Code Jam 2015 - Qualification Round

Analysis: Ominous Omino

Introduction

There are various aspects to solving this problem. First off, there is one key observation about X-ominoes where X>=7 which we will cover shortly. Beyond that, we describe a brute force solution where we generate all possible X-ominoes, simulate Richard picking one of the X-ominoes, then simulate Gabriel placing that particular X-omino in all possible locations on the grid to determine if a winning configuration can be found. We also present an alternative solution that involves an exhaustive case-by-case analysis of input to determine the winner.

Trivial case where X>=7

There is a key observation for X-ominoes where X>=7. We point out that in such cases, it is impossible for Gabriel to win. How is that?

```
###   ####
#.#   #.##
##.   ###.
```

When X>=7, Richard can always choose a X-omino which has a hole in the middle as shown in the figure above (a 7-omino in the left, and a 10-omino in the right). This means that it is impossible for Gabriel to win using that X-omino at least once, as it is impossible to fill the hole in the middle with another X-omino.

Therefore in cases where X>=7, Richard will always win.

Available cells is a multiple of X

Another insight is the following: if R*C is not a multiple of X, then it is impossible for Gabriel to win. Why? The X-omino occupies X cells exactly, so if the total number of cells is not a multiple of X, it is impossible to cover all the cells using only X-ominoes. Therefore, in such cases, Richard will always win.

In fact, we can extend this principle. Let's imagine that we have already placed one X-omino in the RxC grid, and it results in the RxC grid being separated into two (or more) edge-connected regions. In the figure below, we use a 4-omino in a 2x6.

```
.##...
##....
```

Since the sizes of the connected component (connected components can be found by using flood fill) of the blank areas represented as '.' are 1 and 7, and 1 is too small to fit while 7 is not a multiple of X, then it is not possible for Gabriel to win that particular configuration. Therefore we can say that if we ever arrive in a grid configuration where any of the remaining connected components size is not a multiple of X then we can say that Richard will win that particular configuration. If a connected blank area of size M is a multiple of X, it can be guaranteed that there is a way to place M/X X-ominoes to fill in the blank area. It can be proven by using exhaustive case-by-case analysis which is described in the next section.

Now, armed with the above knowledge, we proceed with describing a brute force solution. First, we describe a way to generate all possible X-ominoes. Then we describe the general brute force strategy to test if Richard can pick a X-omino that will guarantee him a win, and if he is unable to do so then Gabriel will win.

Generating all possible X-ominoes

To generate all possible X-ominoes, we describe a recursive strategy. Let's start with a 1-omino. Well, for a 1-omino there is trivially only 1 possibility which is the following:

```
#
```

Similarly, we can generate the configurations for a 2-omino as follows:

```
#   ##
#
```

How can we do this? We can build the X-ominoes recursively. We start with an empty board (let's say of size 20x20) then place a '#' in middle. Then we can recursively add a '#' adjacent to any of the existing '#' we have already placed, and stop when we have placed X '#' to create a X-omino. But you might have noticed that this recursive process will likely create a lot of duplicate X-omino configurations, e.g. in the 2-omino case, after placing the '#' in the middle, there are 4 possible placements of an adjacent '#' (labeled as 1-4 in the figure below).

```
.1.
4#2
.3.
```

This means that this recursive procedure will generate four 2-ominoes! But we know that there are only two 2-ominoes, as "4#" and "#2" are equivalent, and likewise for "1#" and "3#". Therefore, we can come up with a way to remove duplicate X-omino configurations if needed, which we leave as an exercise.

We can precompute all the X-ominoes for 1<=X<=6. Also note that when generating X-ominoes this way, we will generate all X-ominoes under reflection and rotations which we show with an example below:

```
#.  .#  .##  ##.
##  ##  ##.  .##
.#  #.
```

For our solution, it is fine to generate all such X-ominoes. Note that it is also feasible to generate all possible X-ominoes for X<=6 by hand. We cover this in the "Alternative Solution" section below.

Brute force strategy

Now, after we have generated all possible X-ominoes, we proceed with describing the brute force strategy. As mentioned earlier, if X>=7, we report that Richard will win, and also if R*C is not a multiple of X, we report that Richard will win.

In the brute force strategy, we simulate Richard picking each of the X-ominoes one-by-one, then simulate if it is possible for Gabriel to win with that particular X-omino. If we can find any X-omino that Richard can pick which results in Gabriel being unable to win, then we report that Richard can win. But if Gabriel wins for all X-ominoes that Richard picks, then we report Gabriel as the winner.

Simulating Richard picking a X-omino is straightforward. Richard can pick each of the X-omino one by one. The trickier part is to check if Gabriel can win, we now describe a strategy to perform this check.

We take the RxC grid and we have the X-omino that Richard has required that Gabriel use at least once. We can brute force the placement of this X-omino in the RxC grid (NOTE: We have to try the CxR grid too, we elaborate on the reasons to take the CxR grid in a subsequent paragraph). If it is impossible to place the X-omino (e.g. if the width of the X-omino is bigger than C) in either the RxC grid or the CxR grid then we trivially say that Richard wins for that particular X-omino. If it is possible to place the X-omino in a particular location, we still need to check whether it is possible for Gabriel to win. Letâ€™s see some examples. Suppose that the X-omino Richard chose is the following.

```
.##
##.
```

and we are given a 2x4 grid. In that case, we can place the X-omino in the following two configurations.

```
.##.
##..
```

or

```
..##
.##.
```

We notice that in both configurations, the X-omino has divided the grid into two connected components of '.' of size 1 and 3. Well, we had mentioned earlier in the section "Available cells is a multiple of X" that such configurations imply Gabriel will always lose.

In fact, after placing a X-omino if it results in connected components of size M where all such M is a multiple of X, then we can say that Gabriel wins with that X-omino placed at that particular location.

As another example, if we use the following 4-omino:

```
##
##
```

and we are given a 2x6 grid, and if we placed the 4-omino as follows:

```
.##...
.##...
```

then it is not possible for Gabriel to win (there are two connected regions of size 2 and 6, and 2 and 6 are not multiples of 4). But if we placed the 4-omino as follows:

```
##....
##....
```

or

```
..##..
..##..
```

Then in both configurations we say that Gabriel wins (in the first case, there is one connected component of size 8, and 8 is multiple of 4; in the second case, there are two connected components of size 4 each, and 4 is a multiple of 4).

As mentioned in an earlier paragraph, we try the CxR grid too. The reason is because we have to check for a 90 degree rotation of the X-omino that Richard selected. Remember, Gabriel can rotate or reflect the X-omino when initially placing it on the grid. Instead of rotating the X-omino, we can instead just rotate the grid! Therefore it suffices to check if we can fill a RxC grid, or a CxR grid with that X-omino. Note that we don't need to consider all possible reflections and rotations of the X-omino because because in grid space, things are symmetric therefore it suffices to check for both the RxC and CxR grid.

Therefore to generalize it, after placing the selected X-omino at a particular place, we can check the sizes of the edge-connected '.' components, and if all such components have size M is a multiple of X, then it means that we can always fill the M space with M/X X-ominoes. In such a case, we say that that particular grid configuration is one for which Gabriel can win. An explanation of this winning condition for our brute force solution is based off the conclusion that we arrive at from the "Alternative solution" presented below.

Alternative solution

The alternative solution involves careful analysis of various cases. Let **S** = min(R, C) and **L** = max(R, C) so that S<=L. Suppose that the grid dimensions are SxL (if the dimensions are LxS, the grid can be rotated without affecting the win conditions).

Richard can force a win if any of the following conditions hold; otherwise Gabriel will win.

1. X does not divide S*L,
2. X=3 and S=1,
3. X=4 and S<=2,
4. X =5 and either (i) S<=2 or (ii) (S, L) = (3, 5),
5. X=6 and S<=3,
6. X>=7.

We have already explained (1) and (6) in the previous paragraphs. For (2), (3), (4i), (4ii) and (5), Richard can choose the following pieces and always guarantee a win:

(2)

```
#.
##
```

It is impossible for Gabriel to fit the above piece in a grid with S=1. Similar explanation follows for the following two cases.

(3)

```
###
.#.
```

For the above piece, when the grid has S=2 notice that it will divide the '.' cells into two connected regions and that it is impossible for these regions to have a size which is a multiple of 4. It is impossible for Gabriel to fit the above piece in a grid with S=1.

(4i)

```
#..
##.
.##
```

Similar to the above cases, it is impossible for Gabriel to fit the above piece in a grid with S<=2.

(4ii) For this case, Richard can use the same piece as used in (4i). By trying all possibilities, one can see that it is impossible for Gabriel to win with X=5 and a 3x5 grid.

```
#....  .#...  ..#..
##...  .##..  ..##.
.##..  ..##.  ...##
```

(5)

```
.#..
####
.#..
```

A similar explanation follows for the above piece as the explanation for (3).

For all combinations of X, S and L not satisfied by the above condition, Gabriel will win. We provide here an explanation for the X=6 case, and leave the other cases as an exercise. Let's consider the 4x6 grid, which is the smallest grid for which S>3 and S*L is a multiple of X. We show below Gabriel's strategy for filling the 4x6 grid, and then generalize for cases where the grid is bigger than 4x6.

First off, we point out that for a 6-omino, there are only <a href="http://en.wikipedia.org/wiki/Hexomino" >35 choices. In fact, listed below are the number of choices for each X-omino:

- 1-omino, 1 choice,
- 2-omino, 1 choice,
- 3-omino, 2 choices,
- 4-omino, 5 choices,
- 5-omino, 12 choices.

Since the 6-omino only has 35 choices, we list out below a case-by-case analysis of the 35 choices that Richard can make, and Gabriel's strategy for filling up the rest of the cells to guarantee a win for himself. '#' denotes the original piece that Richard chooses while 'a', 'b', and 'c' are 6-ominoes that Gabriel chooses.

```
######   a#####   a#####   a#####   ####cc   ##abbb   ##abbb
aaaaaa   aaaaa#   aaaa#c   abb#cc   aab##c   ##abbb   #aabbb
bbbbbb   bbbbbb   abbbbc   aabbcc   aabbcc   #aaccc   ##accc
cccccc   cccccc   bbcccc   aabbcc   aabbbc   #aaccc   #aaccc

##abbb   #aabbb   ###bbb   #aaaaa   #aaaaa   aaa#bb   aa#bbb
#aabbb   ##abbb   #aabbb   ###cca   ####ca   a####b   a####b
#aaccc   ##accc   #aaccc   #ccccb   #ccccc   aacc#b   aaac#b
##accc   #aaccc   #aaccc   #bbbbb   bbbbbb   ccccbb   cccccb

a#bbbb   aa#bbb   aa#bbb   a###bb   ###bbb   ###bbb   ###bbb
a####b   a####b   a####b   aaa##b   #a##bb   aa###b   a###bb
aacc#b   aaa#bb   aa#ccb   acc#bb   aaaccb   aacccb   aaaccb
aacccc   cccccc   accccb   accccb   aacccc   aacccb   aacccc

###aaa   aaa#bb   aaa###   aa#bbb   ###bbb   ###bbb   aaaaaa
###aaa   a###bb   aaab##   a###bb   aa#bbb   aa##bb   ###bbb
cccbbb   aac##b   bbbb#c   aaa##b   aa##cc   aaa#cb   #c#bbb
cccbbb   cccccb   bccccc   cccccc   aacccc   accccc   #ccccc

aaaaab   aabbbb   ##bbbb   aaaaaa   aaaaab   aaaabb   ##aaaa
#a#cbb   a#b#bc   a###bb   #bbbbc   a#bbbb   aa#cbb   c##aab
```

```
###cbb    a###cc    aaa#cc    ##bbcc    ###ccb    ###cbb    cc##bb
#ccccb    aa#ccc    aacccc    ###ccc    ##cccc    ##cccc    cccbbb
```

Whew, that was a lot of cases! Clearly, Gabriel will always win with a 4x6 grid and any 6-omino that Richard chooses.

We now explore the case when the grid is bigger for X=6. Let's pick the 6x8 grid. To win in such cases, Gabriel can use the following strategy. Gabriel can use the top-left corner of the board to place the piece that Richard chooses so that he can complete a 4x6 portion similar to the manner described in the 35 cases above. Then Gabriel can label the remaining cells using a Hamiltonian path (with cells as vertices, and adjacent cells as neighbors). 'Z' denotes the 4x6 portion on the top-left corner of the grid, and the rest is filled with a sequence of 'a'-'x' which denotes the Hamiltonian path. In this case, Gabriel can start the path from the top-right corner and 'snake' back and forth. In general, depending on the parity of the number of rows of the 'Z' region (let's call the number of these rows T), Gabriel can start from the top-right (when T is even), or from the cell adjacent to the top-right corner of the Z-region (when T is odd) then 'snake' back and forth to fill the rest of the cells.

```
ZZZZZZba
ZZZZZZcd
ZZZZZZfe
ZZZZZZgh
ponmlkji
qrstuvwx
```

Observe that the cells 'a'-'x' forms a chain. We can simply chop off this chain into sizes of 6 each (i.e. a 6-omino!). This way, Gabriel can win for X=6 and grids that are bigger than 4x6.

Similar to above, we can list out cases for the other X-ominoes, which we leave as an exercise.

Here is a sample implementation in Python:

```python
def richard_wins(X, R, C):
  S = min(R, C)
  L = max(R, C)
  if (S * L) % X != 0: return True
  if X == 3 and S == 1: return True
  if X == 4 and S <= 2: return True
  if X == 5 and (S <= 2 or (S, L) == (3, 5)): return True
  if X == 6 and S <= 3: return True
  if X >= 7: return True
  return False

for tc in range(input()):
  X, R, C = map(int, raw_input().split())
  print "Case #%d: %s" % (tc + 1,
    "RICHARD" if richard_wins(X, R, C) else "GABRIEL")
```