GCPC 2022
Presentation of solutions

Jury and Testers

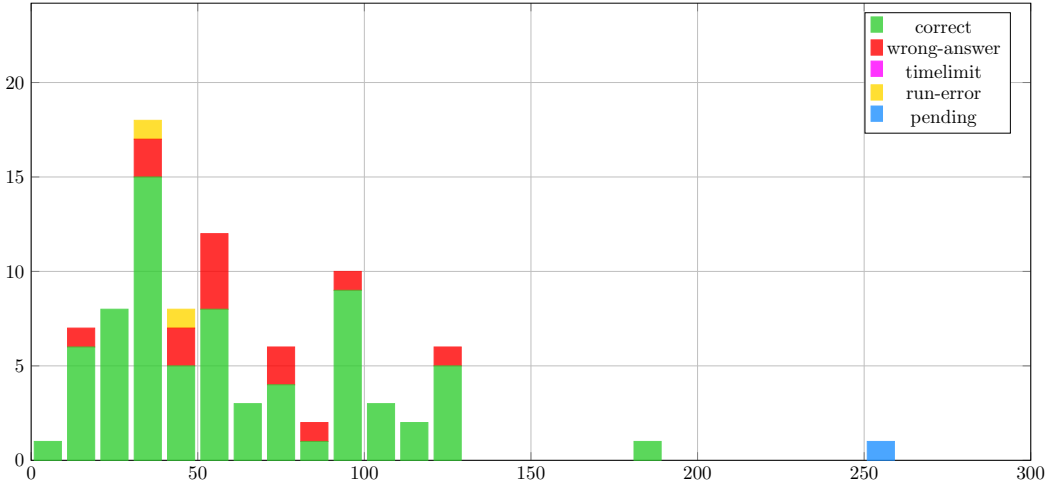
Thanks to the jury:

- Lucas Alber (KIT)
- Julian Baldus (Saarbrücken)
- Paul Jungeblut (KIT)
- Felicia Lucke (CPUlm)
- Nathan Maier (CPUlm)
- Jannik Olbrich (CPUlm)
- Michael Ruderer (CPUlm)
- Marcel Wienöbst (Lübeck)
- Paul Wild (FAU)
- Wendy Yi (KIT)
- Michael Zündorf (KIT)

Thanks to our test readers:

- Andreas Grigorjew (Helsinki)
- Hans Spath

K – K.O. Kids


correct
wrong-answer
timelimit
run-error
pending
0
5
10
15
20
0
50
100
150
200
250
300

K – K.O. Kids

Problem

k players walk over a bridge of length n, where each step consists of one real and one fake plate. The i-th player knows the real plate for all steps entered by players $1, \ldots, i-1$. All players switch sides at each step, the first player begins with the left plate. How many players make it over the bridge for a given layout s?

K – K.O. Kids

Problem

k players walk over a bridge of length n, where each step consists of one real and one fake plate. The i-th player knows the real plate for all steps entered by players $1, \ldots, i-1$. All players switch sides at each step, the first player begins with the left plate. How many players make it over the bridge for a given layout s?

Solution

- Game can be simulated naively in $O(n \cdot k)$ and also in $O(n + k)$ by starting at the last known step. This is fast enough.

K – K.O. Kids

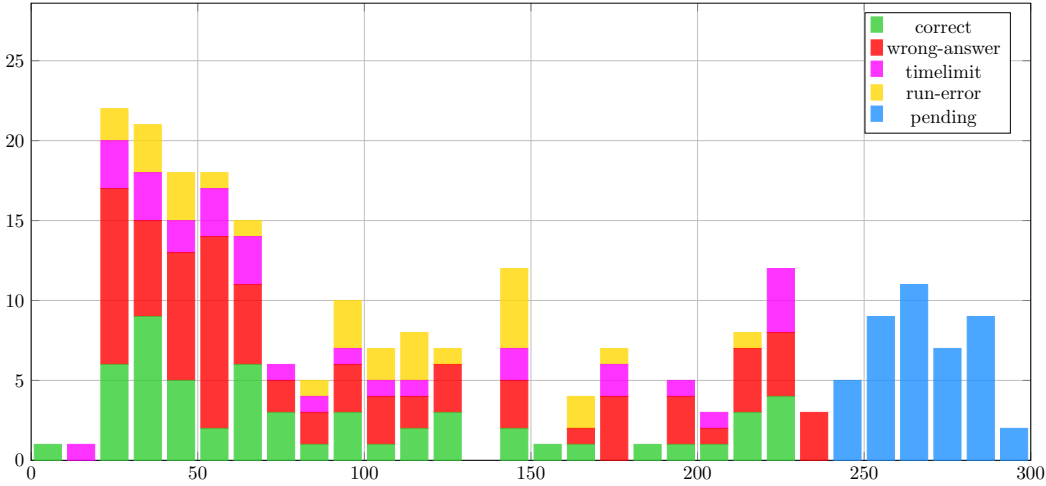
Problem

k players walk over a bridge of length n, where each step consists of one real and one fake plate. The i-th player knows the real plate for all steps entered by players $1, \ldots, i-1$. All players switch sides at each step, the first player begins with the left plate. How many players make it over the bridge for a given layout s?

Solution

- Game can be simulated naively in $O(n \cdot k)$ and also in $O(n + k)$ by starting at the last known step. This is fast enough.
- An $O(n)$ solution is possible as well:
  - Add an imaginary zeroth step $s[0] = \mathtt{R}$ to handle the first player starting on the left.
  - Let d be the number of steps with $s[i] = s[i-1]$ for $i = 1, \ldots, n$.
  - The solution is $\max(k - d, 0)$.

E – Enjoyable Entree



E – Enjoyable Entree

Problem

A canteen has a soup on offer whose recipe changes every day:

- On each of the first two days, the soup has some given type.
- On each subsequent day, the soup is an equal mix of the two previous days' soups.

Find the percentages of the first two days' soups that can be found in the nth soup.

E – Enjoyable Entree

Problem

A canteen has a soup on offer whose recipe changes every day:

- On each of the first two days, the soup has some given type.
- On each subsequent day, the soup is an equal mix of the two previous days' soups.

Find the percentages of the first two days' soups that can be found in the nth soup.

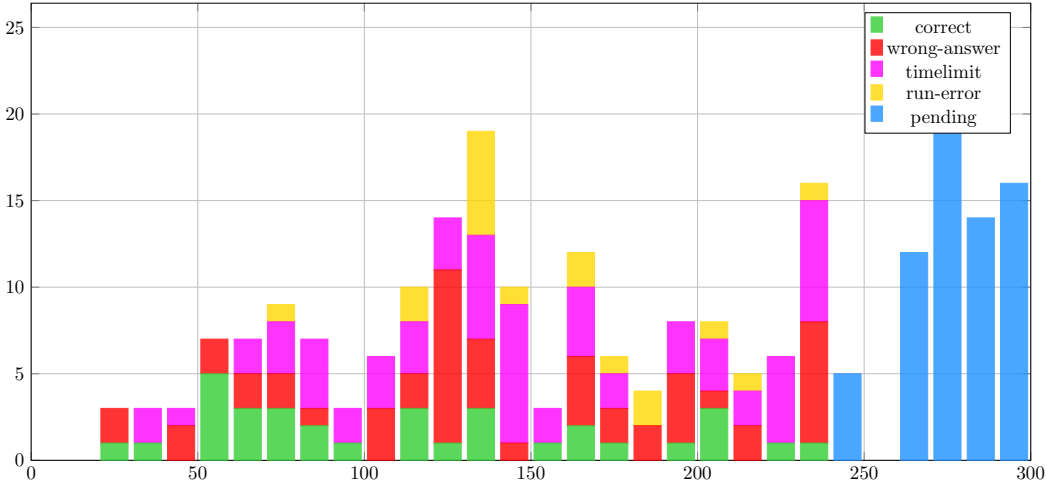
Solution

- We need to solve the following recurrence:

$$v_1 = (1, 0) \qquad v_2 = (0, 1) \qquad v_n = \tfrac{1}{2}(v_{n-1} + v_{n-2}) \qquad (n \geq 3)$$

- The number n may be very large, so that simple iteration will be too slow.
- However, the sequence quickly converges to its limit, $(\frac{1}{3}, \frac{2}{3})$.
- If we stop after 100 steps, the answer will have the required precision of 10^{-6}.

C – Chaotic Construction



C – Chaotic Construction

Problem

On a circular street where some positions are blocked by roadwork, find out whether there is a path between given pairs (a, b) of positions.

C – Chaotic Construction

Problem

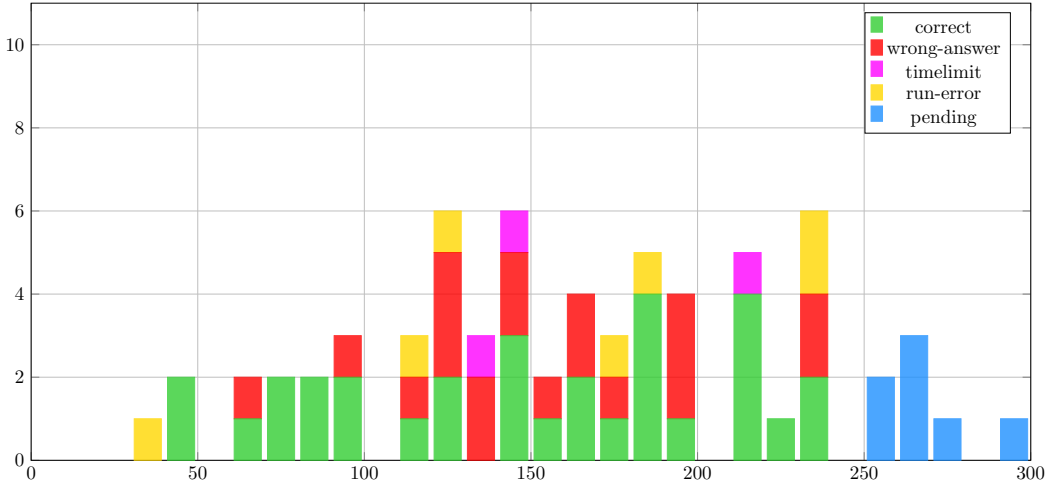
On a circular street where some positions are blocked by roadwork, find out whether there is a path between given pairs (a, b) of positions.

Solution

- Store blocked positions in a balanced binary search tree / segment tree.
- Use the tree to determine in $O(\log n)$ whether there is a blocked road ...
  - ... between a and b
  - ... at a position $\geq b$ or $\leq a$

If both conditions are met, the result is `impossible`, otherwise it is `possible`.

I – Improving IT



I – Improving IT

Problem

Given a list of CPU prices over n months and their resale values for the following m months, find the lowest cost to operate the system over n months.

I – Improving IT

Problem

Given a list of CPU prices over n months and their resale values for the following m months, find the lowest cost to operate the system over n months.

Solution

- The problem can be transformed into the shortest path problem:

I – Improving IT

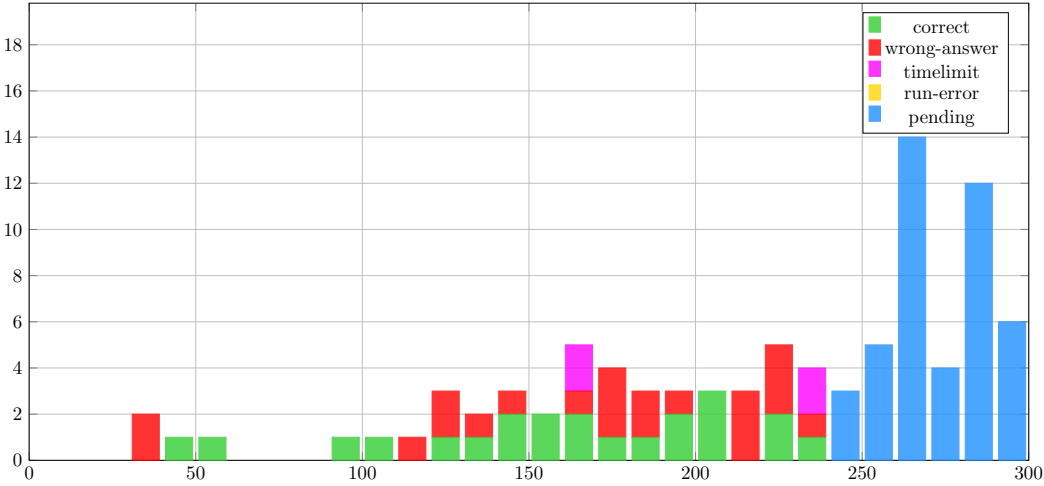
Problem

Given a list of CPU prices over n months and their resale values for the following m months, find the lowest cost to operate the system over n months.

Solution

- The problem can be transformed into the shortest path problem:
- Build a graph with nodes $1, \ldots, n+1$ and edges $(i, i+j)$ with weights $d_{i,j}$, where $d_{i,j}$ is the cost for buying a CPU in month i and keeping it for j months.
- Compute the shortest path from node 1 to $n+1$ by visiting the nodes in order $1, \ldots, n+1$.

D – Diabolic Doofenshmirtz



D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x.
However, the t values must be strictly increasing.

D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x. However, the t values must be strictly increasing.

Solution

Try to find t such that $x \leq t < 2 \cdot x$. The answer is then simply $t - (t \bmod x)$.

D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x.
However, the t values must be strictly increasing.

Solution

Try to find t such that $x \leq t < 2 \cdot x$. The answer is then simply $t - (t \bmod x)$.

How to find the right t:

- Since x could be 1, we can only try $t = 1$ as the first query.

D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x.
However, the t values must be strictly increasing.

Solution

Try to find t such that $x \leq t < 2 \cdot x$. The answer is then simply $t - (t \bmod x)$.

How to find the right t:

- Since x could be 1, we can only try $t = 1$ as the first query.
- If this t is less than x, we know that x is at least 2.
- Thus, the largest possible t we can now try is $t = 3$.

D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x.
However, the t values must be strictly increasing.

Solution

Try to find t such that $x \leq t < 2 \cdot x$. The answer is then simply $t - (t \bmod x)$.

How to find the right t:

- Since x could be 1, we can only try $t = 1$ as the first query.
- If this t is less than x, we know that x is at least 2.
- Thus, the largest possible t we can now try is $t = 3$.
- If this t is less than x, we know that x is at least 4.
- Thus, the largest possible t we can now try is $t = 7$.

D – Diabolic Doofenshmirtz

Problem

You can query t and the judge responds with $t \bmod x$. Your goal is to find x.
However, the t values must be strictly increasing.

Solution

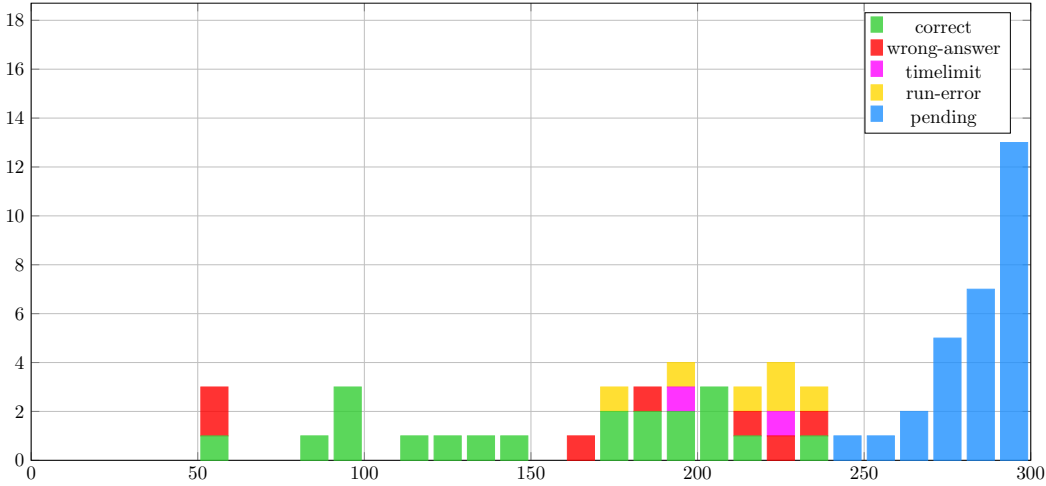
Try to find t such that $x \leq t < 2 \cdot x$. The answer is then simply $t - (t \bmod x)$.

How to find the right t:

- Since x could be 1, we can only try $t = 1$ as the first query.
- If this t is less than x, we know that x is at least 2.
- Thus, the largest possible t we can now try is $t = 3$.
- If this t is less than x, we know that x is at least 4.
- Thus, the largest possible t we can now try is $t = 7$.

$\Rightarrow$ Try $t = 2^i - 1$ until $t \bmod x \neq t$. Then answer with $t - (t \bmod x)$.

H – Hardcore Hangman



H – Hardcore Hangman

Problem

Hangman with the possibility to query multiple letters at once. Queries are answered with the set of all indices containing one of the queried letters. Find the hidden word with at most 7 queries.

Solution

- First, find the length of the hidden word by querying all letters.

H – Hardcore Hangman

Problem

Hangman with the possibility to query multiple letters at once. Queries are answered with the set of all indices containing one of the queried letters. Find the hidden word with at most 7 queries.

Solution

- First, find the length of the hidden word by querying all letters.
- Afterwards, ensure that it is possible to uniquely identify the positions of each letter by assigning each (there are 26) a distinct binary string of length 5. The i-th bit indicates whether the letter should be included in the i-th query.

H – Hardcore Hangman

Problem

Hangman with the possibility to query multiple letters at once. Queries are answered with the set of all indices containing one of the queried letters. Find the hidden word with at most 7 queries.

Solution

- First, find the length of the hidden word by querying all letters.
- Afterwards, ensure that it is possible to uniquely identify the positions of each letter by assigning each (there are 26) a distinct binary string of length 5. The i-th bit indicates whether the letter should be included in the i-th query.
- To reconstruct the word, find, for each letter ℓ, the indices which are in the answers of the queries with ℓ included and not in the other ones.

H – Hardcore Hangman

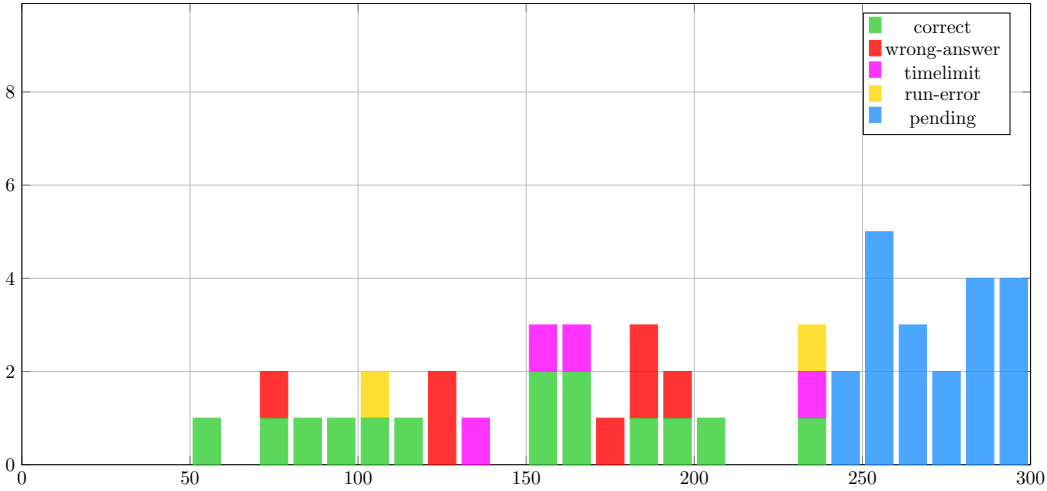
Problem

Hangman with the possibility to query multiple letters at once. Queries are answered with the set of all indices containing one of the queried letters. Find the hidden word with at most 7 queries.

Solution

- First, find the length of the hidden word by querying all letters.
- Afterwards, ensure that it is possible to uniquely identify the positions of each letter by assigning each (there are 26) a distinct binary string of length 5. The i-th bit indicates whether the letter should be included in the i-th query.
- To reconstruct the word, find, for each letter ℓ, the indices which are in the answers of the queries with ℓ included and not in the other ones.
- Runtime is $O(n)$ for a hidden word of length n and the strategy uses exactly 7 queries. (A solution with 6 queries exists as well.)

L – Lots of Land



L – Lots of Land

Problem

Given integers h, w, n ($1 \leq h, w \leq 100$, $1 \leq n \leq 26$), find a way to split an $h \times w$ rectangle into n rectangular parts of equal area and with integer side lengths.

L – Lots of Land

Problem

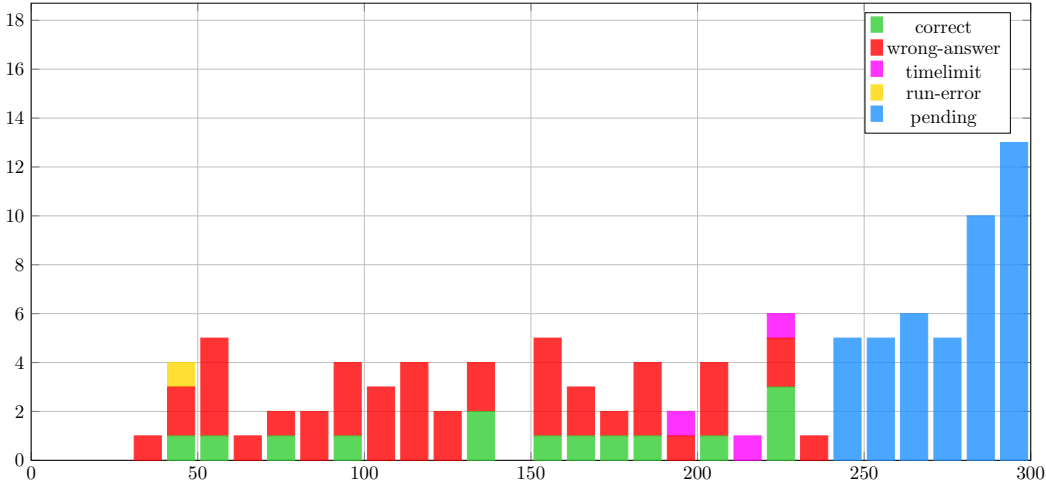
Given integers h, w, n ($1 \leq h, w \leq 100$, $1 \leq n \leq 26$), find a way to split an $h \times w$ rectangle into n rectangular parts of equal area and with integer side lengths.

Solution

- If $h \cdot w$ is not divisible by n, no solution exists, so output `impossible`.
- Otherwise, it's always possible to split the large rectangle into a *grid* of rectangles.
- Can divide into $a \times b$ rectangles $\iff$ a divides h, b divides w and $n \cdot a \cdot b = h \cdot w$.
- Once a and b are found, many possible ways to construct the answer, e.g.:

```
answer[i][j] = 'A' + (i/a)*(w/b) + (j/b)
```

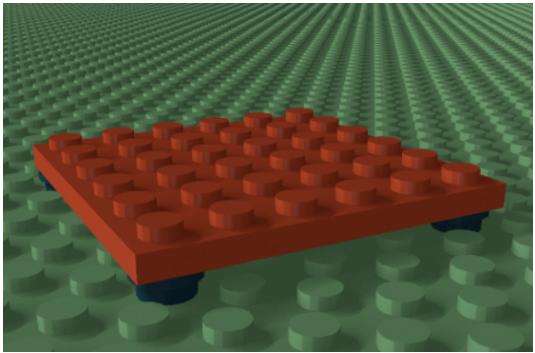
A – Alternative Architecture



A – Alternative Architecture

Problem

How many ways are there to place a Lego plate of dimensions $a \times b$ on an infinite base plate, if smaller 1×1 plates placed in the corners are used as buffer?



A – Alternative Architecture

Problem

How many ways are there to place a Lego plate of dimensions $a \times b$ on an infinite base plate, if smaller 1×1 plates placed in the corners are used as buffer?

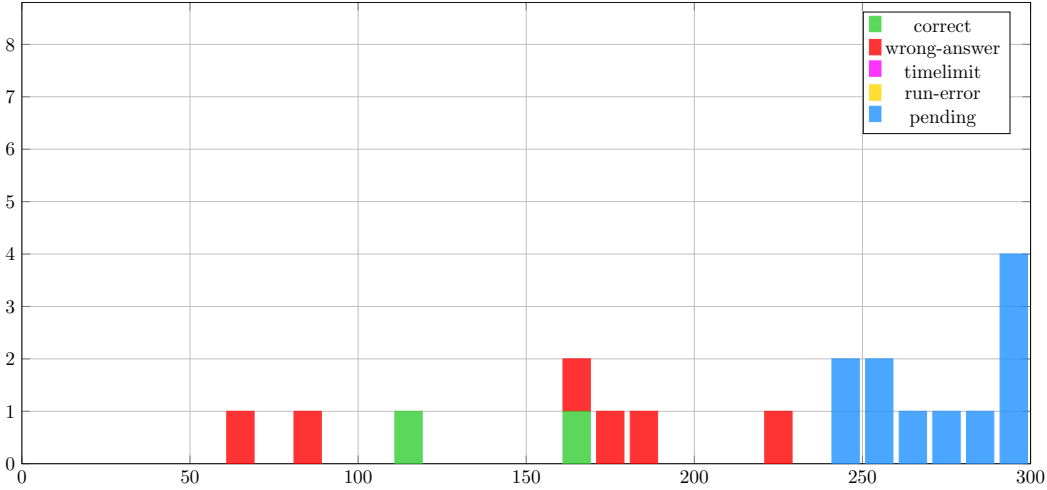
Equivalent Problem

Find the number of ways to draw an $(a-1) \times (b-1)$ rectangle in the plane so that all of its corners have integer coordinates.

Solution

- Place one of the corners at $(0, 0)$.
- The next corner must be at some point (x, y) with $x^2 + y^2 = (a-1)^2$.
- Find all candidate points, e.g. by first looping over all x and then solving for y.
- Compute the coordinates of the two other corners and check if they are integral.
- Make sure to avoid double counting if the rectangle is a square ($a = b$).

J – Jesting Jabberwocky



J – Jesting Jabberwocky

Problem

Given a sequence of cards, how many cards need to be inserted at some other position to obtain a sequence where the cards are grouped by suit?

J – Jesting Jabberwocky

Problem

Given a sequence of cards, how many cards need to be inserted at some other position to obtain a sequence where the cards are grouped by suit?

Solution

- Observe that since we can reinsert a card at any position, we only have to count how many cards need to be removed.

J – Jesting Jabberwocky

Problem

Given a sequence of cards, how many cards need to be inserted at some other position to obtain a sequence where the cards are grouped by suit?

Solution

- Observe that since we can reinsert a card at any position, we only have to count how many cards need to be removed.
- For a fixed order of suits, we can calculate the minimum number of cards to be removed using dynamic programming.

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.
- If a card i is already in the correct group, i.e. of suit s, we can leave it there.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`, `dp[i - 1][s - 1]`$\}$

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.
- If a card i is already in the correct group, i.e. of suit s, we can leave it there.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$,$ `dp[i - 1][s - 1]`$\}$
- If a card i is not in the correct group, i.e. not of suit s, we have to remove it.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$+ 1,$ `dp[i - 1][s - 1]`$\}$

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.
- If a card i is already in the correct group, i.e. of suit s, we can leave it there.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$,$ `dp[i - 1][s - 1]`$\}$
- If a card i is not in the correct group, i.e. not of suit s, we have to remove it.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$+ 1,$ `dp[i - 1][s - 1]`$\}$

There are $k!$ possibilities to arrange k suits. We can try all.

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.
- If a card i is already in the correct group, i.e. of suit s, we can leave it there.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$,$ `dp[i - 1][s - 1]`$\}$
- If a card i is not in the correct group, i.e. not of suit s, we have to remove it.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$+ 1,$ `dp[i - 1][s - 1]`$\}$

There are $k!$ possibilities to arrange k suits. We can try all.

Running time: $\mathcal{O}(k! \cdot n \cdot k)$, which is fast enough for $k = 4$.

J – Jesting Jabberwocky

Solution

Dynamic programming for a fixed order of suits:

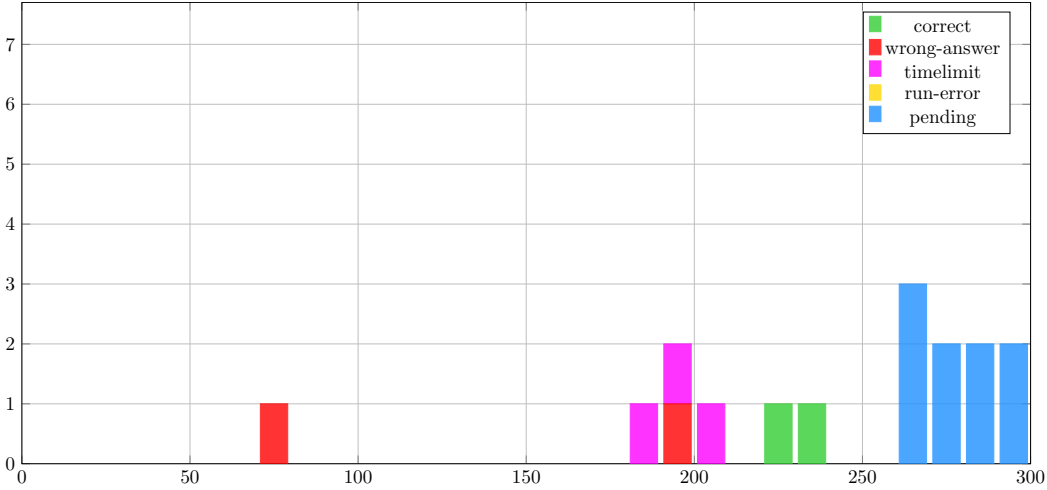
- Let `dp[i][s]` be the minimum number of cards to remove such that the remaining of the first i cards are ordered correctly and the last card is of suit s.
- If a card i is already in the correct group, i.e. of suit s, we can leave it there.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$,$ `dp[i - 1][s - 1]`$\}$
- If a card i is not in the correct group, i.e. not of suit s, we have to remove it.
  $\Rightarrow$ `dp[i][s]` $= \min\{$`dp[i - 1][s]`$+ 1,$ `dp[i - 1][s - 1]`$\}$

There are $k!$ possibilities to arrange k suits. We can try all.

Running time: $\mathcal{O}(k! \cdot n \cdot k)$, which is fast enough for $k = 4$.

This problem can also be solved in $\mathcal{O}(k \cdot 2^k \cdot n)$.

B – Breeding Bugs



B – Breeding Bugs

Problem

Given n positive integers, how many can you select such that no two selected integers sum up to a prime.

B – Breeding Bugs

Problem

Given n positive integers, how many can you select such that no two selected integers sum up to a prime.

Solution

First, observe that we can select at most one 1 since $1 + 1 = 2$ and 2 is prime.

B – Breeding Bugs

Problem

Given n positive integers, how many can you select such that no two selected integers sum up to a prime.

Solution

First, observe that we can select at most one 1 since $1 + 1 = 2$ and 2 is prime.

Let's rephrase the problem in terms of graph theory: Given a graph with n vertices where two vertices are connected iff their sum is prime, find a maximum independent set.

B – Breeding Bugs

Problem

Given n positive integers, how many can you select such that no two selected integers sum up to a prime.

Solution

First, observe that we can select at most one 1 since $1 + 1 = 2$ and 2 is prime.

Let's rephrase the problem in terms of graph theory: Given a graph with n vertices where two vertices are connected iff their sum is prime, find a maximum independent set.

Now observe the following properties:

- No two vertices sum up to 2.
- All primes that some vertices sum up to are odd.
- The sum of two numbers is only odd if we sum up an even and an odd number.

B – Breeding Bugs

Problem

Given n positive integers, how many can you select such that no two selected integers sum up to a prime.

Solution

First, observe that we can select at most one 1 since $1 + 1 = 2$ and 2 is prime.

Let's rephrase the problem in terms of graph theory: Given a graph with n vertices where two vertices are connected iff their sum is prime, find a maximum independent set.

Now observe the following properties:

- No two vertices sum up to 2.
- All primes that some vertices sum up to are odd.
- The sum of two numbers is only odd if we sum up an even and an odd number.

$\Rightarrow$ The given graph is *bipartite*.

B – Breeding Bugs

Solution

- The complement of a maximum independent set is a minimum vertex cover.
- For bipartite graphs, the minimum vertex cover and the matching have the same size.

B – Breeding Bugs

Solution

- The complement of a maximum independent set is a minimum vertex cover.
- For bipartite graphs, the minimum vertex cover and the matching have the same size.

$\Rightarrow$ The answer is $n - |\text{matching}|$.

B – Breeding Bugs

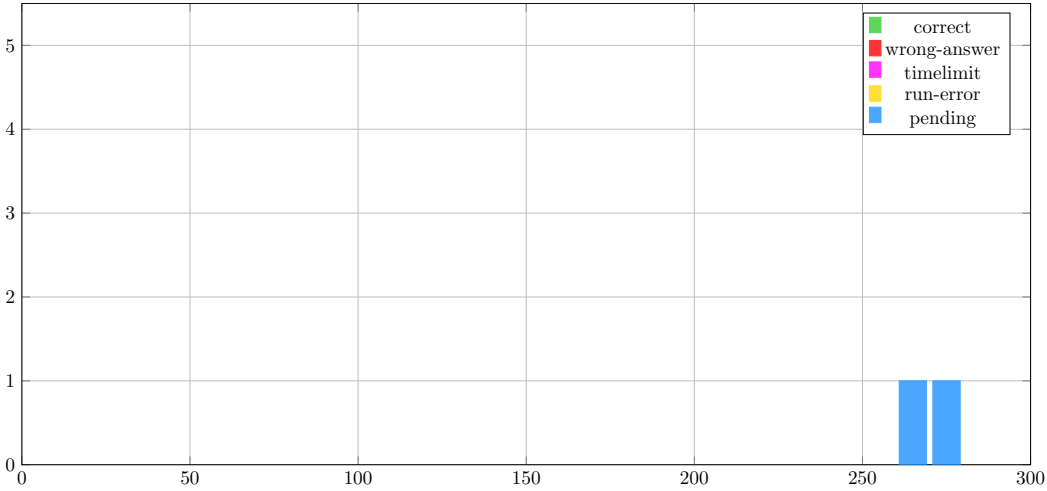
Solution

- The complement of a maximum independent set is a minimum vertex cover.
- For bipartite graphs, the minimum vertex cover and the matching have the same size.

$\Rightarrow$ The answer is $n - |\text{matching}|$.

- A maximum matching can be found in many ways:
  - Any flow algorithm
  - Kuhn's algorithm
  - Hopcroft Karp's algorithm

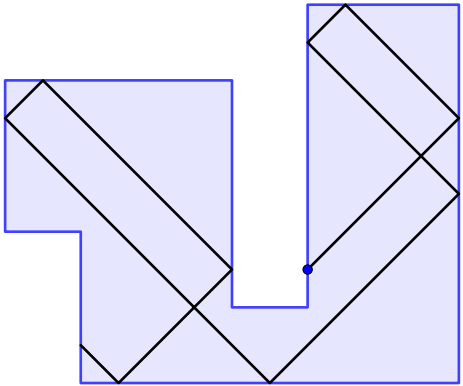
M – Mirror Madness



M – Mirror Madness

Problem

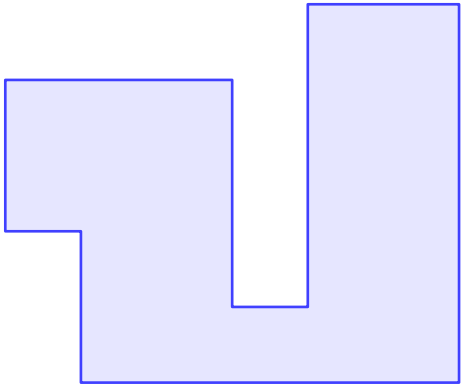
Simulate the path of a laser beam inside an axis-aligned polygon.



M – Mirror Madness

Solution

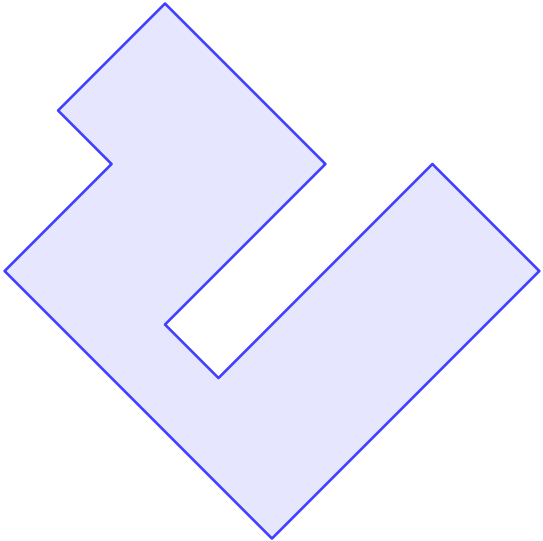
- Simple ray tracing will be too slow, so we need to be more clever.



M – Mirror Madness

Solution

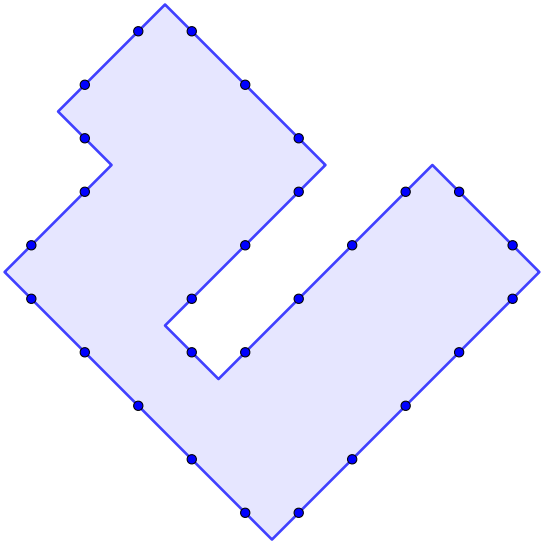
- Simple ray tracing will be too slow, so we need to be more clever.
- Rotate the whole figure by 45°.



M – Mirror Madness

Solution

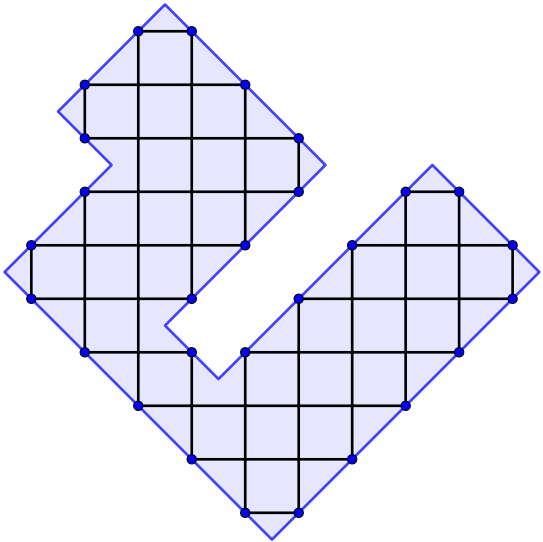
- Simple ray tracing will be too slow, so we need to be more clever.
- Rotate the whole figure by $45°$.
- There are at most 10^6 possible bounce locations.



M – Mirror Madness

Solution

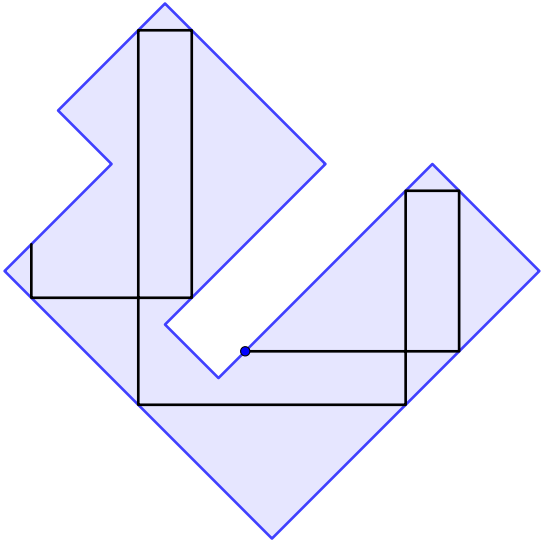
- Simple ray tracing will be too slow, so we need to be more clever.
- Rotate the whole figure by 45°.
- There are at most 10^6 possible bounce locations.
- Connect them up by forming pairs within each row and column.



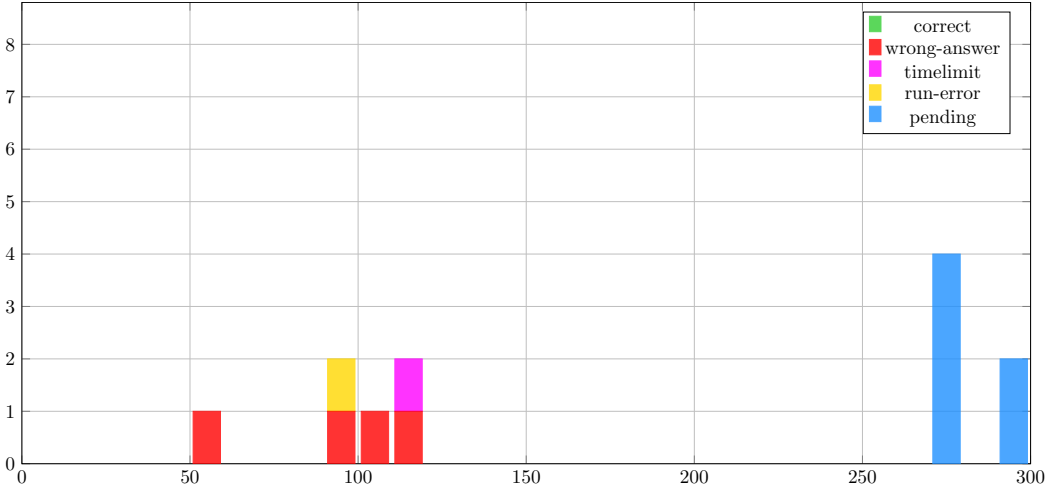
M – Mirror Madness

Solution

- Simple ray tracing will be too slow, so we need to be more clever.
- Rotate the whole figure by 45°.
- There are at most 10^6 possible bounce locations.
- Connect them up by forming pairs within each row and column.
- Construct the answer step by step by following the pointers between pairs.



G – Guessing Game



G – Guessing Game

Problem

- There is a pool of yes-or-no events (the examinations).
- We are given a list of guesses. There are n guesses each containing predictions of seven distinct events. Alan guesses the outcome of two distinct events.
- Is there an outcome of the events where Alan's guess contains more *correct* predictions than any other guess?

G – Guessing Game

Problem

Is there an outcome of the events where Alan's guess contains more *correct* predictions than any other guess?

Solution

- If there is such an outcome, there is one where all of Alan's predictions come true
- Then it suffices to ensure that no other guess contains more than one correct prediction

G – Guessing Game

Problem

Is there an outcome of the events where Alan's guess contains more *correct* predictions than any other guess?

Solution

- If there is such an outcome, there is one where all of Alan's predictions come true
- Then it suffices to ensure that no other guess contains more than one correct prediction
- We encode these two conditions into a 2SAT formula

G – Guessing Game

Problem

Is there an outcome of the events where Alan's guess contains more *correct* predictions than any other guess?

Solution

- Each event has two possible results.
  $\Rightarrow$ Boolean variable X_e for all events e
- Prediction P for event e can be expressed as either X_e or $\neg X_e$
- "Alan predictions [P_1 and P_2] must both be correct" $\Rightarrow P_1 \wedge P_2$
- "A following submission [consisting of $P'_1, \ldots, P'_7$] must not contain two correct predictions"

$$\Rightarrow \bigwedge_{1 \leq i < j \leq 7} (\neg P'_i \vee \neg P'_j)$$

G – Guessing Game

Problem

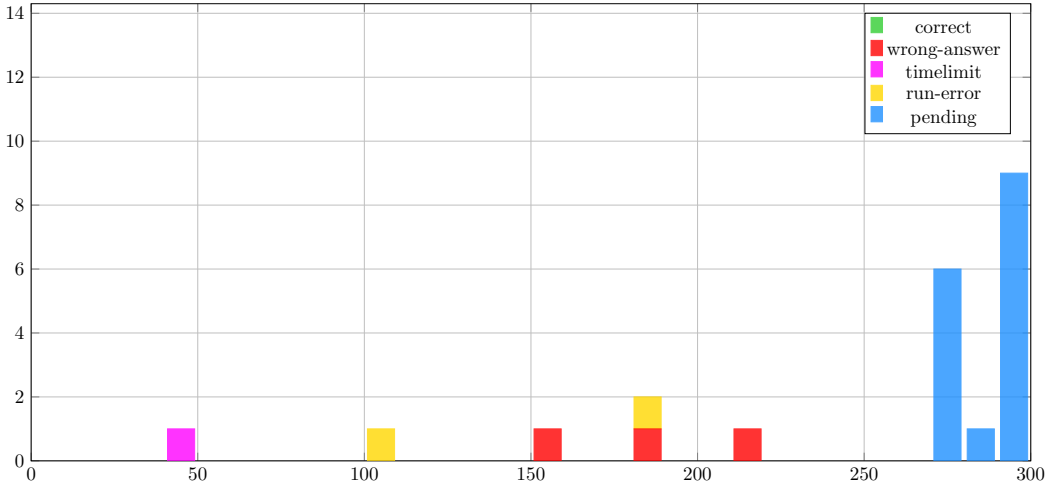
Is there an outcome of the events where Alan's guess contains more *correct* predictions than any other guess?

Solution

$$\Rightarrow \quad P_1 \wedge P_2 \wedge \bigwedge_{k=1}^{n} \bigwedge_{i=1}^{7} \bigwedge_{j=i+1}^{7} \left(\neg P_i^{(k)} \vee \neg P_j^{(k)} \right)$$

- Alan can win if and only if this formula is satisfiable
- Use your favourite 2SAT algorithm to check this
- Running time: $\mathcal{O}(n)$

F – Formula Flatland



F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

Solution

- First, we find a combinatorial embedding.
- Simply sort the coordinates of the neighbors of a vertex in ccw order.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

Solution

- First, we find a combinatorial embedding.
- Simply sort the coordinates of the neighbors of a vertex in ccw order.
- We can now traverse the faces of the graph.
- We start with an edge, take the head vertex of the edge, find the next edge in cyclic order and continue.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

Solution

- First, we find a combinatorial embedding.
- Simply sort the coordinates of the neighbors of a vertex in ccw order.
- We can now traverse the faces of the graph.
- We start with an edge, take the head vertex of the edge, find the next edge in cyclic order and continue.
- The smallest face we see is the answer.
- Runtime: $\mathcal{O}(n \log(n))$ for sorting and $\mathcal{O}(n)$ to enumerate all faces.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

Solution

- First, we find a combinatorial embedding.
- Simply sort the coordinates of the neighbors of a vertex in ccw order.
- We can now traverse the faces of the graph.
- We start with an edge, take the head vertex of the edge, find the next edge in cyclic order and continue.
- The smallest face we see is the answer.
- Runtime: $\mathcal{O}(n \log(n))$ for sorting and $\mathcal{O}(n)$ to enumerate all faces.
- Verdict: Wrong Answer

F – Formula Flatland

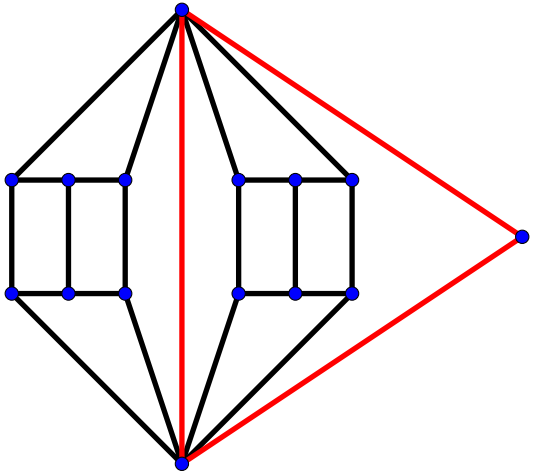
Problem

Given a planar graph with minimum degree at least 3, find the length of the smallest circle. This value is also called the *girth* of a graph.

Solution

- First, we find a combinatorial embedding.
- Simply sort the coordinates of the neighbors of a vertex in ccw order.
- We can now traverse the faces of the graph.
- We start with an edge, take the head vertex of the edge, find the next edge in cyclic order and continue.
- The smallest face we see is the answer.
- Runtime: $\mathcal{O}(n \log(n))$ for sorting and $\mathcal{O}(n)$ to enumerate all faces.
- Verdict: Wrong Answer?!

F – Formula Flatland



F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution

- Orient the edges such that each vertex has outdegree ≤ 5 and the graph is a DAG.
- Can be done by recursively removing a vertex with degree ≤ 5.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution

- Orient the edges such that each vertex has outdegree ≤ 5 and the graph is a DAG.
- Can be done by recursively removing a vertex with degree ≤ 5.
- Observe that the girth is either 3, 4 or 5.

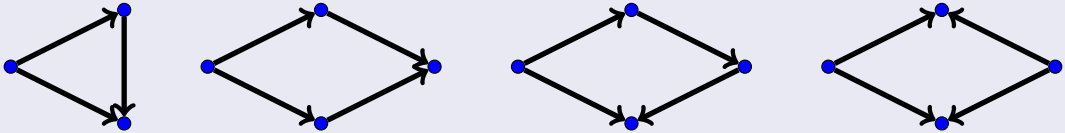
$\Rightarrow$ We only need to check if a cycle of length 3 or 4 exists.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution

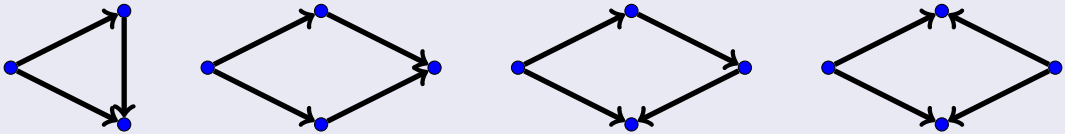


F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution



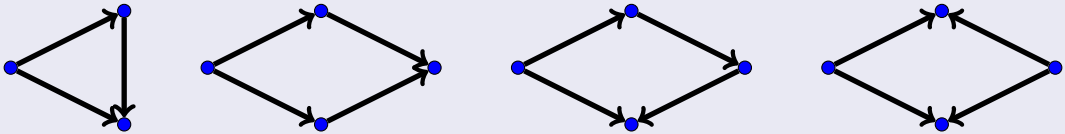
- The first three cases can be solved with a limited BFS from each vertex.
- We go until depth 3, so the runtime is $\mathcal{O}(5^3 n)$.

F – Formula Flatland

Problem

Given a planar graph with minimum degree at least 3, find the smallest circle. This property is also called the *girth* of a graph.

Real Solution



- The fourth case can be solved by enumerating all pairs of outgoing edges and checking for duplicates.
- We have limited degree, so the runtime is $\mathcal{O}(5^2 n \cdot \log(n))$.