第7回 日本情報オリンピック本選 解説[1]

2008 年 12 月 22 日

情報オリンピック日本委員会

1 碁石ならべ

本問題は，問題文に指定され碁石の置き方を実行したときに，最終的に白い碁石が何個置かれているかその数を出力する問題である．解法としては，配列の一つ一つの要素に各碁石の色を表した値を格納してシュミレーションを行う方法と，配列の一つ一つの要素に連続する同じ色の碁石の数を格納してシュミレーションを行う方法の計算量が異なった二つの方法が考えられる．

解法１

一つ一つの要素に，碁石一つ一つの色を表す値を格納する長さ n の配列を用いて，シュミレーションを行う．

つまり，偶数である i 番目に碁石を置く際には，$i-1$ 番目が i 番目の碁石と異なる色の碁石であった場合，同じ色の碁石になるまで右から左に碁石を置き換える処理を繰り返す．

全ての碁石を置き終わった後に，並んでいる白の碁石の数を求める．

解法２

一つ一つの要素に，連続する白または黒の碁石の数を表す値を格納する長さ n の配列を用いて，シュミレーションを行う．

最初の碁石を置く処理は以下のようになる．

```
右端の碁石の index = 0;
配列 [右端の碁石の index] = 1;
左端の碁石の色 = 直前に置いた碁石の色 = 一番最初に置く碁石の色;
```

二つ目以降に置く各碁石の処理は，たとえば以下のように場合分けをすることになる．

```
if (直前に置いた碁石の色 == 新たに置く碁石の色) {
    // 連続した色の碁石の数を一つ増やす
    配列 [ 右端の碁石の index ]++;
} else if (偶数番目に碁石を置く場合) {
    if( 右端の碁石の index == 0 ) {
        // テーブルに置いてある碁石は一色であり,
        // 新たに置く碁石の色と異なるので,
        // 現在デーブルに置いてある碁石を,
        // 新たに置く碁石と同じ色の碁石に，全ての置き換える
        配列 [ 右端の碁石の index ]++;
        左端の碁石の色 = 新たに置く碁石の色;
    } else {
        // 直前の連続した色の碁石を,
        // 新たに置く碁石の色と同じ色に置き換える
```

```
        配列 [ 右端の碁石の index - 1 ] += 配列 [ 右端の碁石の index ] + 1;
        右端の碁石の index--;
    }
} else {
    // 直前に置いた碁石とは異なった色の碁石を新たに置く
    右端の碁石の index++;
    配列 [ 右端の碁石の index ] = 1;
}
直前に置いた碁石の色 = 新たに置く碁石の色;
```

全ての碁石を置き終わった後に，左端の碁石の色をもとに，並んでいる白の碁石の数を求める．

解析

解法 1 の場合，たとえば偶数番目の碁石を置くたびに，テーブルに置いてある全ての碁石を，新しく置く碁石の色に置き換えるとすると，全体で $\lfloor \frac{n}{2} \rfloor^2$ 回の碁石の置き換えが発生する．n の上限は 100000 なので，25 憶回の置き換えが発生する可能性がある．つまり，解法 1 の計算量は $O(n^2)$ であり，制限時間内で全てのテストデータに正しい答えを出力することはできない．

解法 2 の場合，碁石を置く処理にかかる計算は，テーブルに置いてある碁石の数が多くなっても変わらず，入力のデータサイズに依存しないので，その計算量は $\mathrm{O}(1)$ である．つまり，解法 2 のような計算量が $\mathrm{O}(n)$ のアルゴリズムを用いることによって，100% の得点を得られるように，本選の制限時間と採点用入力データを調整した．

2 共通部分文字列

1 解法

文字列 x の長さを $|x|$ と書く. 与えられた二つの文字列を s と t とする. s と t のうち長い方の長さを $n = \max\{|s|, |t|\}$ とおく.

素朴な解法 1 ($O(n^4)$ 時間)

s に含まれる文字列すべてについて, t に含まれるかどうかを調べればよい.

s に含まれる文字列は, 空文字列を除きすべて s の i 文字目から j 文字目まで $(1 \leqq i \leqq j \leqq |s|)$ という形をしているので, 全部で $O(|s|^2)$ 個ある. 文字列 x が文字列 t に含まれるかどうかを調べる (文字列検索) にはいくつかの方法がある. 簡単な方法は, 各 k $(1 \leqq k \leqq |t| - |x| + 1)$ について, 文字列 t の k 文字目から $|x|$ 文字分が x に一致するかどうかを調べ, どれかが一致すれば x が y に含まれると判定することである. この方法では文字列検索が最悪でも $O(|x||t|)$ 時間で行える. 実際には, 各 k について $|x|$ 文字全部調べる必要があるとは限らず, もっと速い場合もあるが, ほとんど同じ文字列が何度も繰り返し現れる場合は最悪計算量に近い時間がかかる. これを用いると, 問題は全体で $O(|s|^3|t|)$ 時間で解ける.

s と t を入れ替えても答えは変わらないので, 短い方を s として上のアルゴリズムを用いれば全体の時間計算量が $O(|s||t| \min\{|s|^2, |t|^2\})$ に改善されるが, 採点用データではどれも s と t の長さがほぼ同じなのでこの改善は効かない.

なお, 文字列検索には様々なアルゴリズムがあり, 上で述べた簡単な方法よりも速い方法としては, $O(|x| + |t|)$ 時間の Boyer–Moore 法などがある. このような方法を用いれば, 問題が $O(|s|^2|t|)$ 時間で解ける.

素朴な解法 2 ($O(n^3)$ 時間)

s の部分文字列をすべて考える代わりに, 共通部分文字列の s と t における開始位置をすべて考える.

すなわち, $1 \leqq i \leqq |s|$ と $1 \leqq j \leqq |t|$ を満たす i と j の組それぞれについて, s の i 文字目から始まる部分文字列と t の j 文字目から始まる部分文字列が何文字まで共通であるかを調べる. こうすると, i と j の組が $|s||t|$ 通りあり, 指定された開始位置から始まる共通部分文字列の長さを調べるのに $\min\{|s|, |t|\}$ 時間かかるので, 全体で $O(|s||t| \min\{|s|, |t|\})$ 時間で解ける.

高速な解法 ($O(n^2)$ 時間)

s と t の共通部分列は, s と t を何文字かずらせば完全に一致する部分のことなので, ずらす量の選び方をすべて考える.

すなわち, $0 \leqq i \leqq |s|$ に対し, s の先頭から i 文字除いた文字列 s' と t を上下に並べて比較する. s' の j 文字目と t の j 文字目が一致するとき $a_j = 1$, そうでないとき $a_j = 0$ とすると, 数列 $a_1, \ldots, a_{|t|}$ の中で連続する 1 の個数の最大値は $O(|t|)$ 時間で求められる. 問題文中の例 1 で $i = 1$ とした場合を図にすると次のようになる.

$s' =$ BRACADABRA
$t =$ KCADADADABRBCRDARA
$a =$ 0010<u>11111</u>0000000

これをすべての i に対して求め, さらに s と t の役割を入れ替えて同じことをして, 最大値を出力すればよい. 全体で $O(|s||t|)$ 時間である.

2 採点用データについて

$n = 50$ 程度の小さなデータを 3 個, $n = 4000$ 程度の大きなデータを 7 個, 計 10 個のデータを用いて採点した.

小さなデータは上述のどの解法でも解けることを想定している. また, 大きなデータであっても, 文字列中に繰り返しが少ない場合, アルゴリズムが最悪時間計算量より速く終了することが多いため, 素朴な解法でも正解できる可能性が高い.

番号	$\|s\|$	$\|t\|$	答え	備考
1	50	50	2	ランダム
2	50	49	17	出現する文字は 2 種類 (G と L), ほとんど G
3	50	50	0	答えが 0
4	4000	4000	5	ランダム
5	4000	3980	1951	出現する文字は 2 種類 (E と T), ほとんど E
6	3984	3992	1440	出現する文字は 4 種類 (F, K, I, Y), ほとんど KF の繰り返し
7	4000	4000	904	出現する文字は 3 種類 (V, G, H), ほとんど V
8	3992	3994	719	出現する文字は 3 種類 (K, F, V), ほとんど K
9	3995	3996	3995	全部 C
10	4000	4000	0	答えが 0

表: 問題 2 の採点用データ

3 ダーツ

1 解法

入力として的（まと）に書かれた点数 $P_1, \ldots, P_N$ が与えられる．4 本以下の矢を投げたときの合計点のうち，M を超えないものの最大値を求める問題である．

まず,「矢を投げない」場合を簡単に処理するために，次のように問題を言い換えておこう．$P_0 = 0$ とおく．的に $N+1$ 個の点数 $P_0, P_1, \ldots, P_N$ が書かれていて，4 本の矢を投げると考えよう．すなわち,「矢を投げない＝0 点の的に矢が刺さった」と考えるのである．

さて，4 本の矢が番号 a, b, c, d $(0 \leqq a, b, c, d \leqq N)$ の的に刺さった場合の得点 $S_{a,b,c,d}$ は

$$S_{a,b,c,d} = \begin{cases} P_a + P_b + P_c + P_d & P_a + P_b + P_c + P_d \leqq M \text{ のとき} \\ 0 & P_a + P_b + P_c + P_d > M \text{ のとき} \end{cases}$$

で与えられる．最大値

$$\max_{0 \leqq a,b,c,d \leqq N} S_{a,b,c,d}$$

を求めればよい．

効率の悪い解法で部分点を取るのは難しくないが，時間制限が厳しいので，高得点を得るには，データをあらかじめソートしておくなどのテクニックが必要であろう．

解法 1 ($O(N^4)$ 時間)

矢の刺さる場所は（「投げない = 0 点の的に刺さる」も含めると）$N+1$ 通りある．$(N+1)^4$ 通りの矢の刺さり方を全て調べれば最も単純な解法が得られる．これは，4 重ループを書くだけで容易に実装できる．$O(N^4)$ 時間かかる．

なお，矢を投げる順番は問わないので，$0 \leqq a \leqq b \leqq c \leqq d \leqq N$ の範囲で探索すれば十分である．こうすることで，計算時間を少しだけ節約することができる（$4! = 4 \times 3 \times 2 \times 1 = 24$ 分の 1 の計算時間で済む）．ただし，これでは計算時間のオーダーは変わらない．

$N = 100$〜200 程度のデータであれば，制限時間内に正解を出力することができる．

解法 2 ($O(N^3 \log N)$ 時間)

「4 本の矢を投げる」と考えるのでなく,「まず矢を 3 本投げておいてから，最後の 1 本の矢を最適な場所に投げる」と考えよう．

3 本の矢の得点が P_a, P_b, P_c であると仮定する．このとき，4 本目の矢を投げる最適な場所は，

$$P_a + P_b + P_c + P_d \leqq M$$

となるような d のうち，左辺がもっとも大きくなるところである．言い換えると，

$$P_d \leqq M - (P_a + P_b + P_c)$$

をみたす d のうち，P_d が最も大きくなるものを求めればよい．このような d は，$P_0, \ldots, P_N$ をあらかじめソートしておけば，二分探索によって，$O(\log N)$ 時間で求めることができる．

下準備のソートにかかる時間は $O(N \log N)$ だから，全体では $O(N^3 \log N)$ 時間かかる．

この方法であれば，$N = 300$ 程度のデータに対して，制限時間内に正解を出力することができる．

解法 3 — 想定解 ($O(N^2 \log N)$ 時間)

解法 2 を改良することで，さらに効率の良い解法が得られる．

今度は,「4 本の矢を投げる」ことを,「まず 2 本の矢をまとめて投げて，次に 2 本の矢をまとめて投げる」と考えることにしよう．

矢を 2 本投げることで得られる点数は $(N+1)^2$ 通り以下である．まず，下準備として，これらの可能性をすべて調べてソートしておく．ソートした結果を $Q_1 \leqq Q_2 \leqq \cdots \leqq Q_r$ とおく ($r \leqq (N+1)^2$)．この下準備にかかる時間は $O(N^2 \log N)$ である．

さて，最初の 2 本の矢の点数の合計が Q_i であったと仮定しよう．残りの 2 本の矢（3 本目，4 本目の矢）の最適な得点を求めるためには,

$$Q_i + Q_j \leqq M$$

をみたす j のうち，Q_j が最も大きくなるものを求めればよい．このような j は二分探索により $O(\log N)$ 時間で求めることができる．各 i ($1 \leqq i \leqq r$) に対して最適な j を求めればよいので，全体として $O(N^2 \log N)$ 時間かかる．

下準備の時間も含めて，この解法では，全体で $O(N^2 \log N)$ 時間かかる．

この問題の設定では $N \leqq 1000$ なので，この方法であれば，すべてのデータに対して制限時間内に正解を出力することができる．

2　採点用データについて

以下に採点用データの N, M の値を挙げる．

番号	1	2	3	4	5	6	7	8	9	10
N	50	100	300	300	300	1000	1000	1000	1000	1000
M	2×10^6	2×10^7	10^8	10^8	10^8	2×10^8	2×10^8	2×10^8	2×10^8	2×10^8

解法 1 なら最初の 2 つのデータに対して，解法 2 なら最初の 5 つのデータに対して制限時間内に正解を出力することができる．

解法 3 であれば，すべての入力データに対して，制限時間内に正解を出力することができる．

4 ぴょんぴょん川渡り

はじめに

この解説では，3 つの解法を紹介することにする．解法 1 は，そのままでは効率が悪く満点を得ることはできないが，後の効率の良い解法に対する大きな第一歩となっている解法である．解法 2 は，解法 1 に少し手を加えることにより効率を劇的に向上した解法である．解法 3 は，解法 1,2 の考え方を異なったアプローチにより実現する解法である．

解法 1（深さ優先探索）

考え方

計算の効率を考えることをひとまずおいておき，まずは正しい答えが得られるアルゴリズムを考えることから始めよう．

とりあえず，全ての経路を試してみれば，危険度の合計が最小になる経路はその中のどれかなので，正しい答えが必ず求まるはずである．どの石を飛ばすか，それぞれの行でどの石に乗るかの全ての場合を試してみればよい．

実現

上記の考え方のアルゴリズムは，深さ優先探索により実現される．

$\texttt{Danger}(i,j,l)$ を，i 行目 j 番目の石に居て，そこまでの一行飛ばしのジャンプの回数が l であるときの，そこから向こう岸までの危険度の合計の最小値を求める関数とする．$\texttt{Danger}(i,j,l)$ は，岸に到達可能な時は 0 を返せばよい．岸に到達可能な時とは，$i=n$ であるか，$i=n-1$ かつ $l<m$ であるときである．

そうでない時は，まず普通のジャンプの場合として，$1 \leqq j' \leqq k_{i+1}$ を満たす全ての j' について

$$(d_{i,j}+d_{i+1,j'}) \times |x_{i,j}-x_{i+1,j'}| + \texttt{Danger}(i+1,j',l)$$

と，$l<m$ なら，一行飛ばしのジャンプの場合として，$1 \leqq j'' \leqq k_{i+2}$ を満たす全ての j'' について

$$(d_{i,j}+d_{i+2,j''}) \times |x_{i,j}-x_{i+2,j''}| + \texttt{Danger}(i+2,j'',l+1)$$

を計算し，その最も小さいものを返せばよい．

後は，$1 \leqq j \leqq k_1$ を満たす全ての j について $\texttt{Danger}(1,j,0)$ と，$m>0$ なら，$1 \leqq j' \leqq k_2$ を満たす全ての j' について $\texttt{Danger}(2,j',1)$ を呼び出し，その返り値の最も小さいものを答えとすればよい．

計算量

簡単のため，全ての行が k 個の石を含むとする．この解法は，経路の全通りを試している．その場合の数は，

$$\sum_{i=0}^{m} {}_n\mathrm{C}_i \times k^{n-i} \quad \text{通り}$$

である．これは n の増加に伴い指数的に増加してしまい，すぐにとても大きな数になってしまう．この解法はとても効率が悪いと言える．

この解法では，問題文に記されている，配点の 20%分のサイズの小さな入力に対してのみ得点を得ることができる．

解法2（メモ化探索）

考え方

解法 1 はなぜ効率が悪いのだろうか．それは，同じ計算を何度もしているからである．

例えば，ある行に石 A があり，その前の行に石 B,C があるとする．石 A は $i+1$ 行目の j_a 番目の石であり，石 B, C は i 行目の j_b, j_c 番目の石であるとする．

石 B について $\texttt{Danger}(i, j_b, l)$ を計算する際，石 A について $\texttt{Danger}(i+1, j_a, l)$ が呼び出され，A からの危険度の合計の最小を一度計算する．しかし，次に 石 C について $\texttt{Danger}(i, j_c, l)$ を計算する際にも，石 A について $\texttt{Danger}(i+1, j_a, l)$ が呼び出され，そこで，前に一度した A からの危険度の計算を全く同じくもう一度してしまう．

よって，そのような無駄を無くすようにすればよい．

実現

上記の無駄を無くすためには，解法 1 のプログラムに少し手を入れればよい．一度目に計算した際に，その値を記憶しておき，二度目からはその記憶しておいてある値を返すようにする．

例えば，3 次配列 memo を用意し，その全要素を -1 で初期化しておく．$\texttt{Danger}(i, j, l)$ が呼び出されたとき，

$\texttt{memo}[i, j, l] = -1 \rightarrow$ 解法 1 の時と同様に値を計算し，その値を $\texttt{memo}[i, j, l]$ に記憶した後に返す

$\texttt{memo}[i, j, l] \neq -1 \rightarrow \texttt{memo}[i, j, l]$ に記憶されている値を返す

というようにすればよい．

計算量

簡単のため，全ての行が k 個の石を含むとする．再帰呼び出しの回数の大体の見積もりをする．

関数 Danger が自身を呼び出す回数は，引数 i, j, l の 3 つの数字の組み合わせ 1 つについて，一度目なら $2k$ 回（通常のジャンプの場合，一行飛ばしの場合，それぞれ k 回ずつ），二度目以降は 0 回である．

よって，Danger の再帰呼び出しの回数は，

$$i, j, l \text{ の組み合わせの個数} \times 2k = n \times k \times (m+1) \times 2k \fallingdotseq 2nmk^2 \quad \text{回}$$

程度であると分かる．この解法の計算量は $O(nmk^2)$ である．

今回の問題の制限下では，全ての入力ファイルに対し制限時間内に計算が終わるであろうことがわかる．

解法 3（動的計画法）

考え方

解法 2 のアルゴリズムが行っていることは，配列 memo を，どんどんと順番に，値が既に入っている部分の値を使って，新しい値を計算し代入していっていることに他ならない．よって，再帰関数を用いず，ループ処理で memo を埋めてゆくこともできる．

ある石のそこからの反対側の岸までの危険度は，その石の一行先と二行先の全ての石についてそこからの危険度が計算されていれば求めることができる．よって，反対側の岸に近いほうの行から順に memo を埋めてゆけばよい．

実現

n 行目の全ての石についてまず初期値を代入する．$1 \leqq j \leqq k_n, 0 \leqq l \leqq m$ の全ての j, l について，$\mathtt{memo}[n, j, l] = 0$ を代入すればよい．また，$n-1$ 行目の全ての石についても初期値を代入する．$1 \leqq j' \leqq k_{n-1}, 0 \leqq l' \leqq m-1$ の全ての j', l' について，$\mathtt{memo}[n, j', l'] = 0$ を代入する．$l' = m$ の場合については，下の手順と同様に計算する．

i 行目の j 番目の石について，そこまでに l 回の一行飛ばしを行っている際の，そこから反対側の岸までの危険度 $\mathtt{memo}[i, j, l]$ を計算することを考える．$1 \leqq j' \leqq k_{i+1}$ を満たす全ての j' について

$$(d_{i,j} + d_{i+1,j'}) \times |x_{i,j} - x_{i+1,j'}| + \mathtt{memo}[i+1, j', l]$$

と，$l < m$ なら，$1 \leqq j'' \leqq k_{i+2}$ を満たす全ての j'' について

$$(d_{i,j} + d_{i+2,j''}) \times |x_{i,j} - x_{i+2,j''}| + \mathtt{memo}[i+2, j'', l+1]$$

を計算し，その最も小さいものを $\mathtt{memo}[i, j, l]$ の値として代入すればよい．（解法 1 で見た形と近いことに気づかれよう．）

この計算を大きい i から順次行い，後は，$1 \leqq j \leqq k_1$ を満たす全ての j について $\mathtt{memo}[1, j, 0]$ の値と，$m > 0$ なら，$1 \leqq j' \leqq k_2$ を満たす全ての j' について $\mathtt{memo}[2, j', 1]$ の値を見て，最も小さいものを答えとすればよい．

計算量

簡単のため全ての行が k 個の石を含むとする．ループの回数の大体の見積もりをする．

i 行目 j 番目の一つの石について，そこまでの一行飛ばしの回数が l 回の時の，そこからの危険度を確定する際，$2k$ 回（通常のジャンプの場合，一行飛ばしの場合，それぞれ k 回ずつ）のループが必要である．これを，$0 \leqq l \leqq m$ についての全ての l について，つまり $m+1$ 回行う．さらに，それを i 行目の全ての石について，つまり $1 \leqq j \leqq k$ の全ての j について，よって k 回行い，さらに，それを全ての行について，つまり $1 \leqq i \leqq n$ の全ての i について，よって n 回行う．

よって，ループの回数は，

$$2k \times (m+1) \times k \times n \fallingdotseq 2nmk^2 \quad \text{回}$$

程度となる．ここで見積もられたされたループの回数は解法 2 で見積もられた再帰呼び出しの回数と等しい．この解法の計算量も $O(nmk^2)$ であり，同様にこの解法でも問題の制限下では全ての入力に対して得点を得られるであろう事が分かる．

動的計画法とは

この解法 3 のように，そこまでに計算した値を用いてさらに計算を進めることを繰り返すような手法は**動的計画法**と呼ばれる．（あるいは解法 2 も動的計画法であると考えられることもある．）

一般的には，動的計画法とは，問題を複数の部分問題に分割し，部分問題らを解いてその答えを記憶しておき，それを使って元の大きな問題を解く方法である，などと定義される．このような定義はいまいち分かりにくいが，動的計画法はアルゴリズムを考える際のとても重要なアプローチの一つである．動的計画法が用いられる典型問題として,「ナップサック問題」「最長共通部分列問題」「最長増加部分列問題」等が挙げられる．動的計画法という言葉を聞いたことの無かった人や，動的計画法にあまり慣れていないという人は，それらを調べてみるなどして，どういうものであるかを把握しておくのがよいであろう．

5

ペンキの色

解法

この問題は指定された長方形の領域以外がいくつの領域に分割されるかを問う問題で，長方形の領域は最終的には大きな数値となるので座標圧縮が必要であることを最初に見抜き，その上でテープの領域の塗り方の工夫を考える必要があった．

座標圧縮

端も含めて全ての出てくる座標を整列し $c_0, c_1, \cdots, c_n$ とする．問題文より全てのテープは板に対して平行な辺を持つ長方形であるので，任意の $i(0 \leq i < n)$ で $c_i < a_1, a_2 < c_{i+1}$ となる任意の 2 つの数値 a_1, a_2 をとる時，a_1, a_2 は重なりの枚数 (または重なっていない状態) が同一である．なぜならば，もし違っている時その違いの区切りが出てくる座標が存在するはずであり $c_{0\cdots n}$ の中に区切りの位置が含まれていないというのは矛盾．よって成立する．よって $c_0, c_1, \cdots, c_n$ を $0, 1, \cdots, n$ に置き換えても同様の結果が得られることがわかる．

元の c_n の上限が 1000000 であるのに対し，置き換えた場合はたかだかテープの最大数 n とした場合 $2n + 2$ である．テープで塗られているかいないかの判定は格子毎に内容を保持しておく必要があるので，座標圧縮をするかしないかでメモリー領域が取れるか取れないかの差がある．ただし 30% では $w, h \leqq 100$ であるのでこれに対しては座標圧縮をしなくても点数が取れる．

テープの工夫

テープをすでに貼ったか否かというのを判定する時，新しいテープが来た時に処理する計算量は，テープの領域を全て埋めるという素朴な方法だと n 枚に対して $n \times n$ の領域を埋める必要性がありオーダー $O(n^3)$ となる．

しかし，ここで格子 $m(x, y)$ に対してテープ $i(0 \leq i < n)$ による (a_i, b_i) から (c_i, d_i) までの領域を埋める時に，$m'(x, y)$ に対して $(a_i, b_i \cdots d_i)$ に 1 を増やし，$(c_i + 1, b_i \cdots d_i)$ から 1 を引き，最後に $m(x, y) = \sum_{k=0}^{x} m'(k, y)$ とすればたかだか $O(n^2)$ で処理は完了する．

色を幅優先で塗る

$m(x, y) = 0$ であればその格子はテープが貼られていない，という状態になれば後は $m(x, y) = 0$ の連続領域の数を求めれば良い．

素朴な方法としては $m(x, y) = 0$ となる任意の x, y を探し出しそこから塗り始め，再帰を用いて $m(x, y)$ を塗った時，$m(x \pm 1, y), m(x, y \pm 1)$ を塗れるならば塗り，そのそれら $m(x \pm 1, y), m(x, y \pm 1)$ に対しても塗れるならば $m(x, y)$ と同様の処理をする．塗るべき領域がなくなれば別の $m(x, y) = 0$ となる任意の x, y を探し出しそこから同様のことを繰り返し，$m(x, y) = 0$ となる x, y が存在しなくなるまで繰り返せば良い．ここで塗り始めた回数が連続領域の数であり答えとなる．

ただしこれは再帰であるため場合によっては再帰できる深さをこえて，スタックオーバーフローを引き起こす場合がある．そこで全ての問題で正しい答えを出すには再帰関数を使わずに自前でスタックを組むか，幅優先探索により塗るかをしなければならない．

まとめ

上記 3 点全てを踏まえればテープの枚数を n として全体の計算量は $O(n^2)$ で実装できる．

この問題は計算量をシビアに見る問題であった．気付かなかった人は与えられる上限の条件を見て良く考える癖をつけていってほしい．