培正 喇沙 編程挑戰賽
LA SALLE – PUI CHING PROGRAMMING CHALLENGE

2ND · 2017

AUGUST 15, 2017 (TUESDAY)

LA SALLE COLLEGE

SOLUTIONS

ID	NAME	AUTHOR	DIFFICULTY
A	Ambiguous Dates	Tung Kam Chuen	★★★☆☆
B	Bacteria Experiment	Wong Tsz Chun	★★★★☆
C	Cheering	Tung Kam Chuen	★☆☆☆☆
D	Distribution of Days	Wong Tsz Chun	★★★☆☆
E	Expected Score	Poon Lik Hang	★★★☆☆
F	Frustrating Game	Chow Kwan Ting Jeremy	★★★★☆
G	Gravitational Tetris	Wong Man Hang	★★★★★
H	Hit!	Wong Yik Chun	★★☆☆☆
I	Inverted Signs	Wong Yik Chun	★★☆☆☆
J	Juicy Candies	Tung Kam Chuen	★★★★★
K	Knights	Wong Tsz Chun	★☆☆☆☆
L	Let Me Count The Ways	Tung Kam Chuen	★★★★★

The authors expect that the problems were to be solved by

the following percentage of teams

From 1 to 5 stars: >75%, 50-75%, 25-50%, 5-25%, < 5%

A - Ambiguous Dates

Solution

This problem may look intimidating at first, but with the following key observation, it becomes quite easy:

> **One optimal way of rearrangement is to sort $D[]$ in ascending order.**

The rough idea is that for larger $D[i]$, putting it nearer to the end of the year can help reduce the number of ambiguous dates (as, perhaps indirectly, hinted in sample 2).

To calculate the answer, it suffices to count the number of ambiguous dates (D_0, M_0, Y_0) with $D_0 > M_0$, and the answer will be equal to two times the count.

Suppose this fact is true (we will outline the proof later). Notice that we only need to count the number of indices i, j, such that $1 \leq i < j \leq M$, $D[i] \geq j$, and $D[j] \geq i$. As $D[]$ is increasing, $D[i] \geq j$ implies $D[j] \geq i$ (since $D[j] \geq D[i] \geq j \geq i$).

For a given j, let $\alpha(j)$ be the largest index such that $D[\alpha(j)] < j$. (Set $\alpha(j) = 0$ if $D[1] \geq j$.) Then, we add $j - \alpha(j) - 1$ to the answer if (and only if) $\alpha(j) < j$.

Because of the monotonicity of $D[]$, finding $\alpha(j)$ can be done by either binary search or the method of two pointers. Thus, the time complexity is $O(N \, log \, N)$, assuming sorting is done in $O(N \, log \, N)$ time.

(Outline of) Proof of the observation

To prove the key observation which is central to our solution, it suffices to show that, if $D[i] > D[i+1]$ for some i, then swapping $D[i]$ and $D[i+1]$ does not make the answer worse.

Showing that this fact is true amounts to doing simple calculations in different cases, for example:

- $D[i] \leq i$
- $D[i+1] > i$
- $D[i] > i$ and $D[i+1] \leq i$

Challenge

Do you know how to solve the problem if rearrangement of $D[]$ is **not** allowed?

B - Bacteria Expermient

Solution

As there are many edges to be formed, we cannot just simulate the evolution of the graph, but try to observe some patterns.

We may start from simplier form of initial graph like chain. In this case, we can observe that nodes with distance of 2 can be connected by an edge at t = 1, distance of 3 or 4 can be connected at t = 2, and so on. As at time T, nodes with distance at most 2^T can be connected. Therefore, for the two ends of a chain with distance D, the total time required should be $\lceil log_2 D \rceil$

With this observation, we can try to find the longest chain in the initial graph(a.k.a. diameter of the graph) and use the formula above. It can be easily proved that the edge connecting two ends of diameter must be one of latest edge formed. Otherwise, there must be a longer chain contained in the graph.

So, we can just find the length of diameter(D), and the answer should be $\lceil log_2 D \rceil$

Implementation

There are two simple way to find the diameter of a graph in $O(|V|)$:

The first way is implementing a tree DP. We can perform a DFS on the graph from an arbitrary node. For each node, we have to calculate the two longest distance to the leaves, and use their sum to update the diameter. The longest distance can be maintained by using the information from the children and add one to them.

The second way is two perform DFS twice. We can start at an arbitrary node(let's call it node u), and find one of the farthest node from node u, let's call it node v. And then, we should start a DFS again to find the farthest node from node v, node w. The diameter of the tree is the distance between node v and node w. Proof of correctness for this algorithm can be easily searched on Google

You may replace the DFSs with BFSs as well.

Comments

The author intends to prepare an easier graph problem than those requiring advanced techniques(e.g. Dijkstra, SPFA, LCA) so teams which can observe the pattern are able to solve the task.

Disappointingly, some teams know how to find the diameter of a tree, but getting wrong formulas: $\lceil 0.5D \rceil$ or $D-1$. It seems many teams tried to observe some patterns, but did not verify their guess with larger cases or think of a reasonable proof.

C - Cheering

This is intended to be the easiest problem of the whole contest. Nothing tricky, just input the string, calculate A and B as required, and output accordingly. The psuedo-code may look like:

```
input S
set A = 0, B = 0
for i from 0 to length(S) - 1
   if S[i...i+2] = "LSC"
      A ← A + 1
   if S[i...i+3] = "PCMS"
      B ← B + 1
end
if A > B
   print "LSC"
if A < B
   print "PCMS"
if A = B
   print "Tie"
```

D - Distribution of Days

Solution

The given constriants on years (S and E) are so large and with many queries (at most 5000 queries) as well, so we cannot just simply answer each query by iterating through every year between S and E as it costs so much time. Therefore, we have to figure out a faster way of answering queries.

Key observation is that as the pattern of leap years forms a cycle with length 400, and the pattern of days of the week is a cycle with length 7. The distribution of days must form a cycle every $7 \times 400 = 2800$ years. In other words, distribution of days in the year 2000 must be same as that in the year 4800.

With this observation, we can pre-compute the distribution of days in every 2800 years. Then for each query, we can first find number of cycles within the years S and E, which is $\frac{E-S+1}{2800}$, multiply it with the pre-computed value. The only thing left is the remaining years after last cycle. We can obtain these values easily by just iterating all the remaining years, with at most 2799 years in worst case. Thefore the time complexity of this method is roughly $O(Q \times L + 366L)$ where L is the cycle length (2800 in this case). This solution is fast enough to pass all tests during the contest.

Further Optimization

Actually, the number of days within 400 consecutive years is $97 \times 366 + 303 \times 365 = 146097$. As 146097 is a multiple of seven(7×20871), the length of cycle L can be reduced to 400 instead of 2800.

Actually if we only consider years kL to $(K+1)L - 1$, where k is arbitrary integers, as cycles, we will have to handle the years before the first cycle within range, and also the years after the last cycle within range. These two parts each has at most $L - 1$ possibilties only. Therefore, we can pre-compute the values of these two parts as well. Then the time complexity can be reduced to $O(Q + 366L)$

Comments

As what the author heard from the contestants, some teams are able to notice that the solution should consider cycles in order to reduce time complexity. Unfortunately, only one team can manage to code the solution without bugs. Obviously, this task is mainly testing contestants' coding skills. For instance, contestants must carefully handle cases with 29 February.

To be honest, the author expected more teams were able to solve this task.

E – Expected Score

Firstly, we can divide the $2N$ chests into two types. 1. "Opened chest" (i.e. number in the chest is known) 2. "Unopened chest" (i.e. number in the chest is not known).
Let's denote the 2 chests marked as the same number as a "Chest group".
Note that the player has three strategies in the last round. 1. Open two "Opened chests" 2. Open one "Opened chest" and one "unopened chest" 3. Open two "Unopened chests"
Then, we can calculate the expected score of the whole game for each strategy separately.

Sample Case

For better understanding, we use an example to demonstrate the calculation process.

Consider the following game state:
3 ? 4 4 ? ? 5 ? 2 2 ? ? ? ?
We have 14 chests totally, "unopened chest" is denoted as an $?$.
The score we get in the previous round is 2.

We can first classify each chest group as "Both opened", "One opened" and "Both unopened".

Chest groups with score = 2 *and* 4 belong to "Both opened"
Chest groups with score = 3 *and* 5 belong to "One opened"
Chest groups with score = 1, 6 *and* 7 belong to "Both unopened"

We also denotes M as the number of unopened chest. In this case, $M = 8$

Case 1:

We have 100% chance to get score x in the last round if chest group with score x belongs to “Both opened”.
We have 0% chance to get score x if chest group with score x does not belong to “Both opened”.
It is obvious that we will choose the largest x belongs to “Both opened”. In the sample above, it is 4.
Then our final expected score = $max(4, 2) = 4$

Case 2:

We have $\frac{1}{\mathrm{M}}$ chance to get score x in the last round only if and only if chest group with score x belongs to “One opened”.
We have 0% chance to get score x if chest group with score x does not belong to “One opened”.
It is also obviously that we will choose the largest x belongs to “One opened”. In the sample, it is 5 and $M = 8$.
Then our final expected score $= \frac{1}{8} \times max(5, 2) + \frac{7}{8} \times 2 = 2.125$

Case 3:

We have $\frac{2}{\mathrm{M}} \times \frac{1}{M-1}$ chance to get score x if and only if chest group with score x belongs to “Both unopened”.
We have 0% chance to get score x if chest group with score x does not belong to “Both unopened”.
Note that if x is smaller than the score we got in the previous round, getting score x in the last round will not increase our expected final score. So, we only consider the “Both unopened” chest groups which score greater than x. They are 6 and 7 in this case.
Therefore, the final expected score $= \frac{2}{8} \times \frac{1}{7} \times 6 + \frac{2}{8} \times \frac{1}{7} \times 7 + other \times 2$ Where $other = 1 - \left(\frac{2}{8} \times \frac{1}{7} + \frac{2}{8} \times \frac{1}{7} \right)$, $ans = 2.321$

The answer of the whole problem is maximum of the answer in the three cases above.

General Solution

Case 1:

```
for (all x which is both opened)
    Answer = max(Answer, x);
Answer = max(Answer, previous_round_score)
```

Case 2:

```
for (all x which is one opened)
    Answer = max(Answer, x);
Answer = max(Answer, previous_round_score);
Answer = 1 / m * Answer + (1 – 1 / m) * previous_round_score;
```

Case 3:

```
for (all x which is both unopened)
    if (x > previous_round_score)
        Answer += (2 / m) * (1 / (m - 1)) * x;
        other += (2 / m) * (1 / (m - 1));
Answer = Answer + (1 – other) * previous_round_score
```

Comment

All of the problem setters agree that this is not a difficult task and rank it as medium difficulty comparing with other problems in this contest.
Unexpectedly only 2 + 1 teams solve this task.
Although the problem statement are complicated, it is suggested to read all problems in an ACM contest.
When encounters case handling problem, it is a good strategy to list out all cases on a paper before coding.

F – Frustrating Game

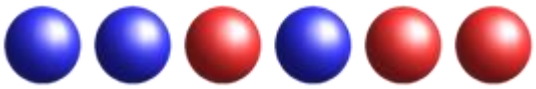
Solution:

Firstly, we should determine whether it is possible to achieve the goal. Let the number of red balls be ***R*** and the number of blue balls be ***B***. If one of them is satisfied, then it is impossible to turn all balls into white.

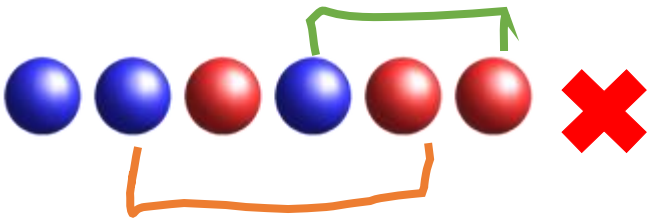
1. ***R*** is not divisible by ***X***
2. ***B*** is not divisible by ***Y***
3. $\frac{R}{X}$ is not equal to $\frac{B}{Y}$

Because if one of them is satisfied, you will not be able to select ***X*** red balls and ***Y*** balls at some time, while not all balls are in white. Other than those cases, it is always possible to achieve the goal. The proof will be discussed later.

Let's think about an easier problem, where ***X*** and ***Y*** is always equal to 1. Because **(3)** not satisfied, so ***R*** = ***B***. Let look at the Sample 1.



Because of the restrictions of the spell, there cannot be two spells ***{a, b}, {c, d}***, where $a < c$, such that $b > c$ *and* $b < d$ as not matter how we arrange the spells, it is always invalid.



So, we only can have spells {a, b}, {c, d} such that $b < c$ or $b > d$.

Now it looks like the problem which is about assigning a valid parentheses sequence, which one end is red ball and one end is blue ball.

We can assign such valid parentheses sequence to Sample 1.

To get the correct order of spells, we just need to output the index of parentheses from outside to inside, which is ***{1, 6}, {2, 3}, {4, 5}***. We can simulate this easily by stack. When there are 1 red ball and 1 blue ball on the top of the stack, we pop them out to use a spell. After processing all balls, we reverse the order of the spells. Then we can get a valid spell sequence.

Back to the original problem, we can extend the idea of ***X*** = 1 and ***Y*** = 1. The problem is again similar to assigning a valid parentheses sequence, but this time the parentheses have ***X + Y*** parts instead of 2 parts and there must be ***X*** red parts and ***Y*** blue parts. We maintain a stack, where each element of the stack records how many red and blue balls are in the previous ***X + Y*** balls of it in the stack. When there are ***X*** red balls and ***Y*** blue balls on the top of the stack, we pop them out to use a spell. Finally, we reverse the order of the spells.

Notice that we may need window sliding technique to maintain the information of the element of the stack, so we will not only access the top of the stack. Therefore hand-written stack is preferred as STL stack can only access the top element of the stack.

So why when ***(1), (2), (3)*** are not satisfied, it is always possible? At this point, ***R = kX*** and ***B = kY,*** where ***k*** is a positive integer. In a sequence that consists of ***kX*** red balls and ***kY*** blue balls, we can always find an interval that contain ***X*** red balls and ***Y*** blue balls. After popping them out, our sequence consists of ***(k – 1)X*** and ***(k – 1)Y***, we can find an interval that contain ***X*** red balls and ***Y*** blue balls again. The process is end when ***k = 0***. So, by induction, it is always possible to turn all balls into white.

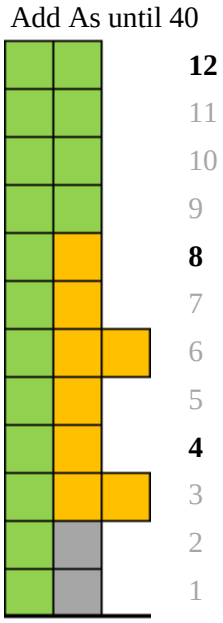
G – Gravitational Tetris

When the problem states that the input is guaranteed to be a valid game state, what does it mean? Every piece consists of 4 blocks, and there are 10 columns. Therefore, we can conclude that the number of remaining blocks in the game must be even.

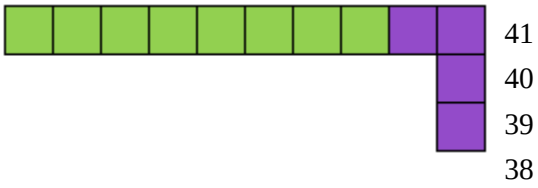
Specifically, if even number of lines have been eliminated, the number of remaining blocks will be a multiple of 4. If odd number of lines have been eliminated, the remainder of number of remaining blocks divided by 4 will be 2. **To win the game, an even number of lines are to be eliminated.**

Once we understand the condition of the valid game states, we need to come up with a construction that can cover all cases, is easy to implement, and satisfy the constraints (time, memory and output limit). Below describes one such solution.

Let's say we aim for eliminating 40* lines. This is equivalent to making all 10 elements in the array to become 40. We do so by going through columns 1 to 9 one by one. Use a combination of As and (at most 3) Gs so that the column becomes 40. i.e. solve for $a[i] + 4x + 3y = 40$. Note that for every G you add, you need to add one block to the next column $a[i + 1]$.



For column 10, there are two possibilities: $a[10]$ mod 4 = 0 or 2. For $a[10]$ mod 4 = 0, it's easy because you can just add As to make $a[10] = 40$. For $a[10]$ mod 4 = 2, you can make $a[10] = 38$, then use two Bs and an E to finish the game.

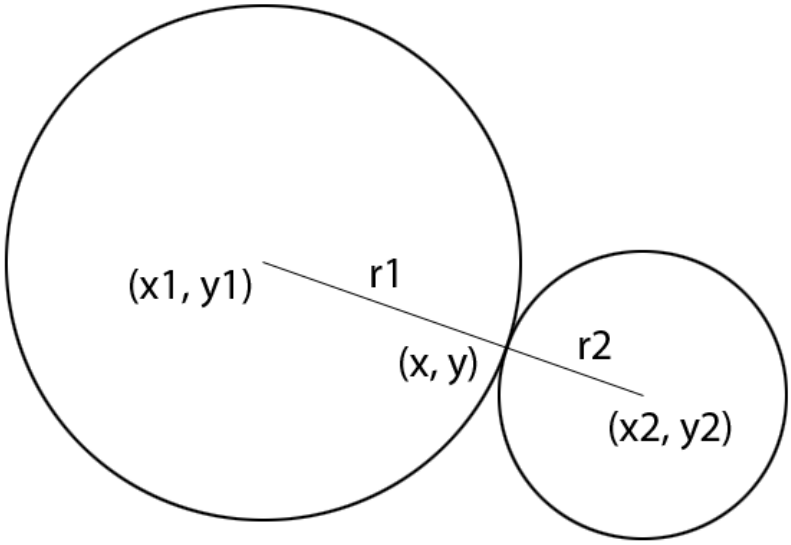


Some contestants may approach this problem as a graph search problem. However, the large number of states requires heavy optimization and is unlikely to fit inside the time limit.

* 40 is *not* the minimum, but we usually pick a number that would certainly work by intuition.

H – Hit!

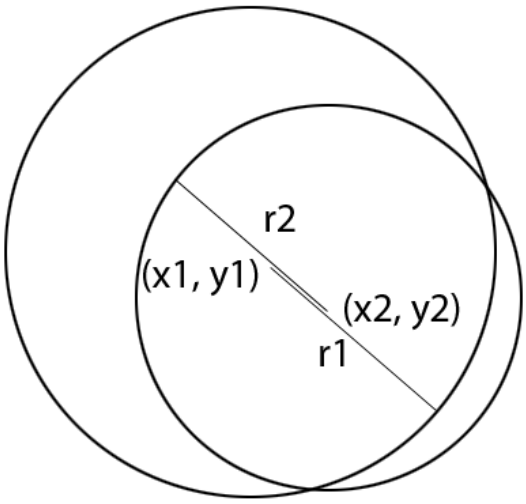
Let’s consider the hardest case: the two circles are just touching.



Can you find (x, y)?

Using the ratio formula, we can obtain $(x, y) = (\frac{x_1 r_2 + x_2 r_1}{r_1 + r_2}, \frac{y_1 r_2 + y_2 r_1}{r_1 + r_2})$

Can this formula solve all the cases, such as the one below? This is left as an exercise for you. However, during contest time, it is not a bad strategy to simply give it a try.



Time complexity would be O(1).

I – Inverted Signs

Assume we flip a subarray from `H[l]` to `H[r]`, one can notice that the changes between `l` and `r`, do not contribute any difference to the new chaos index. For example, when we change 7 and -4 to -7 and 4, their absolute difference is still 11.

Therefore, only two pairs (`H[l-1]`, `H[l]`) and (`H[r]`, `H[r+1]`) would contribute to the update. Denote old chaos index as C, we know that the new chaos index would be

C – abs(`H[l-1]`– `H[l]`)
  + abs(`H[l-1]`– (–`H[l]`))
  – abs(`H[r]`– `H[r+1]`))
  + abs(–(`H[r]`) – `H[r+1]`))

So we can do a linear scan in the array, with `i`=1,2,…,N, maintaining `l` that
– abs(`H[l-1]`– `H[l]`) + abs(`H[l-1]`– (–`H[l]`)) is least. The answer would be the minimum of
C – abs(`H[l-1]`– `H[l]`)
  + abs(`H[l-1]`– (–`H[l]`))
  – abs(`H[i]`– `H[i+1]`))
  + abs(–(`H[i]`) – `H[i+1]`))
`i`=1,2,…,N and `l<=i`

Time complexity would be O(N).

J - Juicy Candies

This is a standard, yet quite difficult, problem on dynamic programming (DP).

The main idea is that, if we know how to calculate (quickly) the number of strings *with a given prefix*, we would be done. So, the following discussion is divided into two parts: how to calculate the number of strings with a given prefix, and how to use the calculation to solve the problem.

Number of strings with a given prefix

Note that we are only concerned with two things:

- The number of occurrences of `B`, `R`, and `S` in the prefix
- The last character (if any) of the prefix

So we do a simple dynamic programming **on suffixes**. For a nonempty suffix, we need to know:

- The number of occurrences of `B`, `R`, and `S` in the suffix
- The fisrt character of the suffix

So we define, for a given state (b, r, s, f), $dp[b][r][s][f]$ to be the number of suffixes with b `B`s, r `R`s, and s `S`s, with first character indicated by f (for example, $f = 0$ for `B`, $f = 1$ for `R`, $f = 2$ for `S`).

Our base case would be $dp[1][0][0][0] = dp[0][1][0][1] = dp[0][0][1][2] = 1$. For transition from (b, r, s, f) to (b', r', s', f'), we append a character to the front of the original suffix.

If we append `B`, make sure $f \neq 0$ and $b < B$. The new state is $(b', r', s', f') = (b+1, r, s, 0)$, so we add $dp[b][r][s][f]$ to $dp[b+1][f][s][0]$. Similar rules apply for appending `R` and `S`.

The number of states is $O(BRS)$ and transition is done in $O(1)$, so the time complexity for this step is $O(BRS)$.

Beware that the numbers can be very large, so you should upper-bound them by a large enough number (for example, 10^{18}, or just K), to avoid overflow.

Finding the answer

If $dp[B][R][S][0] + dp[B][R][S][1] + dp[B][R][S][2] < K$, output `None`. From now on, assume that the sought string does exist.

We try to find out the characters from left to right. For a given prefix with b `B`s, r `R`s, and s `S`s, and the last character being $c \in \{$`B`,`R`,`S`$\}$, we try to append a character to it. Which character to append? We follow the procedure below:

1. If $b < B$ and $c \neq$ `B`, append `B` to the prefix if $dp[B-b][R-r][S-s][0] \geq K$ and go to step 4. Otherwise, subtract $dp[B-b][R-r][S-s][0]$ from K.
2. If $r < R$ and $c \neq$ `R`, append `R` to the prefix if $dp[B-b][R-r][S-s][1] \geq K$ and go to step 4. Otherwise, subtract $dp[B-b][R-r][S-s][1]$ from K.
3. Append `S` to the prefix.
4. Update the values of b, r, s, and c.

Repeat the steps for $(B+R+S)$ times, and we will have the desired string. For this part, the time complexity is $O(B+R+S)$.

Combining, the time complexity for the problem is $O(BRS)$.

K - Knights

Solution

The problem has long explanation on the rules of the game. However, it should be not hard to notice that only the four corners are important.

The solution to this problem is to just check the number of unoccupied corners. We should place knights only on these places.

As the corners cannot be occupied by ways other than directly sending knights there, corners are neccesary for you to send knights directly. On the other hand, if four corners are occupied, every other cell will be occupied as well. Therefore, it is obvious that the solution both optimal and correct.

Implementation

The idea is easy, so the only thing left is to implement it. Actually there are many ways: after reading each pair of coordinates, use four `if`-statements to check if it is any of the corners. Or, marking those places as `TRUE` in a 2-D boolean array.

The only thing requries our attention should be when any of N and M equal one, the number of corners is reduced to 2 or 1. Some solutions without special handling will double-count these corners.

Comments

It is happy to see that nearly all teams were able to notice the key observation and implemented their idea carefully, successfully obtained a red balloon.

However, many teams attempted many times before getting an `Accepted`, this is mainly because teams did not check their programs carefully with some boundary cases mentioned above. The sample case with $N = M = 1$ should remind you taking care of the special cases.

L - Let Me Count The Ways

There are two parts to this problem, and **both** parts are challenging. That no (official) teams were able to solve this problem during the contest was as expected.

For problems with such a simple combinatorial setting, one would expect that the answer would be given by an equally simple formula. Finding such a formula is the first part.

It turns out that, given such a formula, computing the answer efficiently would still be tricky. Computing the answer efficiently is the second part.

Finding the formula

How to count the ways? Here are some ideas which may help.

Idea 1: Relaxing the conditions

For succinctness, we introduce the following definitions:

- Call an arrangement *row-regular*, if numbers on both of the rows are sorted in descending order.
- Call an arrangement *column-regular*, if numbers on all the columns are sorted in descending order.

Then, the answer is given by the number of arrangements that are both row-regular and column-regular.

The number of row-regular arrangements is $\binom{N+M}{N}$, because each arrangement corresponds to a way to choose N numbers from $1, 2, \ldots, N+M-1, N+M$ for the first row.

Therefore, the answer is $\binom{N+M}{N}$, minus the number of row-regular arrangements that are **not** column-regular.

Idea 2: Special case $N = M$

When $N = M$, the answer is given by the well-known Catalan numbers (if you don't know what it is, just google it). Its formula is given by $\frac{1}{N+1} \times \binom{2N}{N}$.

We can rewrite it as $\binom{N+N}{N} - \binom{N+N}{N+1}$. Maybe the answer is similar-looking in the general case...?

Idea 3a: Testing with the sample cases

Let answer be $\binom{N+M}{N} - R(N, M)$. Let us try the ideas on the first four sample cases:

- Sample 1: $N = 5, M = 3$.
  Answer $= 28 = \binom{5+3}{5} - R(5,3) = 56 - R(5,3)$.
  So $R(5,3) = 28$.
- Sample 2: $N = 2, M = 1$.
  Answer $= 2 = \binom{2+1}{2} - R(2,1) = 3 - R(2,1)$.
  So $R(2,1) = 1$.
- Sample 3: $N = 4, M = 4$.
  Answer $= 14 = \binom{4+4}{4} - R(4,4) = 70 - R(4,4)$.
  So $R(4,4) = 56$.
- Sample 4: $N = 10, M = 6$.
  Answer $= 3640 = \binom{10+6}{10} - R(10,6) = 8008 - R(10,6)$.
  So $R(10,6) = 4368$.

Note the interesting relation between $\binom{N+M}{N}$ and $R(N,M)$. In fact, one may observe that $R(N,M) = \binom{N+M}{N} \times \frac{M}{N+1} = \binom{N+M}{N+1}$. Note that this fits with our knowledge of the Catalan numbers (i.e. when $N = M$).

This is of course correct, and so we finally have the answer: $\binom{N+M}{N} - \binom{N+M}{N+1}$. For a proof of the formula, see the section below.

Idea 3b: Direct combinatorial proof

It will be more convenient to consider another combinatorial object: paths on the Cartesian coordinate plane.

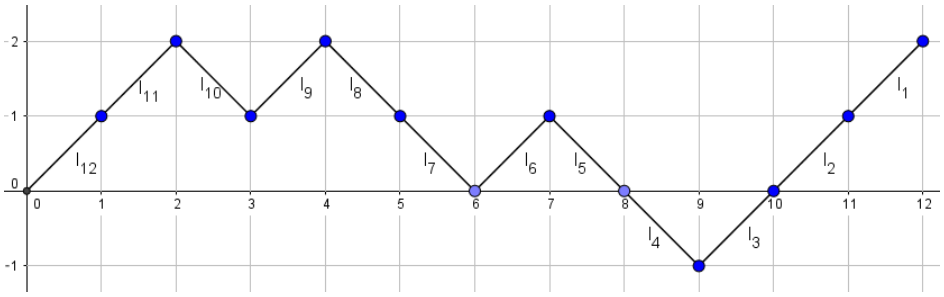
For each row-regular arrangement, we assign to it a path from $(0,0)$ to $(N+M, N-M)$. Our starting point is $(0,0)$ and we move in the following manner:

```
for i from (N + M) downto 1
   if i is found in the first row
      move from (X, Y) to (X + 1, Y + 1)
   else
      move from (X, Y) to (X + 1, Y - 1)
```

For example, the table below

12	11	9	6	3	2	1
10	8	7	5	4		

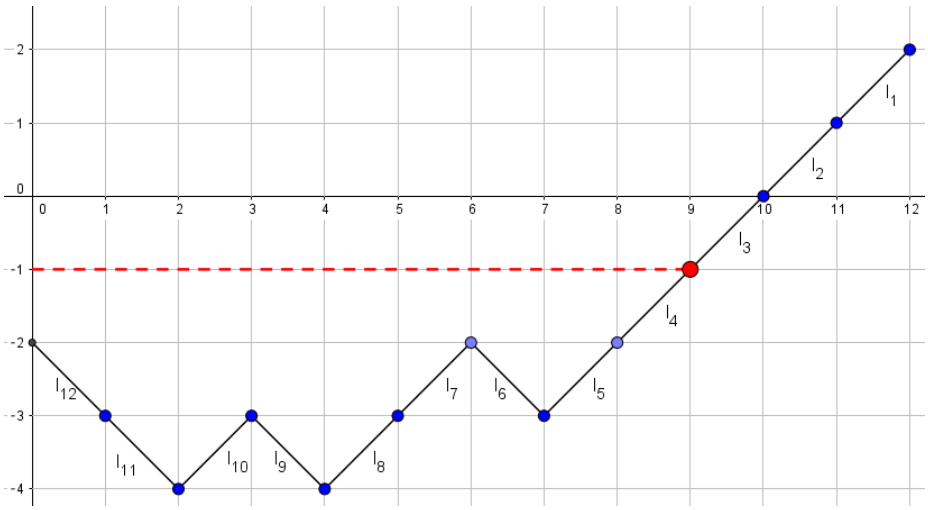
would correspond to this path:



Our answer would be the number of such paths from $(0, 0)$ to $(N+M, N-M)$, which does not cross the x-axis ('good' paths).

The total number of paths is $\binom{N+M}{N}$, as we know. We shall show that the number of paths which *crosses* the x-axis ('bad' paths) equals $\binom{N+M}{N+1}$.

For each 'bad' path, it contains a point of the form $(x, -1)$. Find the left-most such point, and reflect the portion of the path to the left of the point, along the line $y = -1$. For example, the path above would become:



In fact, this construction provides a *bijection* between:

- the number of 'bad' paths from $(0, 0)$ to $(N + M, N - M)$, and
- the number of paths from $(0, -2)$ to $(N + M, N - M)$.

Hence, the number of 'bad' paths equals $\binom{N+M}{N+1}$, and we are done.

Computing the answer efficiently

Finally, we need to compute $\binom{N+M}{N} - \binom{N+M}{N+1}$ modulo $P = 1000000007$ efficiently.

Let us first consider the following solution, which works for $N \leq 10^7$, but not for $N \leq 10^9$.

- Precompute `fact[k] := k!` (k factorial)
- Calculate $\binom{n}{r}$ using the formula $\binom{n}{r} = \frac{n!}{r!\,(n-r)!}$
- Get answer!

Now we address several difficulties and improve the solution above.

Difficulty 1. How to *divide* a number (modulo P)? In other words, how to find the inverse of a number (modulo P)? Well, since P is a prime, Fermat's little theorem tells us that $ab \equiv 1 \pmod{P}$ is equivalent to $a \equiv b^{P-2} \pmod{P}$. Finding $b^{P-2} \pmod{P}$ can be done in $O(\log P)$ time, using fast exponentiation.

Thus, to divide by $r!$, we only need to know `fact[r]`.

Difficulty 2. It seems that precomputing the factorial table is unavoidable. What must be done to fit the solution into the time limit? Well, we can precompute the factorial table on our machine, and store *a portion* of the values in our new program. For example, we can store 1000 numbers: the values of $1000000!, 2000000!, ..., 999000000!, 1000000000!$, all modulo P.

Then, it takes at most 1000000 iterations to compute any value of `fact[k]` (if `k` is too large, `fact[k]` is just zero), and the solution fits into the time limit.