NCPC 2016
Presentation of solutions

The Jury

2016-10-08
### NCPC 2016 Jury

- Per Austrin (KTH Royal Institute of Technology)
- Pål Grønås Drange (Statoil ASA)
- Antti Laaksonen (CSES)
- Ulf Lundström (Excillum)
- Jimmy Mårdell (Spotify)
- Lukáš Poláček (Google)
- Mathias Rav (Aarhus University)
- Pehr Söderman (Kattis)
- Jon Marius Venstad (Yahoo!)
Problem

Simple problem with solutions by the jury in all languages available in the contest.

Some solution (guess the language)

```prolog
solve(N1, N2) :-
    (N2-N1 > 180 -> Ans is N2-N1-360;
     N2-N1 > -180 -> Ans is N2-N1;
      Ans is N2-N1+360),
    write(Ans), nl.
```

Statistics: 535 submissions, 266 accepted, first after 00:03
### Problem
Simulate ranking system of some vaguely familiar game.

### Solution
1. Read and understand the rules.
2. Keep track of current rank, current number of stars and current number of consecutive wins.
3. Update accordingly.
4. Don’t try to be clever.
G - Game Rank

Problem
Simulate ranking system of some vaguely familiar game.

Solution
1. Read and understand the rules.
2. Keep track of current rank, current number of stars and current number of consecutive wins.
3. Update accordingly.
4. Don’t try to be clever.

Statistics: 960 submissions, 218 accepted, first after 00:17
D - Daydreaming Stockbroker

Problem

Play the stock market when knowing the future.

Solution (guess the language)

```php
fscanf(STDIN, "%d", $days);
$money = 100;
$prev = 1<<30;
for ($i = 0; $i < $days; ++$i) {
    fscanf(STDIN, "%d", $cur);
    if ($cur > $prev)
        $money += min(floor($money/$prev), 100000)*($cur-$prev);
    $prev = $cur;
}
echo $money;
```
**Problem**

Play the stock market when knowing the future.

**Solution (guess the language)**

```c
fscanf(STDIN, "%d", $days);
$money = 100;
$prev = 1<<30;
for ($i = 0; $i < $days; ++$i) {
    fscanf(STDIN, "%d", $cur);
    if ($cur > $prev)
        $money += min(floor($money/$prev), 100000)*($cur-$prev);
    $prev = $cur;
}
echo $money;
```

Statistics: 740 submissions, 183 accepted, first after 00:12
Problem

When drawing \( p \) items out of \( n + x \) items, what is probability that exactly one out of the first \( x \) items is drawn?

What is the maximum such probability over all \( x \)?

Solution
Problem

When drawing \( p \) items out of \( n + x \) items, what is probability that \textit{exactly one} out of the first \( x \) items is drawn?

What is the maximum such probability over all \( x \)?

Solution

1. Probability is

\[
\frac{\binom{x}{1} \cdot \binom{n}{p-1}}{\binom{n+x}{p}} = \{...some calculations...\} = \frac{x \cdot p}{n+1} \cdot \prod_{i=2}^{x} \frac{n-p+i}{n+i}
\]
Problem

When drawing $p$ items out of $n + x$ items, what is probability that exactly one out of the first $x$ items is drawn?

What is the maximum such probability over all $x$?

Solution

1. Probability is

$$\frac{x \cdot \binom{n}{p-1}}{\binom{n+x}{p-1}} = \{\text{...some calculations...}\} = \frac{x \cdot p}{n+1} \cdot \prod_{i=2}^{x} \frac{n-p+i}{n+i}$$

2. When going from $x-1$ to $x$, probability changes by factor

$$\frac{x}{x-1} \cdot \frac{n-p+x}{n+x}$$
Problem
When drawing $p$ items out of $n + x$ items, what is probability that exactly one out of the first $x$ items is drawn?

What is the maximum such probability over all $x$?

Solution
1. Probability is
   \[
   \frac{x}{1} \cdot \binom{n}{p-1} \cdot \binom{n+x}{p} = \{...some calculations...\} = \frac{x \cdot p}{n+1} \cdot \prod_{i=2}^{x} \frac{n-p+i}{n+i}
   \]
2. When going from $x - 1$ to $x$, probability changes by factor
   \[
   \frac{x}{x-1} \cdot \frac{n-p+x}{n+x}
   \]
3. Some calculus $\Rightarrow$ increase if $x < \frac{n}{p-1}$, decrease otherwise
   $\Rightarrow$ max happens at $x = \lceil n/(p-1) \rceil$. 

Problem Author: Per Austrin
NCPC 2016 solutions
Problem

When drawing $p$ items out of $n + x$ items, what is probability that exactly one out of the first $x$ items is drawn?

What is the maximum such probability over all $x$?

Solution

Linear $O(n/p)$ time solution:

```c
int n, p;
scanf("%d%d", &n, &p);
int x = n/(p-1);
double res = double(x*p) / (n+1);
for (int i = 2; i <= x; ++i)
    res *= double(n-p+i) / (n+i);
printf("%.9lf\n", res);
```
Problem

When drawing $p$ items out of $n + x$ items, what is probability that exactly one out of the first $x$ items is drawn?

What is the maximum such probability over all $x$?

Solution

Constant time solution:

```c
int n, p;
scanf("%d%d", &n, &p);
int x = n++/(p-1);
printf("%.9lf\n", x*p*exp(lgamma(n-p+x)-lgamma(n-p+1)-lgamma(n+x)+lgamma(n));
```

(But in order to do this in languages that don’t provide full ISO C support, one may have to implement the $\Gamma$ function oneself)
Problem

When drawing $p$ items out of $n + x$ items, what is probability that exactly one out of the first $x$ items is drawn?

What is the maximum such probability over all $x$?

Solution

Constant time solution:

```c
int n, p;
scanf("%d%d", &n, &p);
int x = n++/(p-1);
printf("%.9lf\n", x*p*exp(lgamma(n-p+x)-lgamma(n-p+1)
        -lgamma(n+x)+lgamma(n));
```

(But in order to do this in languages that don’t provide full ISO C support, one may have to implement the $\Gamma$ function oneself)

Statistics: 349 submissions, 68 accepted, first after 00:17
Problem

What is minimum number of cards to move to get list of cards in some form of order?

Solution

Try all $4! = 24$ possible ways of ordering the 4 suits.

Try all $2^4 = 16$ possible ways of choosing ascending/descending order within suits.

Now we have a fixed total order on the cards.

Maximum number of cards that can remain in place is length of longest increasing subsequence with respect to the chosen ordering.

Statistics: 77 submissions, 23 accepted, first after 00:29

Problem Author: Ulf Lundström

NCPC 2016 solutions
C - Card Hand Sorting

Problem
What is minimum number of cards to move to get list of cards in some form of order?

Solution
1. Try all $4! = 24$ possible ways of ordering the 4 suits.
2. Try all $2^4 = 16$ possible ways of choosing ascending/descending order within suits.
# Problem
What is minimum number of cards to move to get list of cards in some form of order?

## Solution
1. Try all $4! = 24$ possible ways of ordering the 4 suits.
2. Try all $2^4 = 16$ possible ways of choosing ascending/descending order within suits.
3. Now we have a fixed total order on the cards.
Problem
What is minimum number of cards to move to get list of cards in some form of order?

Solution
1. Try all $4! = 24$ possible ways of ordering the 4 suits.
2. Try all $2^4 = 16$ possible ways of choosing ascending/descending order within suits.
3. Now we have a fixed total order on the cards.
4. Maximum number of cards that can remain in place is length of longest increasing subsequence with respect to the chosen ordering.
## Problem
What is minimum number of cards to move to get list of cards in some form of order?

## Solution
1. Try all $4! = 24$ possible ways of ordering the 4 suits.
2. Try all $2^4 = 16$ possible ways of choosing ascending/descending order within suits.
3. Now we have a fixed total order on the cards.
4. Maximum number of cards that can remain in place is length of longest increasing subsequence with respect to the chosen ordering.

Statistics: 77 submissions, 23 accepted, first after 00:29
Problem
Fill horizontal and vertical blocks of squares in an initially empty grid, and output the number of unfilled connected components after each operation.

Solution
Problem

Fill horizontal and vertical blocks of squares in an initially empty grid, and output the number of unfilled connected components after each operation.

Solution

1. The problem can be modelled as a graph where each node represents a square in the grid, and two nodes are connected if the squares belong to the same component.
Problem
Fill horizontal and vertical blocks of squares in an initially empty grid, and output the number of unfilled connected components after each operation.

Solution
1. The problem can be modelled as a graph where each node represents a square in the grid, and two nodes are connected if the squares belong to the same component.
2. Each operation can be divided into modifications of single squares in the grid.
Problem

Fill horizontal and vertical blocks of squares in an initially empty grid, and output the number of unfilled connected components after each operation.

Solution

1. The problem can be modelled as a graph where each node represents a square in the grid, and two nodes are connected if the squares belong to the same component.

2. Each operation can be divided into modifications of single squares in the grid.

3. The process can be simulated efficiently using a union-find structure when all operations are done in the reverse order.
Problem

Fill horizontal and vertical blocks of squares in an initially empty grid, and output the number of unfilled connected components after each operation.

Solution

1. The problem can be modelled as a graph where each node represents a square in the grid, and two nodes are connected if the squares belong to the same component.

2. Each operation can be divided into modifications of single squares in the grid.

3. The process can be simulated efficiently using a union-find structure when all operations are done in the reverse order.

Statistics: 134 submissions, 17 accepted, first after 01:14
## Problem

Type a word using autocorrect.

## Solution

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
4. To type word, find node corresponding to longest prefix of word
5. Answer for word is distance to node + #remaining letters
6. Time complexity: linear in size of input.

Statistics: 70 submissions, 12 accepted, first after 00:43
<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type a word using autocorrect.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Realisation: may need multiple autocorrects for a single word</td>
</tr>
</tbody>
</table>

Statistics: 70 submissions, 12 accepted, first after 00:43
Problem

Type a word using autocorrect.

Solution

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects

Graph for dictionary “flame”, “flaming”, “play”, “player”
B - Bless You Autocorrect!

Problem

Type a word using autocorrect.

Solution

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS

Statistics: 70 submissions, 12 accepted, first after 00:43
Problem
Type a word using autocorrect.

Solution
1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
Type a word using autocorrect.

**Solution**

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
B - Bless You Autocorrect!

Problem
Type a word using autocorrect.

Solution
1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS

Statistics: 70 submissions, 12 accepted, first after 00:43
Problem Author: Jimmy Mårdell
NCPC 2016 solutions
Problem
Type a word using autocorrect.

Solution
1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
B - Bless You Autocorrect!

Problem

Type a word using autocorrect.

Solution

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS

Statistics: 70 submissions, 12 accepted, first after 00:43

Problem Author: Jimmy Mårdell
NCPC 2016 solutions
B - Bless You Autocorrect!

**Problem**
Type a word using autocorrect.

**Solution**
1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
4. To type word \( w \), find node corresponding to longest prefix of \( w \)
Problem
Type a word using autocorrect.

Solution
1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
4. To type word $w$, find node corresponding to longest prefix of $w$
5. Answer for $w$ is distance to node + #remaining letters

Problem Author: Jimmy Mårdell
Type a word using autocorrect.

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
4. To type word $w$, find node corresponding to longest prefix of $w$
5. Answer for $w$ is distance to node + #remaining letters
6. Time complexity: linear in size of input.
B - Bless You Autocorrect!

**Problem**

Type a word using autocorrect.

**Solution**

1. Realisation: may need multiple autocorrects for a single word
2. Build trie of dictionary, plus shortcut edges for autocorrects
3. Find shortest distance to each node with BFS
4. To type word \( w \), find node corresponding to longest prefix of \( w \)
5. Answer for \( w \) is distance to node + \( \# \) remaining letters
6. Time complexity: linear in size of input.

Statistics: 70 submissions, 12 accepted, first after 00:43
Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 1

1. Split the walks into intervals during which the two dogs don’t switch line segments.
2. Movement is relative: the two dogs walking from $P$ to $P + \Delta P$ and from $Q$ to $Q + \Delta Q$ is equivalent to one standing still at $P$ and other moving from $Q$ to $Q + \Delta Q - \Delta P$.
3. Closest distance in each such interval boils down to distance between point and line segment, basic geometric primitive.
4. Time complexity: $O(n)$. 

Statistics: 95 submissions, 11 accepted, last after 01:47
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 1

1. Split the walks into intervals during which the two dogs don’t switch line segments.
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 1

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. Movement is relative: the two dogs walking from $P$ to $P + \Delta P$ and from $Q$ to $Q + \Delta Q$ is equivalent to one standing still at $P$ and other moving from $Q$ to $Q + \Delta Q - \Delta P$.
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 1

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. Movement is relative: the two dogs walking from \( P \) to \( P + \Delta P \) and from \( Q \) to \( Q + \Delta Q \) is equivalent to one standing still at \( P \) and other moving from \( Q \) to \( Q + \Delta Q - \Delta P \).

3. Closest distance in each such interval boils down to distance between point and line segment, basic geometric primitive.
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 1

1. Split the walks into intervals during which the two dogs don’t switch line segments.
2. Movement is relative: the two dogs walking from $P$ to $P + \Delta P$ and from $Q$ to $Q + \Delta Q$ is equivalent to one standing still at $P$ and other moving from $Q$ to $Q + \Delta Q - \Delta P$.
3. Closest distance in each such interval boils down to distance between point and line segment, basic geometric primitive.
4. Time complexity: $O(n)$.
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 2 (more or less the same but different perspective)

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. In an interval where dogs walk from $P$ to $P + \Delta P$ and $Q$ to $Q + \Delta Q$, square dist. after fraction $t \in [0, 1]$ of the time is

$$\|P - Q + t(\Delta P - \Delta Q)\|_2^2 = \|P - Q\|_2^2 + 2t\langle P - Q, \Delta P - \Delta Q\rangle + t^2\|\Delta P - \Delta Q\|_2^2$$

Minimum happens at $t = -\langle P - Q, \Delta P - \Delta Q\rangle / \|\Delta P - \Delta Q\|_2^2$ (basic calculus)

Truncate to $t \in [0, 1]$, be careful with $\Delta P = \Delta Q$. 

Time complexity: $O(n)$. 

Statistics: 95 submissions, 11 accepted, first after 01:47
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 2 (more or less the same but different perspective)

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. In an interval where dogs walk from $P$ to $P + \Delta P$ and $Q$ to $Q + \Delta Q$, square dist. after fraction $t \in [0, 1]$ of the time is

$$\|P - Q + t(\Delta P - \Delta Q)\|^2_2 = \|P - Q\|^2_2 + 2t \langle P - Q, \Delta P - \Delta Q \rangle + t^2 \|\Delta P - \Delta Q\|^2_2$$

3. Minimum happens at $t = \frac{-\langle P - Q, \Delta P - \Delta Q \rangle}{\|\Delta P - \Delta Q\|^2_2}$ (basic calculus)

Truncate to $t \in [0, 1]$, be careful with $\Delta P = \Delta Q$. 
Problem

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 2 (more or less the same but different perspective)

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. In an interval where dogs walk from \( P \) to \( P + \Delta P \) and \( Q \) to \( Q + \Delta Q \), square dist. after fraction \( t \in [0, 1] \) of the time is

\[
\| P - Q + t(\Delta P - \Delta Q) \|_2^2 = \| P - Q \|_2^2 + 2t \langle P - Q, \Delta P - \Delta Q \rangle + t^2 \| \Delta P - \Delta Q \|_2^2
\]

3. Minimum happens at \( t = \frac{-\langle P - Q, \Delta P - \Delta Q \rangle}{\| \Delta P - \Delta Q \|_2^2} \) (basic calculus)

   Truncate to \( t \in [0, 1] \), be careful with \( \Delta P = \Delta Q \).

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

Two dogs move around along straight line segments, what is the closest they get to each other?

Solution 2 (more or less the same but different perspective)

1. Split the walks into intervals during which the two dogs don’t switch line segments.

2. In an interval where dogs walk from \( P \) to \( P + \Delta P \) and \( Q \) to \( Q + \Delta Q \), square dist. after fraction \( t \in [0, 1] \) of the time is
   \[
   \|P - Q + t(\Delta P - \Delta Q)\|_2^2 = \|P - Q\|_2^2 + 2t \langle P - Q, \Delta P - \Delta Q \rangle + t^2 \|\Delta P - \Delta Q\|_2^2
   \]

3. Minimum happens at \( t = \frac{-\langle P - Q, \Delta P - \Delta Q \rangle}{\|\Delta P - \Delta Q\|_2^2} \) (basic calculus)

   Truncate to \( t \in [0, 1] \), be careful with \( \Delta P = \Delta Q \).

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

Statistics: 95 submissions, 11 accepted, first after 01:47
## Problem

Compute $f(n) \mod m$, where $f(1) = 1$, $f(n) = n^{f(n-1)}$.

## Solution

1. **If $n \leq 5$:** just compute $f(n - 1)$ and then $n^{f(n-1)} \mod m$ with modular exponentiation.

2. **If $n > 5$:** ????????????
Problem
Compute \( f(n) \mod m \), where \( f(1) = 1 \), \( f(n) = n^{f(n-1)} \).

Solution
1. If \( n \leq 5 \): just compute \( f(n - 1) \) and then \( n^{f(n-1)} \mod m \) with modular exponentiation.
2. If \( n > 5 \): ???????????

Lemma
For all \( n \) and \( m \), and \( e \geq \log_2(m) \) it holds that

\[
n^e \mod m = n^\phi(m) + e \mod \phi(m) \mod m.
\]

(\( \phi(m) = Euler's \ totient \ function. \))

Proof: ugly and does not fit on slide. (Boils down to Chinese Remainder Theorem and \( \phi \) being multiplicative.)
## Problem
Compute $f(n) \mod m$, where $f(1) = 1$, $f(n) = n^{f(n-1)}$.

## Solution
1. If $n \leq 5$: just compute $f(n-1)$ and then $n^{f(n-1)} \mod m$ with modular exponentiation.
2. If $n > 5$: compute $z = f(n-1) \mod \phi(m)$ recursively. The lemma then says that $f(n) \mod m = n^{\phi(m)+z} \mod m$
Problem

Compute $f(n) \mod m$, where $f(1) = 1$, $f(n) = n^{f(n-1)}$.

Solution

1. If $n \leq 5$: just compute $f(n-1)$ and then $n^{f(n-1)} \mod m$ with modular exponentiation.

2. If $n > 5$: compute $z = f(n-1) \mod \phi(m)$ recursively. The lemma then says that $f(n) \mod m = n^{\phi(m)+z} \mod m$

3. Time complexity:
   - each recursive call dominated by time to compute $\phi(m)$: $O(\sqrt{m})$ using naive factorization.
   - recursing until $n$ becomes $\leq 5$ hopelessly slow...
Problem

Compute $f(n) \mod m$, where $f(1) = 1$, $f(n) = n^{f(n-1)}$.

Solution

1. If $n \leq 5$: just compute $f(n - 1)$ and then $n^{f(n-1)} \mod m$ with modular exponentiation.

2. If $n > 5$: compute $z = f(n - 1) \mod \phi(m)$ recursively. The lemma then says that $f(n) \mod m = n^{\phi(m)+z} \mod m$

3. Time complexity:
   - each recursive call dominated by time to compute $\phi(m)$: $O(\sqrt{m})$ using naive factorization.
   - recursing until $n$ becomes $\leq 5$ hopelessly slow...
   - ...but we can stop when we reach $m = 1$!

Problem Author: Per Austrin  NCPC 2016 solutions
**Problem**
Compute $f(n) \mod m$, where $f(1) = 1$, $f(n) = n^{f(n-1)}$.

**Solution**

1. If $n \leq 5$: just compute $f(n-1)$ and then $n^{f(n-1)} \mod m$ with modular exponentiation.
2. If $n > 5$: compute $z = f(n-1) \mod \phi(m)$ recursively. The lemma then says that $f(n) \mod m = n^{\phi(m)+z} \mod m$.
3. Time complexity:
   - each recursive call dominated by time to compute $\phi(m)$: $O(\sqrt{m})$ using naive factorization.
   - recursing until $n$ becomes $\leq 5$ hopelessly slow...
   - ...but we can stop when we reach $m = 1$!

**Lemma**: $\phi(\phi(\cdots \phi(m)))$ reaches 1 after $O(\log m)$ iterations

**Proof**: cute but does not fit on slide.
Problem

Compute \( f(n) \mod m \), where \( f(1) = 1 \), \( f(n) = n^{f(n-1)} \).

Solution

1. If \( n \leq 5 \): just compute \( f(n-1) \) and then \( n^{f(n-1)} \mod m \) with modular exponentiation.

2. If \( n > 5 \): compute \( z = f(n-1) \mod \phi(m) \) recursively. The lemma then says that \( f(n) \mod m = n^{\phi(m)+z} \mod m \)

3. Time complexity:
   - each recursive call dominated by time to compute \( \phi(m) \): \( O(\sqrt{m}) \) using naive factorization.
   - recursing until \( n \) becomes \( \leq 5 \) hopelessly slow...
   - ...but we can stop when we reach \( m = 1! \)

   **Lemma**: \( \phi(\phi(\cdots \phi(m))) \) reaches 1 after \( O(\log m) \) iterations

   **Proof**: cute but does not fit on slide.

Statistics: 47 submissions, 3 accepted, first after 00:45
Problem

Given a set of rectangles, build a tower of maximum height.

Solution

1. Make a graph: vertices = lengths, edges = given rectangles.

Orientation of rectangles gives a valid tower if no width occurs more than once ⇔ all nodes have outdegree ≤ 1.
Problem
Given a set of rectangles, build a tower of maximum height.

Solution
1. Make a graph: vertices = lengths, edges = given rectangles.

2. Encode orientation of a rectangle as direction of an edge.

Problem Author: Andreas Björklund  NCPC 2016 solutions
Given a set of rectangles, build a tower of maximum height.

Solution

1. Make a graph: vertices = lengths, edges = given rectangles.

2. Encode orientation of a rectangle as direction of an edge.

3. Orientation of rectangles gives a valid tower if no width occurs more than once ⇔ all nodes have outdegree ≤ 1
Problem Reformulated

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize $\sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v)$

Solution

In connected component with $v$ vertices and $e$ edges, average out-degree is $e/v$, so we must have $e \leq v$.

Each component is a tree, or a tree plus one edge.

Case 1: $e = v$ (tree plus one edge): each node must get out-degree exactly 1 so $\text{indeg}(v) = \text{deg}(v) - 1$.

Case 2: $e = v - 1$ (tree): one node will have out-degree 0, the rest out-degree 1. Let node with highest value get out-degree 0 in order to maximize height.

Time complexity: $O(n)$ (assuming $O(1)$ dictionary lookup).
Problem Reformulated

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize \( \sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v) \)

Solution

1. In connected component with \( v \) vertices and \( e \) edges, average out-degree is \( e/v \), so we must have \( e \leq v \)

   *Each component is a tree, or a tree plus one edge.*
Problem Reformulated

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize \( \sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v) \)

Solution

1. In connected component with \( v \) vertices and \( e \) edges, average out-degree is \( e/v \), so we must have \( e \leq v \)

   Each component is a tree, or a tree plus one edge.

2. **Case 1**: \( e = v \) (tree plus one edge): each node must get out-degree exactly 1 so \( \text{indeg}(v) = \text{deg}(v) - 1 \).
Problem Reformulated

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize \( \sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v) \)

Solution

1. In connected component with \( v \) vertices and \( e \) edges, average out-degree is \( e/v \), so we must have \( e \leq v \)

   Each component is a tree, or a tree plus one edge.

2. Case 1: \( e = v \) (tree plus one edge): each node must get out-degree exactly 1 so \( \text{indeg}(v) = \text{deg}(v) - 1 \).

3. Case 2: \( e = v - 1 \) (tree): one node will have out-degree 0, the rest out-degree 1. Let node with highest value get out-degree 0 in order to maximize height.

Time complexity: \( O(n) \) (assuming \( O(1) \) dictionary lookup).

Statistics: 38 submissions, 3 accepted, first after 02:22

Problem Author: Andreas Björklund

NCPC 2016 solutions
**Problem Reformulated**

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize \( \sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v) \)

**Solution**

1. In connected component with \( v \) vertices and \( e \) edges, average out-degree is \( e/v \), so we must have \( e \leq v \)
   
   *Each component is a tree, or a tree plus one edge.*

2. **Case 1:** \( e = v \) (tree plus one edge): each node must get out-degree exactly 1 so \( \text{indeg}(v) = \text{deg}(v) - 1 \).

3. **Case 2:** \( e = v - 1 \) (tree): one node will have out-degree 0, the rest out-degree 1. Let node with highest value get out-degree 0 in order to maximize height.

4. Time complexity: \( O(n) \) (assuming \( O(1) \) dictionary lookup).
**Problem Reformulated**

Given an undirected graph, direct edges so that each node has at most one out-going edge and maximize \( \sum_{v \in V} \text{value}(v) \cdot \text{indeg}(v) \)

**Solution**

1. In connected component with \( v \) vertices and \( e \) edges, average out-degree is \( e/v \), so we must have \( e \leq v \).  
   *Each component is a tree, or a tree plus one edge.*

2. **Case 1**: \( e = v \) (tree plus one edge): each node must get out-degree exactly 1 so \( \text{indeg}(v) = \text{deg}(v) - 1 \).

3. **Case 2**: \( e = v - 1 \) (tree): one node will have out-degree 0, the rest out-degree 1. Let node with highest value get out-degree 0 in order to maximize height.

4. Time complexity: \( O(n) \) (assuming \( O(1) \) dictionary lookup).

Statistics: 38 submissions, 3 accepted, first after 02:22
I - Interception

Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

1. Try all 4 possible ways of using the two crossings.
   2. Case 1, use both crossings: problem decomposes into two separate problems on a line, simple greedy.
   3. Case 2, use one crossing: a bit of work, better to skip and then revisit with ideas from the harder Case 3.
   4. Case 3, don't use crossings: main challenge to handle.

In order to improve on Case 1, can use at most 1 extra device for the sides. One side must use minimum number of crossings.
Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

1. Try all 4 possible ways of using the two crossings.
### Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

### Solution 1

1. Try all 4 possible ways of using the two crossings.
2. **Case 1, use both crossings**: problem decomposes into two separate problems on a line, simple greedy.
**Problem**
Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

**Solution 1**
1. Try all 4 possible ways of using the two crossings.
2. **Case 1, use both crossings**: problem decomposes into two separate problems on a line, simple greedy.
3. **Case 2, use one crossing**: a bit of work, better to skip and then revisit with ideas from the harder Case 3.
Problem
Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1
1. Try all 4 possible ways of using the two crossings.
2. **Case 1, use both crossings**: problem decomposes into two separate problems on a line, simple greedy.
3. **Case 2, use one crossing**: a bit of work, better to skip and then revisit with ideas from the harder Case 3.
4. **Case 3, don’t use crossings**: main challenge to handle.
   - In order to improve on Case 1, can use at most 1 extra device for the sides.
   - One side must use minimum number of crossings.
Problem
Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1
Guess first position $P$ to use after first crossing, $O(n)$ choices.
Problem
Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1
Try to squeeze the rest as close as possible to the crossing, $O(1)$ choices given $P$. 
Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

When one side is decided, the positions next to the crossings determine which crossing calls remain uncovered.
Problem

Given a large set of paths in a large graph with very special structure, find a minimum set of edges that hit all paths.

Solution 1

When one side is decided, the positions next to the crossings determine which crossing calls remain uncovered.
I - Interception

Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

Now want optimal solution to the other side with up to 4 extra intervals added – there are $O(1)$ choices to try for the key positions.
Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

...one horrible implementation and 20 bugs later: success!
Problem

Given large set of paths in large graph with very special structure, find minimum set of edges that hit all paths.

Solution 1

...one horrible implementation and 20 bugs later: success!
Time complexity: $O((n + m)\log n)$ (though can be made linear at cost of making implementation even more horrible)
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

Solution 2

Cover all paths contained in one of the four tails optimally, get first device as close to beginning as possible. Greedy.

Add one extra device to some tails to cut off all paths from the tail to the rest of the graph. Only 16 possibilities; try them all.

Graph and remaining paths now truncated to a circle. Can be solved with dynamic programming.

Time complexity: $O(n + m)$ (assuming bucket sort or similar)

Statistics: 7 submissions, 0 accepted, first after N/A

Problem Author: Per Austrin
NCPC 2016 solutions
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

Solution 2

1. Cover all paths contained in one of the four “tails” optimally, get first device as close to beginning as possible. *Greedy.*
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

**Solution 2**

1. Cover all paths contained in one of the four “tails” optimally, get first device as close to beginning as possible. *Greedy.*

2. Add one extra device to some tails to cut off all paths from the tail to the rest of the graph. *Only 16 possibilities; try them all.*
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

**Solution 2**

1. Cover all paths contained in one of the four “tails” optimally, get first device as close to beginning as possible. *Greedy.*

2. Add one extra device to some tails to cut off all paths from the tail to the rest of the graph. *Only 16 possibilities; try them all.*

3. Graph and remaining paths now truncated to a circle. *Can be solved with dynamic programming.*
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

**Solution 2**

1. Cover all paths contained in one of the four “tails” optimally, get first device as close to beginning as possible. *Greedy.*

2. Add one extra device to some tails to cut off all paths from the tail to the rest of the graph. *Only 16 possibilities; try them all.*

3. Graph and remaining paths now truncated to a circle. *Can be solved with dynamic programming.*

4. Time complexity: $O(n + m)$ (assuming bucket sort or similar)
If you prefer to remain sane and don’t want to take the mildly masochistic path of solution 1...

Solution 2

1. Cover all paths contained in one of the four “tails” optimally, get first device as close to beginning as possible. *Greedy.*

2. Add one extra device to some tails to cut off all paths from the tail to the rest of the graph. *Only 16 possibilities; try them all.*

3. Graph and remaining paths now truncated to a circle. *Can be solved with dynamic programming.*

4. Time complexity: $O(n + m)$ (assuming bucket sort or similar)

Statistics: 7 submissions, 0 accepted, first after N/A
Random numbers

296 teams with 722 contestants.

3045 submissions, 802 accepted (26\%)\(^1\)

34 number of seconds before end that last accepted submission was submitted.

482 number of lines of code used in total by the shortest jury solutions to solve the entire problem set.

(154 of those lines for I Interception)

\(^1\)These numbers only count submissions up to the first accepted solution on each problem for each team.
Random facts

- All but one of the problems have **near-linear solutions**
  Exception:
All but one of the problems have **near-linear solutions**

Exception: E (**Exponial**). Basic solution $O(\sqrt{m}\log m)$, input size $O(\log m)$. Very unlikely to have near-linear time solution. (Asymptotically, F (**Raffle**) is probably also an exception.)
Random facts

- All but one of the problems have **near-linear solutions**
  
  Exception: E (**Exponial**). Basic solution $O(\sqrt{m}\log m)$, input size $O(\log m)$. Very unlikely to have near-linear time solution. (Asymptotically, F (**Raffle**) is probably also an exception.)

- Required precision for K (**Dogs**) changed from $10^{-6}$ to $10^{-4}$ just a few days before the contest. Many solutions have poor precision when answer is $\approx 0$. Problem meant to be easy, not a floating point trap. Unfortunately this change was insufficient, some teams were still tripped up by precision issues.
Random facts

- All but one of the problems have near-linear solutions
  Exception: E (Exponial). Basic solution $O(\sqrt{m} \log m)$, input size $O(\log m)$. Very unlikely to have near-linear time solution. (Asymptotically, F (Raffle) is probably also an exception.)

- Required precision for K (Dogs) changed from $10^{-6}$ to $10^{-4}$ just a few days before the contest. Many solutions have poor precision when answer is $\approx 0$. Problem meant to be easy, not a floating point trap. Unfortunately this change was insufficient, some teams were still tripped up by precision issues.

- I (Interception) had the largest number of test cases (125), largest amount of test data ($\approx 325$ MB), and largest number of intentionally incorrect judge solutions (41).
All but one of the problems have near-linear solutions. Exception: E (Exponial). Basic solution $O(\sqrt{m} \log m)$, input size $O(\log m)$. Very unlikely to have near-linear time solution. (Asymptotically, F (Raffle) is probably also an exception.)

Required precision for K (Dogs) changed from $10^{-6}$ to $10^{-4}$ just a few days before the contest. Many solutions have poor precision when answer is $\approx 0$. Problem meant to be easy, not a floating point trap. Unfortunately this change was insufficient, some teams were still tripped up by precision issues.

I (Interception) had the largest number of test cases (125), largest amount of test data ($\approx 325$ MB), and largest number of intentionally incorrect judge solutions (41).

The jury wrote Python solutions for almost all problems. Exceptions: C (Card Hand Sorting) for no good reason, and I (Interception) because painful.
What now?

- Northwestern Europe Contest: November 18 in Bath (UK). Teams from Nordic, Benelux, Germany, UK, Ireland.

- Each university sends up to three(?) teams to fight for spot in World Finals (May, in Rapid City, South Dakota, USA)