Winter Contest 2026 Presentation of Solutions

Winter Contest 2026

The Winter Contest Jury

February 1, 2026

Winter Contest 2026 Jury

- **Andreas Grigorjew**
  Paris Dauphine FR, CPUlm
- **Yvonne Kothmeier**
- **Felicia Lucke**
  ENS Lyon FR, CPUlm
- **Jannik Olbrich**
  Ulm University, CPUlm
- **Jeroen Op de Beek**
  Delft University of Technology
- **Michael Ruderer**
  University of Augsburg, CPUlm
- **Lucas Schwebler**
  Karlsruhe Institute of Technology
- **Christopher Weyand**
  MOIA GmbH, CPUlm
- **Paul Wild**
  Friedrich-Alexander University
  Erlangen-Nürnberg, CPUlm
- **Yidi Zang**
  Karlsruhe Institute of Technology, CPUlm
- **Michael Zündorf**
  Karlsruhe Institute of Technology, CPUlm

Winter Contest 2026 Test Solvers

- **Maarten Sijm**
  CHipCie (Delft University of Technology) / lynk.so

Winter Contest 2026 Technical Team

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

A: Animal Appendages

Problem author: Yidi Zang

Problem

Given how many distinguishable gestures each finger can do, up to what number can you count from zero.

Solution

- We are given "active" gestures per finger, so we need to add 1 to each finger for total gestures.
- With only two fingers with x and y total gestures, there are $x \cdot y$ gestures together.
- Extending that to ten finger: $\prod_{i=1}^{10}(a_i + 1)$.
- Subtract one since we include zero.

B: Bewitched Broomstick

Problem author: Yidi Zang

Problem

- Given a string s ($|s| \leq 2 \cdot 10^5$) and a length ℓ.
- Append $\ell - 1$ characters to the string.
- Maximize the appearance of the most frequent substring of length ℓ.

B: Bewitched Broomstick

Problem author: Yidi Zang

Problem

- Given a string s ($|s| \leq 2 \cdot 10^5$) and a length ℓ.
- Append $\ell - 1$ characters to the string.
- Maximize the appearance of the most frequent substring of length ℓ.

Solution Idea

- First try every substring t of length ℓ.
- Find the maximum overlap ℓ of t and a suffix of s.
- After that the remaining characters repeat the period of t.
- Count of the number of appearances and choose the maximum.
- For shorter substring of the suffix they just repeat their period.
- How to find maximum overlap and period of t?

B: Bewitched Broomstick

Problem author: Yidi Zang

Solution Idea

- First try every substring t of length ℓ.
- Find the maximum overlap ℓ of t and a suffix of s.
- After that the remaining characters repeat the period of t.
- Keep count of the number of appearances and choose the maximum.
- How to find maximum overlap and period of t?

Maximum Overlap

- Maximum overlap is the "easy" part.
- Reverse the string and add a '#' at the ℓth position.
- Find the longest match with KMP prefix function.

Period of Substrings

- How find the period of a single string?
- Find the first position i where $i + |\text{common prefix}| \geq \ell$.
- Using z-function for each substring is too slow.

B: Bewitched Broomstick

Problem author: Yidi Zang

Period of Substrings

- How find the period of a single string?
- Find the first position i where $i + |\text{common prefix}| \geq \ell$.
- Using z-function for each substring is too slow.
- Instead use Suffix Tree, that is a trie of all suffixes.
- The depth of a node is the length of the common prefix.
- The period can be calculated with a smaller into larger merging of (implicit) segment trees.
- The (annoying) details are left as an exercise to the reader.
- Total Runtime: $\mathcal{O}(n \log^2(n))$.

Notes

- There are also $\mathcal{O}(n\sqrt{n})$ solutions.
- The finding maximum overlap step can also be done in the same Suffix Tree.

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.

C: Cinderella’s Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.
- For a third point, there are only three possibilities:
  - Left of 0,
  - right of $d_{1,2}$, or
  - between 0 and $d_{1,2}$.

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.
- For a third point, there are only three possibilities:
  - Left of 0,
  - right of $d_{1,2}$, or
  - between 0 and $d_{1,2}$.
- At most one option can be consistent with both $d_{1,3}$ and $d_{2,3}$!
  $\implies$ Determine which is correct and just continue with the next point.

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.
- For a third point, there are only three possibilities:
  - Left of 0,
  - right of $d_{1,2}$, or
  - between 0 and $d_{1,2}$.
- At most one option can be consistent with both $d_{1,3}$ and $d_{2,3}$!
  $\implies$ Determine which is correct and just continue with the next point.
- If no option is consistent, the answer is impossible.

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.
- For a third point, there are only three possibilities:
  - Left of 0,
  - right of $d_{1,2}$, or
  - between 0 and $d_{1,2}$.
- At most one option can be consistent with both $d_{1,3}$ and $d_{2,3}$!
  $\implies$ Determine which is correct and just continue with the next point.
- If no option is consistent, the answer is impossible.
- After placing the points, check that they match the distance matrix.

C: Cinderella's Chore

Problem author: Jannik Olbrich

Problem

Given a matrix with pairwise distances, place n points on a line such that their distances are as in the matrix (or determine that this is impossible)

Solution

- Placing two points is trivial: Place one point at position 0 and the second at position $d_{1,2}$.
- For a third point, there are only three possibilities:
  - Left of 0,
  - right of $d_{1,2}$, or
  - between 0 and $d_{1,2}$.
- At most one option can be consistent with both $d_{1,3}$ and $d_{2,3}$!
  $\implies$ Determine which is correct and just continue with the next point.
- If no option is consistent, the answer is impossible.
- After placing the points, check that they match the distance matrix.
- Shift the coordinates s.t. they are in the allowed range.

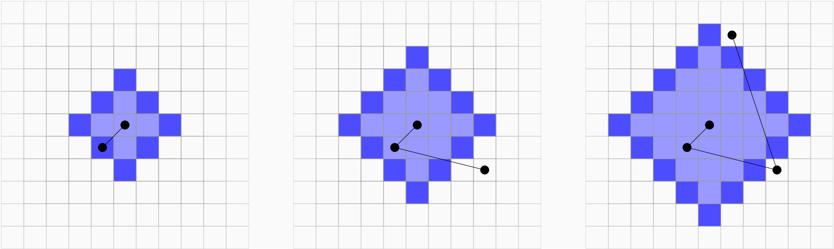
D: Delicious Disaster

Problem author: Jannik Olbrich, Paul Wild

Problem

Find the size of an ever expanding pile of magic porridge in an infinite grid.

- The porridge occupies all cells (x, y) such that $|x| + |y| \leq r$.
- You can query locations (x, y) to check if they are inside or not.
- The size r increases by 1 for each query you ask.
- Successive queries can be at most 1000 steps apart.
- Initial size: $r_0 \leq 10^6$, maximal number of queries: 5000.



D: Delicious Disaster

Problem author: Jannik Olbrich, Paul Wild

Solution

- The bound on successive queries makes binary search impossible.
- Instead, start by moving away from the origin by steps of 1000.
- This way, it takes at most ~ 1000 queries to reach an outside cell.
- You could now use binary search, while carefully handling the growing of the porridge.
- Instead, keep querying the same location until the porridge catches up to you.
- This takes another ~ 1000 queries.

E: Evening Entertainment

Problem author: Michael Zündorf

Problem

Play n rounds of a card game with every player scoring zero points.

E: Evening Entertainment

Problem author: Michael Zündorf

Problem

Play n rounds of a card game with every player scoring zero points.

Solution

- It is impossible to score 0 points in a round $\Rightarrow$ Case $n = 1$ is impossible
- Each player needs to score points and lose them all in the last round

E: Evening Entertainment

Problem author: Michael Zündorf

Problem

Play n rounds of a card game with every player scoring zero points.

Solution

- It is impossible to score 0 points in a round $\Rightarrow$ Case $n = 1$ is impossible
- Each player needs to score points and lose them all in the last round
- Who takes the tricks does not matter. Let the first player take all tricks
- The first player has $p_0 = \sum_{i=0}^{n-1} 20 + 10 \cdot i$ points before the last round
- All other players have $p_j = 20 * (n - 1)$ points before the last round

E: Evening Entertainment

Problem author: Michael Zündorf

Problem

Play n rounds of a card game with every player scoring zero points.

Solution

- It is impossible to score 0 points in a round $\Rightarrow$ Case $n = 1$ is impossible
- Each player needs to score points and lose them all in the last round
- Who takes the tricks does not matter. Let the first player take all tricks
- The first player has $p_0 = \sum_{i=0}^{n-1} 20 + 10 \cdot i$ points before the last round
- All other players have $p_j = 20 * (n - 1)$ points before the last round
- To lose all points in the last round, each player needs to bet $points_j/10$ tricks more than they take
- This number is always an integer and therefore possible

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1
- for each node except node 1, the answer is the minimum node among all descendants

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1
- for each node except node 1, the answer is the minimum node among all descendants
- for node 1 the answer is the minimum of all subtrees that do not include node 2

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1
- for each node except node 1, the answer is the minimum node among all descendants
- for node 1 the answer is the minimum of all subtrees that do not include node 2
- both can be computed with a DFS from node 1

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1
- for each node except node 1, the answer is the minimum node among all descendants
- for node 1 the answer is the minimum of all subtrees that do not include node 2
- both can be computed with a DFS from node 1
- **Runtime:** $O(n)$

F: Forgotten Fragments

Problem author: Christopher Weyand

Problem

Given a tree and a starting node. You can visit the starting node plus one adjacent subtree. For each possible start, maximize the MEX of the visited nodes.

Solution

- simulating it in $\Omega(n^2)$ is too slow!
- **Observation 1:** the MEX of a subtree is the minimum of everything else
- **Observation 2:** you always go towards node 1 (unless you are node 1)
- let's root the tree at node 1
- for each node except node 1, the answer is the minimum node among all descendants
- for node 1 the answer is the minimum of all subtrees that do not include node 2
- both can be computed with a DFS from node 1
- **Runtime:** $O(n)$
- other solutions involve segment trees or smaller into larger

G: Grimms' Fairy Tales

Problem author: Jeroen Op de Beek

Problem

Given a name of one of the problems in this contest, print the page where this problem starts.

G: Grimms' Fairy Tales

Problem author: Jeroen Op de Beek

Problem

Given a name of one of the problems in this contest, print the page where this problem starts.

Solution

- Naive solution: We can tabulate all the problem names, and carefully check on which page they start, and hardcode this table.

G: Grimms' Fairy Tales

Problem author: Jeroen Op de Beek

Problem

Given a name of one of the problems in this contest, print the page where this problem starts.

Solution

- Naive solution: We can tabulate all the problem names, and carefully check on which page they start, and hardcode this table.
- Instead, we can also notice that each problem name starts with the i'th capital letter in the alphabet, and all the problem statements are 2 pages. So a correct python code would be:

```python
print(1 + 2 * (ord(input()[0])- ord('A')))
```

H: Hansel and Gretel

Problem author: Lucas Schwebler

Problem

Uniquely mark a vertex in the graph to identify it in the second pass

H: Hansel and Gretel

Problem author: Lucas Schwebler

Problem

Uniquely mark a vertex in the graph to identify it in the second pass

Solution

- Observation: A graph can have exactly one of:
  - a vertex without edges
  - a vertex connected to every other vertex

H: Hansel and Gretel

Problem author: Lucas Schwebler

Problem

Uniquely mark a vertex in the graph to identify it in the second pass

Solution

- Observation: A graph can have exactly one of:
  - a vertex without edges
  - a vertex connected to every other vertex
- In the first pass:
  - If there is no vertex with zero edges, add one
  - Otherwise, add one and connect it to every other vertex

H: Hansel and Gretel

Problem author: Lucas Schwebler

Problem

Uniquely mark a vertex in the graph to identify it in the second pass

Solution

- Observation: A graph can have exactly one of:
  - a vertex without edges
  - a vertex connected to every other vertex
- In the first pass:
  - If there is no vertex with zero edges, add one
  - Otherwise, add one and connect it to every other vertex
- In the second pass:
  - If there is a vertex with zero edges, output it
  - Otherwise, find the vertex with $n-1$ edges

I: Ignoble Imp

Problem author: Jannik Olbrich

Problem

Given two strings a, b of length n, find a string c of length n such that $\max\{h(a,c), h(b,c)\}$ is minimal, where $h(a,c)$ is the Hamming distance between a and c.

I: Ignoble Imp

Problem author: Jannik Olbrich

Problem

Given two strings a, b of length n, find a string c of length n such that $\max\{h(a,c), h(b,c)\}$ is minimal, where $h(a,c)$ is the Hamming distance between a and c.

Solution

- When $a[i] = b[i]$, just set $c[i] = a[i]$.

I: Ignoble Imp

Problem author: Jannik Olbrich

Problem

Given two strings a, b of length n, find a string c of length n such that $\max\{h(a,c), h(b,c)\}$ is minimal, where $h(a,c)$ is the Hamming distance between a and c.

Solution

- When $a[i] = b[i]$, just set $c[i] = a[i]$.
- For half of the other indices, choose the character from a. For the other half of the indices, choose the character from b.

J: Jaded Journey

Problem author: Yidi Zang

Problem

- Minimize the cost to travel n distance units by ship.
- Rowing cost x per distance unit.
- Repairing sail cost r but will break again after d distance units.
- Wind is required to use sail.

J: Jaded Journey

Problem author: Yidi Zang

Problem

- Minimize the cost to travel n distance units by ship.
- Rowing cost x per distance unit.
- Repairing sail cost r but will break again after d distance units.
- Wind is required to use sail.

Solution

- We can model it as dp, where dp[i] represents minimum cost to get to i with a broken sail.
- There are 2 transitions, rowing one unit and repairing sail.
- Rowing from $i \to i + 1$: costs x.
- Repairing sail $i \to i + d$: costs $r + (\#\text{ of units without wind}) \cdot x$.

J: Jaded Journey

Problem author: Yidi Zang

Problem

- Minimize the cost to travel n distance units by ship.
- Rowing cost x per distance unit.
- Repairing sail cost r but will break again after d distance units.
- Wind is required to use sail.

Solution

- We can model it as dp, where dp[i] represents minimum cost to get to i with a broken sail.
- There are 2 transitions, rowing one unit and repairing sail.
- Rowing from $i \to i + 1$: costs x.
- Repairing sail $i \to i + d$: costs $r + (\#\text{ of units without wind}) \cdot x$.
- Efficiently calculate ($\#$ of units without wind) using prefix sums or sliding window.
- Total Runtime: $\mathcal{O}(n)$.

K: Knavish Knockout

Problem author: Michael Ruderer

Problem

Input: $n \cdot k$ apples with distinct sizes and poison values

Task:

- Partition the apples into k groups of size n.
- Maximize total consumed poisoned, when the smallest apple in each group is eaten.

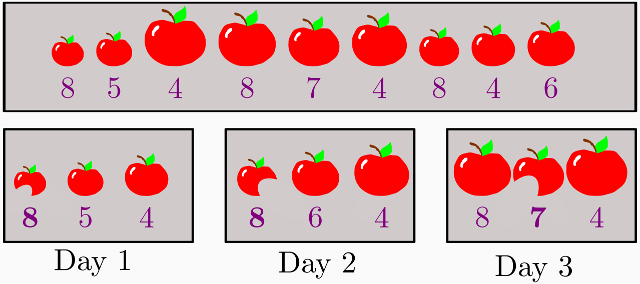
Problem

Input: $n \cdot k$ apples with distinct sizes and poison values

Task:

- Partition the apples into k groups of size n.
- Maximize total consumed poisoned, when the smallest apple in each group is eaten.

E.g. Sample 2 with $n = k = 3$:



K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

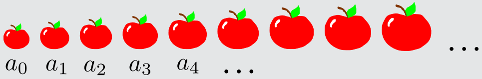
- Let $a_0, a_1, \ldots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest

K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \ldots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest

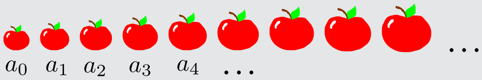


K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \ldots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest



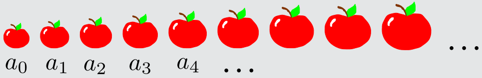
- First Observation: In any solution a_0 has to be eaten

K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \ldots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest



- First Observation: In any solution a_0 has to be eaten
- In general: Among $a_0, \ldots, a_{n \cdot i}$, at least $i + 1$ apples have to be eaten (Pigdeonhole-like argument)
  This condition is sufficient to construct a valid solution!
  Any set E of k apples is the set of apples eaten in a feasible solution if and only if

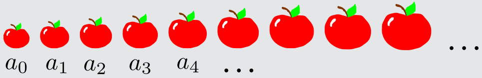
$$|E \cap \{a_0, \ldots, a_{n \cdot i}\}| \geq i + 1 \text{ for all } i = 0 \ldots k - 1$$

K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \dots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest



- First Observation: In any solution a_0 has to be eaten
- In general: Among $a_0, \dots, a_{n \cdot i}$, at least $i+1$ apples have to be eaten (Pigdeonhole-like argument)
  This condition is sufficient to construct a valid solution!
  Any set E of k apples is the set of apples eaten in a feasible solution if and only if

$$|E \cap \{a_0, \dots, a_{n \cdot i}\}| \geq i+1 \text{ for all } i = 0 \dots k-1$$

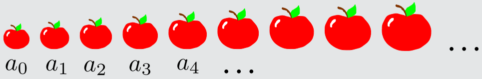
Algorithm: Greedily construct optimal set E satisfying this condition.
For $i = 0, \dots, k-1$, add the most poisonous apple from $\{a_0, \dots, a_{n \cdot i}\} \setminus E$ to E.

K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \ldots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest



- First Observation: In any solution a_0 has to be eaten
- In general: Among $a_0, \ldots, a_{n \cdot i}$, at least $i + 1$ apples have to be eaten (Pigdeonhole-like argument)
  This condition is sufficient to construct a valid solution!
  Any set E of k apples is the set of apples eaten in a feasible solution if and only if

$$|E \cap \{a_0, \ldots, a_{n \cdot i}\}| \geq i + 1 \text{ for all } i = 0 \ldots k - 1$$

Algorithm: Greedily construct optimal set E satisfying this condition.
For $i = 0, \ldots, k - 1$, add the most poisonous apple from $\{a_0, \ldots, a_{n \cdot i}\} \setminus E$ to E.

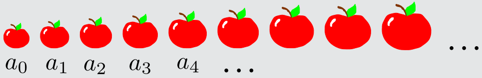
Implementation: Use a max-heap to efficiently find the most poisonous apple in each step.

K: Knavish Knockout

Problem author: Michael Ruderer

Solution Idea

- Let $a_0, a_1, \dots, a_{n \cdot k-1}$ be the apples sorted by size, smallest to largest



- First Observation: In any solution a_0 has to be eaten
- In general: Among $a_0, \dots, a_{n \cdot i}$, at least $i+1$ apples have to be eaten (Pigdeonhole-like argument)
  This condition is sufficient to construct a valid solution!
  Any set E of k apples is the set of apples eaten in a feasible solution if and only if

$$|E \cap \{a_0, \dots, a_{n \cdot i}\}| \geq i+1 \text{ for all } i = 0 \dots k-1$$

Algorithm: Greedily construct optimal set E satisfying this condition.
For $i = 0, \dots, k-1$, add the most poisonous apple from $\{a_0, \dots, a_{n \cdot i}\} \setminus E$ to E.

Implementation: Use a max-heap to efficiently find the most poisonous apple in each step.

Complexity: $O(n \cdot k \cdot \log(n \cdot k))$

L: Lucky Hans

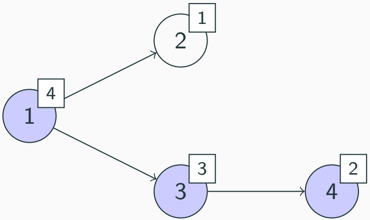
Problem author: Lucas Schwebler

Problem

- DAG on n vertices and m edges, vertex 1 can reach every other vertex.
- An integer k ($2 \leq n \leq 3000$, $1 \leq m \leq 9000$, $1 \leq k \leq n$).

Problem: Find a permutation $p_1, \ldots, p_N$ such that

- Topological order: For every edges $(u, v) \in E$, we have $p_u > p_v$
- Greedy path which starts at 1 and takes the maximum finishes at a vertex v with $p_v = k$

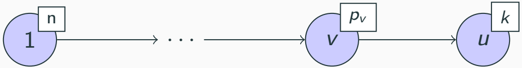


L: Lucky Hans

Problem author: Lucas Schwebler

Solution

- Label greedy path $1, \ldots, v, u$
- Necessary conditions:

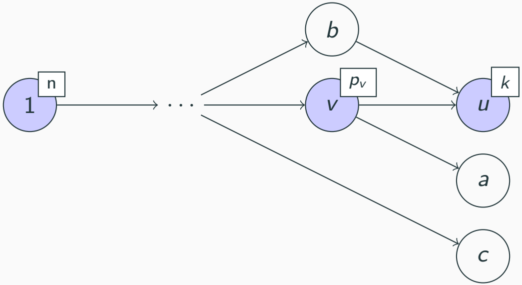


L: Lucky Hans

Problem author: Lucas Schwebler

Solution

- Label greedy path $1, \ldots, v, u$
- Necessary conditions:

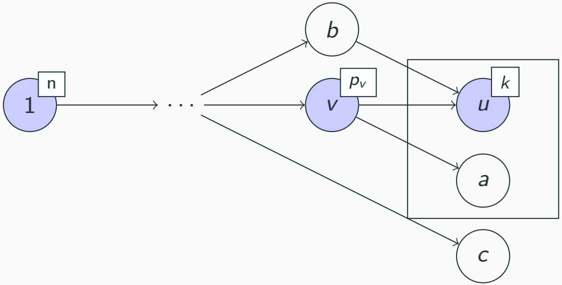


L: Lucky Hans

Problem author: Lucas Schwebler

Solution

- Label greedy path $1, \ldots, v, u$
- Necessary conditions:
  - v can reach $\leq k$ vertices (excluding v)

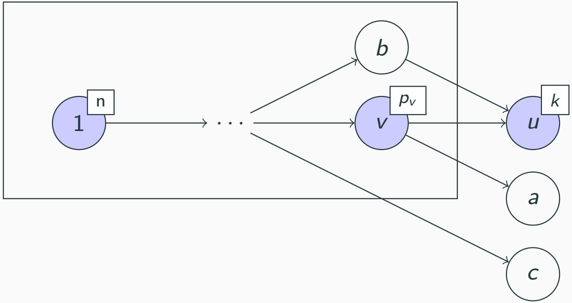


L: Lucky Hans

Problem author: Lucas Schwebler

Solution

- Label greedy path $1, \ldots, v, u$
- Necessary conditions:
  - v can reach $\leq k$ vertices (excluding v)
  - u is reached by $\leq n - k$ vertices (excluding u)

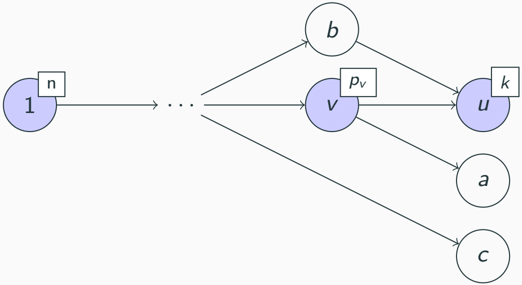


L: Lucky Hans

Problem author: Lucas Schwebler

Solution

- Label greedy path $1, \ldots, v, u$
- Necessary conditions:
  - v can reach $\leq k$ vertices (excluding v)
  - u is reached by $\leq n - k$ vertices (excluding u)
- Also sufficient!



L: Lucky Hans

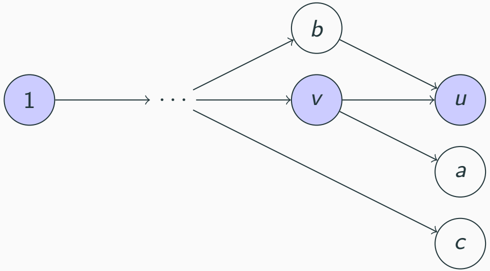
Problem author: Lucas Schwebler

Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:



L: Lucky Hans

Problem author: Lucas Schwebler

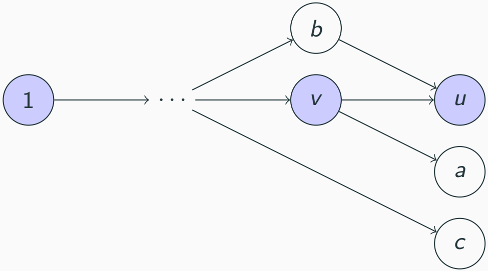
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n-k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k-1$ vertices including everything reachable from v, then u



L: Lucky Hans

Problem author: Lucas Schwebler

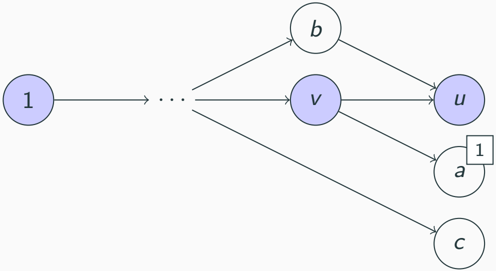
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u



L: Lucky Hans

Problem author: Lucas Schwebler

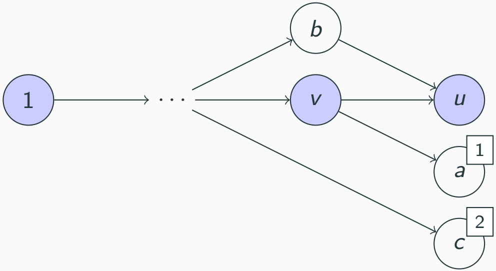
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u



L: Lucky Hans

Problem author: Lucas Schwebler

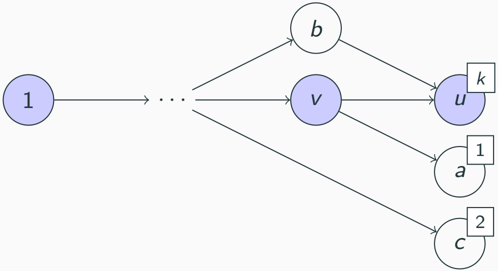
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u



L: Lucky Hans

Problem author: Lucas Schwebler

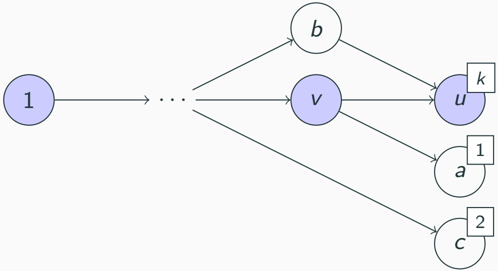
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n-k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k-1$ vertices including everything reachable from v, then u
- all remaining vertices, but v is not allowed



L: Lucky Hans

Problem author: Lucas Schwebler

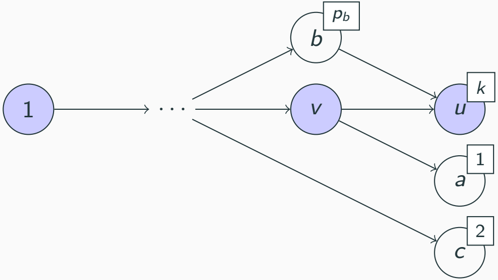
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u
- all remaining vertices, but v is not allowed



L: Lucky Hans

Problem author: Lucas Schwebler

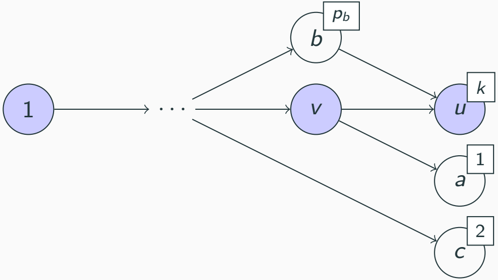
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u
- all remaining vertices, but v is not allowed
- all remaining vertices



L: Lucky Hans

Problem author: Lucas Schwebler

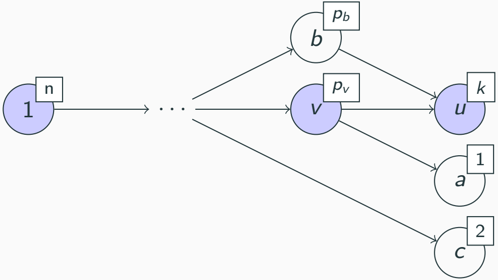
Necessary conditions

- v can reach $\leq k$ vertices (excluding v)
- u is reached by $\leq n - k$ vertices (excluding u)

Construction

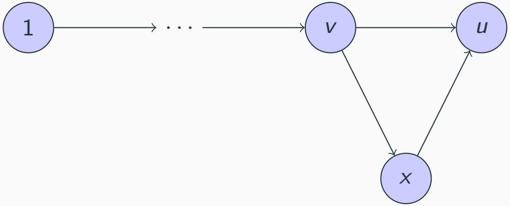
Pop vertices with outdegree 0 in this order:

- $k - 1$ vertices including everything reachable from v, then u
- all remaining vertices, but v is not allowed
- all remaining vertices



Note

In the reconstruction, it's possible that v is *not* directly before u.



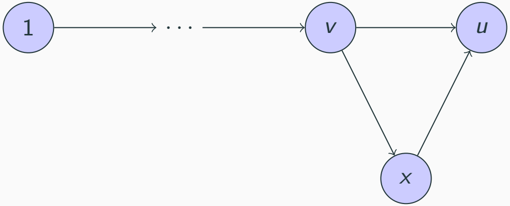
L: Lucky Hans

Problem author: Lucas Schwebler

Note

In the reconstruction, it's possible that v is *not* directly before u.

- Reconstruction is still valid

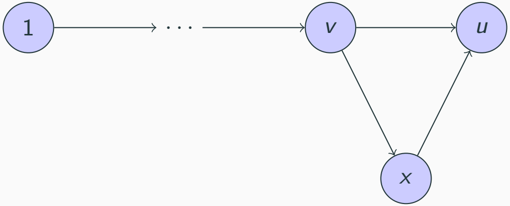


Problem author: Lucas Schwebler

Note

In the reconstruction, it's possible that v is *not* directly before u.

- Reconstruction is still valid
- Alternatively, among all possible v, pick the one which is "closest" to u.



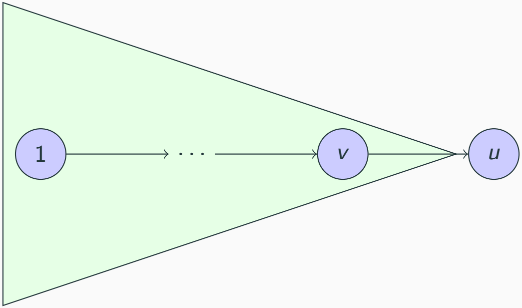
L: Lucky Hans

Problem author: Lucas Schwebler

Proof Idea

- Let S be the set of vertices which can reach v (including v).
- $\min_{a \in S}(p_a) > \max_{b \in V \setminus S}(p_b)$.
- The graph induced by S has only one vertex without outgoing edge: v.

$\rightsquigarrow$ Greedy path contains v.



L: Lucky Hans

Problem author: Lucas Schwebler

Complexity

- For each vertex, count the number of reachable vertices in $O(nm)$
- Check necessary conditions for all edges in $O(m)$
- Reconstruction in $O(m)$

L: Lucky Hans

Problem author: Lucas Schwebler

Complexity

- For each vertex, count the number of reachable vertices in $O(nm)$
- Check necessary conditions for all edges in $O(m)$
- Reconstruction in $O(m)$

Possible speedup (not required): bitsets for reachability in $O(\frac{nm}{W})$

Statistics: ... submissions, ... accepted, ... unknown

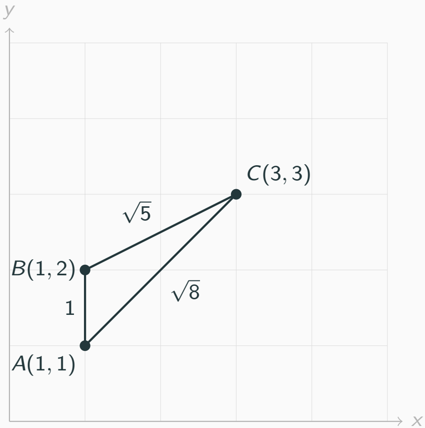
M: Mother Hulda

Problem author: Jannik Olbrich

Problem

Given a matrix with squared pairwise distances, place n points in 2D with integer coordinates such that their distances are as in the matrix (or determine that this is impossible)

$$M = \begin{pmatrix} 0 & 1 & 8 \\ 1 & 0 & 5 \\ 8 & 5 & 0 \end{pmatrix}$$



Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with “Cinderella’s Chore,” we can just place the first point at the origin.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with “Cinderella’s Chore,” we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.
  - Pick any candidate that has a non-collinear placement. If there is none, the points must be either on a line, or the answer is impossible.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.
  - Pick any candidate that has a non-collinear placement. If there is none, the points must be either on a line, or the answer is impossible.
- Iterate over all $\mathcal{O}(k)$ valid placements of the chosen three points. (When two points are placed, there are at most 2 valid placements for a third point)

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.
  - Pick any candidate that has a non-collinear placement. If there is none, the points must be either on a line, or the answer is impossible.
- Iterate over all $\mathcal{O}(k)$ valid placements of the chosen three points. (When two points are placed, there are at most 2 valid placements for a third point)
- For each, place the other points if possible

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.
  - Pick any candidate that has a non-collinear placement. If there is none, the points must be either on a line, or the answer is impossible.
- Iterate over all $\mathcal{O}(k)$ valid placements of the chosen three points. (When two points are placed, there are at most 2 valid placements for a third point)
- For each, place the other points if possible
- After placing the points, check that they match the distance matrix & shift the coordinates s.t. they are in the allowed range.

Solution

- Key insight: Placing three non-collinear points uniquely determines the remaining positions.
- As with "Cinderella's Chore," we can just place the first point at the origin.
- For all $d_{1,i}$ find all possible vectors (x, y) with $x^2 + y^2 = d_{1,i}$. Let the maximum number of valid vectors be k.
- Find a point that is not collinear with the first two points:
  - For each candidate, consider all possible positions of the second point and this candidate.
  - Either all placements matching the distance matrix are collinear, or none are.
  - Pick any candidate that has a non-collinear placement. If there is none, the points must be either on a line, or the answer is impossible.
- Iterate over all $\mathcal{O}(k)$ valid placements of the chosen three points. (When two points are placed, there are at most 2 valid placements for a third point)
- For each, place the other points if possible
- After placing the points, check that they match the distance matrix & shift the coordinates s.t. they are in the allowed range.
- Time complexity: $\mathcal{O}(k^2 \cdot n)$, and k is at most $\approx 10^2$.

Random facts

Jury work

- 770 secret test cases ($\approx$ 59 per problem)

Random facts

Jury work

- 770 secret test cases ($\approx$ 59 per problem)
- 108 jury solutions

Random facts

Jury work

- 770 secret test cases (≈ 59 per problem)
- 108 jury solutions
- The minimum number of lines the jury needed to solve all problems is

$$1 + 131 + 11 + 3 + 6 + 26 + 1 + 12 + 3 + 11 + 17 + 93 + 51 = 366$$

  On average 28.2 lines per problem

Random facts

Jury work

- 770 secret test cases ($\approx$ 59 per problem)
- 108 jury solutions
- The minimum number of lines the jury needed to solve all problems is

$$1 + 131 + 11 + 3 + 6 + 26 + 1 + 12 + 3 + 11 + 17 + 93 + 51 = 366$$

  On average 28.2 lines per problem
- The minimum number of characters the jury needed to solve all problems is

$$8 + 3257 + 321 + 132 + 255 + 659 + 33 + 210 + 126 + 303 + 420 + 1976 + 1430$$

  On average 702 characters per problem