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)

B — Baby Bites

Problem

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

B — Baby Bites

Problem

Check that input list is $1, 2, \ldots, n$ 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").
```

B — Baby Bites

Problem

Check that input list is $1, 2, \ldots, n$ 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

C — Code Cleanups

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

C — Code Cleanups

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

C — Code Cleanups

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 ≥ 20, we needed a cleanup the evening before and reset everything

C — Code Cleanups

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 ≥ 20, we needed a cleanup the evening before and reset everything
3. If dirt left at end of year, need an extra cleanup

C — Code Cleanups

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 ≥ 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

H — House Lawn

Problem

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

Solution

H — House Lawn

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

H — House Lawn

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.

H — House Lawn

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

I — 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

I — 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*: $a_i > \sum_{j=1}^{i-1} a_j$

I — 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)

I — 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

I — 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)

I — 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)

Statistics: 328 submissions, 91 accepted, first after 00:24

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

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)

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

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)
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.

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)
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).

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)
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

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

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.

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

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 $\binom{5+7-1}{7-1}^2 = 213\,444$

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 $\binom{5+7-1}{7-1}^2 = 213\,444$

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

K — King's Colors

Problem

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

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:

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

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)

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

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

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

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

3. Compute in your favorite way in $O(nk)$ 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

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

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

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

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

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)

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 $L_D(i+j)$

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

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

Statistics: 40 submissions, 10 accepted, first after 01:25

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

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.

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).

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 $1 \leq w \leq w_i - 1$

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 $O(n \log n + \sum w_i)$

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

G — Game Scheduling

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).

G — Game Scheduling

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

G — Game Scheduling

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.

G — Game Scheduling

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:

G — Game Scheduling

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.

G — Game Scheduling

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.

G — Game Scheduling

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.

G — Game Scheduling

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.

G — Game Scheduling

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.

G — Game Scheduling

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 Δ is max degree.* Exactly what we need.

G — Game Scheduling

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 Δ is max degree.* Exactly what we need.
4. Use efficient algorithm for finding a $(\Delta + 1)$-edge coloring (Misra-Gries). Naive implementation sufficiently fast.

G — Game Scheduling

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 Δ is max degree.* Exactly what we need.
4. Use efficient algorithm for finding a $(\Delta + 1)$-edge coloring (Misra-Gries). Naive implementation sufficiently fast.

Statistics: 10 submissions, 0 accepted

F — 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 ℓ.

Solution (1/3)

F — 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 ℓ.

Solution (1/3)

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

F — 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 ℓ.

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.

F — 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 ℓ.

Solution (2/3)

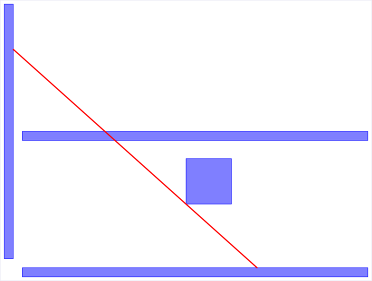
1. Possible **pitfall**: assume optimal solution passes through two corners.

Problem

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

Solution (2/3)

1. Possible **pitfall**: assume optimal solution passes through two corners. This is false:



F — 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 ℓ.

Solution (3/3)

1. **Lemma**: can assume that optimal solution
   1. passes through a corner of some rectangle, and

F — 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 ℓ.

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).

F — 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 ℓ.

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.

F — 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 ℓ.

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 ℓ through some point P, with one endpoint on line L_1 and another on line L_2

F — 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 ℓ.

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 ℓ 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)

F — 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 ℓ.

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 ℓ 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.

What next?

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)