NCPC 2018
Presentation of solutions

The Jury

2018-10-06
NCPC 2018 Jury

- Per Austrin (KTH Royal Institute of Technology)
- Andreas Björklund (Lund University)
- Markus Dregi (Equinor/Webstep)
- Bjarki Ágúst Guðmundsson (Syndis)
- Antti Laaksonen (CSES)
- Jimmy Mårdell (Spotify)
- Lukáš Poláček (Google)
- Torstein Strømme (University of Bergen)
- Pehr Söderman (Kattis)
- Jon Marius Venstad (Oath)
Problem

Check that input list is 1, 2, ..., n except that some elements may be replaced by “mumble”.

Obligatory Prolog Solution

solve([“mumble”|Tail], Pos) :-
    NewPos is Pos+1,
    solve(Tail, NewPos).

solve([Head|Tail], Pos) :-
    number_string(Pos, Head),
    NewPos is Pos+1,
    solve(Tail, NewPos).

solve([], _) :- write(“makes sense”).

solve(_, _) :- write(“something is fishy”).

Statistics: 453 submissions, 225 accepted, rst after 00:02
### Problem
Check that input list is $1, 2, \ldots, n$ except that some elements may be replaced by “mumble”.

### Obligatory Prolog Solution
```prolog
solve(["mumble"|Tail], Pos) :-
    NewPos is Pos+1,
    solve(Tail, NewPos).
solve([Head|Tail], Pos) :-
    number_string(Pos, Head),
    NewPos is Pos+1,
    solve(Tail, NewPos).
solve([], _) :- write("makes sense").
solve(_, _) :- write("something is fishy").
```
Problem

Check that input list is 1, 2, ..., $n$ except that some elements may be replaced by “mumble”.

Obligatory Prolog Solution

```
solve(["mumble"|Tail], Pos) :-
    NewPos is Pos+1,
    solve(Tail, NewPos).
solve([Head|Tail], Pos) :-
    number_string(Pos, Head),
    NewPos is Pos+1,
    solve(Tail, NewPos).
solve([], _) :- write("makes sense").
solve(_, _) :- write("something is fishy").
```

Statistics: 453 submissions, 225 accepted, first after 00:02
Problem

Given list of days of dirty pushes, each increasing dirtiness by 1 per day, calculate how many cleanups are needed to keep dirtiness below 20 at all times.

Solution
Problem
Given list of days of dirty pushes, each increasing dirtiness by 1 per day, calculate how many cleanups are needed to keep dirtiness below 20 at all times.

Solution
1. Simulate the process day by day
Problem

Given list of days of dirty pushes, each increasing dirtiness by 1 per day, calculate how many cleanups are needed to keep dirtiness below 20 at all times.

Solution

1. Simulate the process day by day
2. Whenever dirtiness reaches $\geq 20$, we needed a cleanup the evening before and reset everything
Problem
Given list of days of dirty pushes, each increasing dirtiness by 1 per day, calculate how many cleanups are needed to keep dirtiness below 20 at all times.

Solution
1. Simulate the process day by day
2. Whenever dirtiness reaches $\geq 20$, we needed a cleanup the evening before and reset everything
3. If dirt left at end of year, need an extra cleanup
Problem

Given list of days of dirty pushes, each increasing dirtiness by 1 per day, calculate how many cleanups are needed to keep dirtiness below 20 at all times.

Solution

1. Simulate the process day by day
2. Whenever dirtiness reaches $\geq 20$, we needed a cleanup the evening before and reset everything
3. If dirt left at end of year, need an extra cleanup

Statistics: 669 submissions, 198 accepted, first after 00:11
### Problem

Given specs for a bunch of lawnmowers, find cheapest ones with sufficiently high capacity for given lawn size.

### Solution
Problem
Given specs for a bunch of lawnmowers, find cheapest ones with sufficiently high capacity for given lawn size.

Solution
1. Mower with cutting rate $c$, cutting time $t$, recharge time $r$ cuts on average $10080ct/(t + r)$ square meters per week.
Problem
Given specs for a bunch of lawnmowers, find cheapest ones with sufficiently high capacity for given lawn size.

Solution
1. Mower with cutting rate $c$, cutting time $t$, recharge time $r$ cuts on average $10080ct/(t + r)$ square meters per week
2. Among those where this is at least lawn size, print names of cheapest mowers.
Problem

Given specs for a bunch of lawnmowers, find cheapest ones with sufficiently high capacity for given lawn size.

Solution

1. Mower with cutting rate $c$, cutting time $t$, recharge time $r$ cuts on average $10080ct/(t + r)$ square meters per week
2. Among those where this is at least lawn size, print names of cheapest mowers.

Statistics: 842 submissions, 155 accepted, first after 00:25
Intergalactic Bidding

Problem
Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

Solution
Doubling property $\Rightarrow$ sequence superincreasing: $a_i > \sum_{j=1}^{i-1} a_j$.
Implies solution must use the largest $a_i \leq t$ (because the smaller ones don't have a large enough sum).
Rinse and repeat: keep greedily adding largest number that does not cause us to exceed $t$.
Numbers are large, either use language that has big integers or write yourself (only need addition and comparisons, which are very easy to implement).

Statistics: 328 submissions, 91 accepted, 1st after 00:24

Problem Author: Bjarki Ágúst Guðmundsson
NCPC 2018 solutions
Problem
Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

Solution
1. Doubling property $\Rightarrow$ sequence superincreasing: $a_i > \sum_{j=1}^{i-1} a_j$
Problem
Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

Solution
1. Doubling property $\Rightarrow$ sequence superincreasing: $a_i > \sum_{j=1}^{i-1} a_j$
2. Implies solution must use the largest $a_i \leq t$ (because the smaller ones don’t have a large enough sum)
Problem

Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

Solution

1. Doubling property $\Rightarrow$ sequence superincreasing: $a_i > \sum_{j=1}^{i-1} a_j$
2. Implies solution must use the largest $a_i \leq t$ (because the smaller ones don’t have a large enough sum)
3. Rinse and repeat – keep greedily adding largest number that does not cause us to exceed $t$
Intergalactic Bidding

Problem
Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

Solution
1. Doubling property $\Rightarrow$ sequence superincreasing: $a_i > \sum_{j=1}^{i-1} a_j$
2. Implies solution must use the largest $a_i \leq t$ (because the smaller ones don’t have a large enough sum)
3. Rinse and repeat – keep greedily adding largest number that does not cause us to exceed $t$
4. Numbers are large, either use language that has big integers or write yourself (only need addition and comparisons, which are very easy to implement)
### Problem

Given sequence of positive integers $a_1, \ldots, a_n$ such that $a_i \geq 2a_{i-1}$, find subset that sums to $t$.

### Solution

1. Doubling property $\Rightarrow$ sequence *superincreasing*: $a_i \geq \sum_{j=1}^{i-1} a_j$
2. Implies solution must use the largest $a_i \leq t$ (because the smaller ones don’t have a large enough sum)
3. Rinse and repeat – keep greedily adding largest number that does not cause us to exceed $t$
4. Numbers are large, either use language that has big integers or write yourself (only need addition and comparisons, which are very easy to implement)

Statistics: 328 submissions, 91 accepted, first after 00:24
<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine how many 0's and 1's the string has. <strong>Example:</strong> ( x ) zeros gives ( x ) ( \times ) ( x ) ( - ) ( 1 ) ( \times ) ( x ) ^{2} ) many 00, solve for ( x ). Greedily find suitable positions for 0's from left to right. All other positions will have 1's. If this process fails, there are no solutions. Watch out for small special cases: if number of 00 is 0, two solutions ( x = 0 ) and ( x = 1 ) solution must be non-empty (you can also handle small cases using brute force).</td>
</tr>
</tbody>
</table>
J — Jumbled String

Problem
Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.

Solution
1. Determine how many 0’s and 1’s the string has
   ($x$ zeros gives $\frac{x(x-1)}{2}$ many “00”, solve for $x$)
## Problem

Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.

## Solution

1. Determine how many 0’s and 1’s the string has
   \( x \text{ zeros gives } \frac{x(x-1)}{2} \text{ many “00”, solve for } x \)
2. Greedily find suitable positions for 0’s from left to right. All other positions will have 1’s.
Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.

1. Determine how many 0’s and 1’s the string has. \(x\) zeros gives \(\frac{x(x-1)}{2}\) many “00”, solve for \(x\).
2. Greedily find suitable positions for 0’s from left to right. All other positions will have 1’s.
3. If this process fails, there are no solutions.
Problem

Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.

Solution

1. Determine how many 0’s and 1’s the string has
   \( x \) zeros gives \( \frac{x(x-1)}{2} \) many “00”, solve for \( x \)

2. Greedily find suitable positions for 0’s from left to right. All other positions will have 1’s.

3. If this process fails, there are no solutions.

4. Watch out for small special cases:
   - if number of “00” is 0, two solutions \( x = 0 \) and \( x = 1 \)
   - solution must be non-empty
   (you can also handle small cases using brute force).
Problem
Create a bit string that has given numbers of subsequences “00”, “01”, “10” and “11”, or report that this is not possible.

Solution
1. Determine how many 0’s and 1’s the string has
   \( x \) zeros gives \( \frac{x(x-1)}{2} \) many “00”, solve for \( x \)
2. Greedily find suitable positions for 0’s from left to right. All other positions will have 1’s.
3. If this process fails, there are no solutions.
4. Watch out for small special cases:
   - if number of “00” is 0, two solutions \( x = 0 \) and \( x = 1 \)
   - solution must be non-empty
   (you can also handle small cases using brute force).

Statistics: 359 submissions, 38 accepted, first after 00:22
Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing $d$ damage one at a time randomly to living minions?

Solution

Way to slowing exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.

Speed up using dynamic programming / memoisation.

Problem: still slow, number of states is $7^{10} \approx 300$ millions

Optimization: order of your/opp onent’s minions does not affect answer noralize healths to be in sorted order. Number of possible states is reduced to $(5 + 7 - 1)(7 - 1) = 213444$
Problem
Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing $d$ damage one at a time randomly to living minions?

Solution
1. Way too slow exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.
E — Explosion Exploit

Problem
Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing $d$ damage one at a time randomly to living minions?

Solution
1. Way too slow exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.
2. Speed up using dynamic programming / memoisation.
### Problem
Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing $d$ damage one at a time randomly to living minions?

### Solution
1. Way too slow exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.
2. Speed up using dynamic programming / memoisation.

**Problem:** still slow, number of states is $7^{10} \approx 300$ millions
### Problem

Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing \(d\) damage one at a time randomly to living minions?

### Solution

1. Way too slow exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.

2. Speed up using dynamic programming / memoisation.  
   **Problem:** still slow, number of states is \(7^{10} \approx 300\) millions

3. **Optimization:** order of your/opponent’s minions does not affect answer – normalize healths to be in sorted order.  
   Number of possible states is reduced to \(\left(\frac{5+7-1}{7-1}\right)^2 = 213\,444\)
E — Explosion Exploit

Problem

Given healths of your own and your opponent’s minions in a card game, what is probability that all opponent’s minions die after dealing $d$ damage one at a time randomly to living minions?

Solution

1. Way too slow exhaustive search solution: for each living minion, decrease its health by one and recursively compute answer for the updated healths, then average to get answer.

2. Speed up using dynamic programming / memoisation. **Problem:** still slow, number of states is $7^{10} \approx 300$ millions

3. **Optimization:** order of your/opponent’s minions does not affect answer – normalize healths to be in sorted order. Number of possible states is reduced to \( \left( \frac{5 + 7 - 1}{7 - 1} \right)^2 = 213444 \)

Statistics: 144 submissions, 41 accepted, first after 00:34

Problem Author: Jimmy Mårdell

NCPC 2018 solutions
Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?
Problem
Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 1 [Dynamic Programming]
1. For any leaf $v$, two possibilities:
Problem
Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 1 [Dynamic Programming]

1. For any leaf $v$, two possibilities:
   1. it is the only node with its color:
      
      $$k \cdot f( T \setminus v, k - 1)$$ such colorings
## Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

## Solution 1 [Dynamic Programming]

1. For any leaf $v$, two possibilities:
   1. **it is the only node with its color:**
      \[ k \cdot f(T \setminus v, k - 1) \] such colorings
   2. **some other node has same color:**
      \[ (k - 1) \cdot f(T \setminus v, k) \] such colorings
      (can pick any color except the color of parent node)
K — King’s Colors

Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 1 [Dynamic Programming]

1. For any leaf $v$, two possibilities:
   1. it is the only node with its color:
      $k \cdot f(T \setminus v, k - 1)$ such colorings
   2. some other node has same color:
      $(k - 1) \cdot f(T \setminus v, k)$ such colorings
      (can pick any color except the color of parent node)

2. We see that answer only depends on $n$ and $k$, not on structure of $T$ and get recurrence

$$f(n, k) = k \cdot f(n - 1, k - 1) + (k - 1) \cdot f(n - 1, k)$$
Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 1 [Dynamic Programming]

1. For any leaf $v$, two possibilities:
   1. it is the only node with its color:
      $k \cdot f(T \setminus v, k - 1)$ such colorings
   2. some other node has same color:
      $(k - 1) \cdot f(T \setminus v, k)$ such colorings
      (can pick any color except the color of parent node)

2. We see that answer only depends on $n$ and $k$, not on structure of $T$ and get recurrence

   $$f(n, k) = k \cdot f(n - 1, k - 1) + (k - 1) \cdot f(n - 1, k)$$

3. Compute in your favorite way in $O(nk)$ time
Problem
Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 2 [Inclusion-Exclusion]
1. Number of $c$-colorings (not necessarily using all $c$ colors) is $c(c - 1)^{n-1}$: root can have any color and as we go down the tree each node has $c - 1$ choices
Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 2 [Inclusion-Exclusion]

1. Number of $c$-colorings (not necessarily using all $c$ colors) is $c(c - 1)^{n-1}$: root can have any color and as we go down the tree each node has $c - 1$ choices.

2. By principle of inclusion-exclusion, answer is

$$f(n, k) = \sum_{c=1}^{k} (-1)^{k-c} \binom{k}{c} c(c - 1)^{n-1}$$
Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 2 [Inclusion-Exclusion]

1. Number of $c$-colorings (not necessarily using all $c$ colors) is $c(c - 1)^{n-1}$: root can have any color and as we go down the tree each node has $c - 1$ choices.

2. By principle of inclusion-exclusion, answer is

$$f(n, k) = \sum_{c=1}^{k} (-1)^{k-c} \binom{k}{c} c(c - 1)^{n-1}$$

3. Can compute in $O(k \log n)$ time
K — King’s Colors

Problem

Given tree $T$ on $n$ vertices, how many $k$-colorings does it have that use all $k$ colors?

Solution 2 [Inclusion-Exclusion]

1. Number of $c$-colorings (not necessarily using all $c$ colors) is $c(c-1)^{n-1}$: root can have any color and as we go down the tree each node has $c-1$ choices.

2. By principle of inclusion-exclusion, answer is

$$f(n, k) = \sum_{c=1}^{k} (-1)^{k-c} \binom{k}{c} c(c-1)^{n-1}$$

3. Can compute in $O(k \log n)$ time

Statistics: 123 submissions, 33 accepted, first after 00:30
D — Delivery Delays

Problem
Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution

1. Find shortest distances between all pairs of nodes in the graph (using Dijkstra's algorithm for each vertex)
2. Binary search for answer
3. To check if delay possible, let $L_D(i)$ be latest possible start time for delivering orders $i, i + 1, \ldots, k$ with max delay $D$
4. Compute $L_D(i)$: guess how many orders $j$ to bring together with $i$, simulate delivering them, then use value of $L_D(i + j)$

Time complexity is $O(nm \log m + k^2 \log D_{\max})$.
Problem

Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution

1. **Find shortest distances between all pairs of nodes** in the graph (using Dijkstra’s algorithm for each vertex)
Problem
Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution
1. Find shortest distances between all pairs of nodes in the graph (using Dijkstra’s algorithm for each vertex)
2. Binary search for answer $D$
D — Delivery Delays

Problem

Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution

1. Find shortest distances between all pairs of nodes in the graph (using Dijkstra’s algorithm for each vertex)
2. Binary search for answer $D$
3. To check if delay $D$ possible, let $L_D(i)$ be latest possible start time for delivering orders $i, i + 1, \ldots, k$ with max delay $D$
D — Delivery Delays

Problem

Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution

1. **Find shortest distances between all pairs of nodes** in the graph (using Dijkstra’s algorithm for each vertex)
2. **Binary search for answer** $D$
3. To check if delay $D$ possible, let $L_D(i)$ be latest possible start time for delivering orders $i, i + 1, \ldots, k$ with max delay $D$
4. **Compute $L_D(i)$**: guess how many orders $j$ to bring together with $i$, simulate delivering them, then use value of $L_D(i + j)$

Time complexity is $O(nm \log m + k^2 \log D_{\text{max}})$. 
Delivery Delays

Problem
Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution
1. Find shortest distances between all pairs of nodes in the graph (using Dijkstra’s algorithm for each vertex)
2. Binary search for answer \( D \)
3. To check if delay \( D \) possible, let \( L_D(i) \) be latest possible start time for delivering orders \( i, i+1, \ldots, k \) with max delay \( D \)
4. Compute \( L_D(i) \): guess how many orders \( j \) to bring together with \( i \), simulate delivering them, then use value of \( L_D(i+j) \)
5. Time complexity is \( O(nm \log m + k^2 \log D_{\text{max}}) \).
Problem

Given list of orders made and when they are ready to be delivered from origin to destination, what is smallest possible maximum delay to deliver them in first-come-first-served order?

Solution

1. **Find shortest distances between all pairs of nodes** in the graph (using Dijkstra’s algorithm for each vertex)

2. **Binary search for answer** $D$

3. To check if delay $D$ possible, let $L_D(i)$ be latest possible start time for delivering orders $i, i+1, \ldots, k$ with max delay $D$

4. **Compute $L_D(i)$**: guess how many orders $j$ to bring together with $i$, simulate delivering them, then use value of $L_D(i+j)$

5. Time complexity is $O(nm \log m + k^2 \log D_{\text{max}})$.

Statistics: 40 submissions, 10 accepted, first after 01:25
Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

Solution

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.
2. Maintain an array $H[w] = \text{height of highest frog pile that can carry a weight of } w$ (for $w$ up to max weight).
3. For each frog $(l_i, w_i, h_i)$ by decreasing weight (time reversal):
   1. Frog escapes if $l_i + H[w_i] > d$
   2. Update $H[w] = \max(H[w], h_i + H[w_i + w_i])$ for $1 \leq w \leq w_i - 1$
4. Time complexity is $O(n \log n + \sum w_i)$
**Problem**

Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

**Solution**

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.
Problem

Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

Solution

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.

2. Maintain array $H[w] =$ height of highest frog pile that can carry a weight of $w$ (for $w$ up to max weight).
Problem

Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

Solution

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.

2. Maintain array $H[w] =$ height of highest frog pile that can carry a weight of $w$ (for $w$ up to max weight).

3. For each frog $(l_i, w_i, h_i)$ by decreasing weight (time reversal):

   1. Frog escapes if $l_i + H[w_i] > d$
   2. Update $H[w] = \max(H[w], h_i + H[w_i + w])$ for $1 \leq w \leq w_i - 1$
Problem

Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

Solution

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.
2. Maintain array $H[w] =$ height of highest frog pile that can carry a weight of $w$ (for $w$ up to max weight).
3. For each frog $(l_i, w_i, h_i)$ by decreasing weight (time reversal):
   1. Frog escapes if $l_i + H[w_i] > d$
   2. Update $H[w] = \max(H[w], h_i + H[w_i + w])$ for $1 \leq w \leq w_i - 1$
4. Time complexity is $O(n \log n + \sum w_i)$
A — Altruistic Amphibians

Problem
Given leap capacities, weights, and heights of a set of frogs, decide how many frogs can escape a pit of given depth $d$ if they build piles of frogs to elevate each other. No frog can carry its own weight.

Solution

1. A frog can only help lighter ones, so can assume the frogs leave pit in order of increasing weight.
2. Maintain array $H[w] =$ height of highest frog pile that can carry a weight of $w$ (for $w$ up to max weight).
3. For each frog $(l_i, w_i, h_i)$ by decreasing weight (time reversal):
   1. Frog escapes if $l_i + H[w_i] > d$
   2. Update $H[w] = \max(H[w], h_i + H[w_i + w])$ for $1 \leq w \leq w_i - 1$
4. Time complexity is $O(n \log n + \sum w_i)$

Statistics: 55 submissions, 1 accepted, first after 04:29
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 1 [Explicit construction]

1. The $n = 1$ case: classic round robin scheme
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 1 [Explicit construction]

1. The $n = 1$ case: classic round robin scheme
2. Idea: make “pseudorounds” where all players with index $i$ play against all players with index $j \neq i$ on other teams, using basic round robin schedule on $n$ players.
Problem
Given \( m \) teams with \( n \) players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 1 [Explicit construction]

1. The \( n = 1 \) case: classic round robin scheme
2. Idea: make “pseudorounds” where all players with index \( i \) play against all players with index \( j \neq i \) on other teams, using basic round robin schedule on \( n \) players.
3. Add games where players with index \( i \) meet each other via round robin schedule on \( m \) teams:
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 1 [Explicit construction]

1. The $n = 1$ case: classic round robin scheme
2. Idea: make “pseudorounds” where all players with index $i$ play against all players with index $j \neq i$ on other teams, using basic round robin schedule on $n$ players.
3. Add games where players with index $i$ meet each other via round robin schedule on $m$ teams:
   - $n$ even: just add them after pseudorounds.
**Problem**

Given \( m \) teams with \( n \) players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

**Solution 1 [Explicit construction]**

1. The \( n = 1 \) case: classic round robin scheme
2. Idea: make “pseudorounds” where all players with index \( i \) play against all players with index \( j \neq i \) on other teams, using basic round robin schedule on \( n \) players.
3. Add games where players with index \( i \) meet each other via round robin schedule on \( m \) teams:
   1. \( n \) even: just add them after pseudorounds.
   2. \( n \) odd: schedule them during pseudoround where \( i \) had bye.
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 1 [Explicit construction]

1. The $n = 1$ case: classic round robin scheme
2. Idea: make “pseudorounds” where all players with index $i$ play against all players with index $j \neq i$ on other teams, using basic round robin schedule on $n$ players.
3. Add games where players with index $i$ meet each other via round robin schedule on $m$ teams:
   1. $n$ even: just add them after pseudorounds.
   2. $n$ odd: schedule them during pseudoround where $i$ had bye.
   3. $n$ and $m$ odd: use one last round to collect remaining games.
Problem

Given \( m \) teams with \( n \) players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 2 [General solution]

1. Construct graph with \( m \cdot n \) nodes representing all players, with edges between players from different teams.
**Problem**

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

**Solution 2 [General solution]**

1. Construct graph with $m \cdot n$ nodes representing all players, with edges between players from different teams.
2. A schedule using $d = n(m - 1) + 1$ days is an *edge coloring* of the graph with $d$ colors.
Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

Solution 2 [General solution]

1. Construct graph with $m \cdot n$ nodes representing all players, with edges between players from different teams.

2. A schedule using $d = n(m - 1) + 1$ days is an edge coloring of the graph with $d$ colors.

3. **Vizing’s Theorem**: every graph has an edge coloring using $\Delta + 1$ colors where $\Delta$ is max degree. Exactly what we need.
## Problem
Given \( m \) teams with \( n \) players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

## Solution 2 [General solution]

1. Construct graph with \( m \cdot n \) nodes representing all players, with edges between players from different teams.

2. A schedule using \( d = n(m-1) + 1 \) days is an edge coloring of the graph with \( d \) colors.

3. **Vizing’s Theorem**: every graph has an edge coloring using \( \Delta + 1 \) colors where \( \Delta \) is max degree. Exactly what we need.

4. Use efficient algorithm for finding a \((\Delta + 1)\)-edge coloring (Misra-Gries). Naive implementation sufficiently fast.
### Problem

Given $m$ teams with $n$ players per team, construct a round based schedule so all players play against all players from all other teams, such that each player has at most one bye (free round).

### Solution 2 [General solution]

1. Construct graph with $m \cdot n$ nodes representing all players, with edges between players from different teams.

2. A schedule using $d = n(m - 1) + 1$ days is an edge coloring of the graph with $d$ colors.

3. **Vizing’s Theorem**: *every graph has an edge coloring using $\Delta + 1$ colors where $\Delta$ is max degree.* Exactly what we need.

4. Use efficient algorithm for finding a $(\Delta + 1)$-edge coloring ([Misra-Gries](https://en.wikipedia.org/wiki/Misra%E2%80%93Gries_algorithm)). Naive implementation sufficiently fast.

Statistics: 10 submissions, 0 accepted
Firing the Phaser

Problem
Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (1/3)
Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (1/3)

1. Idea: given the line on which the optimal segment lies, we get a relatively easy one-dimensional problem about intervals.
## Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

## Solution (1/3)

1. **Idea:** given the line on which the optimal segment lies, we get a relatively easy one-dimensional problem about intervals.

2. **So we just have to find a small candidate set of lines (= pairs of points) to try.**
Firing the Phaser

Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (2/3)

1. Possible pitfall: assume optimal solution passes through two corners.
Problem
Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length \( \ell \).

Solution (2/3)
1. Possible **pitfall**: assume optimal solution passes through two corners. **This is false:**
Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)

**Lemma:** can assume that optimal solution

- passes through a corner of some rectangle, and
Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)

1. **Lemma**: can assume that optimal solution
   1. passes through a corner of some rectangle, and
   2. either passes through another corner, or ends along the sides of two other rectangles (as in previous picture).
Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)

1. **Lemma**: can assume that optimal solution
   1. passes through a corner of some rectangle, and
   2. either passes through another corner, or ends along the sides of two other rectangles (as in previous picture).

2. So we can take as candidates all lines of these two types. **First type** is trivial to generate.
Problem

Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)

1. **Lemma**: can assume that optimal solution
   - passes through a corner of some rectangle, and
   - either passes through another corner, or ends along the sides of two other rectangles (as in previous picture).

2. So we can take as candidates all lines of these two types. **First type** is trivial to generate.

3. For **type 2**, need to find all lines of length $\ell$ through some point $P$, with one endpoint on line $L_1$ and another on line $L_2$. 
Firing the Phaser

Problem
Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)

1. **Lemma**: can assume that optimal solution
   1. passes through a corner of some rectangle, and
   2. either passes through another corner, or ends along the sides of two other rectangles (as in previous picture).

2. So we can take as candidates all lines of these two types. **First type** is trivial to generate.

3. For **type 2**, need to find all lines of length $\ell$ through some point $P$, with one endpoint on line $L_1$ and another on line $L_2$.

4. Working out the math this becomes a degree 4 polynomial equation (can also solve it numerically with less math).
Problem
Given set of axis-aligned rectangles, find max number of rectangles that can be intersected by a straight line segment of length $\ell$.

Solution (3/3)
1. **Lemma**: can assume that optimal solution
   1. passes through a corner of some rectangle, and
   2. either passes through another corner, or ends along the sides of two other rectangles (as in previous picture).
2. So we can take as candidates all lines of these two types. **First type** is trivial to generate.
3. For **type 2**, need to find all lines of length $\ell$ through some point $P$, with one endpoint on line $L_1$ and another on line $L_2$.
4. Working out the math this becomes a degree 4 polynomial equation (can also solve it numerically with less math).

Statistics: 19 submissions, 0 accepted
Random statistics

232 submitting teams

3149 total number of submissions (792 accepted)

6 programming languages used by teams

Ordered by popularity: Python 2/3 (1400), Java (892), C++ (740), C# (105), C (6), Haskell (6)

(Top 3 languages are in reverse order from the “usual” one! Python, Java and C# increased in popularity, all other languages decreased.)

381 number of lines of code used in total by the shortest jury solutions to solve the entire problem set. (Much smaller than usual.)
What next?

Northwestern Europe Regional Contest (NWERC)

Nov. 23-25 in Eindhoven (Netherlands)

Teams from Nordic, Benelux, Germany, UK, Ireland, and Estonia.
Northwestern Europe Regional Contest (NWERC)

Nov. 23-25 in Eindhoven (Netherlands)

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 (April 2019 in Porto, Portugal)