DDCC 2020 予選 解説

E869120, square1001

2019 年 11 月 23 日

For International Readers: English editorial starts on page 12.

DISCO presents ディスカバリーチャンネル コードコンテスト 2020 予選 解説

Writer: E869120, square1001

2019.11.23

問題 A – 「DDCC Finals」

＜解説＞

この問題は、プログラミング初心者にとっては設定が少し複雑で一見難しそうに見える問題かもしれませんが、少し整理してみると簡単に見えてくると思います。

DISCO 君が本戦で獲得する賞金は、以下の 3 つの合計になります。

- 「コード部門」で獲得する賞金 (30 万円, 20 万円, 10 万円, 0 円のいずれか)
  - $X = 1$ だと 30 万、$X = 2$ だと 20 万、$X = 3$ だと 10 万、それ以外で 0 円
- 「装置実装部門」で獲得する賞金 (30 万円, 20 万円, 10 万円, 0 円のいずれか)
  - $Y = 1$ だと 30 万、$Y = 2$ だと 20 万、$Y = 3$ だと 10 万、それ以外で 0 円
- 「両部門同時優勝」でのボーナス賞金 (40 万円, 0 円のいずれか)
  - 「$X = 1$ かつ $Y = 1$」だと 40 万円、それ以外で 0 円

それぞれの賞金額を計算するプログラムは、条件分岐 (多くのプログラミング言語では if-else 文) を使うと書けます。3 つの賞金額が求まれば、あとはこの 3 つを足し算するだけで DISCO 君が得られる賞金額が計算できます！

ちなみに、実際の「DDCC 2020」では、コード部門で 1 位, 2 位, 3 位になった人が得られる商品はそれぞれ「30 万円以下の PC」、「20 万円以下の PC」、「10 万円以下の PC」です。

＜必要なプログラミングの知識＞

- 入力・出力
- 条件分岐 (if-else 文)

＜サンプルコード (Python 3)＞

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8559133

問題 B –「Iron Bar Cutting」

＜解説＞

この問題は、問題文をよく読むと、以下の問題と同じ問題であることが分かります。

> 長さ N の整数列 $A = \{A_1, A_2, A_3, \ldots, A_N\}$ がある。
> あなたはこの数列に対して、以下の操作のいずれかを行うことができる。
> - 整数 t を選び、A_t に 1 を加算する。
> - 整数 t を選び、A_t に 1 を減算する。減算前の時点で $A_t \geq 1$ でなければならない。
>
> そのとき、$B_1 + B_2 + \cdots + B_s = B_{s+1} + B_{s+2} + \cdots + B_N$ となるような s が存在するようにしたいです。操作回数の最小値を求めてください。

上の問題について考えていきましょう。

操作前の時点で、$B_1 + B_2 + B_3 + \cdots + B_s = t_1$、$B_{s+1} + B_{s+2} + \cdots + B_N = t_2$ とするとき、以下の操作を行うことで $|t_1 - t_2|$ 回の操作が達成でき、これが最小です。

- $B_1 + B_2 + \cdots + B_s < B_{s+1} + B_{s+2} + \cdots + B_N$ のとき
  - B_1 に 1 を加算する。
- $B_1 + B_2 + \cdots + B_s > B_{s+1} + B_{s+2} + \cdots + B_N$ のとき
  - B_N に 1 を加算する。

つまり、$|(B_1 + B_2 + \cdots + B_s) - (B_{s+1} + B_{s+2} + \cdots + B_N)|$ の最小値が答えであるということが分かります。この値は普通に s を全探索してそれぞれについて値を求めると、計算量 $O(N^2)$ かかってしまい間に合いませんが、累積和を用いることによって間に合います。

具体的には、$B_1 + B_2 + B_3 + \cdots + B_i$ の和と $B_i + B_{i+1} + B_{i+2} + \cdots + B_N$ の和を $1 \leq i \leq N$ すべてについて持っておくとよいです。例えば $A = \{2, 4, 6, 1, 3, 5\}$ の場合、以下のようになります。

i の値	1	2	3	4	5	6
$B_1 + B_2 + B_3 + \cdots + B_i$	2	6	12	13	16	21
$B_i + B_{i+1} + B_{i+2} + \cdots + B_N$	21	19	15	9	8	5

＜サンプルコード (C++)＞

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8320272

問題 C –「Strawberry Cakes」

例えば入出力例 1 を見ても、どうやってうまくジグソーパズルのように“敷き詰めて” $H \times W$ の長方形にできるのか分かりそうにないと思います。そこで、別の視点から考えてみましょう。

ここでは、2 つの解法を載せておきました。解法 #2 の方が簡単だと思いますが、解法 #1 を見ると再帰関数に対する知見が広がると思うので、両方見ることをおすすめします。

★ 解法 #1：再帰的に考える

どんなイチゴの配置でも、長方形のショートケーキに分けてそれぞれイチゴが 1 個ずつ乗っている、というような切り分け方はあります。これは問題文中に書かれています。これをヒントにして考えてみましょう。

$H \times W$ のケーキを 2 等分することを考えます。縦に切る時は $H \times t$ と $H \times (W - t)$ の 2 つの長方形のケーキに分かれ、横に切る時は $s \times W$ と $(H - s) \times W$ の長方形のケーキに分かれます。

分けられた 2 つのケーキそれぞれにイチゴが 1 個以上あれば、必ずその分けられたケーキに対して「条件を満たす分け方」があります。

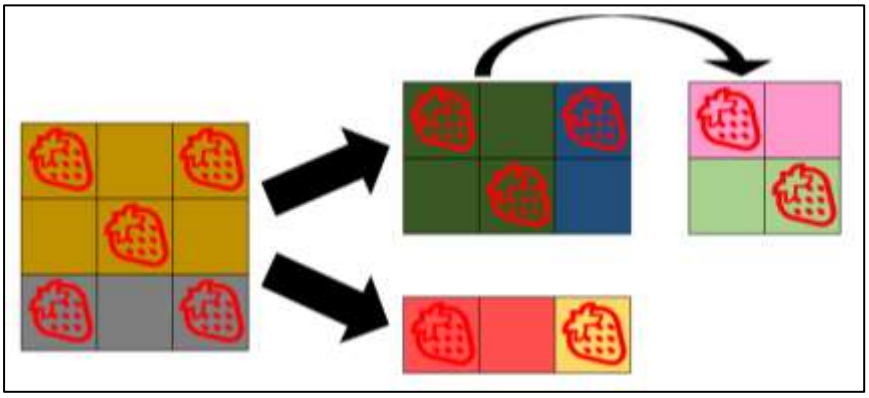


図 1. 入力例 1 で再帰的に分け方を求める例

ケーキにイチゴが 2 個以上ある場合、イチゴがどんな配置をしていても、両長方形にイチゴが 1 個以上含まれるようにできるのでしょうか？答えは Yes です。

イチゴが 2 個の場合簡単です。2 つのイチゴが違う行にある場合、この間で横に切ることができ、違う列にある場合はその間で縦に切ることができるからです。

イチゴが 3 個以上の場合、2 つのイチゴだけを見てこれが分かれるような切り方をすれば (このような切り方は 2 個の場合と同じなのでできる) 両長方形にイチゴが 1 個以上含まれるようにできます。

これで、どんな入力に対しても条件を満たすような分け方を見つける方法が分かりました！実装は、再帰関数などを使うことでできます。計算回数は $O\big(H \times W \times (H + W)\big)$ 程度なので、適切に実装すれば間に合います。

★ 解法 2. 横に切って、その後縦に切る

次のような方法でケーキを K 個に切り分けることを考えてみましょう。

＜ステップ #1：ケーキをできるだけ横に切り分ける＞

まず、できるだけ横に切り分けることを考えます。全部の行に切れ目を入れて H 個に分割すると、イチゴが全く乗っていないピースがあるかもしれません。

ですのでその代わりに、このようなピースがないような中でできるだけ多く横に切ります。**そうすると、各ピースでイチゴは全部同じ行に集中することになります。**

＜ステップ #2：各ピースを縦に切り分ける＞

各ピースでイチゴは全部同じ行に集中するので、同じ列には 2 つ以上のイチゴが乗ることはありません。したがって、できるだけ縦に切り分けると各ピースにちょうど 1 個のイチゴが乗っているようにすることができます。

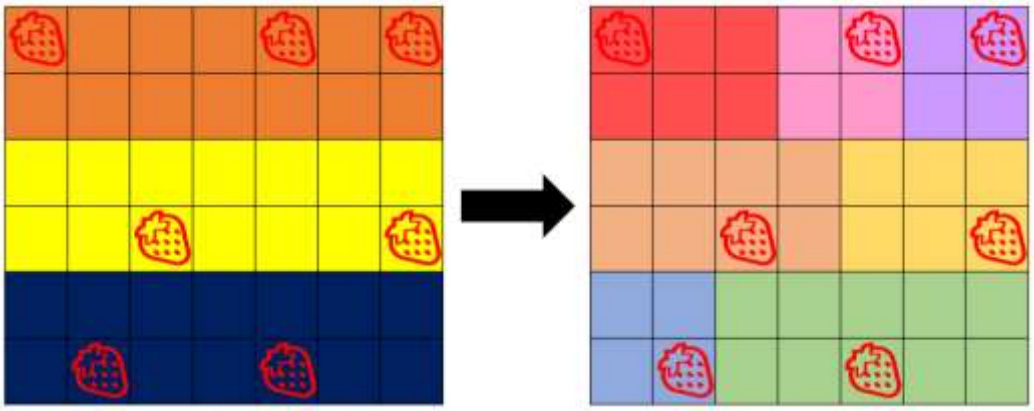


図 2. 解法 #2 での切り方（左がステップ #1、右がステップ #2）

「できるだけ切り分ける」実装は、まだ切り分けられるところが見つかったなら切る、見つからなかったらこれで打ち切る、というようにできます。この実装だと、計算回数は $O\big(H \times W \times (H+W)\big)$ となります。

ただ、「何もイチゴがない行・列を他の行・列のピースにつなげる」という感じで「できるだけ切り分ける」処理を適切に実装すると、$O(H \times W)$ の計算回数で答えを求めることができます。

サンプルコード（C++）

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8320839

ボーナス問題

解法 #1 の計算回数は $O\big(H \times W \times (H+W)\big)$ と書かれていますが、この計算回数でできる証明は少し難しいです。証明してみましょう。

問題 D –「Digit Sum Replace」

＜解説＞

例えば、$N = 229$ 人の人が予選に参加する場合を考えます。そのとき、3 通りの方法があります。

- 229 → 49 → 13 → 4 人、と減らす。
- 229 → 211 → 31 → 4 人、と減らす。
- 229 → 211 → 22 → 4 人、と減らす。

どのやり方でも 3 回の予選が開かれることになるのが分かるでしょう。

実は、N がどのような値であっても、開かれる予選の回数は同じです。これはなぜでしょうか？

【パターン #1】合計が 9 以下となる連続する 2 つの桁を選ぶと、次のラウンドの参加者の人数の桁数は 1 減り、各桁の数字の和は変わりません。

【パターン #2】合計が 10 以上となる連続する 2 つの桁を選ぶと、次のラウンドの参加者の人数の桁数は変わらず、各桁の数字の和は 9 減ります。

表にまとめてみると、次のようになります。

パターン	桁数の減少	各桁の数字の和の減少
#1	1	0
#2	0	9

つまり、N の桁数を D、各桁の数字の和を S とすると：

- 桁数は 1 で終わるので、パターン #1 はちょうど $D - 1$ 回行われる
- 各桁の数字の和は 1～9 のどれかで終わるので、パターン #2 はちょうど「$S - 1$ を 9 で割った商」回行われる

よって、予選の回数は $(D - 1) + \lfloor (S - 1) \div 9 \rfloor$ となります。($\lfloor x \rfloor$ は x を整数で切り捨てた値です) これを求めるプログラムを書けば、正解となります。

＜サンプルコード (Python)＞

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8562444

問題 E –「Majority of Balls」

＜解説＞

まず、以下の条件を満たす N 個の球の集合を求めることを考えましょう。

- N 個の球 $\{V_1, V_2, \dots, V_{N-1}, W\}$ において、赤のボールの方が多い。（球の集合 S_1 とする）
- N 個の球 $\{V_1, V_2, \dots, V_{N-1}, W\}$ において、青のボールの方が多い。（球の集合 S_2 とする）

これは、二分探索によって求めることができます。まずは、1 以上 $N+1$ 以下のすべての i について、ボール $i, i+1, \dots, i+N-1$ の中で赤と青どちらが多いかを考えてみましょう。例えば、$S =$ "$RBRBRBBRRB$" の場合：

i の値	1	2	3	4	5	6
赤の数 / 青の数	3 / 2	2 / 3	2 / 3	2 / 3	3 / 2	2 / 3
どちらが多いか	赤	青	青	青	赤	青

さて、i の値が決まった時の「どちらが多いか」を f_i とします。例えば、上の例では $f_1 = red, f_2 = Blue$ です。そこでよく考えてみると、f_1 と f_{N+1} は**必ず異なる**ことが分かります。なぜなら、ボール $1, 2, \dots, N$（f_1 に関係しているボール）とボール $N+1, N+2, \dots, 2N$（f_{N+1} に関係しているボール）の合計は、必ず赤 N 個・青 N 個になるからです。

そこで、$f_i \neq f_{i+1}$ を満たすような i を見つけるためには、以下の操作を $R - L = 1$ となるまで繰り返せばよいです。最初、$L = 1, R = N+1$ とします。尚、この操作において、$f_L \neq f_R$ は常に成り立ちます。

- $M = \frac{L+R}{2}$ とおく。
  - $f_M = f_L$ の場合、$L = M$ として R の値は変えない。
  - $f_M = f_R$ の場合、$R = M$ として L の値は変えない。

そうすると、$\log_2 N$ 回程度の操作で $f_i \neq f_{i+1}$ が求まり、集合 S_1, S_2 は以下のようになります。

- $S_1 = \{i, i+1, i+2, \dots, i+N-2,\ i+N-1\}$
- $S_2 = \{i+1, i+2, i+3, \dots, i+N-1, i+N\}$　（赤は共通部分）

特に、以下の球の集合については、赤の球と青の球が同数存在します。

- 球の集合 T_1: $\{i+1, i+2, i+3, \dots, i+N-1\}$
- 球の集合 T_2: $\{i+N+1, i+N+2, \dots, 2N, 1, 2, \dots, i-1\}$

さて、球 t が赤か青かを 1 回で判定するには、以下の 4 つに条件分岐をして質問を行えばよいです。（球 $1, 2, \ldots, 2N$ はすべて 4 つの条件のうちいずれか 1 つにあてはまります。）

- $t = i$ の場合：この球の色は前の $\log_2 N$ 回の質問で特定できている。
- $t = i + N$ の場合：この球の色は前の $\log_2 N$ 回の質問で特定できている。
- t が T_1 に入っている場合：T_2 と球 t の合計 N 個の球について質問する。T_1 は赤と青が同数あるため、質問に対する答えが赤だった場合は球 t が赤であることが分かる。また、質問に対する答えが青だった場合は球 t が青であることが分かる。
- t が T_2 に入っている場合：T_1 と球 t の合計 N 個の球について質問する。T_1 は赤と青が同数あるため、質問に対する答えが赤だった場合は球 t が赤であることが分かる。また、質問に対する答えが青だった場合は球 t が青であることが分かる。

合計して、$2N - 2 + \log_2 N$ 回の質問ですべての球の色が分かります。

$N \leq 99$ より、$198 - 2 + 7 = 203$ 回の質問で答えを導くことができます。

<サンプルコード>

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8323705

問題 F –「DISCOSMOS」

この問題は最終問題なので難しいです。最初全く取り掛かれないように思えるかもしれませんが、「$T = 1$ の場合が解ければ、あとは動的計画法で計算できる」ということに気づけば、方針が立ちやすいと思います。ただ、$T = 1$ の場合も、この問題特有の性質を生かし、「斜めに注目する」ことをひらめくのが重要になってくる難問です。

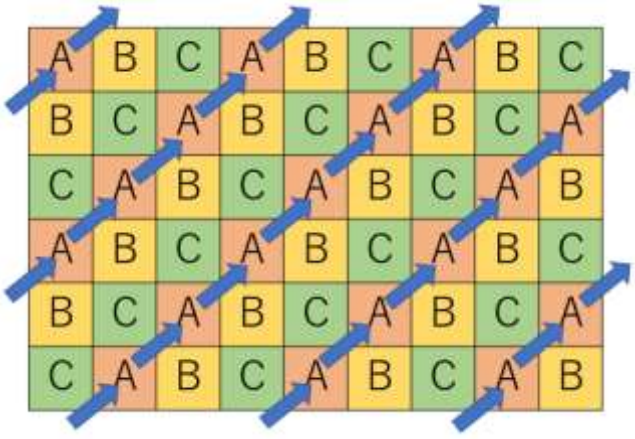
それに対し、他の多くの数え上げ問題と違い、mod p 上の逆元や二項係数 $C(n,k)$ を求めるなどという高等的な知識は必要ありません。

一つ注意ですが、この解説では (N,y) のマスから 1 マス下に動くと $(1,y)$ に着き、$(x,1)$ のマスから 1 マス左に動くと (x,N) のマスに着く、のように端を超えたら逆側に行くものとします。

★ ステップ #1：$T = 1$ の場合、2 種類までしかロボットはない

あるマス (x,y) から「右に 1 マス、上に 1 マス進む」ことを繰り返すと、いずれ元のマスに戻ってきます。これを「(x,y) のゾーン」ということにします。明らかに、(x,y) のゾーンに (p,q) が属するとき、逆に (p,q) のゾーンも (x,y) のゾーンに属します。

このとき、G を「H と W の最大公約数」として、ゾーンは G 種類あり、それぞれ $(0,0),(0,1),(0,2),\ldots,(0,G-1)$ のゾーンであることが分かると思います。

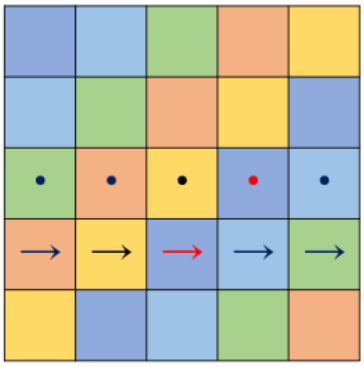


上図のように、$H = 6, W = 9$ の場合、「ゾーン」は $G = 3$ 種類あり、それぞれ $(0,0),(0,1),(0,2)$ のゾーン（それぞれ A, B, C としている）なことが分かると思います。

ここで、**「同じゾーンに右移動型のロボットと下移動型のロボットがある」場合、いずれ同じマスにたどり着きます。**これは、右移動型のロボットの位置から何回か「右に 1 マス、上に 1 マス進む」ことで下移動型のロボットにたどり着くからです。

なので、$D = 1$ の場合、同じゾーンには「右移動型のロボット」と「下移動型のロボット」の両方があることはない、ということが分かります。

次に、同じゾーンに「右移動型のロボット」と「停止型のロボット」が両方ある場合を考えます。この場合、必ず「停止型のロボット」から「1 個上、1 個右に進んだ位置」に「右移動型のロボット」があるようなマスが存在します。



このとき、停止型のロボットと同じ行のマスは**すべて**停止型のロボットにならなければ条件を満たしません。

右図の $H = 5, W = 5$ の例を見てみましょう。例えば、黒文字の 2 マスが（左から順に）右移動型・停止型であることが決まっているものとします。そのとき、赤文字の 2 マスの種類は（左から順に）右移動型・停止型の組み合わせです。残り 8 通りの組み合わせでは、いずれ同じマスに重なります。

これで黒文字・赤文字の 4 つのマスのロボットの種類が決まりました。1 個ずつ右に広げていくと、最終的に元の位置に戻り、青文字のマスもすべて「右移動型・停止型」でなければならない、ということが分かります。

したがってこの場合、すべての列に停止型ロボットがあることになります。1 つのゾーンすべてが「停止型のロボット」の場合でも、すべての列に停止型ロボットがあるので、次のことが言えます。

- $H \times W$ 個のロボットの中に停止型・右移動型両方のロボットがあれば、下移動型のロボットはない

また、行と列を逆にすると、同じように次のことが言えます。

- $H \times W$ 個のロボットの中に停止型・下移動型両方のロボットがあれば、右移動型のロボットはない

よって、**$T = 1$ の場合、ロボットは 2 種類しかない**ことが分かりました！これに気づけばこの問題は半分解けたようなものです！

(解説は次ページへ続く)

★ ステップ #2：$D = 1$ の場合何通りあるか数えてみよう

次の 2 パターンに分けて、何通りあるか数えてみましょう。

＜パターン #1：停止型がない場合＞

最初の方で、「同じゾーンには右移動型と下移動型のロボット両方はない」ことは既に言いました。しかし、右移動型・下移動型のロボットは 1 秒経つと“1 個先のゾーン”に行くので、別のゾーンのロボットが重なることはありません。

よって、G 個のゾーンに「全部右移動型」か「全部下移動型」か設定することになるので、2^G 通りです。

＜パターン #2：停止型が 1 個でもある場合＞

次に、停止型が 1 個でもある場合を考えます。

右移動型と停止型両方がある場合、H 個の行に「全部右移動型」か「全部停止型」か設定することになるので、「全部『全部右移動型』」と「全部『全部停止型』」の 2 通りを除いた $2^H - 2$ 通りになります。

同様に、下移動型と停止型両方がある場合、W 個の列に「全部下移動型」か「全部停止型」か設定することになるので、「全部『全部下移動型』」と「全部『全部停止型』」の 2 通りを除いた $2^W - 2$ 通りになります。

最後に、全部停止型の場合が 1 通りあるので、パターン #2 は $2^H + 2^W - 3$ 通りであることが分かりました。

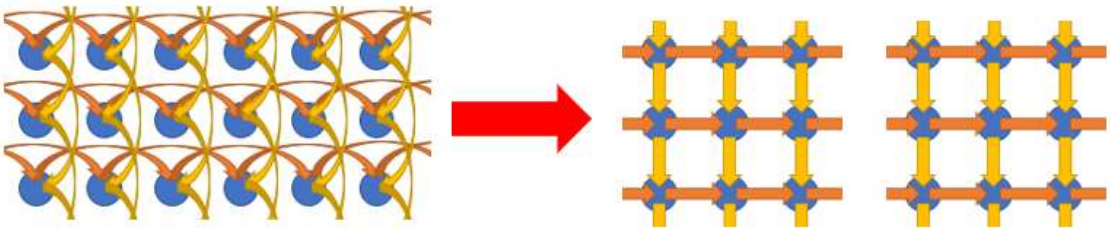
よって、$K = 1$ の場合の答えは $2^H + 2^W + 2^G - 3$ 通りです。特に $H = W = N$ の場合、$3 \times 2^N - 3$ 通りであることが分かるでしょう。最大公約数 G はユークリッドの互除法、2^x を $10^9 + 7$ で割った余りは繰り返し 2 乗法を使って、それぞれ対数時間で計算できます。

(解説は次のページに続きます)

★ ステップ #3. T が大きい場合でも解いてみよう

T が 2 以上の場合でも、先ほどのような T が 1 のときの問題に帰着させることができます。

$H \times W$ の各マスからマスに矢印を書くことを考えましょう。まず、各マスから「T マス右に移動した場所」にオレンジの矢印を書きます。続いて、各マスから「T マス下に移動した場所」に黄色の矢印を書きます。



マス目の場所や矢印をうまく組み替えると、なんと $T = 1$ のときの形が何個か、みたいにできます。例えば上の図は $(H, W, T) = (3, 6, 2)$ の場合ですが、3×3 の $T = 1$ の場合 2 つに“分解”できました。そうすると、$(H, W, T) = (3, 6, 2)$ の場合の答えは $21^2 = 441$ 通りであることがすぐに分かります。

ただ、特殊な値の場合だけで考えてはいけません。一般の (H, W, T) の場合はどのように分解されるのでしょうか？ここで、G_H を H と K の最大公約数、G_W を W と K の最大公約数として、黄色の矢印をたどると $H \div G_H$ 回、オレンジの矢印をたどると $W \div G_W$ 回で元の場所に戻ることが分かると思います。したがって、先ほどのような矢印の形は、$(H \div G_H) \times (W \div G_W)$ のマス目が $G_H \times G_W$ 個、に分解できることが分かりました。

したがって、$R_X = H \div G_H$、$R_Y = W \div G_W$ とすると、答えは $\left(2^{R_X} + 2^{R_Y} + 2^{\gcd(R_X, R_Y)} - 3\right)^{G_X \times G_Y}$ となることが分かりました！

これはユークリッドの互除法・繰り返し 2 乗法を使うことによって、対数時間で計算することができます。

＜実装例 – Python 3 による実装＞

- https://atcoder.jp/contests/ddcc2020-qual/submissions/8568515

A: DDCC Finals

The total consists of three parts:

- The money earned in Coding Contest: $300000, 200000, 100000$, or 0 yen
- The money earned in Robot Maneuver: $300000, 200000, 100000$, or 0 yen
- The bonus for winning both competitions: 400000 or 0 yen

Find each of these with if-else statements and sum them up to compute the answer.

Sample Code in Python3: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8559133`

B: Iron Bar Cutting

Before we start doing the operations, let us first decide where to cut. Assume that we will do the cut at the i-th notch from the left.

Let L_i and R_i be the original total length of the bar to the left and right of the i-th notch, respectively, that is, $L_i = A_1 + A_2 + \ldots + A_i$ and $R_i = A_{i+1} + A_{i+2} + \ldots + A_N$. Then, it is optimal to keep adding 1 to A_1 as long as $L_i < R_i$ and keep adding 1 to A_N as long as $L_i > R_i$. Thus, the minimum amount of money needed if we do the cut at the i-th notch is $|L_i - R_i|$ yen.

We can try all $i = 1, 2, \ldots, N-1$ and find the answer to the problem as the minimum value of $|L_i - R_i|$. However, since N can be up to 200000, using a double-nested loop to do this would run out of time. Instead, we can use the fact $L_{i+1} = L_i + A_{i+1}$ to sequentially find all of $L_1, L_2, \ldots, L_N$ in this order in linear time. Then, R_i can also be found as $(A_1 + A_2 + \ldots + A_N) - L_i$ (we should compute $A_1 + A_2 + \ldots + A_N$ just once and use the result repeatedly).

Sample Code in C++: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8320272`

C: Strawberry Cakes

Solution 1: Recursion

The problem statement says: "We can show that this is always possible, regardless of the number and positions of the strawberries." This is actually a big hint to a recursive solution. We can process our cake recursively, as follows:

- If our cake contains exactly one strawberry: Submit this cake without making any cut.
- If our cake contains two or more strawberries: Choose any two of those strawberries. If they belong to different rows, make any horizontal cut between them. Otherwise, they should belong to different columns, so make any vertical cut between them. Now, we have two smaller rectangular cakes, each with one or more strawberries. Recursively process each of these cakes.

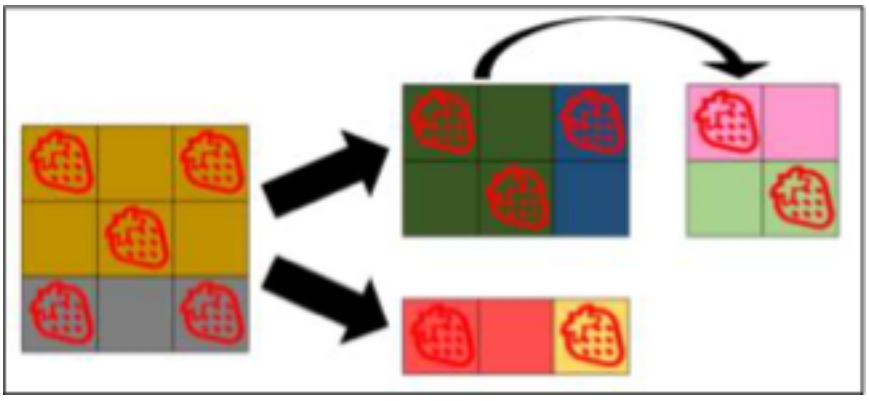


Fig 1 Illustration of Solution 1

The time complexity of this solution is $O(HW(H+W))$ (may vary depending on implementation), which runs in time if properly implemented.

Solution 2: Horizontal Cuts then Vertical Cuts

Let us first make as many horizontal cuts as possible without producing a piece with no strawberry. Then, in each piece, all the strawberries should belong to the same row, and we can just make some vertical cuts to separate them.

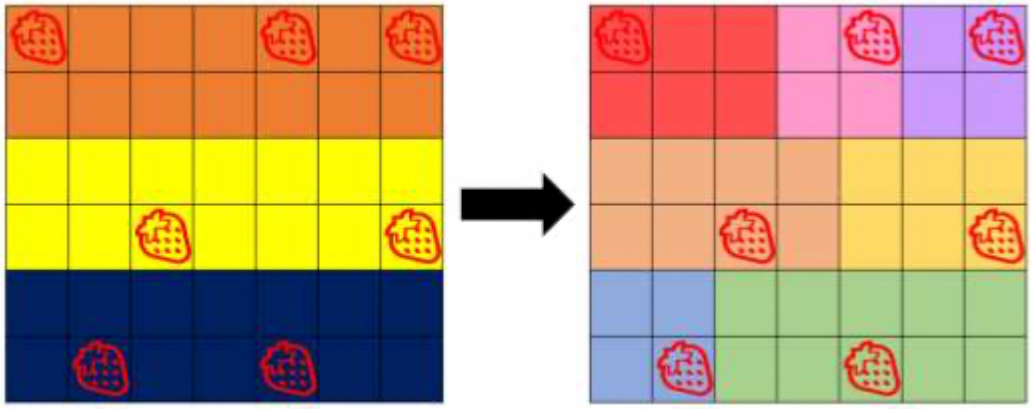


Fig 2　Illustration of Solution 2

Sample Code in C++: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8320839`

Exercise: In Solution 1, prove the time complexity $\mathrm{O}(HW(H+W))$.

D: Digit Sum Replace

The conclusion is: for a fixed value of N, the number of rounds is always the same.

Let $d(N)$ be the number of the digits in N, and $s(N)$ be the sum of the digits in N. (For example, $d(2019) = 4$ and $s(2019) = 12$.) If the organizer chooses two digits whose sum is 9 or less, $d(N)$ decreases by 1 and $s(N)$ does not change. On the other hand, if they choose two digits whose sum is 10 or greater, $d(N)$ does not change and $s(N)$ decreases by 9. Thus, before N becomes 9 or less, that is, before $d(N)$ becomes 1 and $s(N)$ becomes 9 or less, we always have $(d(N) - 1) + \lceil (s(N) - 9)/9 \rceil$ rounds.

Sample Code in Python3: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8562444`

E: Majority of Balls

In this editorial, the balls are numbered 0 to $2N-1$.

Let $f(i)$ be a function that returns `R` if there are more red balls than blue balls among the N balls $i, i+1, \ldots, i+N-1$, and `B` otherwise.

We use one question to ask $f(0)$ to the judge. Without loss of generality, we assume that the response is `R`. We can see that $f(N) = $ `B`, because we have N red balls and N blue balls in total. We can also see that there exists an index i $(0 \leq i < N)$ such that $f(i) = $ `R` and $f(i+1) = $ `B`. Let us do binary search to find this i in $\lceil \log N \rceil$ questions. Now, Ball i should be red, Ball $i+N$ should be blue and there should be $(N-1)/2$ red balls and $(N-1)/2$ blue balls among the $N-1$ balls $i+1, i+2, \ldots, i+N-1$. Also, there should be $(N-1)/2$ red balls and $(N-1)/2$ blue balls among the remaining $N-1$ balls $0, 1, \ldots, i-1, i+N+1, i+N+2, \ldots, 2N-1$.

We can now identify the color of each of the $2N-2$ balls other than Ball i and $i+N$ in one question, as follows:

- Ball $i+1, i+2, \ldots, i+N-1$: For each of these balls j, ask a question with the N balls $j, 0, 1, \ldots, i-1, i+N+1, i+N+2, \ldots, 2N-1$.
- Ball $0, 1, \ldots, i-1, i+N+1, i+N+2, \ldots, 2N-1$: For each of these balls j, ask a question with the N balls $j, i+1, i+2, \ldots, i+N-1$.

We have now identified the colors of all balls in $1 + \lceil \log N \rceil + (2N-2) \leq 204$ questions.

Sample Code in C++: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8323705`

F: DISCOSMOS

In this editorial, the squares are labeled from $(0, 0)$ to $(H - 1, W - 1)$. Also, we assume that the grid is cyclic by default. For example, $(0, j)$ lies just below $(H - 1, j)$, and $(i, W - 1)$ can be referred to as $(i, -1)$.

Let us first solve the problem for the case $T = 1$. From a square (x, y), if we continue moving diagonally by $(-1, +1)$, we will eventually come back to (x, y). Let us define the *zone* of (x, y) as the set of squares visited along the way. We can see that the grid is divided into G zones of $(0, 0), (0, 1), \ldots, (0, G - 1)$, where $G = \gcd(H, W)$, as shown below:

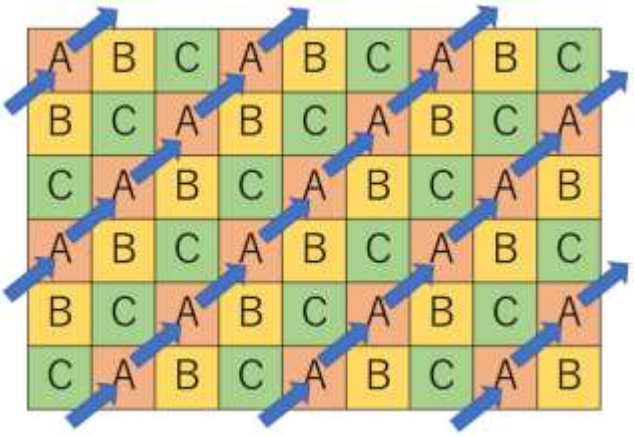


Fig 3 A grid with $(H, W) = (6, 9)$ divided into $G = 3$ zones

Here, it is never the case that a Type-R robot and a Type-D robot are both placed in the same zone (otherwise, they will eventually meet each other, which must be avoided when $T = 1$).

We will now show that, if the grid contains both a Type-H robot and a Type-R robot, the grid should not contain a Type-D robot. If there is a zone completely filled with Type-H robots, there is a Type-H robot in every column, so no Type-D robot should be placed. Otherwise, there should be a zone with both Type-R robots and Type-H robots. In this zone, there always exists a square (i, j) such that a Type-R robot is placed at (i, j) and a Type-H robot is placed at $(i - 1, j + 1)$. From this, we can show that there is no Type-D robot in the whole grid, as follows. See the following figure:

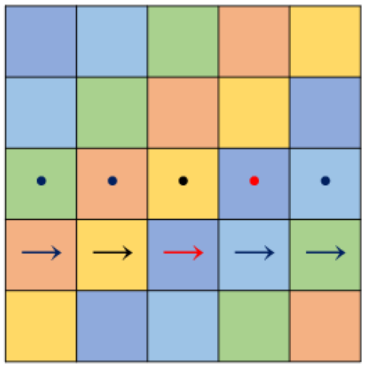


Fig 4 A grid with $(H, W) = (5, 5)$

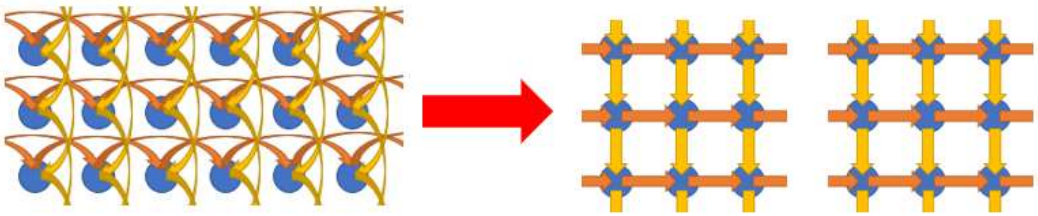
Let us place a Type-R robot in $(3, 1)$ and a Type-H robot in $(2, 2)$. Then, $(3, 2)$ and $(2, 3)$ must be occupied by a Type-R robot and a Type-H robot, respectively (in all of the remaining eight ways to place robots in $(3, 2)$ and $(2, 3)$, there will be eventually two robots meeting each other). Similarly, $(3, 3)$ and $(2, 4)$ must also be occupied by a Type-R robot and a Type-H robot, respectively, and so on. Eventually, we see that there must be a Type-R robot in every zone, from which it follows that there is no Type-D robot in the whole grid. (A similar argument holds for general H and W.)

Therefore, if the grid contains both a Type-H robot and a Type-R robot, the grid should not contain a Type-D robot. Similarly, if the grid contains both a Type-H robot and a Type-D robot, the grid should not contain a Type-R robot. Now, let us count the valid placements:

- The case with no Type-H robot: As mentioned above, a Type-R robot and a Type-D should not coexist in the same zone, so each zone should be either completely filled with Type-R robots or completely filled with Type-D robots. On the other hand, all of those 2^G placements are valid.
- The case with one or more Type-H robots: We have one placement with Type-H robots only, $2^H - 2$ placements with both Type-H and Type-R robots (for each row, we choose to completely fill it with Type-H robots or completely fill it with Type-R robots), and $2^W - 2$ placements with both Type-H and Type-D robots.

Therefore, we have $2^H + 2^W + 2^G - 3$ valid placements in total when $T = 1$.

Finally, the case $T \leq 2$ can be reduced to the case $T = 1$. For example, the case $(H, W, T) = (3, 6, 2)$ can be decomposed into two separate instances with $(H, W, T) = (3, 3, 1)$, as shown below:



(From each point (i, j), yellow and orange arrows point to $(i + T, j)$ and $(i, j + T)$.)

Fig 5 Decomposing the case $(H, W, T) = (3, 6, 2)$

In general, let $G_H = \gcd(H, T), G_W = \gcd(W, T)$. If we start at some square in the $H \times W$ grid and continue moving down by T (corresponding to the yellow arrows above), we will get back to where we started after H/G_H moves. Also, if we continue moving right by T (corresponding to the orange arrows above), we will get back to where we started after W/G_W moves. Thus, the case with general (H, W, T) can be decomposed into $G_H \times G_W$ separate instances with $(H, W, T) = (H/G_H, W/G_W, 1)$. Therefore, the answer to the problem is $(2^{R_X} + 2^{R_Y} + 2^{\gcd(R_X, R_Y)} - 3)^{G_X \times G_Y}$, where $R_X = H/G_H, R_Y = W/G_W$.

Sample Code in Python3: `https://atcoder.jp/contests/ddcc2020-qual/submissions/8568515`