CMIMC 2021
TCS Round

INSTRUCTIONS

1. Do not look at the test before the proctor starts the round.
2. This test consists of several problems, some of which are short-answer and some of which require proofs, to be solved within a time frame of **60 minutes**. There are **70 points** total.
3. Answers should be written and clearly labeled on sheets of blank paper. Each numbered problem should be *on its own sheet*. If you have multiple pages, number them as well (e.g. 1/3, 2/3).
4. Write your team ID on the upper-right corner and the problem and page number of the problem whose solution you are writing on the upper-left corner on each page you submit. Papers missing these will not be graded. Problems with more than one submission will not be graded.
5. Write legibly. Illegible handwriting will not be graded.
6. In your solution for any given problem, you may assume the results of previous problems, even if you have not solved them. You may not do the same for later problems.

Balance the Board

You place n^2 indistinguishable pieces on an $n \times n$ chessboard, where $n = 2^{90} \approx 1.27 \times 10^{27}$. Of these pieces, n of them are slightly lighter than usual, while the rest are all the same standard weight, but you are unable to discern this simply by feeling the pieces.

However, beneath each row and column of the chessboard, you have set up a scale, which, when turned on, will tell you *only* whether the average weight of all the pieces on that row or column is the standard weight, or lighter than standard. On a given step, you are allowed to rearrange every piece on the chessboard, and then turn on all the scales simultaneously, and record their outputs, before turning them all off again. (Note that you can only turn on the scales if all n^2 pieces are placed in different squares on the board.)

Find an algorithm that, in at most k steps, will always allow you to rearrange the pieces in such a way that every row and column measures lighter than standard on the final step.

Scoring

An algorithm that completes in at most k steps will be awarded:

- 1 pt for $k > 10^{55}$
- 10 pts for $k = 10^{55}$
- 30 pts for $k = 10^{30}$
- 50 pts for $k = 10^{28}$
- 65 pts for $k = 10^{20}$
- 80 pts for $k = 10^{5}$
- 90 pts for $k = 2021$
- 100 pts for $k = 500$

You are allowed to prove multiple bounds, and will receive points for the best bound that is correctly proven. **Please state which bound you are trying to prove above any proof, or you may not be awarded points for that bound.**

Partial points may be awarded for progress towards the next bound, and partial points may be deducted for logical flaws or lack of rigor in proofs. To get full points, you must demonstrate that your algorithm always produces a correct result, and always runs in at most the stated number of moves.

Solutions

For all of these solutions, we will define a row/column to be "empty" if it does not contain a light piece, and "full" otherwise, so our goal is to make every row and column full. In addition, we will assume without loss of generality that the starting configuration contains at least one empty row/column, as otherwise the board can be balanced in 1 step, which trivially satisfies every possible bound.

Solution for $k = 10^{55}$ (10 pts). If we initially have $k > 0$ empty columns, then we have at most $n(n-k)$ squares we don't know the weight of. We may check these squares one by one by swapping them out with a square from the empty column, and seeing if the column becomes light or not, so after $n^2 - nk + 1 < n^2 < 10^{55}$ steps, we can balance the board.

Solution for $k = 10^{30}$ (30 pts). We can speed up the previous solution by binary searching for a light piece on each column, i.e. take half of a light column and swap it out with half of the empty column. If the empty column becomes light, repeat this procedure with half of the section we just swapped in. Otherwise, put that section back and swap in half of the section we didn't use, and so on. In this way, we can guarantee we will discover one light piece in at most $\log_2(n)$ moves, at which point we can swap it with a random piece in the leftmost column, and continue searching (if the piece from the leftmost column we swapped with happened to also be light, then we have just found another light piece in $1 < \log_2(n)$ moves, so we again store this in the left column and continue as usual). Since there are n light pieces total, we can find them all in at most $n\log_2(n)$ moves, or approximately $10^{27} \cdot 90 < 10^{30}$.

Solution for $k = 10^{28}$ (50 pts). Again, note that we have at least one empty column.

Now split the remaining grid squares into $2n$ groups each of size $\leq n/2$. For each group, swap it into the empty column and check if that group contains a light piece. (This takes $2n$ measurements). As there are only n light pieces, at most n of the groups contain a light piece, so there are at most $n \cdot n/2$ light piece candidates, and therefore at least $n^2/2$ squares are guaranteed to be empty.

Now we do the same process, but this time we can form $n/2$ empty columns (using the $n^2/2$ empty squares) and can therefore do $n/2$ checks in parallel. Thus each step of the process (at least) halves the number of remaining squares, and takes only $\frac{2n}{n/2} = 4$ measurements.

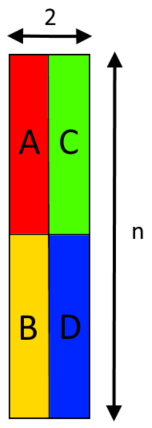
As we start out with $n^2/2$ "light" candidates and need to end up with n "light" candidates (ie. the light squares) we need to run the process $\log_2(n/2)$ times, or 89, and then one final step to put the light squares in their places. Thus the overall number of measurements is $2n + 89 \cdot 4 + 1$ which is about $2.54 \cdot 10^{27} < 10^{28}$.

Solution for $k = 10^5$ (80 pts). You might have noticed that the second half of the last solution was clearly fast enough to get all the bounds. All we need to do is to get to $n/2$ empty columns faster.

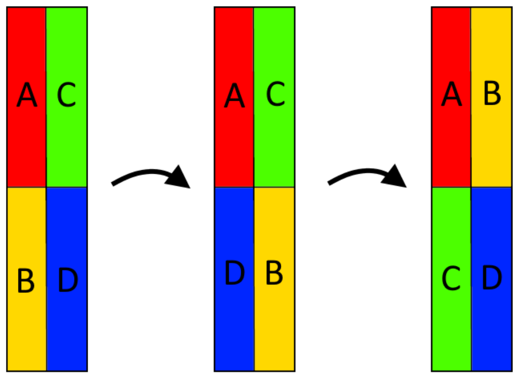
At any point in time, if we have k free columns, we may run k instances of binary search in parallel. Each binary search will either make a column empty, or remove a duplicate element from some column with more than one light piece. Since our initial number of duplicate elements is bounded at our initial value of k, we can define the number of free columns after i parallel binary searches, or c_i, to satisfy the recurrence $c_{i+1} = 2c_i - d_i$ (as long as $c_i < n/2$), where d_i is the number of duplicates found on the ith move. We can see that the sum of all d_i is simply our initial k value (THIS IS IMPORTANT: if there's a lot of duplicates in columns, then that means that we started out with a lot of empty columns to begin with). It is easy to show c_i increases exponentially (in the worst case d_1 is k), thus $c_i \approx n/2$ in $\log(n)$ parallel binary searches, each taking $\log(n)$ time. Once we hit $k = n/2$, this doubling mechanism doesn't really continue to work, so we revert to the original method.

Thus the overall runtime is $90 \cdot 90 + 89 \cdot 4 + 1 < 10^5$.

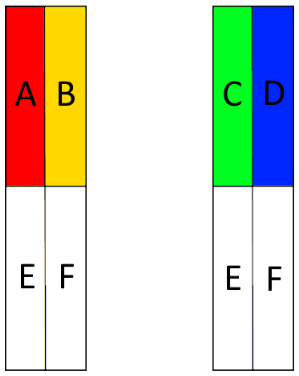
Solution for $k = 500$ (100 pts). It's also possible to get the first half of the problem in only 5 steps. Consider the board as $n/2$ groups of dimension $2 \times n$. Split each group into 4 sub-groups A,B,C,D of size $1 \times n/2$.



We now take 3 steps and measure them in the configurations $\{AB, CD\}, \{AC, BD\}, \{AD, BC\}$.



Now for each $2 \times n$ group, if it contains 2 or less light pieces, then we will know exactly which ones of the subgroups contains a light group. Next, for each group which contains 3 or more light pieces, match it up with an empty column EF (note that we must have generated enough such empty columns). Then in 2 steps measure $\{AE, BF\}, \{CE, DF\}$.



This will tells us exactly which sub-groups have a light square, and as there are at most n of them, there will be $n^2/2$ pieces which we know are not light, and we can re-arrange these pieces into $n/2$ columns guaranteed to be empty.

Thus in 5 moves, we have done the first half, and by the method above we can finish in $4\log_2(n)$ moves, so the overall runtime is $5 + 89 \cdot 4 + 1 < 500$

Multiply to the Moon

You are initially given the number $n = 1$. Each turn, you may choose any positive divisor $d \mid n$, and multiply n by $d+1$. For instance, on the first turn, you must select $d = 1$, giving $n = 1 \cdot (1+1) = 2$ as your new value of n. On the next turn, you can select either $d = 1$ or 2, giving $n = 2 \cdot (1+1) = 4$ or $n = 2 \cdot (2+1) = 6$, respectively, and so on.

Find an algorithm that, in at most k steps, results in n being divisible by the number $2021^{2021^{2021}} - 1$.

Scoring

An algorithm that completes in at most k steps will be awarded:

- 1 pt for $k > 2021^{2021^{2021}}$
- 20 pts for $k = 2021^{2021^{2021}}$
- 50 pts for $k = 10^{10^4}$
- 75 pts for $k = 10^{10}$
- 90 pts for $k = 10^5$
- 95 pts for $k = 6 \cdot 10^4$
- 100 pts for $k = 5 \cdot 10^4$

You are allowed to prove multiple bounds, and will receive points for the best bound that is correctly proven. **Please state which bound you are trying to prove above any proof, or you may not be awarded points for that bound.**

Partial points may be awarded for progress towards the next bound, and partial points may be deducted for logical flaws or lack of rigor in proofs. To get full points, you must demonstrate that your algorithm always produces a correct result, and always runs in at most the stated number of moves.

Solutions

For all of the solutions, we define the following notation. Given a natural number m, we define $f(m)$ to be the least number of steps necessary so that n is divisible by m. We let $N = 2021^{2021^{2021}} - 1$, so the goal of the problem is to bound $f(N)$.

Solution for $k = 2021^{2021^{2021}}$ (20 pts). Suppose $t \mid n$. Using one step, we can multiply n by $t+1$, so n will be divisible by $t+1$. Thus $f(t+1) \le f(t)+1$ for all t. Since $f(1) = 0$, we can apply this repeatedly to see that $f(N) \le N - 1$.

Solution for $k = 10^{10^4}$ (50 pts). We prove that for any m, we have $f(m) \le 2\log_2 m$. Observe that since 1 is always a divisor of n, we can always multiply by 2 at any step, so $f(2t) \le f(t)+1$. The 20-point solution tells us that $f(t+1) \le f(t)+1$. So if m has binary expansion $2^{n_1} + 2^{n_2} + \cdots + 2^{n_p}$ for $n_1 > \cdots > n_p$, we can force m to be a divisor of n within $n_1 + p - 1$ steps (there are n_1 doubling steps and $p-1$ increment-by-1 steps).

Since n_1 and p are each $\le \log_2 m$, we conclude that $f(m) \le 2\log_2 m$. This implies, using the rough estimates $2021 < 10^4$ and $\log_2 2021 < 11$, that

$$f(N) \le 2\log_2 N < 2 \cdot 2021^{2021} \log_2 2021 < 22 \cdot 10^{8084}.$$

Before moving on to the next solution, we introduce an auxiliary function $g(a,b)$ to be the smallest possible integer such that

$$f(x^a(x^b-1)) \le g(a,b) + f(x-1)$$

for all $x \ge 2$. For example, $g(1,1) = 1$ since if $x-1$ divides n, we can ensure that $x(x-1)$ divides n after one step, regardless of the particular values of x and n. Similarly, $g(1,2) = 2$ since if $x-1$ divides n, then after two steps we can ensure that $(x-1)\cdot x \cdot (x+1) = x(x^2-1)$ divides n.

Applying this to $x = 2021$, $a = 0$, and $b = 2021^{2021}$, we have

$$f(N) \le g(0, 2021^{2021}) + f(2020).$$

By the 50-point solution, we have $f(2020) \le 2\log_2 2020 \le 22$. In fact, $f(2020) \le 5$ via the sequence of values $1, 2, 4, 20, 100, 10100$.

Observe that if $a' \le a$, then $g(a', b) \le g(a,b)$, i.e. g is nondecreasing in its first entry. The following lemmas will be useful in all subsequent solutions.

Lemma 1. *Let $d \mid b$ and $a \ge d$. Then $g(a, b+d) \le g(a,b) + 1$.*

Proof. Suppose $x^a(x^b-1)$ divides n. Since $a \ge d$, $x^d(x^b-1)$ also divides n. Since $d \mid b$, $x^d - 1$ divides $x^b - 1$. We then perform the step

$$n \mapsto n \cdot \left(\frac{x^d(x^b-1)}{x^d-1} + 1\right) = \frac{n}{x^d-1} \cdot (x^{b+d} - 1).$$

After this step, $x^{b+d}-1$ divides n, and we have not affected divisibility by x^a. So we're done. □

Lemma 2. *$g(ca, cb) \le g(a,b) + g(0,c)$. In particular, $g(0, cb) \le g(0,b) + g(0,c)$ and $g(0, b^r) \le r \cdot g(0,b)$.*

Proof. If $x-1$ divides n, then $x^c - 1$ can be obtained in $g(0,c)$ steps. Now $x^{ca}(x^{cb}-1) = (x^c)^a((x^c)^b - 1)$ can be obtained from $x^c - 1$ in an additional $g(a,b)$ steps. □

Solution for $k = 10^{10}$ (75 pts). We prove that for $m \geq 1$, we have $g(0,m) \leq m$. We know that $g(1,1) = 1$. Applying Lemma 1 repeatedly with $a = d = 1$, we conclude that $g(1,m) \leq g(1,1)+m-1 = m$, so $g(0,m) \leq m$ as desired.

Thus $g(0,2021) \leq 2021$, so Lemma 2 shows that $g(0,2021^{2021}) \leq 2021^2$. So $f(N) \leq 2021^2 + 5$.

Solution for $k = 10^5$ (90 pts). We will improve on the last solution by showing that $g(0,m) \leq 3\log_2 m$.

Lemma 3. *For $m \geq 1$, write $m = 2^{n_1} + \cdots + 2^{n_p}$ where $n_1 > \cdots > n_p$. Then*

$$g(0,m) \leq g(2^{n_1-1},m) \leq 2n_1 + p - 1.$$

Proof. We know $g(0,2) = g(1,2) = 2$. By Lemma 2 we then get $g(2a,2b) \leq g(a,b)+2$. By Lemma 1, for any $a \geq 1$, we get $g(a,b+1) \leq g(a,b)+1$. So, doubling b costs 2 steps while incrementing b by 1 costs 1 step. Every time we double b, we can also double a.

Starting from $g(1,2) = 2$, we need $n_1 - 1$ doubling operations and $p-1$ increment-by-1 operations to obtain an m in the second slot. It follows that $g(2^{n_1-1},m) \leq 2(n_1-1)+(p-1)+2$. □

Since n_1 and p are each $\leq \log_2 m$, Lemma 3 implies that $g(0,m) \leq 3\log_2 m$. So

$$g(0,2021^{2021}) \leq 3 \cdot 2021 \cdot \log_2 2021 < 66577$$

so $f(N) < 66582$.

Solution for $k = 6 \cdot 10^4$ (95 pts). For this bound, we apply Lemma 3 to $2021 = 11111100101_2$ to conclude that $g(0,2021) \leq 27$. It follows by Lemma 2 that $g(0,2021^{2021}) \leq 27 \cdot 2021 = 54567$, so $f(N) \leq 54572$.

Solution for $k = 5 \cdot 10^4$ (100 pts). We claim that $g(0,2021) \leq 24$. We have by Lemma 3 that $g(16,32) \leq 10$, $g(0,3) \leq 3, g(0,7) \leq 6$. Using these three results together with Lemma 2 we get the chain of inequalities

$$\begin{aligned} g(1008,2016) &\leq g(16,32) + g(0,63) \\ &\leq g(16,32) + g(0,3) + g(0,3) + g(0,7) \\ &\leq 22. \end{aligned}$$

Applying Lemma 1 with $(a,b,d) = (1008,2016,4)$ we get

$$g(1008,2020) \leq g(1008,2016) + 1 \leq 23.$$

Applying Lemma 1 again with $(a,b,d) = (1008,2020,1)$ we get

$$g(1008,2021) \leq g(1008,2020) + 1 \leq 24.$$

So $g(0,2021) \leq 24$. As in the previous solutions, this gives $g(0,2021^{2021}) \leq 24 \cdot 2021 = 48504$, so $f(N) \leq 48509$.

Topologist's Trap the Tiger

There is a tiger (which is treated as a point) in the plane that is trying to escape. It starts at the origin at time $t = 0$, and moves continuously at some speed k. At every positive integer time t, you can place one closed unit disk anywhere in the plane, so long as the disk does not intersect the tiger's current position. The tiger is not allowed to move into any previously placed disks (i.e. the disks block the tiger from moving). Note that when you place the disks, you can 'see' the tiger, (i.e. know where the tiger currently is).

Your goal is to prevent the tiger from escaping to infinity. In other words, you must show there is some radius $R(k)$ such that, using your algorithm, it is impossible for a tiger with speed k to reach a distance larger than $R(k)$ from the origin (where it started).

Find an algorithm for placing disks such that there exists some finite real $R(k)$ such that the tiger will never be a distance more than $R(k)$ away from the origin.

Scoring

An algorithm that can trap a tiger of speed k will be awarded:

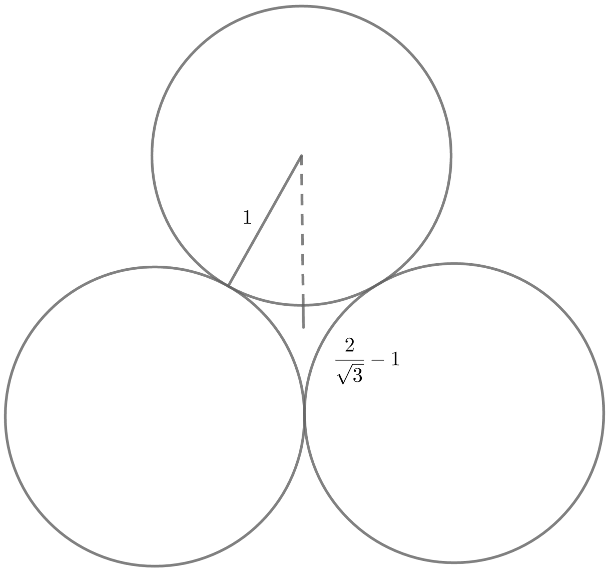
- 1 pt for $k < 0.05$
- 10 pts for $k = 0.05$
- 20 pts for $k = 0.2$
- 30 pts for $k = 0.3$
- 50 pts for $k = 1$
- 70 pts for $k = 2$
- 80 pts for $k = 2.3$
- 85 pts for $k = 2.6$
- 90 pts for $k = 2.9$
- 100 pts for $k = 3.9$

You are allowed to prove multiple bounds, and will receive points for the best bound that is correctly proven. **Please state which bound you are trying to prove above any proof, or you may not be awarded points for that bound.**

Partial points may be awarded for progress towards the next bound, and partial points may be deducted for logical flaws or lack of rigor in proofs. To get full points, you must demonstrate that your algorithm always produces a correct result, and always traps a tiger of the stated speed.

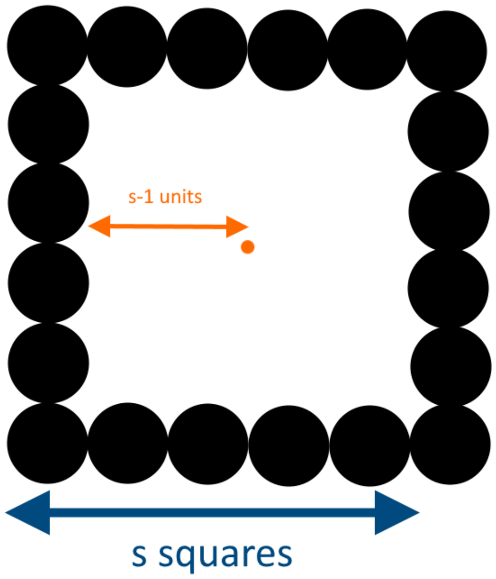
Solutions

Solution for $k = 0.05$ (10 pts). In this case, we may simply construct a triangle around the tiger:



The distance from the tiger to the nearest point on any circle is $\frac{2}{\sqrt{3}} - 1 > 0.1$, and the tiger has only 2 seconds to move between when we place the first and last circles, so it cannot reach the boundary of any of the circles before we place them all if it's speed is $\leq \frac{0.1}{2} = 0.05$.

Solution for $k = 0.2$ (20 pts). We can build off of the previous idea, and instead construct a more efficient enclosure, this time being a square of arbitrarily large size, formed using walls of touching circles:

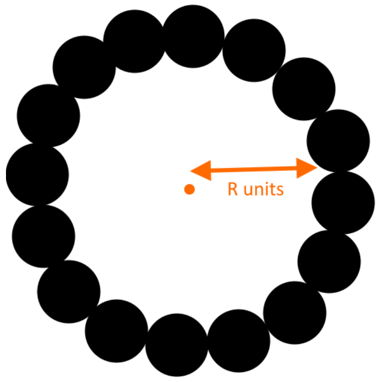


In this case, if our square consists of $4s$ circles, the closest point from the tiger to the boundary is distance $s - 1$, so we can trap any tiger of speed less than $\frac{s-1}{4s}$, which is greater than 0.2 for $s > 5$.

Solution for $k = 0.3$ (30 pts). If we take the logical continuation of the previous constructions, we can see that the most efficient shape in terms of

$$\frac{\text{distance to closest point}}{\text{perimeter}}$$

is a circle, so we can construct a large circle by lining the circumference with our small circles:



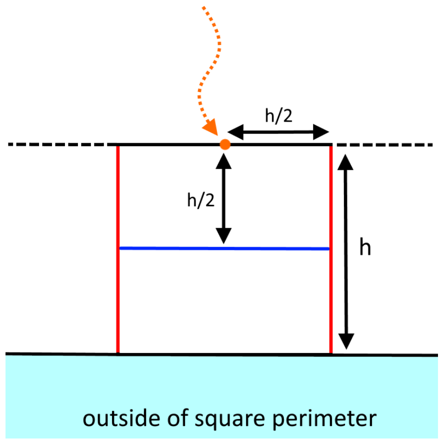
As we make our outer circle larger, the circumference will become locally flat, so each circle will cover a length that is arbitrarily close to 2. The total circumference will be $2\pi R$, requiring πR circles, and the distance to the closest circle will be approximately $R - 1$, so we can trap any tiger with speed less than $\frac{R-1}{\pi R} \approx \frac{1}{\pi}$, which is greater than 0.3, as $\pi < \frac{10}{3}$.

For the remaining solutions, we will instead consider the tiger to move at a fixed speed of 1, and vary the radius r of the circles we place. This will make it easier to see how structures we define in one solution may be applied in another.

The general idea behind the following solution is that we should simply try to place down circles near the tiger's current position, and since we are able to build a wall whose path length increases by 1 unit per second, the tiger should not be able to outrun our wall. The main hurdle with this type of argument, particularly if applied to a non-square perimeter, is ensuring that the *angular speed* of the wall can match that of the tiger, i.e. the tiger cannot gain an advantage by cutting corners and moving further from the perimeter. We handle this by only considering when the tiger is within a constant distance ($h = 32$) of the perimeter. There are simpler constructions that accomplish this task, but it is an important point to consider and verify in any correct solution.

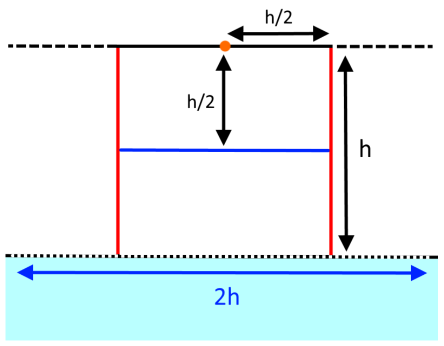
Solution for $r = \frac{1}{2}$ ($k = 2$) (70 pts). For this radius, we will only place circles at points with integer coordinates, and solve the strictly harder problem for a tiger on an integer lattice, who can make a king's move (to one of 8 adjacent squares) every turn. The general idea will be to adaptively construct a large square outer perimeter from which the tiger cannot escape.

We will now define a *layered barrier* of size h to be a square region of height h whose base lies along our outer perimeter. At the moment the tiger first comes within h of the perimeter, we can define this region to have the top side centered at the current position of the tiger:

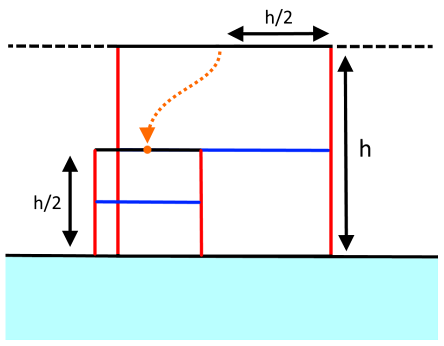


We will pay particular attention to the horizontal mid-line of our region, as well as the side walls. There are now 3 possibilities for what the tiger can do:

- The tiger exits through the top face. In this case, the tiger is no longer within h of the perimeter, so we don't have to do anything.
- The tiger crosses the horizontal mid-line. In this case, since it will take at least $\frac{h}{2}$ time for the tiger to reach this line, we can consider the segment of length $2h$ along the perimeter (centered along the vertical axis of our region), and uniformly cover 1 in every 4 squares along this segment:

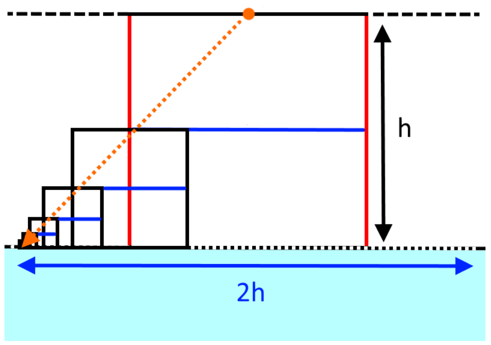


When the tiger finally reaches the mid-line, we can then define a new layered barrier of height $\frac{h}{2}$, centered at the point where the tiger crosses the line, and repeat this procedure recursively:

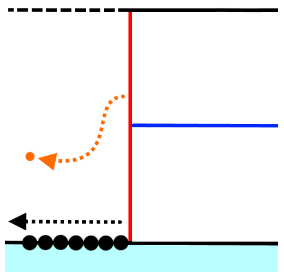


Note that each layer will fill 25% of the squares below it's region, so after 4 such layers, the bottom area of the innermost region will be completely sealed off, at which point we have no need to construct further layers. A layered barrier with no sub-layers can exist for $h = 4$, so in total our original layered barrier only needs to have height $h = 2^3 \cdot 4 = 32$ to ensure that the tiger can never exit through the bottom face of the region, if it entered through the top and never left through the side.

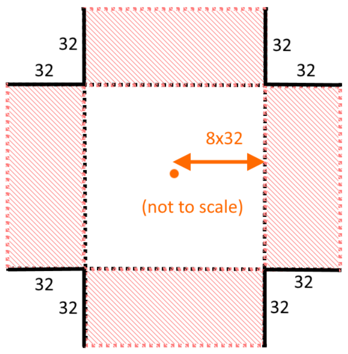
Small aside: the reason we covered a segment of length $2h$ is because the base of every subsequent region is guaranteed to be within this interval, as shown below:



- The tiger exits through a side wall. In this case, we will simply start placing down consecutive circles along the direction the tiger exited, until the point where the tiger either re-enters the original region, or goes further than distance 32 from the perimeter. Since our circles have diameter 1, the tiger will never be able to exceed the horizontal velocity of our barrier-forming process here, so there is no risk of the tiger going below the wall:



Finally, we must cover up the corner areas of our square, so that no two layered barriers facing in perpendicular directions ever intersect, and the above properties always hold. We can easily do this in $8 \cdot 32$ moves, which is not an issue if the original square has side length greater than $9 \cdot 32$:



We are now able to keep our tiger within this square region indefinitely, by following the procedures described above.

Solution for $r = \frac{\pi}{2\pi+1} + \varepsilon$ ($k = 2.3$) (80 pts). We will now relax the square grid restriction, and again try to create a large circular outer perimeter instead (of radius R). For large enough R, the circumference is locally (approximately) flat, so we may build our layered barriers of height h along the circumference without issue. In addition, we already established earlier that we can stack as many nested layers of regions as we need to ensure that the base of the innermost region is fully covered, so this will no longer be a consideration (even though we will need more than 4 layers for subsequent solutions, as our circles have become smaller).

Therefore the only limiting factor left is the speed at which we can build sideways, in the event that the tiger exits the side of our layered barrier, which we will call our "chasing speed". Our strategy will now be to use the $R - h \approx R$ seconds the tiger takes to approach the perimeter as time to uniformly distribute circles around the perimeter, so that $\frac{r}{\pi}$ of the circumference is covered by the time the tiger reaches the boundary. What this effectively does is boost our chasing speed by a factor of $\frac{\pi}{\pi - r}$, since for every $\pi - r$ circles along our wall, we now have a pre-placed circle that we no longer need to place ourselves.

As stated, however, this construction has an issue, namely that it is never efficient to have overlapping circles. In other words, if our goal is to eventually cover our perimeter with $N \approx \frac{\pi R}{r}$ tangent circles, we should only ever place circles at whole number positions along this ring of N circles. Thus, pick an arbitrary circle along this outer ring to be circle 1, and number these circles in order going clockwise as 1, 2, 3, etc.

Now, define a *uniform covering* of density d to be a sequence of integers a_i, where each term is either 0 if we don't want to place a circle at position i, and 1 if we do, satisfying $a_1 = 1$ and $a_{n+1} = 1$ if $\sum_{i=1}^{n} a_i \leq n \cdot d$ and 0 otherwise. For instance, if $d = \frac{1}{3}$, our sequence will look like

$$1, 0, 0, 1, 0, 0, 1, 0, 0, \ldots$$

which is simply placing down one out of every 3 circles, while if $d = \frac{2}{5}$, we would get the sequence

$$1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, \ldots$$

which has $\frac{2}{5}$ of the circles covered. Now, we will demonstrate that these uniform coverings have the desired properties we are looking for.

Lemma 1. *For any $x < y$, the sum $a_{x+1} + a_{x+2} + \cdots + a_y$ is at least $(y - x)d - 2$.*

Proof. Let $f(n)$ be the sum of the first n terms of a_i. We can show that $|f(n) - n \cdot d| < 1$, as $f(n) - n \cdot d$ can increase by at most $1 - d$ each step, and if we ever have $f(n-1) > (n-1) \cdot d$, we must have $a_n = 0$, so this difference will never be more than $1 - d$ (and hence 1) in the positive direction. In the negative direction, note that $f(n) - n \cdot d$ can only decrease by at most d each step, and if we ever have $f(n-1) < (n-1) \cdot d$, $a_n = 1$, so again this difference will never exceed d (and hence 1) in the negative direction. Thus $f(x)$ is within 1 of $x \cdot d$, and $f(y)$ is within 1 of $y \cdot d$, so $f(y) - f(x)$ is within 2 of $(y - x)d$, proving the lemma. $\square$

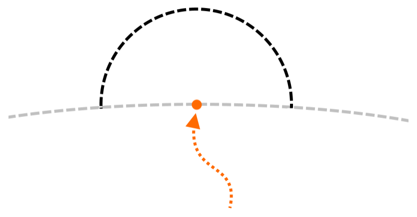
What this tells us is, if we start chasing the tiger down a uniform covering of density d, starting at an arbitrary point, the worst case scenario is that at some point we are 2 circles behind where we expect to be. We can easily resolve this by simply reserving the first 4 moves upon entering a layered barrier to be padding the 2 side exits with 2 extra circles, so that when the chase begins, we have a head start of 2 circles. Of course, this will ever so slightly impact the density of how many circles we can cover along the bottom of our layered barrier, but again, since we can have arbitrarily many stacked layers, this minor loss does not affect anything.

Another point worth briefly mentioning is the fact that, in the event that the tiger chases around the entirety of the circumference at a distance of h from the perimeter (the furthest possible distance that our strategy will still engage in a chase with), it will have gained a relative advantage of $2\pi h$ (the difference between the length of it's path and the circumference of the outer circle). However, if we ensure that our chasing speed s is strictly greater than the tiger's speed of 1, then for large enough R, this constant gain will be irrelevant compared to the $2\pi R(s-1)$ gain we will have acquired by the end of the chase.

It now remains to compute the smallest r for which this construction is feasible. Our speed buff of $\frac{\pi}{\pi-r}$ must bring us above the speed of the tiger, or 1, and since our initial wall building speed is simply (slightly less than) $2r$, we get $\frac{2\pi r}{\pi-r} > 1 \implies 2\pi r > \pi - r \implies r(2\pi+1) > \pi \implies r > \frac{\pi}{2\pi+1}$. We can translate this into any k value less than $2 + \frac{1}{\pi} > 2.3$, thus this strategy will allow us to trap a tiger of speed 2.3.

Remark: One shortcut we can use to compute the appropriate r value for our above solution is to note that, in the event of the tiger moving directly to the perimeter and then proceeding to chase around the entirety of the circumference, it will have traveled a distance of $R(2\pi+1)$ in the time that we covered a barrier of length $2\pi R$, so each of our circles must cover a distance of at least $\frac{2\pi}{2\pi+1}$ along the wall, meaning it's radius must be at least half that length, or $\frac{\pi}{2\pi+1}$. We will use this shortcut in the following solutions.

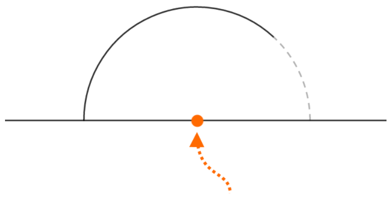
Solution for $r = \frac{\pi}{2\pi+2} + \varepsilon$ $(k = 2.6)$ (85 pts). The main limiting factor of our previous optimization was that we had to uniformly cover an entire circumference. However, we can improve this by only ever needing to uniformly cover semicircles. First, again define a large outer circle with radius R. However, this time, we will not do anything at all until the tiger exactly hits the boundary of the outer circle (if this never occurs, we have trapped the tiger by definition). When this occurs, we will construct a semicircle (plus an arbitrarily small circular arc) of radius $R' \ll R$ centered at the tiger, facing away from our large circle:



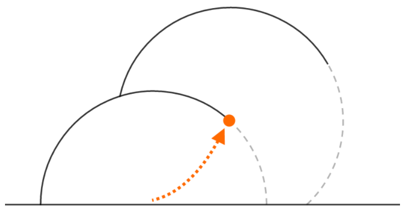
We will now repeat our previous construction of a uniform covering + layered barriers, except only on this semicircle; if at any point the tiger exits the semicircle and goes back into the original circle, we simply wait until the tiger exits again, and repeat this strategy indefinitely.

Thus we simply have to compute the smallest radius on which this strategy will work, which, by the previously mentioned shortcut, will be $r > \frac{\pi}{2\pi+2}$, as our longest possible wall is of length $\pi R'$ along the circumference, and it will take the tiger $R'(1+\pi)$ seconds to perform this chase. Once again, we can translate this into any k value less than $2 + \frac{2}{\pi} > 2.6$, so $k = 2.6$ is attainable.

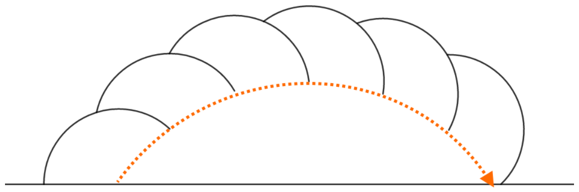
Solution for $r = \frac{\pi}{2\pi+3} + \varepsilon$ ($k = 2.9$) (90 pts). We will push the previous solution to it's logical limit by only relying on uniformly covering arcs of length $120 + \varepsilon$ degrees, which will give us $r > \frac{\pi}{2\pi+3}$ and $k = 2.9$. Once again, we will construct a large outer circle of radius R, and do nothing until the tiger hits this perimeter. At this point, we will construct an arc of θ degrees that has radius $R' \ll R$ and is centered at the tiger, and we will use the same strategies of uniform covering and layered barriers from before:



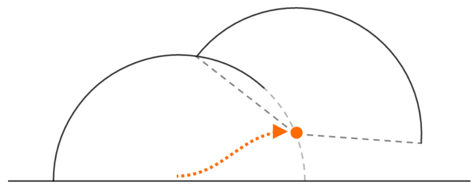
Now, the only possible way the tiger can try to escape this area is by hitting the missing section of our arc (the dotted portion above). In this case, if the tiger intersects this arc at the highest possible point, we will simply construct another arc of the same size on top of the previous arc, centered at the tiger:



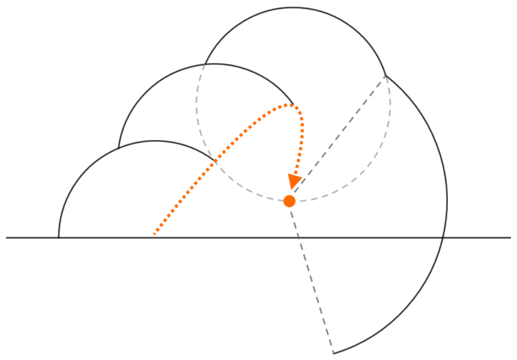
For $\theta > 120$, this arc will be tilted more downwards than the previous arc, so if we repeat this iteratively, eventually our chain of arcs will hit the perimeter of the original circle again, at which point the tiger has no chance of escaping in this direction:



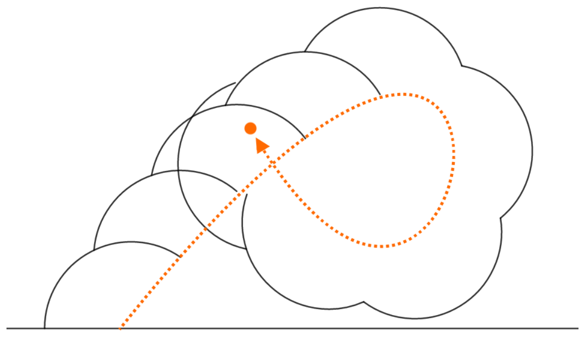
What happens if the tiger doesn't pick the highest possible point every time? Well, if the point of intersection the tiger chooses is within 60 degrees of the highest possible point, we can just construct an arc centered at the tiger and touching the previous arc in the usual way, and here the angular loss will be even greater than if the tiger had chosen the highest possible point, so we will only end up hitting the ground again faster:



And if the tiger tries some more unexpected strategies of choosing points even further away from the highest possible point, we may simply construct a larger arc, centered at the tiger and with one endpoint at this highest point, and here the angular loss is even more extreme:

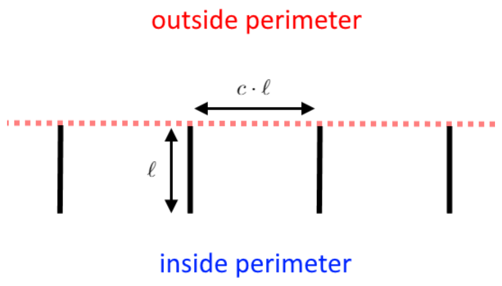


It is clear that these faster angular losses only serve to make our trap work even quicker, so there is no case where the tiger can escape our boundary here. Indeed, if the tiger causes the angular loss to drop to the point of exceeding 180 degrees in magnitude, it will still be accomplishing nothing besides boxing itself in:



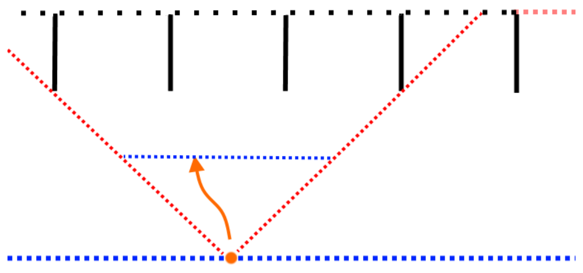
Thus we can conclude, using the same calculations as before, that this strategy gives us $k = 2.9$ as desired.

Solution for $r = \frac{1}{4} + \varepsilon$ ($k = 3.9$) (100 pts). You will notice that the previous 2 solutions, though attaining better bounds, did not take advantage of the fact that we have R idle seconds spent waiting for the tiger to approach the boundary. We will take a different approach, and utilize a recursive construction that allows us to make use of this initial time, and achieve a better bound. Our general setup will be a "sectioned wall", which we will define to be a bunch of "short" vertical walls of length ℓ along the circumference of a large outer circle, equally separated by slightly longer gaps of length $c \cdot \ell$ for some $c > 1$.

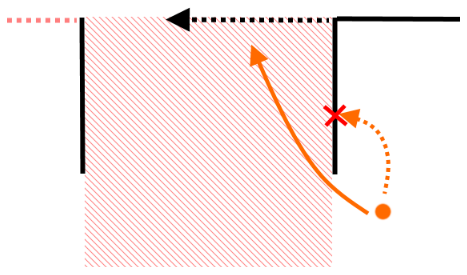


Here, a "wall" is simply our uniformly covered layered barrier setup from before, but with two completely solid walls of height h on the very ends of it, to prevent a "chase" from leaving the active area of the wall through any side but the top. Since the length to width ratio of our wall can be arbitrarily large, these two ends will be negligible for our total calculations.

This sectioned wall will behave similarly to our layered barrier setup, in that, when the tiger approaches within a set distance from the perimeter, we will preemptively start constructing horizontal walls (using the same construction as the vertical walls) to cover up each section:



We can easily adapt our old layering protocol to ensure that this will prevent the tiger from ever breaching the perimeter via travelling directly into it, so once again, the only consideration left is the "chasing" phase of our strategy, when the tiger horizontally crosses into a vertical "section" that has not yet been covered:



In this case, the tiger is unable to enter this section through the vertical walls (by definition), so we can guarantee it is a distance of at least ℓ from the top of the section. Therefore, if we are able to construct a wall of length $c \cdot \ell$ in ℓ seconds, we can block off this entire section before the tiger is able to reach the top of it, again preventing escape. During this time, if the tiger decides to cross through the vertical wall back into the previous section, we ignore it and don't follow the protocol of the vertical wall (until we finish the horizontal wall), since the previous section is by definition already covered, and hence not a risk.

Now, the useful property of this construction is the fact that we were able to create a large "sectioned wall" of length $N \cdot c \cdot \ell$ (where we use N many sections) using N vertical walls of length ℓ. In other words, this sectioned wall is c times as efficient at covering distances than the vertical walls we built it out of. Thus, we can repeatedly use this strategy, by building a sectioned wall out of sectioned walls, and so on, and each time the construction will become c times as efficient, so there is no limit as to how efficient our construction can be, meaning that we can easily exceed the efficiency of 2π units per second required to surround a circle with this construction, if we repeat this enough times.

Hence the only limiting factor on our construction is the guarantee that $c > 1$, so we will now find when this is possible. Observe that a uniformly covered layered barrier of density d can be pre-built at a speed of $\frac{2r}{d}$ units per second, and will work if our individual circles can cover distance $1 - d$ times as fast as the tiger, i.e. $2r = 1 - d$ (plus an arbitrarily small value), or $d = 1 - 2r$, so we can cover at most $\frac{2r}{d} = \frac{2r}{1-2r} = \frac{1}{1-2r} - 1$ distance per second (on average), meaning the necessary c value for our sectioned wall construction is also at most $\frac{1}{1-2r} - 1$ for the initial layer. Thus $c > 1$ only if we have $\frac{1}{1-2r} - 1 > 1$, or $1 - 2r < \frac{1}{2}$, meaning $r > \frac{1}{4}$ is required to build our first layer. All subsequent layers will have walls that can be built c times faster, so if the first layer is possible, all subsequent layers are possible automatically, hence our construction works for any $r > \frac{1}{4}$, corresponding to any $k < 4$, allowing us to attain $k = 3.9$ as desired.

Remark: When restricted to a square grid, this problem is similar to Conway's *Angels and Demons* problem, where it was shown that it is in fact impossible to trap a king of speed 2 (approximately resembling $k = 4$ in our setup, though there are some key differences) in this paper:

András Máthé, *The angel of power 2 wins*, Combin. Probab. Comput. 16(3):363-374, 2007

The above solution for $k = 3.9$ was inspired by another paper published on the subject:

Kutz, M. and P´or, A. (2005) *Angel, Devil, and King.* Computing and combinatorics, Lecture Notes in Comput. Sci. 3595 925–934. Springer, Berlin.

If you found this problem interesting, and would like to read further, these are definitely worth checking out!