\infty (a^{-t} \log a) b^{-t} = \frac{\log a}{\log ab} = \frac{\log a}{\log a + \log b}$$

である.

一般の場合

電池が切れるまでの時間の分布は指数分布であるが, 指数分布のもつ重要な性質として「無記憶性」がある. すなわち, 電池が切れていない任意の時点において, その後電池が切れるまでの時間は, これまでその電池をどれだけの時間使ったかに依存しない. このことは, 既に t_0 時間使った型が c の電池がこれから t 時間以上持つ確率が $\dfrac{c^{-(t_0+t)}}{c^{-t_0}} = c^{-t}$ となることからわかる.

よって, 「2 匹がそれぞれ今何番目の電池を使っているか」のみを状態として保持すれば, その後の情報について得るために十分であることがわかる. 以下の計算量 $O(MN)$ の単純なアルゴリズムによって答えを求めることができる.

for $i := 1$ **to** $M + 1$:
  for $j := 1$ **to** $N + 1$:
    $p_{i,j} := 0$.
$p_{1,1} := 1$.
for $i := 1$ **to** M:
  for $j := 1$ **to** N:
    $p_{i+1,j} := p_{i+1,j} + \dfrac{\log A_i}{\log A_i + \log B_j} p_{i,j}$.
    $p_{i,j+1} := p_{i,j+1} + \dfrac{\log B_j}{\log A_i + \log B_j} p_{i,j}$.
return $\displaystyle\sum_{j=1}^{n} p_{M+1,j}$.

参考

型が c の電池が切れるまでの時間の期待値は $\dfrac{1}{\log c}$ である (積分で求まる). これと無記憶性を用いると, 型が a の電池と型が b の電池を同時に使い始め, 型 a の電池が先に切れる確率を次のようにも求められる：型が a の電池と型が b の電池をそれぞれ切れる度に新しいものに交換して連続して使っていると考えると, 単位時間ごとにそれぞれ $\log a$ 回, $\log b$ 回切れるので, 次に型が a の電池が切れる確率は $\dfrac{\log a}{\log a + \log b}$ である.

Problem G: Rumor with Primes

まず $(-1)^{f(1)}, (-1)^{f(2)}, \ldots$ を求めてから, $g(n)$ を順次計算しながら入力中の N の値を小さい順にみて $g(N)$ (の符号) を答えていくことを考える. 素因数を効率的に調べるために, Eratosthenes の篩が有効である. 始め $x_1 = x_2 = \cdots = +1$ とし, 各素数 p ごとに, p の倍数である n について, ($n > p$ なら n が素数でないことを記録し) n が p で奇数回割り切れるとき $x_n := -x_n$ としていく. この方法の時間計算量は, $n \leq N$ の範囲を調べるとき $O(N \log\log N)$ である.

空間計算量が $O(N)$ であることに注意を要する. 1 つの省メモリ法は, 各 n に対してちょうど 2 ビットのみを使うように実装することである (例えば C++ の bitset). もう 1 つの方法として, 始めに $\sqrt{N}$ 以下の素因数をすべて求めたうえで ($O(\sqrt{N} \log\log N)$ 時間), 区間 $[1, N]$ をいくつかに分割しそれぞれで区間篩のアルゴリズムを適用する方法がある. 長さ L の区間 $\dfrac{N}{L}$ 個に分割した場合, 時間計算量は $O\left(N \log\log N + \dfrac{N\sqrt{N}}{L \log N}\right)$, 空間計算量は $O(L)$ となる. 例えば $L = 10\,000\,000$ を選べば十分である.

参考

$(-1)^{f(n)}$ は Liouville 関数と呼ばれている. $g(n) > 0$ となる n は, $n = 1$ の次に小さいものは $n = 906\,150\,257$ である.

$g(n) = 0$ となる n は $1 \leq n \leq 1\,000\,000\,000$ の範囲では実際には非常に少なく (252 個), それらは知られている[*1]. その値を利用し, N に対して $g(n) = 0$ となる n が N 以下に偶数個あるか奇数個あるかで $g(n)$ の符号を決定する解法も考えられる.

[*1] http://oeis.org/A028488/b028488.txt

Problem H: Read the Input

解法 1

知られている英語の各文字の出現頻度, あるいはさらに隣接 2 文字の出現頻度を用い, S の長さ 30–60 程度の部分文字列に得点づけを行い上位を出力することで, 問題文になっている箇所を発見できる. 得点づけは, 単に出現に対して頻度の和をとるだけでも良い精度となる.

解法 2

問題文に出現しそうな単語を考えて, 検索を行う. 例えば問題文に

> 整数 N は数列 (R_i) の長さを表す. 数列の各要素 R_i $(1 \leq i \leq N)$ は整数である.

とあるが, 単語 “integer”, “sequence”, “element” が実際に問題文中に現れるテストケースがある. 各入力ファイル内で問題の形式はほぼ同じである (ことが予想できる) ので, 検索すべき単語を徐々に増やしていくことができる.

ただし, H2.in の 18 個目のテストケースで若干工夫・ひらめきを要する. 他のケースから類推して, 文字列 “est” を検索して発見した, といった解法が見られた (ある 3 文字が S 中に現れる回数の期待値は 1 より小さい点に注目せよ).