NCPC 2019
Presentation of solutions

2019-10-05
Problems prepared by

- Per Austrin (KTH Royal Institute of Technology)
- Bjarki Ágúst Guðmundsson (Reykjavík University)
- Nils Gustafsson (KTH Royal Institute of Technology)
- Antti Laaksonen (CSES)
- Ulf Lundström (Excillum)
- Jimmy Mårdell (Spotify)
- Torstein Strømme (University of Bergen)
- Pehr Söderman (Kattis)
- Jesper Öqvist (Lund University)

And big thanks to Lukáš Poláček for test-solving the problems!
Problem

Given values $t_1, \ldots, t_n$, which $i$ minimizes $\max(t_i, t_{i+2})$?
H — Hot Hike

**Problem**

Given values $t_1, \ldots, t_n$, which $i$ minimizes $\max(t_i, t_{i+2})$?

**Business Logic Solution**

```
ACCEPT lin
MOVE FUNCTION NUMVAL(lin) TO n
ACCEPT lin
PERFORM VARYING i FROM 1 BY 1 UNTIL i GREATER THAN n
   UNSTRING lin DELIMITED BY SPACE INTO Z(i) WITH POINTER linepos
END-PERFORM
PERFORM VARYING i FROM 1 BY 1 UNTIL i GREATER THAN n - 2
   IF FUNCTION MAX(Z(i), Z(i + 2)) < v THEN
      SET v TO FUNCTION MAX(Z(i), Z(i + 2))
      SET d TO i
   END-IF
END-PERFORM
MOVE v TO t
DISPLAY d, " ", t
```
Given values $t_1, \ldots, t_n$, which $i$ minimizes $\max(t_i, t_{i+2})$?

```
ACCEPT lin
MOVE FUNCTION NUMVAL(lin) TO n
ACCEPT lin
PERFORM VARYING i FROM 1 BY 1 UNTIL i GREATER THAN n
   UNSTRING lin DELIMITED BY SPACE INTO Z(i) WITH POINTER linepos
END-PERFORM
PERFORM VARYING i FROM 1 BY 1 UNTIL i GREATER THAN n - 2
   IF FUNCTION MAX(Z(i), Z(i + 2)) < v THEN
      SET v TO FUNCTION MAX(Z(i), Z(i + 2))
      SET d TO i
   END-IF
END-PERFORM
MOVE v TO t
DISPLAY d, " ", t
```

Statistics: 380 submissions, 219 accepted, first after 00:03
### Problem

Simulate team selection process.

#### Solution

1. Keep kids in a list, remove from list when they are selected.
2. Jump $k - 1$ steps in list every time. (where $k = \text{number of words in rhyme}$)
3. Time complexity $O(n^2)$ (why the square?).
Problem
Simulate team selection process.

Solution
1. Keep kids in a list, remove from list when they are selected.
E — Eeny Meeny

Problem
Simulate team selection process.

Solution
1. Keep kids in a list, remove from list when they are selected.
2. Jump $k - 1$ steps in list every time.
   (where $k =$ number of words in rhyme.)
Problem

Simulate team selection process.

Solution

1. Keep kids in a list, remove from list when they are selected.
2. Jump $k - 1$ steps in list every time. (where $k =$ number of words in rhyme.)
3. Time complexity $O(n^2)$ (why the square?).
Problem
Simulate team selection process.

Solution
1. Keep kids in a list, remove from list when they are selected.
2. Jump $k - 1$ steps in list every time.
   (where $k =$ number of words in rhyme.)
3. Time complexity $O(n^2)$ (why the square?).

Statistics: 346 submissions, 198 accepted, first after 00:11
Problem
Find a winning next move in Word Chain game, or just some valid move if no winning move exists.
Problem
Find a winning next move in Word Chain game, or just some valid move if no winning move exists.

Solution
1. Count how many unused words start with each letter a-z
Problem
Find a winning next move in Word Chain game, or just some valid move if no winning move exists.

Solution
1. Count how many unused words start with each letter a-z
2. For each unused word $x$ that starts with last letter of previous word, check if there are no unused words that start with last letter of $x$ (if so, $x$ is winning).
Problem

Find a winning next move in Word Chain game, or just some valid move if no winning move exists.

Solution

1. Count how many unused words start with each letter a-z
2. For each unused word $x$ that starts with last letter of previous word, check if there are no unused words that start with last letter of $x$ (if so, $x$ is winning).
3. Special case: $x$ starts and ends with same letter.
   (Shown in Sample Input 3.)
Problem
Find a winning next move in Word Chain game, or just some valid move if no winning move exists.

Solution
1. Count how many unused words start with each letter a-z
2. For each unused word $x$ that starts with last letter of previous word, check if there are no unused words that start with last letter of $x$ (if so, $x$ is winning).
3. Special case: $x$ starts and ends with same letter.
   (Shown in Sample Input 3.)
4. Time complexity $O(n)$. 
Problem

Find a winning next move in *Word Chain* game, or just some valid move if no winning move exists.

Solution

1. Count how many unused words start with each letter a-z
2. For each unused word $x$ that starts with last letter of previous word, check if there are no unused words that start with last letter of $x$ (if so, $x$ is winning).
3. Special case: $x$ starts and ends with same letter.
   (Shown in Sample Input 3.)
4. Time complexity $O(n)$.

Statistics: 723 submissions, 189 accepted, first after 00:05
Problem
How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

Solution

$\Rightarrow$ Greedily put new sodas in slots with fewest old sodas.

If untouched slots have $< m$ sodas, impossible.

Time complexity $O(s \log s)$. 

Statistics: 334 submissions, 151 accepted, first after 00:31
**Problem**

How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

**Solution**

1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
Problem
How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

Solution
1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
2. If slot $A$ has more old sodas than slot $B$, it is never better to fill up slot $A$ before slot $B$. 
Problem
How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

Solution
1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
2. If slot $A$ has more old sodas than slot $B$, it is never better to fill up slot $A$ before slot $B$.
3. $\Rightarrow$ Greedily put new sodas in slots with fewest old sodas.
**Problem**

How to put \( n \) new sodas in a fridge with \( s \) partially filled stack-based slots of sodas in a way that maximizes chances that next \( m \) sodas taken from fridge are all old sodas?

**Solution**

1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
2. If slot \( A \) has more old sodas than slot \( B \), it is never better to fill up slot \( A \) before slot \( B \).
3. \( \Rightarrow \) Greedily put new sodas in slots with fewest old sodas.
4. If untouched slots have \( < m \) sodas, impossible.
Problem
How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

Solution
1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
2. If slot $A$ has more old sodas than slot $B$, it is never better to fill up slot $A$ before slot $B$.
3. $\Rightarrow$ Greedily put new sodas in slots with fewest old sodas.
4. If untouched slots have $< m$ sodas, impossible.
5. Time complexity $O(s \log s)$.
Problem
How to put $n$ new sodas in a fridge with $s$ partially filled stack-based slots of sodas in a way that maximizes chances that next $m$ sodas taken from fridge are all old sodas?

Solution
1. If we start putting sodas in a slot, that slot is “lost” and we might as well fill it up completely.
2. If slot $A$ has more old sodas than slot $B$, it is never better to fill up slot $A$ before slot $B$.
3. $\Rightarrow$ Greedily put new sodas in slots with fewest old sodas.
4. If untouched slots have $< m$ sodas, impossible.
5. Time complexity $O(s \log s)$.

Statistics: 334 submissions, 151 accepted, first after 00:31
Problem

Arrange three rectangles of sizes $a_1 \times b_1$, $a_2 \times b_2$ and $a_3 \times b_3$ so that area of enclosing rectangle minimized.
Problem
Arrange three rectangles of sizes $a_1 \times b_1$, $a_2 \times b_2$ and $a_3 \times b_3$ so that area of enclosing rectangle minimized.

Solution
1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
Problem

Arrange three rectangles of sizes $a_1 \times b_1$, $a_2 \times b_2$ and $a_3 \times b_3$ so that area of enclosing rectangle minimized.

Solution

1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
   2. next rectangle to the right with top side aligned.
Problem

Arrange three rectangles of sizes \( a_1 \times b_1 \), \( a_2 \times b_2 \) and \( a_3 \times b_3 \) so that area of enclosing rectangle minimized.

Solution

1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
   2. next rectangle to the right with top side aligned.
   3. last rectangle as high as possible with left side aligned with one of the previous two

\[
\begin{align*}
&\text{or} \\
&\text{or}
\end{align*}
\]
Problem

Arrange three rectangles of sizes $a_1 \times b_1$, $a_2 \times b_2$ and $a_3 \times b_3$ so that area of enclosing rectangle minimized.

Solution

1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
   2. next rectangle to the right with top side aligned.
   3. last rectangle as high as possible with left side aligned with one of the previous two, or to the right of the previous two.

![Diagram showing different arrangements of rectangles with the explanation of the solution steps.](image-url)
Problem

Arrange three rectangles of sizes $a_1 \times b_1$, $a_2 \times b_2$ and $a_3 \times b_3$ so that area of enclosing rectangle minimized.

Solution

1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
   2. next rectangle to the right with top side aligned.
   3. last rectangle as high as possible with left side aligned with one of the previous two, or to the right of the previous two.

2. Try for all $3! \cdot 2^3 = 48$ permutations+rotations of rectangles.
**Problem**

Arrange three rectangles of sizes \(a_1 \times b_1\), \(a_2 \times b_2\) and \(a_3 \times b_3\) so that area of enclosing rectangle minimized.

**Solution**

1. Without loss of generality, can assume solution places:
   1. one rectangle somewhere.
   2. next rectangle to the right with top side aligned.
   3. last rectangle as high as possible with left side aligned with one of the previous two, or to the right of the previous two.

2. Try for all \(3! \cdot 2^3 = 48\) permutations+rotations of rectangles.

Statistics: 232 submissions, 100 accepted, first after 00:33
Problem

Split $n \times m$ bar of chocolate into piles of size $a$ and $n \cdot m - a$ by breaking it horizontally/vertically as few times as possible.

Solution

1. One split succeeds if $a$ is divisible by $n$ or $m$.

2. Two splits succeed if $a$ can be factored into $a = x \cdot y$ where $x \leq n$ and $y \leq m$, or if $n \cdot m - a$ can. (Check by trying all $O(n)$ possible values of $x$.)

3. Three splits are always enough.
Problem

Split $n \times m$ bar of chocolate into piles of size $a$ and $n \cdot m - a$ by breaking it horizontally/vertically as few times as possible.

Solution

1. One split suffices if $a$ is divisible by $n$ or $m$. 
Problem

Split $n \times m$ bar of chocolate into piles of size $a$ and $n \cdot m - a$ by breaking it horizontally/vertically as few times as possible.

Solution

1. One split suffices if $a$ is divisible by $n$ or $m$.
2. Two splits suffice if $a$ can be factored into $a = x \cdot y$ where $x \leq n$ and $y \leq m$, or if $n \cdot m - a$ can. (Check by trying all $O(n)$ possible values of $x$.)
Problem
Split $n \times m$ bar of chocolate into piles of size $a$ and $n \cdot m - a$ by breaking it horizontally/vertically as few times as possible.

Solution
1. One split suffices if $a$ is divisible by $n$ or $m$.
2. Two splits suffice if $a$ can be factored into $a = x \cdot y$ where $x \leq n$ and $y \leq m$, or if $n \cdot m - a$ can. (Check by trying all $O(n)$ possible values of $x$.)
3. Three splits are always enough.
Problem

Split $n \times m$ bar of chocolate into piles of size $a$ and $n \cdot m - a$ by breaking it horizontally/vertically as few times as possible.

Solution

1. One split suffices if $a$ is divisible by $n$ or $m$.
2. Two splits suffice if $a$ can be factored into $a = x \cdot y$ where $x \leq n$ and $y \leq m$, or if $n \cdot m - a$ can.
   (Check by trying all $O(n)$ possible values of $x$.)
3. Three splits are always enough.

Statistics: 642 submissions, 85 accepted, first after 00:20
### Problem

Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

1. **Group of size** $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. **Equivalent problem:** divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$.
3. For the groups of size $< k$, optimal to split the troops evenly.
4. **⇒** Given value of $g$, can do a little math and compute objective value in constant time.
5. **$g$** must be between $n/k - m$ and $n/k$.
6. **Time complexity** $O(m)$ (faster solutions exist).

Statistics: 268 submissions, 19 accepted, first after 01:07

Author: Nils Gustafsson

NCPC 2019 solutions
## Problem

Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

## Solution

1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$. 
Problem

Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

Solution

1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. $\Rightarrow$ Equivalent problem: divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$. 
**Problem**
Divide \( n \) troops into \( \leq m \) groups. Each round we lose up to \( k \) troops from one group. Maximize sum of life lengths of troops.

**Solution**
1. Group of size \( x + k \) equivalent to two groups of sizes \( x \) and \( k \).
2. \( \Rightarrow \) Equivalent problem: divide \( n \) troops into some number \( g \) groups of size \( k \), remaining ones into \( \leq m \) groups of size \( < k \).
3. For the groups of size \( < k \), optimal to split the troops evenly.
Problem

Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

Solution

1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. $\Rightarrow$ Equivalent problem: divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$.
3. For the groups of size $< k$, optimal to split the troops evenly.
4. $\Rightarrow$ Given value of $g$, can do a little math and compute objective value in constant time.
**Problem**

Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

**Solution**

1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. Equivalent problem: divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$.
3. For the groups of size $< k$, optimal to split the troops evenly.
4. Given value of $g$, can do a little math and compute objective value in constant time.
5. $g$ must be between $n/k - m$ and $n/k$. Try all possibilities.
Problem
Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

Solution
1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. $\Rightarrow$ Equivalent problem: divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$.
3. For the groups of size $< k$, optimal to split the troops evenly.
4. $\Rightarrow$ Given value of $g$, can do a little math and compute objective value in constant time.
5. $g$ must be between $n/k - m$ and $n/k$. Try all possibilities.
6. Time complexity $O(m)$ (faster solutions exist).
Problem
Divide $n$ troops into $\leq m$ groups. Each round we lose up to $k$ troops from one group. Maximize sum of life lengths of troops.

Solution
1. Group of size $x + k$ equivalent to two groups of sizes $x$ and $k$.
2. $\Rightarrow$ Equivalent problem: divide $n$ troops into some number $g$ groups of size $k$, remaining ones into $\leq m$ groups of size $< k$.
3. For the groups of size $< k$, optimal to split the troops evenly.
4. $\Rightarrow$ Given value of $g$, can do a little math and compute objective value in constant time.
5. $g$ must be between $n/k - m$ and $n/k$. Try all possibilities.
6. Time complexity $O(m)$ (faster solutions exist).

Statistics: 268 submissions, 19 accepted, first after 01:07
Problem

Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution

1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.

2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.

3. Verify all flows known and correct when done.

4. Time complexity $O(n)$. 

Statistics: 219 submissions, 24 accepted, first after 01:30
Problem

Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution

1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.

2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.

3. Verify all flows known and correct when done.

4. Time complexity \(O(n)\).
Problem

Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution

1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.
Problem

Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution

1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.

2. In top-down order, for known flows:
Problem
Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution
1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.
2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.

Problem

Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution

1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.

2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.

3. Verify all flows known and correct when done.
F — Flow Finder

Problem
Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution
1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.
2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.
3. Verify all flows known and correct when done.
4. Time complexity $O(n)$.

Statistics: 219 submissions, 24 accepted, first after 01:30

Author: Markus Dregi and Nils Gustafsson
NCPC 2019 solutions
Problem
Given rooted tree with water flowing from sources to root, and some known water flows, reconstruct all of them if possible.

Solution
1. In bottom-up order:
   - For unknown flows where all child flows known, flow is sum of child flows.
   - Compute lower bounds on flows: actual flow if known, otherwise max of 1 and sum of lower bounds of child flows.
2. In top-down order, for known flows:
   - If one unknown child, or lower bounds of unknown children adds up to remaining flow, distribute among unknown children.
3. Verify all flows known and correct when done.
4. Time complexity $O(n)$.

Statistics: 219 submissions, 24 accepted, first after 01:30
Problem

Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).
Problem

Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).

Solution

1. #edges from first $t$ nodes to anywhere is $\sum_{i=1}^{t} \text{outdegree}(v_i)$.
## Problem

Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).

## Solution

1. The number of edges from first $t$ nodes to anywhere is $\sum_{i=1}^{t} \text{outdegree}(v_i)$.
2. The number of edges from first $t$ nodes to first $t$ nodes is $\binom{t}{2}$.
Problem

Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).

Solution

1. #edges from first $t$ nodes to anywhere is $\sum_{i=1}^{t} \text{outdegree}(v_i)$.
2. #edges from first $t$ nodes to first $t$ nodes is $\binom{t}{2}$.
3. So #edges from first $t$ to last $n - t$ nodes are $\sum_{i=1}^{t} \text{outdegree}(v_i) - \binom{t}{2}$. 
### Problem
Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).

### Solution
1. #edges from first $t$ nodes to anywhere is $\sum_{i=1}^{t} \text{outdegree}(v_i)$.
2. #edges from first $t$ nodes to first $t$ nodes is $\binom{t}{2}$.
3. So #edges from first $t$ to last $n - t$ nodes are $\sum_{i=1}^{t} \text{outdegree}(v_i) - \binom{t}{2}$.
4. Implies it is optimal to order nodes by increasing out-degree.
Problem

Given complete directed graph (tournament), order nodes so that number of edges from first \( t \) nodes to last \( n - t \) nodes is at most \( k \) for all \( t \). Find minimum value of \( k \) (a.k.a. “directed cutwidth”).

Solution

1. #edges from first \( t \) nodes to anywhere is \( \sum_{i=1}^{t} \text{outdegree}(v_i) \).
2. #edges from first \( t \) nodes to first \( t \) nodes is \( \binom{t}{2} \).
3. So #edges from first \( t \) to last \( n - t \) nodes are \( \sum_{i=1}^{t} \text{outdegree}(v_i) - \binom{t}{2} \).
4. Implies it is optimal to order nodes by increasing out-degree.
5. Time complexity \( O(n \log n) \) after reading the \( n^2 \) size input to compute degrees.
Problem
Given complete directed graph (tournament), order nodes so that number of edges from first $t$ nodes to last $n - t$ nodes is at most $k$ for all $t$. Find minimum value of $k$ (a.k.a. “directed cutwidth”).

Solution
1. #edges from first $t$ nodes to anywhere is $\sum_{i=1}^{t} \text{outdegree}(v_i)$.
2. #edges from first $t$ nodes to first $t$ nodes is $\binom{t}{2}$.
3. So #edges from first $t$ to last $n - t$ nodes are $\sum_{i=1}^{t} \text{outdegree}(v_i) - \binom{t}{2}$.
4. Implies it is optimal to order nodes by increasing out-degree.
5. Time complexity $O(n \log n)$ after reading the $n^2$ size input to compute degrees.

Statistics: 30 submissions, 11 accepted, first after 01:34
Problem

Allocate toys to kid so that they do not start crying.
<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate toys to kid so that they do not start crying.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Kids have preference orderings of which toys they prefer to play with.</td>
</tr>
<tr>
<td>Problem</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Allocate toys to kid so that they do not start crying.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Kids have preference orderings of which toys they prefer to play with.</td>
</tr>
<tr>
<td>2 Envy can be viewed as toys having a preference ordering of which kids they want to be assigned to.</td>
</tr>
</tbody>
</table>
### Problem

Allocate toys to kid so that they do not start crying.

### Solution

1. Kids have preference orderings of which toys they prefer to play with.

2. Envy can be viewed as toys having a preference ordering of which kids they want to be assigned to.

3. This is the *Stable Matching Problem*. 
### Problem

Allocate toys to kid so that they do not start crying.

### Solution

1. Kids have preference orderings of which toys they prefer to play with.
2. Envy can be viewed as toys having a preference ordering of which kids they want to be assigned to.
3. This is the *Stable Matching Problem*.
4. Know how to solve it or figure out algorithm.
Problem

Allocate toys to kid so that they do not start crying.

Solution

1. Kids have preference orderings of which toys they prefer to play with.
2. Envy can be viewed as toys having a preference ordering of which kids they want to be assigned to.
3. This is the Stable Matching Problem.
4. Know how to solve it or figure out algorithm.
5. Time complexity: $O(nm \log m)$ (log factor can be eliminated)
Problem
Allocate toys to kid so that they do not start crying.

Solution

1. Kids have preference orderings of which toys they prefer to play with.
2. Envy can be viewed as toys having a preference ordering of which kids they want to be assigned to.
3. This is the *Stable Matching Problem*.
4. Know how to solve it or figure out algorithm.
5. Time complexity: \( O(nm \log m) \) (log factor can be eliminated)

Statistics: 40 submissions, 5 accepted, first after 03:02
Problem
Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 1: basic setup

1. Explore by walking towards unvisited spaces.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 1: basic setup

1. Explore by walking towards unvisited spaces.
2. Represent current knowledge as a set of map fragments.
### Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

### Solution, part 1: basic setup

1. Explore by walking towards unvisited spaces.
2. Represent current knowledge as a set of map fragments.
3. When we fall into an unknown trap, create a new fragment.
Dungeon Dawdler

Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 1: basic setup

1. Explore by walking towards unvisited spaces.
2. Represent current knowledge as a set of map fragments.
3. When we fall into an unknown trap, create a new fragment.
4. Have some logic to identify when two fragments must be the same and merge fragments when possible.

Many different approaches possible, main challenge is choosing one that minimizes implementation difficulty.
D — Dungeon Dawdler

**Problem**

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

**Solution, part 2: identifying and merging fragments**

1. Observation: if locations always explored in same order, then after falling into new traps 4 times, we have started repeating an ABABAB... or AAAAAA... pattern of traps.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 2: identifying and merging fragments

1. Observation: if locations always explored in same order, then after falling into new traps 4 times, we have started repeating an ABABAB... or AAAAAA... pattern of traps. ⇒ last and third last trap must be the same, can merge.
**Problem**

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

**Solution, part 2: identifying and merging fragments**

1. **Observation:** if locations always explored in same order, then after falling into new traps 4 times, we have started repeating an ABABAB... or AAAAAA... pattern of traps.
   \[ \Rightarrow \text{last and third last trap must be the same, can merge.} \]

2. Can also deduce how to merge two fragments if they:
   - Have traps leading to the same location (must be same trap).
   - Both have two traps.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 2: identifying and merging fragments

1. Observation: if locations always explored in same order, then after falling into new traps 4 times, we have started repeating an ABABAB... or AAAAAA... pattern of traps. ⇒ last and third last trap must be the same, can merge.

2. Can also deduce how to merge two fragments if they:
   - Have traps leading to the same location (must be same trap).
   - Both have two traps.

3. If we merge two fragments, any traps they have in same position must lead to same fragment.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 3: end game

At end we may still have two separate fragments, for two reasons:

1. Only one trap (indistinguishable from two separate identical rooms). Use connectedness guarantee to deduce single trap.
2. Cannot reach both traps from any one point. By connectedness guarantee, traps must be next to each other in a narrow corridor, use this to merge the two fragments. E.g:

```
########
#a..B? and ?A...# => #a..BA...#
#S.### ##..b# #S.###..b#
########
```
D — Dungeon Dawdler

Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 3: end game

At end we may still have two separate fragments, for two reasons:

1. Only one trap (indistinguishable from two separate identical rooms). Use connectedness guarantee to deduce single trap.
Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 3: end game

At end we may still have two separate fragments, for two reasons:

1. Only one trap (indistinguishable from two separate identical rooms). Use connectedness guarantee to deduce single trap.
2. Cannot reach both traps from any one point. By connectedness guarantee, traps must be next to each other in a narrow corridor, use this to merge the two fragments. E.g:

```
#######  #######  #######
#a..B?   and   ?A...#  =>  #a..BA...#
#S.###  ##..b#  #S.###..b#
#######  #######  #######
```
D — Dungeon Dawdler

Problem

Explore and create map of 2D maze with up to two trapdoors/teleporters that cause us to lose our bearings.

Solution, part 3: end game

At end we may still have two separate fragments, for two reasons:

1. Only one trap (indistinguishable from two separate identical rooms). Use connectedness guarantee to deduce single trap.

2. Cannot reach both traps from any one point. By connectedness guarantee, traps must be next to each other in a narrow corridor, use this to merge the two fragments. E.g:

```
#######   #######   #######
#a..B? and ?A...# => #a..BA...#
#S.### ##..b# #S.###..b#
#######   #######   #######
```

Statistics: 1 submissions, 0 accepted
Random statistics

231 submitting teams

3303 total number of submissions (1002 accepted)

9 programming languages used by teams.
Ordered by popularity:

1416 Python 2/3 (2018: 1400)
938 C++ (2018: 740)
775 Java (2018: 892)
105 C# (2018: 105)
36 Rust (2018: N/A)
28 C (2018: 6)
3 Haskell (2018: 6)
2 Ruby (2018: 0)

326 lines of code used in total by the shortest jury solutions to solve the entire problem set.
Northwestern Europe Regional Contest (NWERC)
Nov. 15-17 in Eindhoven.
Teams from Nordic, Benelux, Germany, UK, Ireland, and Estonia.
What next?

Northwestern Europe Regional Contest (NWERC)
Nov. 15-17 in Eindhoven.
Teams from Nordic, Benelux, Germany, UK, Ireland, and Estonia.

Each university sends up to two teams to NWERC to fight for spot in World Finals (June 2020 in Moscow, Russia)