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

4TH · 2019

MAY 13, 2019 (MONDAY)

LA SALLE COLLEGE

SOLUTIONS

ID		NAME	AUTHOR	DIFFICULTY
A		A Billionaire	Poon Lik Hang	★★☆☆☆
B		Blokus Duo	Wong Man Hang	★★★★★
C		camelCaseCounting	Wong Tsz Chun	★★★★★★
D		Drawing Circles	Wong Tsz Chun	★★★★☆
E		Eat More	Wong Yik Chun	★★★★★
F		Find the Base	Chow Kwan Ting Jeremy	★★★★★
G		Guessing Game	Wong Tsz Chun	★★☆☆☆
H		Hacking	Wong Tsz Chun	★☆☆☆☆
I		Ice-cream Sampler	Wong Man Hang	★★★☆☆
J		Just A $10 Note	Wong Tsz Chun	★★★☆☆
K		Kth number in Byteland	Chow Kwan Ting Jeremy	★☆☆☆☆
L		LRTB and TBRL	Wong Tsz Chun	★★★★☆

The authors expect that the problems were to be solved by the following percentage of teams

From 1 to 6 stars: >75%, 50-75%, 30-50%, 15-30%, 5-15%, <5%

A - A Billionaire

Observation

Let C be the sum of cost of all luxuries. Alice needs to accumulate at least C dollars in order to buy all the luxuries. Also let K, E be Alice's initial wealth and gains per day. x be the minimum number of day such that Alice accumulates at least C dollars. It is easy to see:

$$K + xE \geq C$$

$$answer \geq x \geq ceiling(\frac{C-K}{E})$$

As Alice can buy only one object per day. One strategy for Alice is to wait $x-1$ days first, then uses n days to buy all n luxuries. By this strategy, she spends $x+n-1$ days in total. However, this may not be optimal. Intuitively, Alice would like to buy as much luxuries as she can before day x such that she can spend lesser days to buy the remaining luxuries since day x.

In order to buy more luxuries as soon as possible (before day x), it is not hard to see that buying the luxuries in ascending order according to their prices is always optimal.

Solution

By the observation above, we can sort the luxuries' price (c_i) in ascending order first. Iterates over them and for each object, finds how many days are needed for Alice to wait in order to accumulate enough money to buy that object. This can be done by a simple equation. Remember that Alice needs at least one day to buy a new object as she can buy one per day only.

Comments

This is one of the easiest problem in this problem set. The authors expect most of the team should able to come up with the above greedy algorithm to solve this task. However, there exists a simpler solution, which is just to output

$$max(n, ceiling(\frac{C-K}{E}))$$

The reason and prove is left as an exercise.

Fun Facts

Check the song out at `https://youtu.be/8aRor905cCw` !

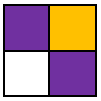
B – Blokus Duo

Observation: Moves for Purple can be considered independently from Orange's. This is because the input is guaranteed to be a valid game state. Once we have determined a player's moves and their relative order. We don't need to care about the move's order relative to the other player's, but we should make sure that we output the moves alternatively.

6 moves for P, 3 moves for O	4 moves for P, 5 moves for O
P[1] O[1] P[2] O[2] P[3] O[3] P[4] P[5] P[6]	P[1] O[1] P[2] O[2] P[3] O[3] P[4] O[4] O[5]

From now on, we consider only pieces from a single player. For Purple, the first piece must be placed on (5, 5) so we start from there. We can use our favourite graph searching algorithm (DFS or BFS) to find all the cells that the piece cover. That would be the first move.

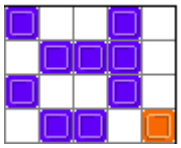
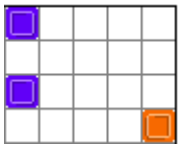
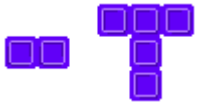
So how to determine the next move? The rule says that a new piece must touch at least one corner of an existing piece. (Again, we can ignore the rule about no touching edges because the board is guaranteed to be valid). Two cells (x, y) and $(x + 1, y + 1)$ are from two different pieces if they are of the same color but the cells $(x, y + 1)$ and $(x + 1, y)$ are of different colors.



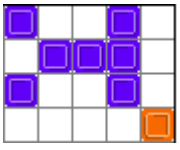
Once we find a cell of the new piece, find all the cell that it covers (using the same algorithm for the first move). Since the task accept any valid topological order, we can append the new piece to the end of the list of moves.

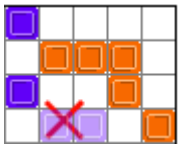
Be careful when maintaining the "visited" state so that you won't add the same piece to the list more than once.

Challenge: In the task, we allow the players to surrender early "by mistake", even when there exist valid moves. What if the rule is changed to require that both players must play a move as long as they are able to? Consider this "corner" case: (Top-right corner of the board)

Input	Current state	Purple's remaining pieces	Orange's remaining pieces
			

In this case, we must place the square first otherwise the other player will be able to make a move, which contradicts with the rules.





C - camelCaseCounting

Observation

All substrings that start with a lowercase letter is valid. Therefore, if the task does not require "unique" substrings, we can iterate through every lowercase letter and it will contribute $|S| - position + 1$ to the answer.

Solution

To tackle with unique substrings, we should not consider the repeated ones, for example: `ana` in `banana` appears twice, but we should count it only once.

There are many standard string algorithms that can do so, for example, Suffix Array and Suffix Tree can count the number of repeated substrings efficiently. Alternatively, Suffix Automaton supports directly counting number of unique substrings. To count only substrings that start with lowercase letters, only slight modifications to the standard algorithms are required.

Comments

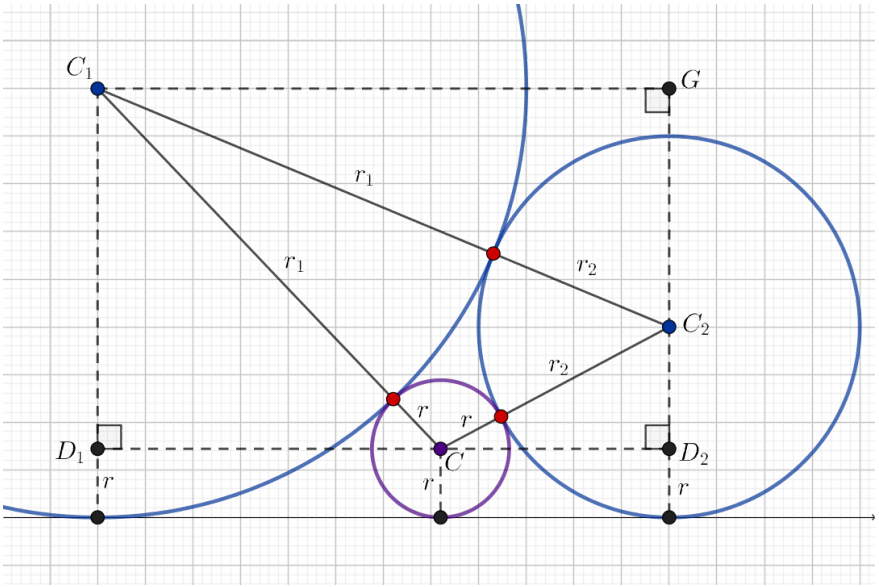
The string length in this problem is rather large (10^6), with only 1 second time limit, meaning that we need fast enough solution. The author was trying hard adjusting the constraints and time limit to let only $O(N)$ solutions (for example, Suffix Automaton or Suffix Tree) pass. (Un)fortunately, in the actual contest, both teams that passed this problem were using $O(N \log N)$ implementation of Suffix Array.

D - Drawing Circles

This problem requires both fundamental geometry and data structure knowledge.

Geometry

We should first find out: given two circles with radii r_1 and r_2, what's the radius r of the new circle.



With a little bit of Maths, and Pythagoras theorem, we can find the formula for calculating r.

Consider triangle GC_1C_2, we have

$$\begin{aligned}(r_1 - r_2)^2 + (CD_1 + CD_2)^2 &= (r_1 + r_2)^2 \\ (CD_1 + CD_2)^2 &= 4r_1r_2 \\ CD_1 + CD_2 &= 2\sqrt{r_1r_2}\end{aligned}$$

Consider triangle CD_1C_1, we have

$$\begin{aligned}(r_1 - r)^2 + (CD_1)^2 &= r_1 + r^2 \\ -2r_1r + (CD_1)^2 &= 2r_1r \\ CD_1 &= 2\sqrt{r_1r}\end{aligned}$$

Similar, by considering triangle CD_2C_2, we have $CD_2 = 2\sqrt{r_2r}$.

Merging these equations, we have

$$\begin{aligned}\sqrt{r_1 r} + \sqrt{r_2 r} &= \sqrt{r_1 r_2} \\ \sqrt{r}(\sqrt{r_1} + \sqrt{r_2}) &= \sqrt{r_1 r_2} \\ r &= \frac{r_1 r_2}{(\sqrt{r_1} + \sqrt{r_2})^2}\end{aligned}$$

Data Structure

After knowing the relationship between previous circles and new circle, things become easier. The only problem remains is that, on each day, how can we find the largest circle to be generated? We can of course maintain the order of every drawn circles, and find the maximum new one by iterating through every possibility, but this is too slow!

We can actually use a max heap to speed up this process. The min heap shall maintain all the potential new circles, together with the two circles that it touches. So the first element to be pushed in is the radius of the cricle generated by the given R_1 and R_2, i.e. $(radius = f(R_1, R_2), previous = \{R_1, R_2\})$, where $f(R_1, R_2)$ is the formula obtained in the previous section. Whenever you pop an element $(radius = r, previous = \{r_1, r_2\})$ from the max heap, you should push two new ones (new potential circles): $(radius = f(r, r_1), previous = \{r, r_1\})$ and $(radius = f(r, r_2), previous = \{r, r_2\})$

Comments

This is a problem which requires two different sets of skills (both parts are not really hard though). In a team contest, it's nice that you have teammates with different strengths (hopefully), so you should try utilizing $1 + 1 + 1 > 3$, and cooperate with your teammates. I believe more than 8 teams have members who know geometry (or good at maths) and members who know data structure (or good at coding), while only 8 teams solved this problem during the contest:(try to communicate with your teammates next time!

E: Eat More

We could first come up with a O(N^2) dp solution.

First we create a prefix sum array S, where S[i] = A[1] + A[2] + ... + A[i]. Let dp(i) = no. of ways to group A[1.. i], and apparently, dp(0) = 1 (only 1 way to group a empty list). Suppose we want to calculate dp(i) and we had already calculated dp(0), dp(1)... dp(i – 1). Then dp(i) = sum(dp(j)) where j < i and |S[i] – S[j]| <= K, as we can transform S[i] – S[j] to (A[1]...A[i]) – (A[1]...A[j]) = A[j + 1]...A[i]. Therefore, dp[n] would be the answer.

In this problem, you have to use data structures such as segment tree or BIT, which support range query and point update. In the naive dp solution, we could find out that calculating dp(i) is actually just summing all dp(j) that fulfil the condition (j < i and |S[i] – S[j]| <= K). Slightly transform the formula, we can see:

|S[i] – S[j]| <= K ->
max(S[i] – S[j], S[j] – S[i]) <= K ->
S[i] – S[j] <= K and S[j] – S[i] <= K

So we could get:

S[i] – K <= S[j] <= S[i] + K

We have to find the sum of all dp[j] that satisfy this inequality. We could use a segment tree, where the x^{th} leaf stores the value of sum of dp(j) where S[j] = x. Thus, when we want to calculate dp(i), we query the sum in range(S[i] – K, S[i] + K), and add dp(i) to the $S[i]^{th}$ node.

Since, S[i] is big, one could use discretization on S[i] or dynamic segment tree.

F – Find the base

We could split the problems into two parts, solve the problem for the base ≤ 317, i.e. $\sqrt{10^5}$, and for the base > 317.

When the base is ≤ 317, we can solve it by brute force, iterating all the base from 2 to 317 and count the number of A_i such that digit sum of A_i = M. This part is done is $O(N\sqrt{\max A_i})$.

When the base is > 317, we can observe all A_i will at most contains two digits as max A_i = 10^5. So we can brute force the x such that digit sum of x = M for some base > 317. Let say we exhaust the ten digit of the x, i, from 0 to max(B – 1, M), the unit digit of x, j, = M – i, x must equal to i * b + j for some $b > \max\{i, j, 317\}$. We can iterate all possible b and count from the frequency table of the N integers. Consider the harmonic series, we can finish this part in $O(\max(B, M) * \log(\max A_i))$.

G - Guessing Game

Solution 1 - Exhaustion

We can exhaust every possible Z, and count the number of values of W that can let you win. Output the Z that produces maximum number of winning values W.

```
for (int Z = 1; Z <= 100; Z++)
  if (Z != X && Z != Y) {
    count = 0;
    for (int W = 1; W <= 100; W++)
      if (abs(W-Z) < abs(W-X) && abs(W-Z) < abs(W-Y))
        count++;
    if (count > best) {
      best = count;
      ans = Z;
    }
  }
```

Solution 2 with Observation

We can observe that the answer Z must be one of the three values (WLOG, assuming $X < Y$): $X-1$, $X+1$ (or equivalently, $Y-1$), or $Y+1$. The number of possible values W that these three Z are going to win are: $X-1$, $\lfloor 0.5(Y-X) \rfloor$, $100-Y$ respectively. You can compare these three values to determine which Z is the best.

H - Hacking

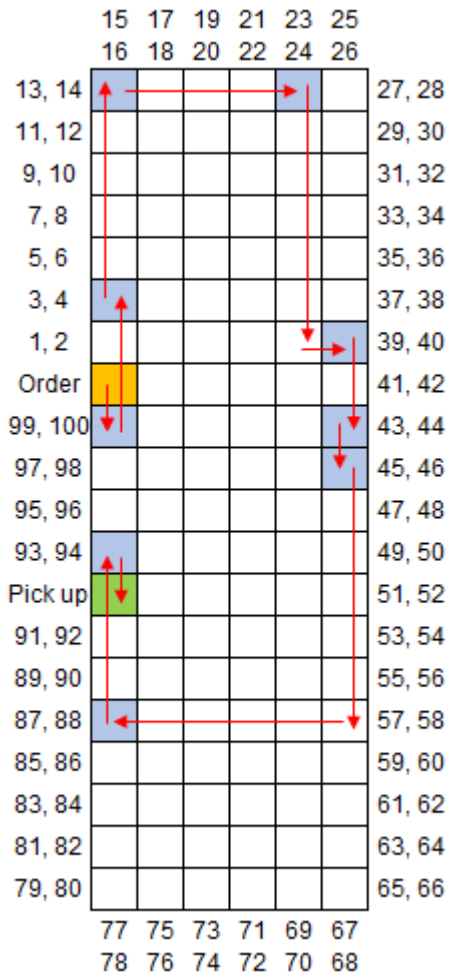
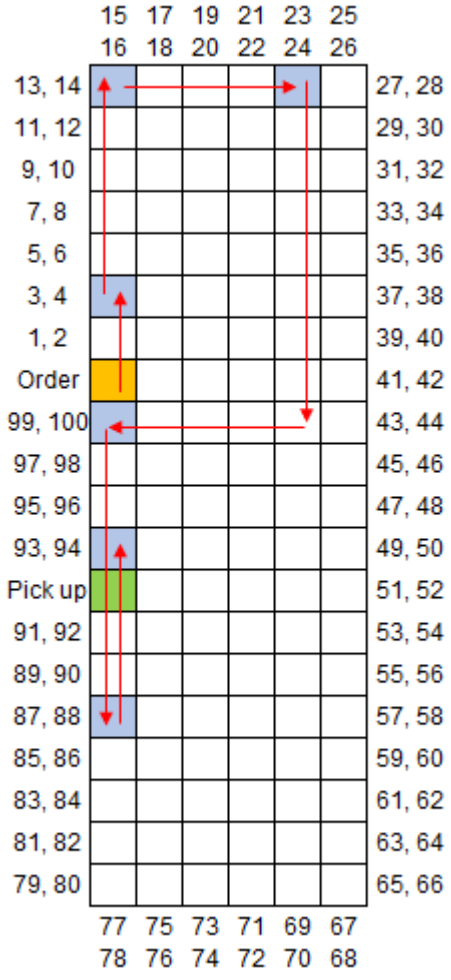
Solution

We can solve it with greedy algorithm. It's intuitive that we should first let the weakest hacker to choose a website to hack, which should be the easiest website that he is skillful enough. We can repeat this process with the second and the third hacker (remember to only consider the unchosen websites). This can be implemented with several `IF`s statements (think carefully!), or with some loops.

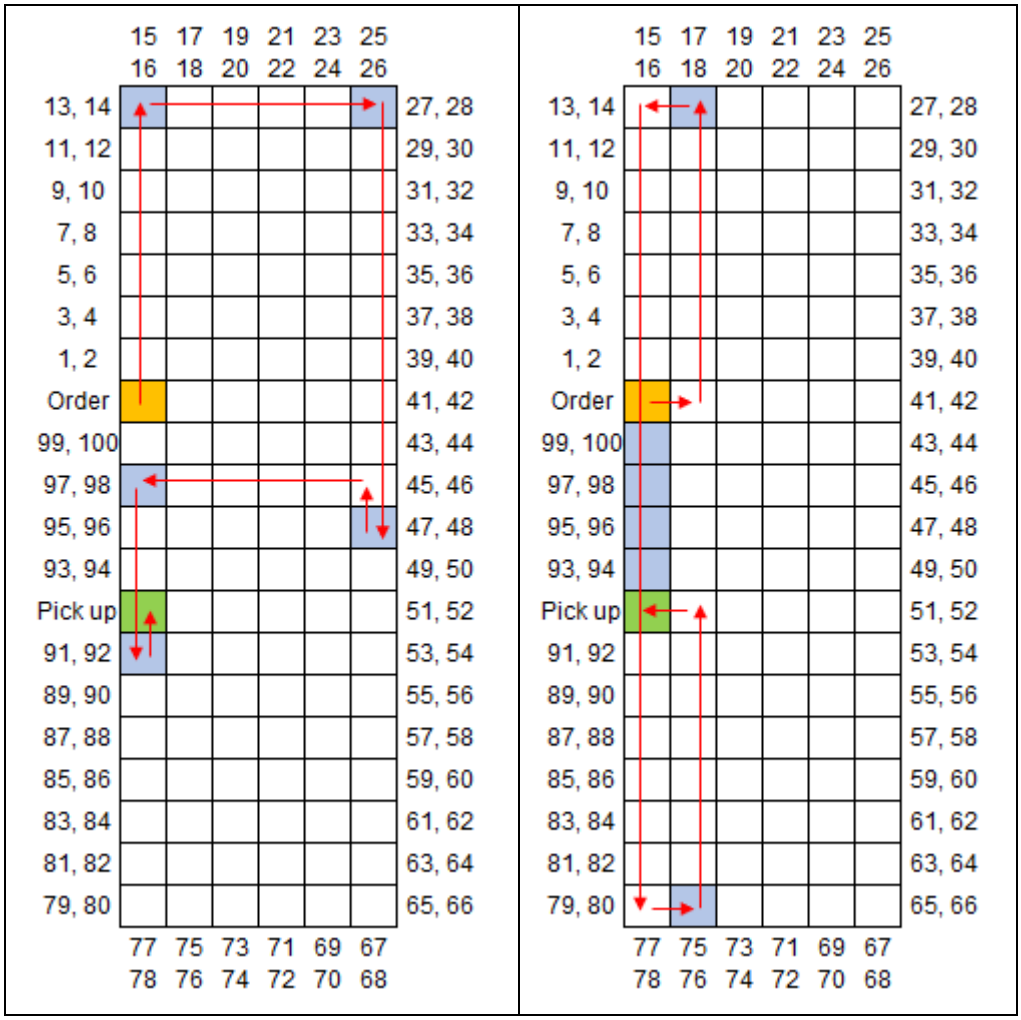
I – Ice-cream Sampler

First, the problem would be much simpler if the input doesn't have flavours 93-100, or if the Order point is right next to the Pick Up point.

Because the Order point is separated from the Pick Up point by flavours 93-100, the problem now has many edge cases like these:

Take 99 / 100 at the beginning. Take 93 / 94 at the end.	If 43-67 is not chosen, go to the left-hand side early.
	

There are even some more complicated ones:



This makes it infeasible to solve the task by handling all these cases. In fact, the author tried to do so but still could not get the correct answer using 120 lines of code.

Luckily, $N = 10$ is small enough for us to try all permutations (ordering). If we have a utility function that converts the flavour number into (row, col), then we can easily compute the distance between two flavours = $\text{abs}(\text{row}_1 - \text{row}_2) + \text{abs}(\text{col}_1 - \text{col}_2)$.

To generate permutations, C/C++ users can use `next_permutation` under `<algorithm>`, but we hshould make sure that the array is initially sorted in non-decreasing order (which is true for this task). Alternatively, we can write a recursive function.

Output the smallest total distance among all $N!$ different orderings.

J - Just A $10 Note

Problem

(Just in case you don't understand the problem) You're required to pay coins for N independent payments, each with an unknown integral value in the range $[0, 9]$. Only \$5, \$2, and \$1 coins are provided, you should prepare the minimum number of coins, such that you are able to fulfill the N payments no matter what the value of them will be.

Solution

For $N = 1$, we definitely need 4 coins, and one of the ways is: $1 \times (\$5), 1 \times (\$2), 2 \times (\$1)$.

For $N = 2$, the optimal way is based on the $N = 1$ solution, and add $1 \times (\$5), 2 \times (\$2)$. Consider several edge cases:

- requirements = {\$8, \$8}:
  a) $\$8 = 1 \times (\$5), 1 \times (\$2), 1 \times (\$1)$
  b) $\$8 = 1 \times (\$5), 1 \times (\$2), 1 \times (\$1)$.
- requirements = {\$8, \$9}:
  a) $\$8 = 1 \times (\$5), 1 \times (\$2), 1 \times (\$1)$
  b) $\$9 = 1 \times (\$5), 2 \times (\$2)$.
- requirements = {\$9, \$9}:
  a) $\$9 = 1 \times (\$5), 1 \times (\$2), 2 \times (\$1)$
  b) $\$9 = 1 \times (\$5), 2 \times (\$2)$.

Other cases can be solved by simply removing a subset of coins from the above arrangements.

Therefore, we can see that for $N = 2$, using 7 coins is possible (and in fact, the optimal).

The general idea for larger values of N is that, for every $(2k+1)$-th payment, prepare additional $1 \times (\$5), 1 \times (\$2), 2 \times (\$1)$. For every $(2k + 2)$-th payment, prepare additional $1 \times (\$5), 2 \times (\$2)$.

The answer is therefore $\lceil 3.5N \rceil$ (it can be shown that this is the optimal number).

K – Kth number in Byteland

We are finding the k^{th} odd length palindrome in the decimal number. We can observe that the answer is always equal to k – 1 concatenating the reversed k – 1 without the last digit as odd length palindrome is constructed by concatenating the reversed first half part of it. For example, the 124^{th} odd length palindrome in the decimal number = “123” + “21” = “12321”.

So we can just read k and decreased k by 1, then convert it to a string. Print it once and print the reversed string without the last digit once. Time complexity = O(|k|).

L - LRTB and TBRL

Solution

Some pairs of cells require the same character to be filled in as they are at the same index in LRTB and TBRL. For example, with $R = 3, C = 2$:

- $(1, 1)$ and $(1, 2)$ (the 1st character in LRTB and TBRL respectively)
- $(1, 2)$ and $(2, 2)$ (the 2nd character in LRTB and TBRL respectively)
- $(2, 2)$ and $(1, 1)$ (the 4th character in LRTB and TBRL respectively)

These pairs will form groups, such that cells within a group should fill in the same character, and cells from different groups are independent. For example, in the case of $R = 3, C = 2$, there are two groups of cells: $\{(1, 1), (1, 2), (2, 2)\}$ and $\{(2, 1), (3, 1), (3, 2)\}$

As different groups are independent from each other, and each group should fill in the same character, there are total $N^{(\text{no. of groups})}$ ways to fill the N characters in the grid.

To form the groups, you may actually form an edge for every pair of cells that require the same character. The problem will then become finding the number of connected components, which can be solved using BFS, DFS, or disjoint-set union-find.

Challenge 1

Find the number of groups in $O(1)$ with the constraint of $R = C$.

Challenge 2

Find the number of groups in $O(1)$ with the constraint of $R = C + 1$.

Challenge 3

Find the number of groups in $O(1)$ with the constraint of $R = C - 1$.

Challenge 4

Find out why, in the problem statement, $N \leq 4762$.