キーエンスプログラミングコンテスト 2020 解説

wo01

2020 年 1 月 18 日

A 問題

1 回のペイント操作で黒く塗られたマスを $\max(H,W)$ 個増やすことができます。よって、求める値は $M=\max(H,W)$ として、$KM \geq N$ を満たす最小の整数 K となります。

M の値を求めるには、条件分岐を用いるか、標準ライブラリとして多くの言語で用意されている関数 max を用いればよいです。

$KM \geq N$ を満たす最小の整数 K の値は、演算 / を整数除算として、

- N が M で割り切れるとき、N/M
- そうでないとき、$N/M+1$

となるので、条件分岐を用いて求めることができます。なお、$(N+M-1)/M$ とすれば条件分岐なしでも求めることができます。

以下に C++ による実装例を示します。

```
#include<cstdio>

using namespace std;

int main(){
    int H, W, N;
    scanf("%d%d%d", &H, &W, &N);
    int m = H > W ? H : W;
    int ans = N / m;
    if(N % m != 0) ans++;
    printf("%d\n", ans);
    return 0;
}
```

B 問題

この問題の原案はキーエンス社の社員により提供されました

$S_i = X_i - L_i, T_i = X_i + L_i$ とします。ロボット i の腕が動く範囲は (S_i, T_i) となります。

ロボット i を残すかどうかは座標 T_i 以上の部分に影響を与えないことに注意すると、以下のような貪欲法により答えを求めることができます。

- ロボットを T_i が小さい順にソートし、$p_1, p_2, \ldots, p_N$ とする。
- $i = 1, 2, \ldots, N$ の順に、以下のことを行う。
  - ロボット p_i の腕がすでに残すと決めたロボットの腕と重ならないならば、ロボット p_i を残すと決める。そうでないなら、残さないと決める。

以下に C++ による実装例を示します。

```
#include<cstdio>
#include<utility>
#include<algorithm>

using namespace std;

typedef pair<int, int> P;

const int MAX_N = 100000;
const int MAX_V = 1000000000;

int N;
int X[MAX_N];
int L[MAX_N];

void input(){
    scanf("%d", &N);
    for(int i = 0; i < N; ++i){
        scanf("%d%d", X + i, L + i);
    }
}

P ps[MAX_N];

int solve(){
    for(int i = 0; i < N; ++i){
```

```
        ps[i] = P(X[i] + L[i], X[i] - L[i]);
    }
    sort(ps, ps + N);
    int cur = -MAX_V;
    int ans = 0;
    for(int i = 0; i < N; ++i){
        if(cur <= ps[i].second){
            ans++;
            cur = ps[i].first;
        }
    }
    return ans;
}

int main(){
    input();
    int ans = solve();
    printf("%d\n", ans);
    return 0;
}
```

C: Subarray Sum

以下のような数列が条件をを満たします。

- $S < 10^9$ のとき、$S, S, \ldots, S, S+1, S+1, \ldots, S+1$
- $S = 10^9$ のとき、$S, S, \ldots, S, 1, 1, \ldots, 1$

ただし S は K 個並べます。

D: Swap and Flip

カードを $p_1, p_2, \ldots, p_N$ の順に並べ替えるために必要な操作回数は、$i < j$ かつ $p_i > p_j$ を満たす (i, j) の個数に一致します。この個数を転倒数といいます。どのようにカードを並べれば表面の整数が単調増加となるか考えましょう。

カード i が何回かの操作のあと左から j 番目の位置に来たとします。このとき、上を向いている面に書かれた値は、操作手順によらず、

- $i - j$ が偶数ならば A_i
- $i - j$ が奇数ならば B_i

となります。

このことを用いると、

- $dp_{mask,i} = mask \subseteq \{1, 2, \ldots, N\}$ に含まれるカードを左から何枚かのカードとして単調増加になるように並べるときの、この部分の転倒数への寄与

という動的計画法により答えを求めることができます。

E: Bichromization

与えられるグラフを G とします。

まず、明らかに不可能なケースを考えましょう。ある頂点 v が存在して、任意の v に隣接する頂点 u に対して $D_v < D_u$ が成り立つ場合、割り当ては不可能です。なぜなら、頂点 v から色が異なる頂点への最小のコストのパスを $v = v_1, v_2, \ldots, v_k$ とするとき、

- $k > 2$ ならばパス $v_2, \ldots, v_k$ が
- $k = 2$ ならばパス v_2, v_1 が

それぞれ頂点 v_2 から色が異なる頂点へのパスとなっているからです。

上記以外の場合は以下のようにして色と重みを割り当てることができます。

まず、各頂点 v に対して、それに隣接する頂点 u のうち D_u の値が最小のもの（複数ある場合、それらのうち番号 u が最小のもの）をとって p_v とします。そして、グラフ G の各頂点 v について辺 $\{v, p_v\}$ を残したグラフ G' を考えます。このグラフは森になります。

次に、頂点の色、および G' に含まれる辺の重みの割り当てを考えます。頂点の色を、各 v について 頂点 v と頂点 p_v が異なる色になるように定めます。さらに、森に含まれる辺 $\{v, p_v\}$ に重み D_v を割り当てます。これでグラフの形が G ではなく G' であれば距離の条件が満たされるようになりました。

最後に G' に含まれない辺に重みを割り当てる必要がありますが、このような辺についてはすべて重み 10^9 を割り当てれば良いことがわかります。

以上で割り当てが得られました。

F: Monochromization

(1/21 公開)

必要な考察をふたつのステップにわけて解説します。第一ステップはダブルカウントを防ぐためのものであり、第二ステップは実際に数え上げを行うものです。最後にこれらをまとめてアルゴリズムの全体像を示します。

第一ステップ: 塗られ方の正規形

便宜上、一回の操作で以下のいずれかの塗り方ができるものと考えます。

操作 1 行をいくつか選び、白または黒の色を選ぶ。選んだ各行について選んだ色に塗る。
操作 2 行をいくつか選ぶ。選んだ各行について白または黒の色を選び、その色に塗る。ただし両方の色が出現するようにする。
操作 3 列をいくつか選び、白または黒の色を選ぶ。選んだ各列について選んだ色に塗る。
操作 4 列をいくつか選ぶ。選んだ各列について白または黒の色を選び、その色に塗る。ただし両方の色が出現するようにする。
操作 5 白または黒の色を選ぶ。行および列をいくつか選び、それらすべてを最初に選んだ色に塗る。

すぐにわかるとおり、異なる操作の結果同じ塗られ方が得られることがあります。操作後の塗られ方 G' から操作列を一意に復元できるようにしましょう。そのためには、以下のように塗られたとみなせばよいです。

- G' において一色だけからなる行、列は最後の操作で塗られたとみなす。
- 上記の行、列を除いて一色だけからなる行、列は最後から二回目の操作で塗られたとみなす。
- 以下同様に、一色だけからなる行、列がなくなるまで繰り返す。

第二ステップ: 塗り方の数え上げ

塗られたとみなす行、列の集合を決めたとき、得られる塗られ方が何通りあるかを数えることを考えます。明らかに塗られ方の個数は塗る行、列の数だけに依存します。r 行 c 列塗って得られる塗られ方の個数を求めるためのふたつの方法を説明します。第一のものはただちに DP をはじめるもので、第二のものはもう少し数学的考察を進めるものです。

- 方法 1
  $dp_{i,j,k}$ = (操作 k からはじめて i 行 j 列を塗る方法の数) という動的計画法を行えばよいです（操作の番号は第一ステップでつけたものです)。
  $r = H$ かつ $c = W$ の場合とそうでない場合で初期状態や遷移が少し異なることに注意してください。
- 方法 2
  まず、$h \times w$ の真っ白なグリッドから問題文の操作により得られるグリッドの個数を求めましょう。得られるグリッドは、階段状の線を切れ目として片側を白く、もう片側を黒く塗って得られるグリッドの行、列を並べ替えたものとなります。このような塗られ方の個数 $p_{h,w}$ は簡単に求められます。

では、もとの問題に戻りましょう。$r = H$ かつ $c = W$ の場合、求める場合の数は $p_{H,W}$ です。$r < H$ または $c < W$ の場合を考えます。黒く塗る行の集合、白く塗る行の集合、塗らない行の集合が R_B, R_W, R_N であったとします。列の集合 C_B, C_W, C_N も同様に定めます。$R_W \times C_B$ および $R_B \times C_W$ 以外のマスが何色に塗られるかは操作の順番によりません。また、これらの部分の塗られ方は階段状の線を切れ目として片側を白く、もう片側を黒く塗って得られるグリッドの行、列を並べ替えたものとなります。さらに、$R_W \times C_B$ の塗られ方は R_W, C_B に対する操作の順番だけに、$R_B \times C_W$ の塗られ方は R_B, C_W に対する操作の順番だけに依存するので、これらは独立に定めることができます。よってこの場合に得られる塗られ方の個数は上で計算した p を用いて計算できます。

まとめ

以上の考察より、行集合 $rmask$ と列集合 $cmask$ の組であって、もとのグリッドから $rmask$ と $cmask$ を取り除いたグリッドに一色だけからなる行、列が存在しないようなものに対して、第二ステップで求めた値を足し合わせることにより、答えを求めることができるとわかります。

Keyence Programming Contest 2020 Editorial

wo01

January 18, 2020

Problem A

In one operation, we can increase the number of black squares by $\max(H, W)$. Thus, the answer is the minimum integer K such that $KM \geq N$, where $M = \max(H, W)$.

To find the value M, we can use a conditional branch or the function max, which is available in standard libraries in many languages.

If we use a conditional branch, the minimum integer K such that $KM \geq N$ can be found as:

- N/M, if M divides N
- $N/M + 1$, otherwise

where the operator / denotes integer division. We can also find it as $(N+M-1)/M$ without a conditional branch.

Sample C++ implementation follows:

```
#include<cstdio>

using namespace std;

int main(){
    int H, W, N;
    scanf("%d%d%d", &H, &W, &N);
    int m = H > W ? H : W;
    int ans = N / m;
    if(N % m != 0) ans++;
    printf("%d\n", ans);
    return 0;
}
```

Problem B

This problem is proposed by KEYENCE.

Let $S_i = X_i - L_i$ and $T_i = X_i + L_i$. The movable range of arms of Robot i is (S_i, T_i).

Noticing that whether Robot i is kept or not does not affect the part with coordinate T_i or greater, we can find the answer with the following greedy algorithm:

- Sort the robots in ascending order of T_i, and let the result be $p_1, p_2, \ldots, p_N$.
- For each $i = 1, 2, \ldots, N$ in this order, do the following:
  - If the movable range of arms of Robot p_i does not intersect with those of the robots that we decided to keep, we decide to keep Robot p_i. Otherwise, we decide to remove this robot.

Sample C++ implementation follows:

```
#include<cstdio>
#include<utility>
#include<algorithm>

using namespace std;

typedef pair<int, int> P;

const int MAX_N = 100000;
const int MAX_V = 1000000000;

int N;
int X[MAX_N];
int L[MAX_N];

void input(){
    scanf("%d", &N);
    for(int i = 0; i < N; ++i){
        scanf("%d%d", X + i, L + i);
    }
}

P ps[MAX_N];

int solve(){
    for(int i = 0; i < N; ++i){
```

```
        ps[i] = P(X[i] + L[i], X[i] - L[i]);
    }
    sort(ps, ps + N);
    int cur = -MAX_V;
    int ans = 0;
    for(int i = 0; i < N; ++i){
        if(cur <= ps[i].second){
            ans++;
            cur = ps[i].first;
        }
    }
    return ans;
}

int main(){
    input();
    int ans = solve();
    printf("%d\n", ans);
    return 0;
}
```

C: Subarray Sum

The following sequence satisfy the conditions.

- If $S < 10^9$, $S, S, \ldots, S, S+1, S+1, \ldots, S+1$.
- If $S = 10^9$, $S, S, \ldots, S, 1, 1, \ldots, 1$.

Here S is repeated K times.

D: Swap and Flip

The number of operations required to rearrange the cards in the order $p_1, p_2, \ldots, p_N$, is equal to the number of pairs (i, j) such that $i < j$ and $p_i > p_j$, which is called the inversion number. Let us consider how we should rearrange the cards so that we have a non-decreasing sequence facing up.

Assume that Card i comes to the j-th position from the left after some number of operations. Facing up on this card is:

- A_i if $i - j$ is even
- B_i if $i - j$ is odd

regardless of the sequence of operations performed.

Based on this fact, we can find the answer with the following dynamic programming:

- $dp_{mask,i}$ = The contribution of the cards in $mask \subseteq \{1, 2, \ldots, N\}$ to the inversion number when these cards are brought to the leftmost positions, showing up a non-decreasing sequence

E: Bichromization

Let G be the given graph.

First, let us exclude a case in which the assignment is obviously impossible. If there exists a vertex v such that $D_v < D_u$ for every vertex u adjacent to v, the assignment is impossible. It follows from the following fact. Let $v = v_1, v_2, \ldots, v_k$ be the path from the vertex v to a vertex of different color, with the minimum cost. Then,

- If $k > 2$, the path $v_2, \ldots, v_k$ goes from the vertex v_2 to a vertex of different color.
- If $k = 2$, the path v_2, v_1 goes from the vertex v_2 to a vertex of different color.

In other cases, the following assignment of colors and weights is possible.

First, for each vertex v, among the vertices adjacent to v, let the vertex u with the minimum value of D_u be p_v. (If there are more than one such vertices, we choose the vertex with the minimum index u.) Then, for each vertex v in the graph G, let us keep the edge $\{v, p_v\}$, and remove the other edges. Let the resulting graph be G', which will be a forest.

Now, let us assign colors to the vertices and weights to edges in G'. We will decide the colors of vertices so that, for each v, the vertices v and p_v have different colors. Then, we will assign the weight D_v to each edge $\{v, p_v\}$ contained in the forest. At this point, the condition would be satisfied if the given graph was G' instead of G.

Lastly, we need to assign weights to the edges not contained in G'. It turns out that we can just assign 10^9 to each of those edges.

We now have a valid assignment.

F: Monochromization

We will solve the problem in two steps: inventing a way to avoid counting the same pattern multiple times, then actually computing the count. Lastly, we will summarize the process and show the whole picture.

Step 1: The normalized form of a painting sequence

For convenience, let us assume that we can do one of the following in one operation:

Operation 1 Choose some rows and a color (black or white). Paint each chosen row in the chosen color.

Operation 2 Choose some rows. For each chosen row, choose a color (black or white) and paint the row in that color. Here we must use both colors at least once.

Operation 3 Choose some columns and a color (black or white). Paint each chosen column in the chosen color.

Operation 4 Choose some columns. For each chosen row, choose a color (black or white) and paint the column in that color. Here we must use both colors at least once.

Operation 5 Choose a color (black or white). Choose some rows and some columns, and paint all of them in the chosen color.

It can be easily seen that the same pattern may be obtained from different sequences of operations. Let us make sure that, for a given pattern G', there is a unique sequence of operations that results in G', by only considering the painting sequence that satisfies the following rules:

- If there is a row or column containing only one color in G', we assume that row or column is painted in the last operation.
- After excluding the row or column above, If there is a row or column containing only one color, we assume that row or column is painted in the second last operation.
- We continue in this way, until there is no row or column containing only one color.

Step 2: Counting the painting sequences

Let us count the patterns that can be obtained when we fix the set of rows and columns that we assume are painted. Obviously, this count only depends on the numbers of rows and columns that are painted. We will introduce two methods to find $cnt_{r,c}$, the number of patterns that can be obtained by painting r rows and c columns. The first one immediately does DP, and the second one makes a little more mathematical observations.

- Method 1
  We can find $cnt_{r,c}$ by doing the following DP: (See Step 1 for descriptions of Operations.) $dp_{i,j,k}$ = (The number of ways to paint i rows and j columns starting with Operation k)
  Note that, when $r = H$ and $c = W$, the initial states and transitions slightly differ from the other

cases.

- Method 2

  First, let us count the patterns that can be obtained from an $h \times w$ completely white grid with the operations in the statement. Those patterns are the patterns that can result from rearranging rows and columns of a grid divided into two parts by a stair-like line, where one of those parts are painted black and the other is painted white. We can easily find the number of those patterns, $p_{h,w}$.

  Let us get back to the original problem. If $r = H$ and $c = W$, the answer is $p_{H,W}$. Consider the case $r < H$ or $c < W$. Let R_B, R_W, R_N be the set of rows painted black, the set of rows painted white, and the set of rows unpainted, respectively. We will define C_B, C_W, C_N similarly. The final colors of the squares that are not in $R_W \times C_B$ or $R_B \times C_W$ do not depend on the order of operations. Also, those parts are painted in a way that can result from rearranging rows and columns of a grid divided into two parts by a stair-like line, where one of those parts are painted black and the other is painted white. Furthermore, the final colors of the squares that are in $R_W \times C_B$ only depend on the relative order of operations on R_W, C_B, and the final colors of the squares that are in $R_B \times C_W$ only depend on the relative order of operations on R_B, C_W, so we can treat these two parts individually. Thus, we can count the patterns that can be obtained in this case using p computed above.

Summary

From the observations above, we can find the answer by summing the values found in Step 2 over all pairs of a row set *rmask* and a column set *cmask* such that there is no row or column containing only one color in the given grid after removing *rmask* and *cmask*.