GCPC 2025 Presentation of Solutions

German Collegiate Programming Contest 2025

The GCPC Jury

August 9, 2025

GCPC 2025 Jury

- **Andreas Grigorjew**
  Paris Dauphine FR, CPUlm
- **Niko Hastrich**
  Saarland University
- **Yvonne Kothmeier**
- **Felicia Lucke**
  Durham University UK, CPUlm
- **Niklas Mohrin**
  Hasso-Plattner-Institut Potsdam
- **Jannik Olbrich**
  Ulm University, CPUlm
- **Lucas Schwebler**
  Karlsruhe Institute of Technology
- **Christopher Weyand**
  MOIA GmbH, CPUlm
- **Paul Wild**
  Friedrich-Alexander University
  Erlangen-Nürnberg, CPUlm
- **Wendy Yi**
  Karlsruhe Institute of Technology, CPUlm
- **Yidi Zang**
  Karlsruhe Institute of Technology
- **Michael Zündorf**
  Karlsruhe Institute of Technology, CPUlm

GCPC 2025 Test Solvers

- **Jonathan Dransfeld**
  Karlsruhe Institute of Technology
- **Paula Marten**
  Hasso-Plattner-Institute Potsdam
- **Michael Ruderer**
  Augsburg University, CPUlm
- **Franz Sauerwald**
  Hasso-Plattner-Institute Potsdam

GCPC 2025 Technical Team

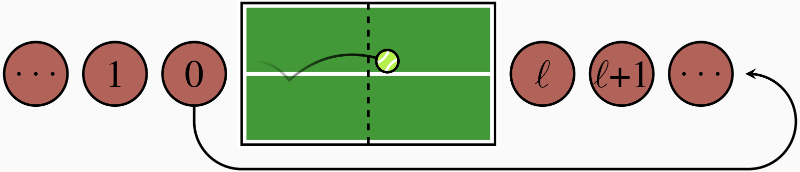
- **Nathan Maier**
  CPUlm
- **Alexander Schmid**
  CPUlm
- **Pascal Weber**
  University of Vienna, CPUlm

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf

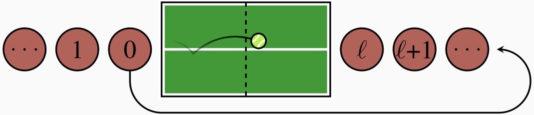
Problem

Given ℓ players on the left of a table and r players on the right, how many different pairs face each other during a game of around-the-table?



A: Around the Table

Problem author: Wendy Yi, Michael Zündorf

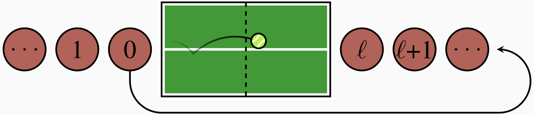


Observations

- Player i on the left may face player $i+\ell$ and player $i+\ell-1 \pmod{n}$
- Player i on the right may face player $i+r$ and player $i+r+1 \pmod{n}$

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf

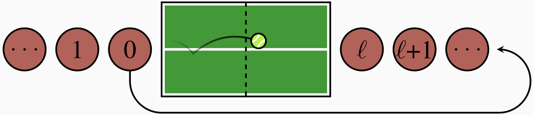


Observations

- Player i on the left may face player $i+\ell$ and player $i+\ell-1 \pmod n$
- Player i on the right may face player $i+r$ and player $i+r+1 \pmod n$
- Each player faces ≤ 4 different players $\implies \leq 2n$ pairs in total with n players

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf

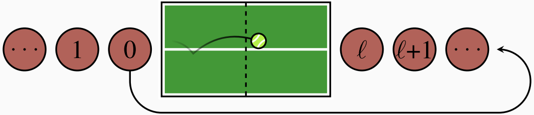


Observations

- Player i on the left may face player $i + \ell$ and player $i + \ell - 1 \pmod{n}$
- Player i on the right may face player $i + r$ and player $i + r + 1 \pmod{n}$
- Each player faces ≤ 4 different players $\implies \leq 2n$ pairs in total with n players
- Some of the four indices may be equal $\implies$ fewer opponents per player in this case

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf



Observations

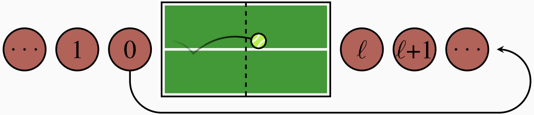
- Player i on the left may face player $i + \ell$ and player $i + \ell - 1 \pmod n$
- Player i on the right may face player $i + r$ and player $i + r + 1 \pmod n$
- Each player faces ≤ 4 different players $\implies \leq 2n$ pairs in total with n players
- Some of the four indices may be equal $\implies$ fewer opponents per player in this case

Solution

- If $r = \ell - 1$, then each player faces two different players $\implies$ n pairs

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf



Observations

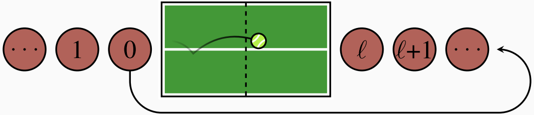
- Player i on the left may face player $i+\ell$ and player $i+\ell-1 \pmod n$
- Player i on the right may face player $i+r$ and player $i+r+1 \pmod n$
- Each player faces ≤ 4 different players $\implies \leq 2n$ pairs in total with n players
- Some of the four indices may be equal $\implies$ fewer opponents per player in this case

Solution

- If $r = \ell - 1$, then each player faces two different players $\implies n$ pairs
- If $r = \ell$ or $r = \ell - 2$, then each player faces three different players $\implies n + \frac{n}{2}$ pairs

A: Around the Table

Problem author: Wendy Yi, Michael Zündorf



Observations

- Player i on the left may face player $i+\ell$ and player $i+\ell-1 \pmod n$
- Player i on the right may face player $i+r$ and player $i+r+1 \pmod n$
- Each player faces ≤ 4 different players $\implies \leq 2n$ pairs in total with n players
- Some of the four indices may be equal $\implies$ fewer opponents per player in this case

Solution

- If $r=\ell-1$, then each player faces two different players $\implies n$ pairs
- If $r=\ell$ or $r=\ell-2$, then each player faces three different players $\implies n+\frac{n}{2}$ pairs
- Else $2n$ pairs

B: Bustling Busride

Problem author: Niko Hastrich

Problem

Partition people queuing for a bus into contiguous segments minimizing the latest arrival time.

All people on the bus need to leave and possibly reenter at every stop (taking some time).

B: Bustling Busride

Problem author: Niko Hastrich

Problem

Partition people queuing for a bus into contiguous segments minimizing the latest arrival time.
All people on the bus need to leave and possibly reenter at every stop (taking some time).

Solution

- Binary Search the answer a^*.

B: Bustling Busride

Problem author: Niko Hastrich

Problem

Partition people queuing for a bus into contiguous segments minimizing the latest arrival time.
All people on the bus need to leave and possibly reenter at every stop (taking some time).

Solution

- Binary Search the answer a^*.
- For each tested a, put as many people into one bus B as possible while not arriving after time a.

B: Bustling Busride

Problem author: Niko Hastrich

Problem

Partition people queuing for a bus into contiguous segments minimizing the latest arrival time.
All people on the bus need to leave and possibly reenter at every stop (taking some time).

Solution

- Binary Search the answer a^*.
- For each tested a, put as many people into one bus B as possible while not arriving after time a.
- Final arrival time of B is

$$\mathrm{Start}(B) + \mathrm{MaxDist}(B) + \sum_{s \in \mathrm{Stops}(B)} w \cdot (\#\mathrm{PeopleLeavingAt}(s) + \#\mathrm{PeopleEnteringAt}(s))$$

B: Bustling Busride

Problem author: Niko Hastrich

Problem

Partition people queuing for a bus into contiguous segments minimizing the latest arrival time.
All people on the bus need to leave and possibly reenter at every stop (taking some time).

Solution

- Binary Search the answer a^*.
- For each tested a, put as many people into one bus B as possible while not arriving after time a.
- Final arrival time of B is

$$\begin{aligned} &\mathrm{Start}(B) + \mathrm{MaxDist}(B) + \sum_{s \in \mathrm{Stops}(B)} w \cdot (\#\mathrm{PeopleLeavingAt}(s) + \#\mathrm{PeopleEnteringAt}(s)) \\ = &\mathrm{Start}(B) + \mathrm{MaxDist}(B) + \sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p))). \end{aligned}$$

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:

Solution (cont.)

- Need to maintain $\sum_{p \in \text{PeopleRiding}(B)} 2w(1 + \#\text{ApproachedStopsBefore}(\text{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\text{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\text{ApproachedStopsBefore}(\text{Dest}(p)))$

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$
- Can be maintained in $O(\log n)$ with (basically) any tree data structure.

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$
- Can be maintained in $O(\log n)$ with (basically) any tree data structure.
- Watch out to not TLE when many buses are needed.

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$
- Can be maintained in $O(\log n)$ with (basically) any tree data structure.
- Watch out to not TLE when many buses are needed.
  - You need might need to rollback the updates to your data structure instead of building it fresh.

Solution (cont.)

- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$
- Can be maintained in $O(\log n)$ with (basically) any tree data structure.
- Watch out to not TLE when many buses are needed.
  - You need might need to rollback the updates to your data structure instead of building it fresh.
- Running time for testing one particular a: $O(n \log n)$

Solution (cont.)

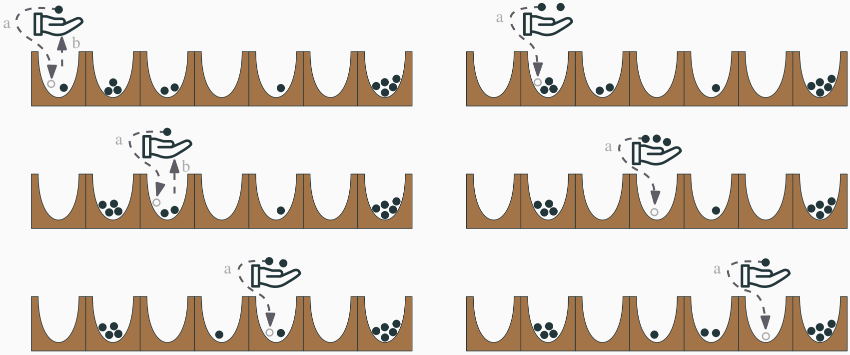
- Need to maintain $\sum_{p \in \mathrm{PeopleRiding}(B)} 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$ for each new person p on the bus
- Two cases:
  - $\mathrm{Dest}(p)$ is already driven to:
    Change in final arrival time $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)))$
  - $\mathrm{Dest}(p)$ is *not* already driven to:
    Change in final arrival time
    $= 2w(1 + \#\mathrm{ApproachedStopsBefore}(\mathrm{Dest}(p)) + \#\mathrm{PeopleLeavingAfter}(\mathrm{Dest}(p)))$
- Can be maintained in $O(\log n)$ with (basically) any tree data structure.
- Watch out to not TLE when many buses are needed.
  - You need might need to rollback the updates to your data structure instead of building it fresh.
- Running time for testing one particular a: $O(n \log n)$
- Running time for solving the whole problem: $O(n \log n \log a^*)$

C: Congklak

Problem author: Lucas Schwebler

Problem

There is a game board with n holes, initially hole i contains a_i stones. Perform the following "game" exactly t times: Drop one stone into hole 1 and simulate according to the rules of Congklak. How many stones are in each hole after playing t of those games?



C: Congklak

Problem author: Lucas Schwebler

Observation

0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	1	0	0	0
3	1	1	1	0	0	0
4	0	2	0	1	1	0

- Look at the process with $a_i = 0$.

C: Congklak

Problem author: Lucas Schwebler

Observation

0	**0**	0	**0**	0	**0**	0
1	**1**	0	**0**	0	**0**	0
2	**0**	1	**1**	0	**0**	0
3	**1**	1	**1**	0	**0**	0
4	**0**	2	**0**	1	**1**	0

- Look at the process with $a_i = 0$.
- Odd indices are counting upwards in binary! (least significant bit is at hole 1)

C: Congklak

Problem author: Lucas Schwebler

Observation

0	**0**	0	**0**	0	**0**	0
1	**1**	0	**0**	0	**0**	0
2	**0**	1	**1**	0	**0**	0
3	**1**	1	**1**	0	**0**	0
4	**0**	2	**0**	1	**1**	0

- Look at the process with $a_i = 0$.
- Odd indices are counting upwards in binary! (least significant bit is at hole 1)
- Even indices contain the number of overflows of the digits.

C: Congklak

Problem author: Lucas Schwebler

Solution 1

- Suppose that holes $1, 3, \ldots, 2k - 1$ are empty.
- Then, the next $2^k - 1$ games are easy to simulate (binary counting).

C: Congklak

Problem author: Lucas Schwebler

Solution 1

- Suppose that holes $1, 3, \ldots, 2k - 1$ are empty.
- Then, the next $2^k - 1$ games are easy to simulate (binary counting).
- After that, simulate one game naively in $O(n)$.
- Then, holes $1, 3, \ldots, 2k + 1$ are empty; repeat as above with larger k.

C: Congklak

Problem author: Lucas Schwebler

Solution 1

- Suppose that holes $1, 3, \ldots, 2k - 1$ are empty.
- Then, the next $2^k - 1$ games are easy to simulate (binary counting).
- After that, simulate one game naively in $O(n)$.
- Then, holes $1, 3, \ldots, 2k + 1$ are empty; repeat as above with larger k.
- If k reaches $\log_2(t)$, we can simulate all remaining games with binary counting.
- Thus, we only need to simulate $O(\log(t))$ games naively.

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 8$

↓					
2	0	0	0	0	0

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 7$

0	1	1	1	0	0

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 3$

		↓			
1	4	1	1	0	0

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 2$

		↓			
1	4	0	2	1	0

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 1$

				↓	
1	4	0	3	1	0

C: Congklak

Problem author: Lucas Schwebler

Solution 2 (Easier to implement!)

- If hole 1 is not empty, simulate one game naively.
- So suppose that hole 1 is empty.
- Suppose that r games are remaining.
- Hole 1 will contain $r \bmod 2$ stones in the end.
- Hole 2 will contain $\left\lfloor \frac{r}{2} \right\rfloor$ additional stones.
- Repeat the same process starting from hole 3 with $\left\lfloor \frac{r}{2} \right\rfloor$ remaining games.
- Time complexity: $O(n \log(t))$

$r = 0$

				↓	
1	4	0	3	0	1

D: Demand for Cycling

Problem author: Jannik Olbrich

Problem

Given an axis-aligned polygon, find an enclosing axis-aligned polygon with minimum circumference

Solution

- Insight: An optimal solution has no two consecutive convex vertices
  $\implies$ Any optimal solution is *rectilinear convex*
- What's the circumference of such a polygon?

D: Demand for Cycling

Problem author: Jannik Olbrich

Problem

Given an axis-aligned polygon, find an enclosing axis-aligned polygon with minimum circumference

Solution

- Insight: An optimal solution has no two consecutive convex vertices
  $\implies$ Any optimal solution is *rectilinear convex*
- What's the circumference of such a polygon? $2(x_{max} - x_{min}) + 2(y_{max} - y_{min})$
- This is the same as the rectangle with corners $\langle x_{min}, y_{min} \rangle$ and $\langle x_{max}, y_{max} \rangle$
- Find x_{min}, x_{max}, y_{min} and y_{max}
- Time complexity: $\mathcal{O}(n)$

E: Engineering Excellence

Problem author: Yidi Zang

Problem

- Given a convex polygon with internal angles $\geq 90^\circ$.
- Move one point to maximize the perimeter.
- The polygon must stay convex (angles $\leq 180^\circ$) and all internal angles $\geq 90^\circ$.

E: Engineering Excellence

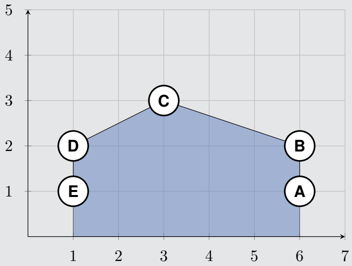
Problem author: Yidi Zang

Problem

- Given a convex polygon with internal angles $\geq 90°$.
- Move one point to maximize the perimeter.
- The polygon must stay convex (angles $\leq 180°$) and all internal angles $\geq 90°$.

Solution

- Try to move every point individually.



E: Engineering Excellence

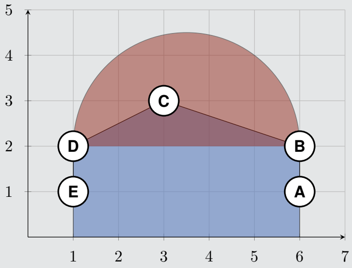
Problem author: Yidi Zang

Problem

- Given a convex polygon with internal angles $\geq 90^\circ$.
- Move one point to maximize the perimeter.
- The polygon must stay convex (angles $\leq 180^\circ$) and all internal angles $\geq 90^\circ$.

Solution

- Try to move every point individually.
- The angle $\angle BCD$ is in $[90^\circ, 180^\circ]$ if C stays within the Thales semicircle.



E: Engineering Excellence

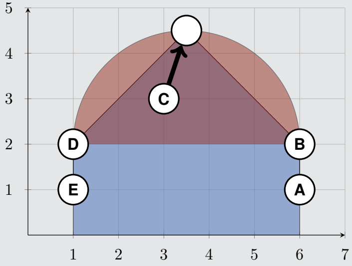
Problem author: Yidi Zang

Problem

- Given a convex polygon with internal angles $\geq 90^\circ$.
- Move one point to maximize the perimeter.
- The polygon must stay convex (angles $\leq 180^\circ$) and all internal angles $\geq 90^\circ$.

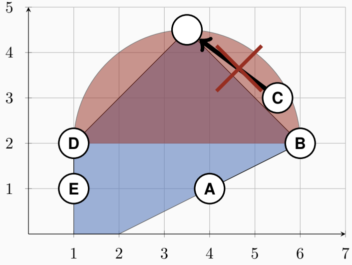
Solution

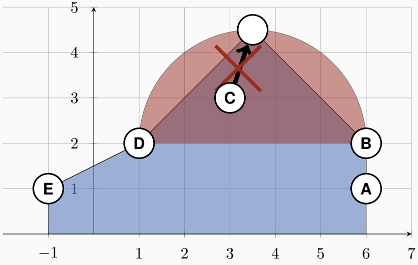
- Try to move every point individually.
- The angle $\angle BCD$ is in $[90^\circ, 180^\circ]$ if C stays within the Thales semicircle.
- Ideally, we move C to the middle top of the Thales semicircle.
- This maximizes the perimeter.



Solution

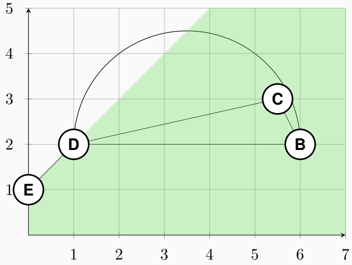
- Ideally, we move C to the middle top of the Thales semicircle.
- This maximizes the perimeter.
- However, this is not always possible because of the angles $\angle ABC$ and $\angle CDE$.





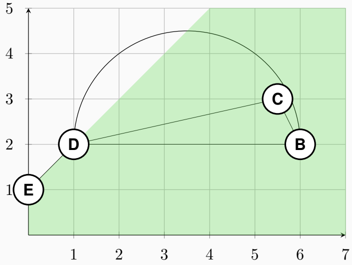
Solution

- The angle $\angle CDE$ is $\leq 180°$, if C stays in the green halfplane.



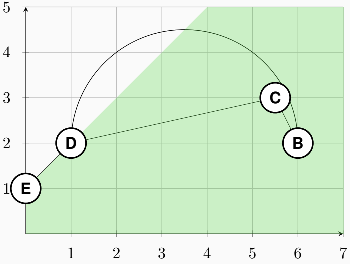
Solution

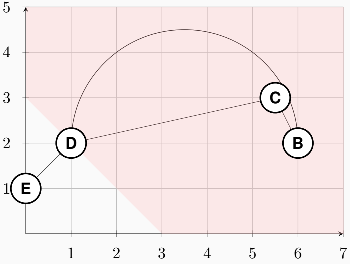
- The angle $\angle CDE$ is $\leq 180°$, if C stays in the green halfplane.
- By intuition or triangle inequality, the intersection point between halfplane line and Thales semicircle is optimal (or middle top of Thales circle).



Solution

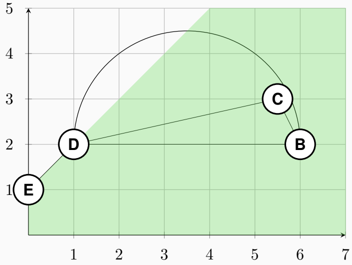
- The angle $\angle CDE$ is $\leq 180°$, if C stays in the green halfplane.
- By intuition or triangle inequality, the intersection point between halfplane line and Thales semicircle is optimal (or middle top of Thales circle).
- There is also a (pink) halfplane for $\angle CDE \geq 90°$ (and $\leq 270°$).

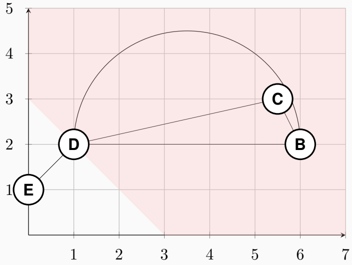




Solution

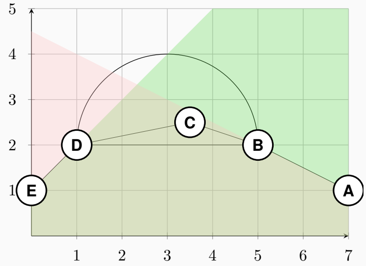
- The angle $\angle CDE$ is $\leq 180^\circ$, if C stays in the green halfplane.
- By intuition or triangle inequality, the intersection point between halfplane line and Thales semicircle is optimal (or middle top of Thales circle).
- There is also a (pink) halfplane for $\angle CDE \geq 90^\circ$ (and $\leq 270^\circ$).
- One of the two halfplanes completely contains the semicircle, ignore that one.





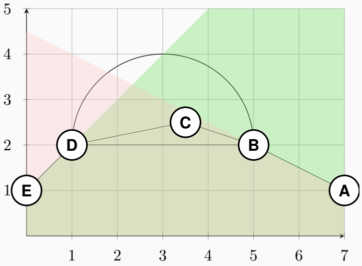
Solution

- Both angles $\angle ABC$ and $\angle CDE$ have one relevant halfplane.
- The halfplane lines might intersect in the semicircle.



Solution

- Both angles $\angle ABC$ and $\angle CDE$ have one relevant halfplane.
- The halfplane lines might intersect in the semicircle.
- Here, the intersection point of the lines is optimal.



Solution

- In total, there are four possible optimal points:

Solution

- In total, there are four possible optimal points:

1. Middle top of the Thales semicircle.
2. Intersection $\angle ABC$ halfplane and Thales semicircle.
3. Intersection $\angle CDE$ halfplane and Thales semicircle.
4. Intersection $\angle ABC$ and $\angle CDE$ halfplane lines.

Solution

- In total, there are four possible optimal points:

1. Middle top of the Thales semicircle.
2. Intersection $\angle ABC$ halfplane and Thales semicircle.
3. Intersection $\angle CDE$ halfplane and Thales semicircle.
4. Intersection $\angle ABC$ and $\angle CDE$ halfplane lines.

- Try all of these points and check whether they are valid.

Solution

- In total, there are four possible optimal points:

1. Middle top of the Thales semicircle.
2. Intersection $\angle ABC$ halfplane and Thales semicircle.
3. Intersection $\angle CDE$ halfplane and Thales semicircle.
4. Intersection $\angle ABC$ and $\angle CDE$ halfplane lines.

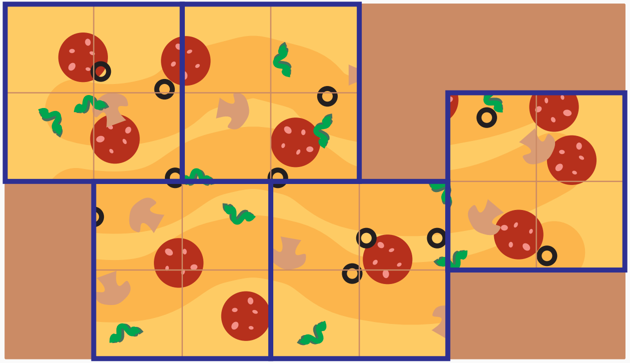
- Try all of these points and check whether they are valid.
- Take care of precision issues: Use `long double` and large ε (e.g. $\varepsilon = 10^{-7}$).
- Runtime: $\mathcal{O}(n)$.

F: Fair and Square

Problem author: Paul Wild

Problem

You are given a region consisting of some cells in a rectangular grid. Find the longest possible length such that the region can be divided into squares of that side length.



F: Fair and Square

Problem author: Paul Wild

Solution

- Testing whether a given length is possible can be done in $\mathcal{O}(h \cdot w)$:
  - Scan over the grid from top left to bottom right
  - Whenever an unmarked cell is encountered, it must be the top left corner of a square
  - Check if a square can be placed here and mark all cells belonging to it, then continue scanning
  - In the end, check if all cells of the region have been marked
  - The time does not depend on the size of the square!
- Testing all side lengths from 1 to $\min(h, w)$ is too slow for the given bounds
- *Key Improvement*: Only test side lengths k such that k^2 divides n, the number of cells
  - The worst case in the given bounds is $n = 2\,822\,400 = 2^8 \cdot 3^2 \cdot 5^2 \cdot 7^2$, which has 40 square divisors
- With this improvement, the solution is fast enough

G: Generating Cool Passwords Company

Problem author: Paul Wild

Problem

Generate a list of n passwords ($1 \leq n \leq 1000$).

1. Passwords must contain at least one each of `a-z`, `A-Z`, `0-9` and a special symbol (e.g. `!` or `#`)
2. Passwords must have pairwise edit distance at least 2
3. Passwords must be between 8 and 12 in length

G: Generating Cool Passwords Company

Problem author: Paul Wild

Problem

Generate a list of n passwords ($1 \leq n \leq 1000$).

1. Passwords must contain at least one each of `a-z`, `A-Z`, `0-9` and a special symbol (e.g. `!` or `#`)
2. Passwords must have pairwise edit distance at least 2
3. Passwords must be between 8 and 12 in length

Solution

- Here's one of the many approaches that work:

  `000GCPC!x000`, `001GCPC!x001`, `002GCPC!x002`, . . . , `999GCPC!x999`

  - The common part `GCPC!x` is used to satisfy rules 1 and 3
  - The two counters are used to satisfy rule 2
- Other approaches include using randomization, permutations, or the digits of π

H: Happy Hookup

Problem author: Andreas Grigorjew

Problem

- We are given a directed graph with n vertices and m edges, and two different vertices u and v.
- Find a vertex c, such that there is a path from u to c and a path from v to c. Or return that no such vertex exists.

H: Happy Hookup

Problem author: Andreas Grigorjew

Problem

- We are given a directed graph with n vertices and m edges, and two different vertices u and v.
- Find a vertex c, such that there is a path from u to c and a path from v to c. Or return that no such vertex exists.

Solution

- Implement a depth first search or a breadth first search.
- Call two searches: one starting from u, one starting from v.

H: Happy Hookup

Problem author: Andreas Grigorjew

Problem

- We are given a directed graph with n vertices and m edges, and two different vertices u and v.
- Find a vertex c, such that there is a path from u to c and a path from v to c. Or return that no such vertex exists.

Solution

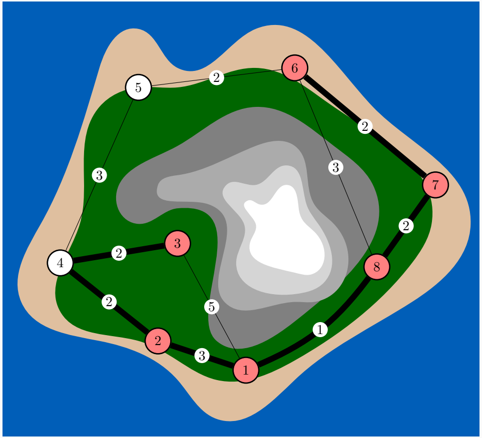
- Implement a depth first search or a breadth first search.
- Call two searches: one starting from u, one starting from v.
- Maintain a boolean array for both searches, indicating for every vertex whether it has been reached by the search.
- Output any vertex for which the entry in both arrays is true, or output "`no`" if no such vertex exists.
- Runtime: $O(n + m)$.

I: Island Urbanism

Problem author: Felicia Lucke, Jannik Olbrich, Paul Wild

Problem

Given a graph G consisting of a large cycle where some edges are replaced by an arbitrary connected graph (a village). Further, given terminal vertices in G such that every village contains at most 7 terminals. Find a Steiner Tree in G, that is, a subtree of G connecting all terminals.



I: Island Urbanism

Problem author: Felicia Lucke, Jannik Olbrich, Paul Wild

Problem

Given a graph G consisting of a large cycle where some edges are replaced by an arbitrary connected graph (a village). Further, given terminal vertices in G such that every village contains at most 7 terminals. Find a Steiner Tree in G, that is, a subtree of G connecting all terminals.

Solution

How do we solve this for few terminals?

- Dynamic Programming: Let $D(S, i)$ be the weight of the smallest tree connecting the terminals in S and vertex i:

$$
\begin{aligned}
D(\emptyset, i) &= 0 && \forall i \\
D(S, i) &\leq D(S \setminus \{i\}, i) && \text{if } i \text{ is a terminal} \\
D(S, i) &\leq D(S, j) + w && \text{if there is an edge } (i, j) \text{ with weight } w \\
D(S, i) &\leq D(A, i) + D(S \setminus A, i) && \forall A \subset S
\end{aligned}
$$

- Time Complexity: $\mathcal{O}(n \cdot 3^k + 2^k \cdot m \cdot \log n)$ for n vertices, m edges, and k terminals

I: Island Urbanism

Problem author: Felicia Lucke, Jannik Olbrich, Paul Wild

Problem

Given a graph G consisting of a large cycle where some edges are replaced by an arbitrary connected graph (a village). Further, given terminal vertices in G such that every village contains at most 7 terminals. Find a Steiner Tree in G, that is, a subtree of G connecting all terminals.

Solution

- The solution is a tree
  $\Longrightarrow$ We must cut the cycle somewhere
- Two cases:
  - Cut inside a village (s.t. the "leftmost" and "rightmost" vertices are disconnected within the village)
  - Cut between two villages ("leftmost" and "rightmost" vertices are connected within the villages)
- Treat "leftmost" and "rightmost" vertices of villages as terminals
  $\Longrightarrow$ cases can be computed with DP
- Just try cutting between any adjacent villages, and within each village!
- Take care with villages without terminals
- Time Complexity: $\mathcal{O}(n \cdot 3^7 + 2^7 \cdot m \cdot \log n)$

J: Jumbled Packets

Problem author: Yidi Zang

Problem

- This is a multi pass problem.
- You are given a binary string s of length n ($1 \leq n \leq 10^5$).
- Encode it into a ternary string of length n.
- After this string is cyclically rotated by some amount, restore the original string s.

J: Jumbled Packets

Problem author: Yidi Zang

Problem

- This is a multi pass problem.
- You are given a binary string s of length n ($1 \leq n \leq 10^5$).
- Encode it into a ternary string of length n.
- After this string is cyclically rotated by some amount, restore the original string s.

Encode

- If the string s consists of only '0', do nothing.

J: Jumbled Packets

Problem author: Yidi Zang

Problem

- This is a multi pass problem.
- You are given a binary string s of length n ($1 \leq n \leq 10^5$).
- Encode it into a ternary string of length n.
- After this string is cyclically rotated by some amount, restore the original string s.

Encode

- If the string s consists of only '0', do nothing.
- Otherwise, find the first '1'.
- Replace everything up to that '1' with '2'.
- For example, replace "00001011" with "22222011".

J: Jumbled Packets

Problem author: Yidi Zang

Encode

- If the string s consists of only '0', do nothing.
- Otherwise, find the first '1'.
- Replace everything up to that '1' with '2'.
- For example, replace "00001011" with "22222011".

Decode

- If the received ternary string consists of only '0', this is already decoded.

J: Jumbled Packets

Problem author: Yidi Zang

Encode

- If the string s consists of only ‘0’, do nothing.
- Otherwise, find the first ‘1’.
- Replace everything up to that ‘1’ with ‘2’.
- For example, replace “00001011” with “22222011”.

Decode

- If the received ternary string consists of only ‘0’, this is already decoded.
- Otherwise, find the substring of ‘2’s.
  - Careful, this might wrap over the end, e.g. “22011222”.

J: Jumbled Packets

Problem author: Yidi Zang

Encode

- If the string s consists of only '0', do nothing.
- Otherwise, find the first '1'.
- Replace everything up to that '1' with '2'.
- For example, replace "00001011" with "22222011".

Decode

- If the received ternary string consists of only '0', this is already decoded.
- Otherwise, find the substring of '2's.
  - Careful, this might wrap over the end, e.g. "22011222".
- Rotate this substring to the beginning, "22011222" $\rightarrow$ "22222011".

J: Jumbled Packets

Problem author: Yidi Zang

Encode

- If the string s consists of only '0', do nothing.
- Otherwise, find the first '1'.
- Replace everything up to that '1' with '2'.
- For example, replace "00001011" with "22222011".

Decode

- If the received ternary string consists of only '0', this is already decoded.
- Otherwise, find the substring of '2's.
  - Careful, this might wrap over the end, e.g. "22011222".
- Rotate this substring to the beginning, "22011222" $\rightarrow$ "22222011".
- Replace the last '2' with '1', all other '2's with '0', "22222011" $\rightarrow$ "00001011".

J: Jumbled Packets

Problem author: Yidi Zang

Encode

- If the string s consists of only '0', do nothing.
- Otherwise, find the first '1'.
- Replace everything up to that '1' with '2'.
- For example, replace "00001011" with "22222011".

Decode

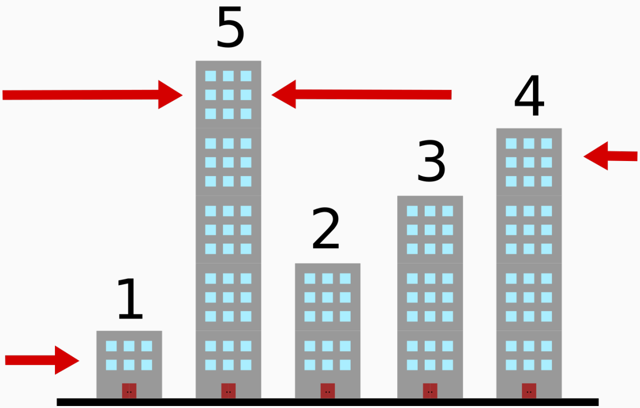
- If the received ternary string consists of only '0', this is already decoded.
- Otherwise, find the substring of '2's.
  - Careful, this might wrap over the end, e.g. "22011222".
- Rotate this substring to the beginning, "22011222" $\rightarrow$ "22222011".
- Replace the last '2' with '1', all other '2's with '0', "22222011" $\rightarrow$ "00001011".
- Encoding and decoding both take $\mathcal{O}(n)$.

K: Karlsruhe Skyline

Problem author: Paul Wild

Problem

Given integers n, a and b, find a permutation of building of heights 1 to n such that a buildings can be seen from the left and b buildings can be seen from the right, or say that none exists.

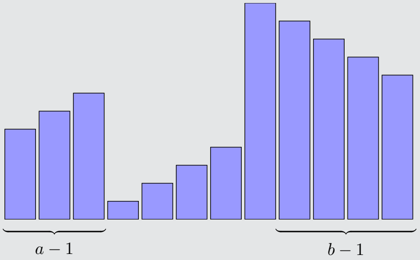


K: Karlsruhe Skyline

Problem author: Paul Wild

Solution

- There are two types of cases where no solution exists:
  - Only building n can be seen from both sides, so $a + b > n + 1$ is impossible
  - Building n must always be next to a 1 clue, so we cannot have $a = b = 1$
- The following setup can be used in the general case:



- If $a = 1$ or $b = 1$ you may need to reverse the middle part.

L: Labour Laws

Problem author: Yvonne Kothmeier

Problem

Given time t_w. Find the minimum time $0 \leq t_b$, such that

- $t_w - t_b \leq 60 * 6$ or
- $t_w - t_b \leq 60 * 9 \wedge t_b \geq 30$ or
- $t_w - t_b \leq 60 * 10 \wedge t_b \geq 45$

L: Labour Laws

Problem author: Yvonne Kothmeier

Problem

Given time t_w. Find the minimum time $0 \leq t_b$, such that

- $t_w - t_b \leq 60 * 6$ or
- $t_w - t_b \leq 60 * 9 \wedge t_b \geq 30$ or
- $t_w - t_b \leq 60 * 10 \wedge t_b \geq 45$

Solution 1

- Case matching
- if $t_w \leq 60 * 6 \Rightarrow 0$
- if $60 * 6 + 30 < t_w \leq 60 * 9 + 30 \Rightarrow 30$
- if $60 * 9 + 45 < t_w \leq 60 * 10 + 45 \Rightarrow 45$
- if $60 * 10 + 45 < t_w \Rightarrow$ Output difference

L: Labour Laws

Problem author: Yvonne Kothmeier

Problem

Given time t_w. Find the minimum time $0 \leq t_b$, such that

- $t_w - t_b \leq 60 * 6$ or
- $t_w - t_b \leq 60 * 9 \wedge t_b \geq 30$ or
- $t_w - t_b \leq 60 * 10 \wedge t_b \geq 45$

Solution 1

- Case matching
- if $t_w \leq 60 * 6 \Rightarrow 0$
- if $60 * 6 + 30 < t_w \leq 60 * 9 + 30 \Rightarrow 30$
- if $60 * 9 + 45 < t_w \leq 60 * 10 + 45 \Rightarrow 45$
- if $60 * 10 + 45 < t_w \Rightarrow$ Output difference
- Fill gaps between cases by adding difference to lower bracket to solution

L: Labour Laws

Problem author: Yvonne Kothmeier

Problem

Given time t_w. Find the minimum time $0 \leq t_b$, such that

- $t_w - t_b \leq 60 * 6$ or
- $t_w - t_b \leq 60 * 9 \wedge t_b \geq 30$ or
- $t_w - t_b \leq 60 * 10 \wedge t_b \geq 45$

Solution 2

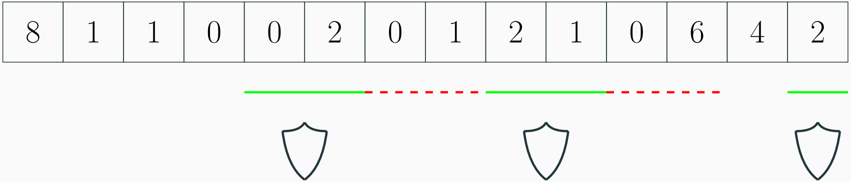
- Loop over t_b from 0 to t_w
- Check if solution is valid
- Output first valid solution

M: Mex Hex

Problem author: Niklas Mohrin

Problem

Given an array p and an integer d, cover length-d intervals of p so that the `mex` of the uncovered numbers is minimized. After a covered interval, the next d numbers cannot be covered.



M: Mex Hex

Problem author: Niklas Mohrin

Problem

Given an array p and an integer d, cover length-d intervals of p so that the $\texttt{mex}$ of the uncovered numbers is minimized. After a covered interval, the next d numbers cannot be covered.

8	1	1	0	0	2	0	1	2	1	0	6	4	2

Observations

- Let p' be the set of uncovered numbers.
- How to achieve $\texttt{mex}(p') = 0$ (the best possible)?

M: Mex Hex

Problem author: Niklas Mohrin

Problem

Given an array p and an integer d, cover length-d intervals of p so that the $\texttt{mex}$ of the uncovered numbers is minimized. After a covered interval, the next d numbers cannot be covered.

8	1	1	0	0	2	0	1	2	1	0	6	4	2

Observations

- Let p' be the set of uncovered numbers.
- How to achieve $\texttt{mex}(p') = 0$ (the best possible)? $\Rightarrow$ Cover all i with $p_i = 0$.

M: Mex Hex

Problem author: Niklas Mohrin

Problem

Given an array p and an integer d, cover length-d intervals of p so that the mex of the uncovered numbers is minimized. After a covered interval, the next d numbers cannot be covered.

8	1	1	0	0	2	0	1	2	1	0	6	4	2

Observations

- Let p' be the set of uncovered numbers.
- How to achieve $\texttt{mex}(p') = 0$ (the best possible)? $\Rightarrow$ Cover all i with $p_i = 0$.
- In general, covering all i with $p_i = x$ implies that $\texttt{mex}(p') \leq x$.

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

M: Mex Hex

Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.

M: Mex Hex

Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i + d) \cap \mathcal{I} = [j, j + d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.

M: Mex Hex

Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i + d) \cap \mathcal{I} = [j, j + d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j + d, j + 2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

M: Mex Hex

Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i + d) \cap \mathcal{I} = [j, j + d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j + d, j + 2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.

M: Mex Hex

Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i + d) \cap \mathcal{I} = [j, j + d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j + d, j + 2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.

		x	x					x					

M: Mex Hex

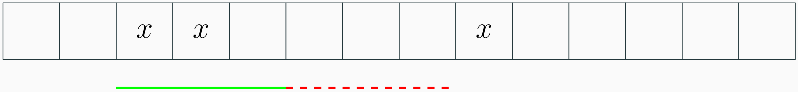
Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i+d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
    - it still covers the same relevant indices, that is, $[i, i+d) \cap \mathcal{I} = [j, j+d) \cap \mathcal{I}$, and
    - it is at least d far away from the previous interval.
- If $[j+d, j+2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.



M: Mex Hex

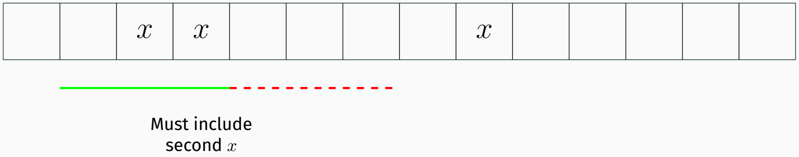
Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i+d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i+d) \cap \mathcal{I} = [j, j+d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j+d, j+2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.



M: Mex Hex

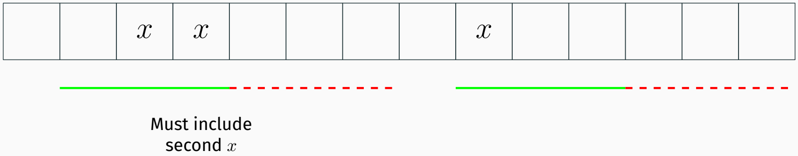
Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i+d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i+d) \cap \mathcal{I} = [j, j+d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j+d, j+2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.



M: Mex Hex

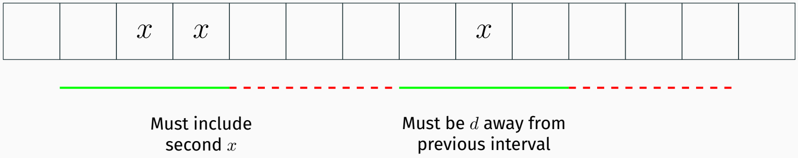
Problem author: Niklas Mohrin

New problem: Check if a set of indices $\mathcal{I} = \{i \mid p_i = x\}$ can be covered.

Greedy algorithm

- Place a covering interval $[i, i + d)$ to start on the first uncovered $i \in \mathcal{I}$.
- Move this interval to the left to start on the smallest index j so that
  - it still covers the same relevant indices, that is, $[i, i + d) \cap \mathcal{I} = [j, j + d) \cap \mathcal{I}$, and
  - it is at least d far away from the previous interval.
- If $[j + d, j + 2d) \cap \mathcal{I} \neq \emptyset$, then $\mathcal{I}$ cannot be covered. Otherwise repeat until all of $\mathcal{I}$ is covered.

Correctness can be proven via greedy stays ahead argument. Time complexity: $\mathcal{O}(|\mathcal{I}|)$.



M: Mex Hex

Problem author: Niklas Mohrin

Solution

- Partition indices into $\mathcal{I}_x = \{i \mid p_i = x\}$.
- Check for $x = 0, 1, \ldots, n$ in increasing order whether $\mathcal{I}_x$ can be covered.

Time complexity: $\sum_{x=0}^{n} \mathcal{O}(|\mathcal{I}_x|) = \mathcal{O}(n)$.

Random facts

Jury work

- 784 secret test cases (≈ 60 per problem)

Random facts

Jury work

- 784 secret test cases ($\approx$ 60 per problem)
- 110 jury solutions, 257 jury submissions

Random facts

Jury work

- 784 secret test cases (≈ 60 per problem)
- 110 jury solutions, 257 jury submissions
- The minimum number of lines the jury needed to solve all problems is

$$4 + 32 + 16 + 7 + 33 + 25 + 1 + 12 + 105 + 16 + 8 + 3 + 16 = 278$$

  On average 21.4 lines per problem

Random facts

Jury work

- 784 secret test cases ($\approx$ 60 per problem)
- 110 jury solutions, 257 jury submissions
- The minimum number of lines the jury needed to solve all problems is

  $$4 + 32 + 16 + 7 + 33 + 25 + 1 + 12 + 105 + 16 + 8 + 3 + 16 = 278$$

  On average 21.4 lines per problem
- The minimum number of characters the jury needed to solve all problems is

  $$125 + 1083 + 409 + 292 + 869 + 613 + 50 + 366 + 2913 + 432 + 312 + 170 + 502 = 8136$$

  On average 625.8 characters per problem