NWERC 2017 presentation of solutions

The Jury

NWERC 2017 Jury

- François Aubry (Université catholique de Louvain)
- Per Austrin (KTH Royal Institute of Technology)
- Gregor Behnke (Ulm University)
- Thomas Beuman (Leiden University)
- Jeroen Bransen (Chordify)
- Jim Grimmett (LifeJak)
- Timon Knigge (Google)
- Robin Lee (Google)
- Lukáš Poláček (Google)
- Johan Sannemo (Google)
- Tobias Werth (Google)
- Paul Wild (FAU Erlangen-Nürnberg)

Big thanks to our test solvers

- Bjarki Ágúst Guðmundsson (Reykjavík University)
- Jan Kuipers (AppTornado)
- Jimmy Mårdell (Spotify)
- Tobias Polzer (Google)

B: Boss Battle

Problem

In a circular room with n pillars, alternatingly:

- You throw a bomb killing anything behind 3 consecutive pillars.
- The boss can move to an adjacent pillar.

Find the minimal number of moves to kill the boss.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.
3. When only 3 pillars remain, a single bomb will defeat the boss.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.
3. When only 3 pillars remain, a single bomb will defeat the boss.
4. Total is thus $n - 3 + 1 = n - 2$.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.
3. When only 3 pillars remain, a single bomb will defeat the boss.
4. Total is thus $n - 3 + 1 = n - 2$.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i+2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.
3. When only 3 pillars remain, a single bomb will defeat the boss.
4. Total is thus $n-3+1=n-2$.

Pitfall

1. If $n \leq 3$ the answer is 1.

B: Boss Battle

Solution

1. Just throw bomb i to pillar $i + 2 \mod n$.
2. Each bomb gives a new position where the boss cannot be.
3. When only 3 pillars remain, a single bomb will defeat the boss.
4. Total is thus $n - 3 + 1 = n - 2$.

Pitfall

1. If $n \leq 3$ the answer is 1.

Statistics: 150 submissions, 118 accepted

D: Dunglish

Problem

Given a sentence and a dictionary, count the number of possible translations or in case of a unique one give the actual translation.

D: Dunglish

Problem

Given a sentence and a dictionary, count the number of possible translations or in case of a unique one give the actual translation.

Solution

1. First do the counting only.
2. Keep two numbers: the number of possible correct translations c, and the total number of possible translations t, both initially 1.
3. For each word in the sentence with c_i correct and w_i incorrect translations, do $c = c * c_i$ and $t = t * (c_i + w_i)$.
4. If t equals 1, do another pass to find the solution, otherwise output c and $t - c$.

D: Dunglish

Possible pitfalls

1. Generating all possible translations (too slow!)
2. The answer can be as big as 2^{60} (does not fit in `int`!)

D: Dunglish

Possible pitfalls

1. Generating all possible translations (too slow!)
2. The answer can be as big as 2^{60} (does not fit in `int`!)

Statistics: 231 submissions, 112 accepted

H: High Score

Problem

In a game with *a*, *b* and *c* tokens, the players score is:

$$a^2 + b^2 + c^2 + 7 \cdot \min(a, b, c)$$

Given *d wildcard* tokens, divide them over *a*, *b* and *c* to get the maximum score.

H: High Score

Problem

In a game with a, b and c tokens, the players score is:

$$a^2 + b^2 + c^2 + 7 \cdot \min(a, b, c)$$

Given d *wildcard* tokens, divide them over a, b and c to get the maximum score.

Solution

1. Let $a \geq b \geq c$
2. Brute-force over $\min(a, b, c)$ and add the rest to a
3. Only needed to check small values of $\min(a, b, c)$

H: High Score

Problem

In a game with a, b and c tokens, the players score is:

$$a^2 + b^2 + c^2 + 7 \cdot \min(a, b, c)$$

Given d *wildcard* tokens, divide them over a, b and c to get the maximum score.

Solution

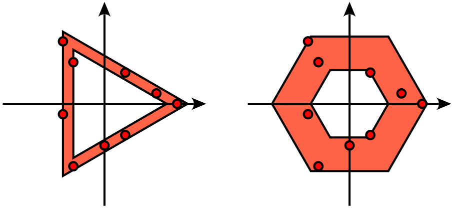
1. Let $a \geq b \geq c$
2. Brute-force over $\min(a, b, c)$ and add the rest to a
3. Only needed to check small values of $\min(a, b, c)$

Statistics: 378 submissions, 99 accepted

G: Glyph Recognition

Problem

- Given some points in the plane, and $3 \leq k \leq 8$, find two regular k-gons containing all the points between them.
- All polygons are centered at the origin.
- All polygons have the same orientation.
- Over all values of k, maximize the ratio $\frac{A_{\text{inner}}}{A_{\text{outer}}}$ of polygon areas.



G: Glyph Recognition

Solution

- Fix a value of k and let p be one of the points.
- Let $r_k(p)$ be the circumradius of the k-gon that has p on its perimeter.
- There are at least two ways to compute $r_k(p)$.

G: Glyph Recognition

Solution

- Fix a value of k and let p be one of the points.
- Let $r_k(p)$ be the circumradius of the k-gon that has p on its perimeter.
- There are at least two ways to compute $r_k(p)$.
  1. Trigonometry: Let (r, α) be the polar coordinates of p.
     - Using rotational symmetry, we can assume $0 \leq \alpha < \frac{2\pi}{k}$.
     - Then
$$r_k(p) = r \cdot \frac{\cos(\alpha - \frac{\pi}{k})}{\cos(\frac{\pi}{k})}.$$

G: Glyph Recognition

Solution

- Fix a value of k and let p be one of the points.
- Let $r_k(p)$ be the circumradius of the k-gon that has p on its perimeter.
- There are at least two ways to compute $r_k(p)$.
  1. Trigonometry: Let (r, α) be the polar coordinates of p.
     - Using rotational symmetry, we can assume $0 \leq \alpha < \frac{2\pi}{k}$.
     - Then
       $$r_k(p) = r \cdot \frac{\cos(\alpha - \frac{\pi}{k})}{\cos(\frac{\pi}{k})}.$$
  2. Binary search on the answer:
     - The vertices of a regular k-gon with radius r are
       $$(r \cdot \cos(i \cdot \frac{2\pi}{k}), r \cdot \sin(i \cdot \frac{2\pi}{k})), \quad 0 \leq i < k.$$
     - Use some pre-written point-in-polygon test to check if this polygon contains p.

G: Glyph Recognition

Solution

- Fix a value of k and let p be one of the points.
- Let $r_k(p)$ be the circumradius of the k-gon that has p on its perimeter.
- There are at least two ways to compute $r_k(p)$.
- With all the $r_k(p)$ found, the answer can be computed with two simple loops.

G: Glyph Recognition

Solution

- Fix a value of k and let p be one of the points.
- Let $r_k(p)$ be the circumradius of the k-gon that has p on its perimeter.
- There are at least two ways to compute $r_k(p)$.
- With all the $r_k(p)$ found, the answer can be computed with two simple loops.

Statistics: 205 submissions, 71 accepted
60% binary search vs. 40% others

K: Knockout Tournament

Problem

Set up a knockout tournament such that Dale's probability of winning is maximized.
Calculate this maximum probability.

K: Knockout Tournament

Solution

1. Suppose that we know the first round pairings.
2. Given i and k it is easy to compute the range of possible opponents of player i for the k-th match.
3. Let $p(k, i) =$ *prob that player i wins his k-th match*
4. By definition $p(0, i) = a(i)/(a(i) + a(j))$ where j is the opponent of i.
5. For $k \geq 1$, we have

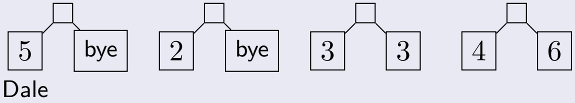
$$p(k, i) = p(k-1, i) \cdot \sum_j p(k-1, j) \cdot a(i)/(a(i) + a(j))$$

where j ranges over the possible opponents of i for match k.

K: Knockout Tournament

Finding the first round pairings

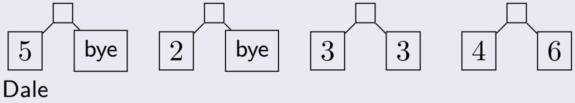
1. We want face strong players as late as possible.
2. That will maximize the chances of them loosing before.
3. Sort the players by non-decreasing rating.
4. Put Dale first and add byes if necessary.
5. **Example:** 6 players with ratings 5, 3, 6, 2, 4, 3.



K: Knockout Tournament

Finding the first round pairings

1. We want face strong players as late as possible.
2. That will maximize the chances of them loosing before.
3. Sort the players by non-decreasing rating.
4. Put Dale first and add byes if necessary.
5. **Example:** 6 players with ratings 5, 3, 6, 2, 4, 3.



Statistics: 82 submissions, 43 accepted

I: Installing Apps

Problem

Each phone app has a download d and a storage size s. Find the largest set of apps that can be installed on a phone.

Solution

1. **Observation:** Fix set S of apps. If S can not be installed in the order given by $(s_i - d_i)$, then it can not be installed at all.
2. Sort all apps by $s_i - d_i$.
3. Knapsack-like dynamic programming: $dp(i, k)$ is the minimum storage space needed to install k apps on the phone out of the first i apps in the sorted order.
4. We can reconstruct the solution by remembering extra information in each field of the DP table.

I: Installing Apps

Problem

Each phone app has a download d and a storage size s. Find the largest set of apps that can be installed on a phone.

Solution

1. **Observation:** Fix set S of apps. If S can not be installed in the order given by $(s_i - d_i)$, then it can not be installed at all.
2. Sort all apps by $s_i - d_i$.
3. Knapsack-like dynamic programming: $dp(i, k)$ is the minimum storage space needed to install k apps on the phone out of the first i apps in the sorted order.
4. We can reconstruct the solution by remembering extra information in each field of the DP table.

Statistics: 49 submissions, ? accepted

A: Ascending Photo

Problem

Cut up and re-order an array to make it weakly increasing.
The number of cuts should be minimal.

A: Ascending Photo

Solution: greedy?

1. Compress unique heights into a contiguous range of $[0, m]$.

A: Ascending Photo

Solution: greedy?

1. Compress unique heights into a contiguous range of $[0, m]$.
2. Ignore repeated heights, ie. $12223343 \rightarrow 12343$.

A: Ascending Photo

Solution: greedy?

1. Compress unique heights into a contiguous range of $[0, m]$.
2. Ignore repeated heights, ie. $12223343 \to 12343$.
3. Now if we only have one of each height, it's easy:

A: Ascending Photo

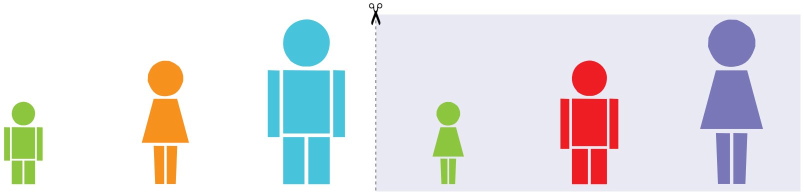
Solution: greedy?

1. Compress unique heights into a contiguous range of $[0, m]$.
2. Ignore repeated heights, ie. $12223343 \rightarrow 12343$.
3. Now if we only have one of each height, it's easy:
   3.1 The answer is the number of $(a, a+1)$ for which $h_a + 1 \neq h_{a+1}$.

A: Ascending Photo

Solution: greedy?

1. Compress unique heights into a contiguous range of $[0, m]$.
2. Ignore repeated heights, ie. $12223343 \to 12343$.
3. Now if we only have one of each height, it's easy:
   3.1 The answer is the number of $(a, a+1)$ for which $h_a + 1 \neq h_{a+1}$.
4. This will **not** work with duplicate heights.



A: Ascending Photo

Solution: attempt 2

1. We'll need to use dynamic programming.

A: Ascending Photo

Solution: attempt 2

1. We'll need to use dynamic programming.
2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

A: Ascending Photo

Solution: attempt 2

1. We'll need to use dynamic programming.
2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

    2.1 If h_p is unique,

$$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q = p + 1 \\ 0, & \text{otherwise} \end{cases} \forall q \mid h_q = h_p + 1)$$

A: Ascending Photo

Solution: attempt 2

1. We'll need to use dynamic programming.
2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

   2.1 If h_p is unique,

   $$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q = p + 1 \\ 0, & \text{otherwise} \end{cases} \forall q \mid h_q = h_p + 1)$$

   2.2 If h_p is not unique,

   $$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p + 1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

A: Ascending Photo

Solution: attempt 2

1. We'll need to use dynamic programming.
2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

    2.1 If h_p is unique,

    $$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q = p + 1 \\ 0, & \text{otherwise} \end{cases} \forall q \mid h_q = h_p + 1)$$

    2.2 If h_p is not unique,

    $$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p + 1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

3. Complexity: $O(N^2)$. Too slow! But it's a start.

A: Ascending Photo

Solution: attempt 2

2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

   2.2 If h_p is not unique,

$$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p+1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

3. Complexity: $O(N^2)$. Too slow! But it's a start.

A: Ascending Photo

Solution: attempt 2

2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

   2.2 If h_p is not unique,

$$\mathrm{dp}[p] = \mathbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p+1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

3. Complexity: $O(N^2)$. Too slow! But it's a start.
4. Note the calculations for a and b are very similar if $h_a = h_b$.

A: Ascending Photo

Solution: attempt 2

2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

   2.2 If h_p is not unique,

$$\mathrm{dp}[p] = \textbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p+1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

3. Complexity: $O(N^2)$. Too slow! But it's a start.
4. Note the calculations for a and b are very similar if $h_a = h_b$.
5. Transform the list of values for a into the values for b quickly by only updating entries for $a + 1$ and $b + 1$.

A: Ascending Photo

Solution: attempt 2

2. Let $\mathrm{dp}[p]$ be the number of non-cuts beginning with person p.

   2.2 If h_p is not unique,

$$\mathrm{dp}[p] = \textbf{max}(\mathrm{dp}[q] + \begin{cases} 1, & \text{if } q \neq p+1 \text{ and } h_{q-1} = h_p \\ 0, & \text{otherwise} \end{cases} \ldots)$$

3. Complexity: $O(N^2)$. Too slow! But it's a start.
4. Note the calculations for a and b are very similar if $h_a = h_b$.
5. Transform the list of values for a into the values for b quickly by only updating entries for $a+1$ and $b+1$.
6. Complexity: $O(N \log N)$.

A: Ascending Photo

Possible pitfalls

1. Greedy. It looks promising, but it's not going to work.

A: Ascending Photo

Possible pitfalls

1. Greedy. It looks promising, but it's not going to work.

Statistics: 72 submissions, ? accepted

F: Factor-Free Tree

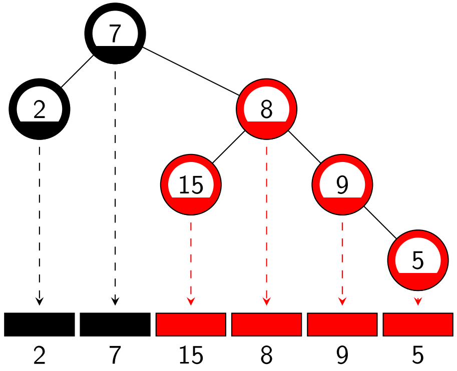
Problem

Find a binary tree that has the given sequence a_i as its inorder traversal.
The value in any vertex must be coprime with the values of each of its ancestors.

F: Factor-Free Tree

Solution

For any vertex i in the final binary tree, the subtree rooted at i will be present as a contiguous subsequence $l \le i \le r$ in the inorder sequence.



F: Factor-Free Tree

Solution

The problem has a recursive structure:

- The root of the tree will be an ancestor of all vertices and must therefore be coprime to all other values.
- After picking a root, the sequence will be split into two halves, representing the left and right subtrees of the root, respectively.
- These subtrees/sequences are also factor-free trees, so now we have split the problem into two smaller subproblems. (Note: if a sequence has multiple valid roots it does not matter which one you pick. Proof left as an exercise for the reader.)

F: Factor-Free Tree

Solution

How do we find a valid root for a (sub)range? Let's solve an easier question first: can a_i be the root of $l \leq i \leq r$?

- Factorize all numbers in the input. Modified sieve of Eratosthenes can do this in $\mathcal{O}(\log k)$ time for a number with k prime factors.
- For each position we can now easily find the nearest positions that share any prime factor with a_i, say $l_i < i$ and $r_i > i$.
- Now position i can be the root of $[l, r]$ if and only if $l_i < l$ and $r < r_i$.

F: Factor-Free Tree

Solution

This suggests the following algorithm (omitting construction):

```
Function Solve (l, r) is
    for i from l to r do
        if i can be root of $[l, r]$ then
            return Solve (l, $i-1$) and Solve ($i+1$, r)
        end
    end
    return $l \geq r$
end
```

How fast does this algorithm run? If the root is at the end of the sequence, the recursion tree will be very skewed with linear work at each level, and we end up doing $\mathcal{O}(n^2)$ work (think worst-case performance of QuickSort).

F: Factor-Free Tree

Solution

The solution: simply start scanning from both sides simultaneously! Intuitively, this is faster because:

- If the root is near one of the ends of the sequence, we split into unequal subproblems, but we can do so quickly.
- If the root is near the middle, it takes a long time to find, but we cut into to roughly equisized subproblems.

This algorithm can be proven to run in $\mathcal{O}(n \log n)$ time, proof omitted for brevity (but you can ask one of your friendly neighbourhood jury members if you are interested).

F: Factor-Free Tree

Solution

The solution: simply start scanning from both sides simultaneously! Intuitively, this is faster because:

- If the root is near one of the ends of the sequence, we split into unequal subproblems, but we can do so quickly.
- If the root is near the middle, it takes a long time to find, but we cut into to roughly equisized subproblems.

This algorithm can be proven to run in $\mathcal{O}(n \log n)$ time, proof omitted for brevity (but you can ask one of your friendly neighbourhood jury members if you are interested).

Statistics: 28 submissions, ? accepted

J: Juggling Troupe

Problem

Jugglers standing in a row throw balls to their neighbours while they have more than a single ball. Give the final configuration.

J: Juggling Troupe

Idea

1. Simplify the problem.

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?
3. Let $L < i$ be the index of the last 0 to the left of i.

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?
3. Let $L < i$ be the index of the last 0 to the left of i.
4. Let $R > i$ be the index of the first 0 to the right of i.

$$\begin{array}{c|c|c|c|c|c|c|c|c|c}
 & L & & & & i & & & R & \\
\hline
\cdots & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 0 & \cdots \\
\hline
\end{array}$$

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?
3. Let $L < i$ be the index of the last 0 to the left of i.
4. Let $R > i$ be the index of the first 0 to the right of i.

$$\begin{array}{c|c|c|c|c|c|c|c|c|c}
 & L & & & & i & & & R & \\
\hline
\cdots & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 0 & \cdots \\
\hline
\end{array}$$

5. The result will have a 0 at position $L + R - i$.

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?
3. Let $L < i$ be the index of the last 0 to the left of i.
4. Let $R > i$ be the index of the first 0 to the right of i.

	L				i			R	
$\cdots$	0	1	1	1	2	1	1	0	$\cdots$

5. The result will have a 0 at position $L + R - i$.
6. All other values in $[L, R]$ are 1's.

J: Juggling Troupe

Idea

1. Simplify the problem.
2. What if we only have one 2 in the input at position i?
3. Let $L < i$ be the index of the last 0 to the left of i.
4. Let $R > i$ be the index of the first 0 to the right of i.

	L				i			R	
$\cdots$	0	1	1	1	2	1	1	0	$\cdots$

5. The result will have a 0 at position $L + R - i$.
6. All other values in $[L, R]$ are 1's.
7. Other indexes remain unchanged.

	L			$L + R - i$				R	
$\cdots$	1	1	1	0	1	1	1	1	$\cdots$

J: Juggling Troupe

Solution

1. Solve the 2's individually.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position *i* with a 2, query the BST for *L* and *R*.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position i with a 2, query the BST for L and R.
4. Keep two sentinel zeros at positions -1 and n.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position *i* with a 2, query the BST for *L* and *R*.
4. Keep two sentinel zeros at positions -1 and *n*.
5. Remove *L* and *R* (don't remove sentinels).

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position i with a 2, query the BST for L and R.
4. Keep two sentinel zeros at positions -1 and n.
5. Remove L and R (don't remove sentinels).
6. Add a zero at $L + R - 1$ if it does not contain another 2.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position i with a 2, query the BST for L and R.
4. Keep two sentinel zeros at positions -1 and n.
5. Remove L and R (don't remove sentinels).
6. Add a zero at $L + R - 1$ if it does not contain another 2.
7. Output 0's for indexes in T and 1's elsewhere.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position i with a 2, query the BST for L and R.
4. Keep two sentinel zeros at positions -1 and n.
5. Remove L and R (don't remove sentinels).
6. Add a zero at $L + R - 1$ if it does not contain another 2.
7. Output 0's for indexes in T and 1's elsewhere.

J: Juggling Troupe

Solution

1. Solve the 2's individually.
2. Keep track of the positions of the zeros in a BST.
3. For each position i with a 2, query the BST for L and R.
4. Keep two sentinel zeros at positions -1 and n.
5. Remove L and R (don't remove sentinels).
6. Add a zero at $L + R - 1$ if it does not contain another 2.
7. Output 0's for indexes in T and 1's elsewhere.

Statistics: 49 submissions, ? accepted

J: Juggling Troupe

Example

$Zeroes = \{4\}$

0	1	2	3	4	5	6	7
1	2	1	1	0	2	1	2

J: Juggling Troupe

Example

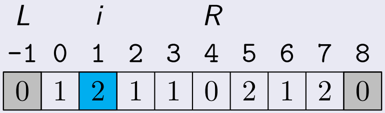
Zeroes $= \{-1, 4, 8\}$

-1	0	1	2	3	4	5	6	7	8
0	1	2	1	1	0	2	1	2	0

J: Juggling Troupe

Example

$Zeroes = \{-1, 4, 8\}$



$L + R - i = -1 + 4 - 1 = 2$

J: Juggling Troupe

Example

$Zeroes = \{-1, 2, 8\}$

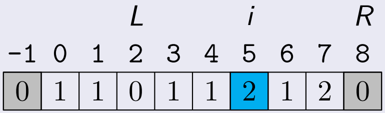
L		i			R				
-1	0	1	2	3	4	5	6	7	8
0	1	1	0	1	1	2	1	2	0

$L + R - i = -1 + 4 - 1 = 2$

J: Juggling Troupe

Example

$Zeroes = \{-1, 2, 8\}$



$L + R - i = 2 + 8 - 5 = 5$

J: Juggling Troupe

Example

$Zeroes = \{-1, 5, 8\}$

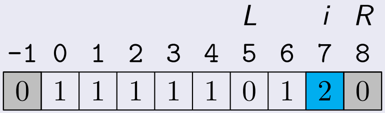
$$\begin{array}{cccccccccc}
 & & & L & & & i & & & R \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\boxed{0} & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 2 & \boxed{0}
\end{array}$$

$L + R - i = 2 + 8 - 5 = 5$

J: Juggling Troupe

Example

$Zeroes = \{-1, 5, 8\}$



$L + R - i = 5 + 8 - 7 = 6$

J: Juggling Troupe

Example

$\mathit{Zeroes} = \{-1, 6, 8\}$

$$\begin{array}{cccccccccc}
 & & & & & & L & & i & R \\
\texttt{-1} & \texttt{0} & \texttt{1} & \texttt{2} & \texttt{3} & \texttt{4} & \texttt{5} & \texttt{6} & \texttt{7} & \texttt{8} \\
\hline
\multicolumn{1}{|c|}{0} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{0} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{0} \\
\hline
\end{array}$$

$L + R - i = 5 + 8 - 7 = 6$

J: Juggling Troupe

Example

$Zeroes = \{-1, 6, 8\}$

0	1	2	3	4	5	6	7
1	1	1	1	1	1	0	1

J: Juggling Troupe

Example

$Zeroes = \{-1, 6, 8\}$

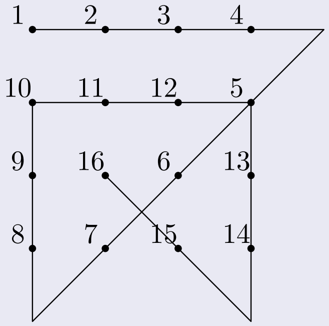
0	1	2	3	4	5	6	7
1	1	1	1	1	1	0	1

Statistics: FIXME submissions, FIXME accepted

C: Connect the Dots

Problem

Draw minimum line segments to cover 16 points in order, without lifting your pen.



C: Connect the Dots

Greedy solution

Let $S(i)$ be the set of angles in which you could enter point i in order to minimize number of segments to get to point i. It turns out that this set is a single interval.
Suppose we go from i to $i + 1$. To compute $S(i + 1)$, consider the ways in which we could leave point i:

- Go directly from point i to $i + 1$. If this is possible you should always do it.

C: Connect the Dots

Greedy solution

If this is not possible with any of the angles in $S(i)$, we need to go through i, then start a new line towards $i+1$. Then $S(i+1)$ can be computed by considering the extremes:

- Continuing arbitrarily far from i along the angle that is the first endpoint of $S(i)$.
- Continuing arbitrarily far from i along the angle that is the second endpoint of $S(i)$.

This will give you the corresponding set $S(i+1)$.

C: Connect the Dots

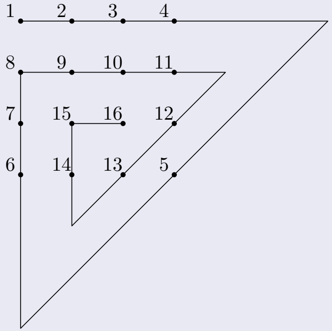
Possible pitfalls

Note that $S(i)$ may either be a single angle, or a half-open interval.

C: Connect the Dots

Possible pitfalls

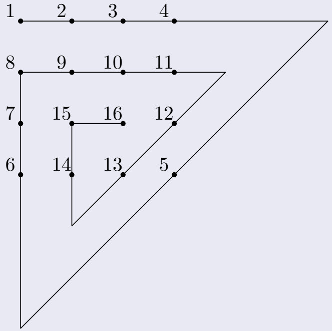
Every segment should pass through two points? No (sample 2)



C: Connect the Dots

Possible pitfalls

Every segment should pass through two points? No (sample 2)



Statistics: 62 submissions, ? accepted

E: English Restaurant

Problem

Random groups of sizes between 1 and g people arrive at the restaurant. Each group occupies the smallest table that fits the group, or leaves if there is no such table.
Find the average occupancy of after t groups have arrived.

Solution, part 1

1. Sort tables by capacity.
2. Calculate expected occupancy $E(i, j)$ for consecutive intervals of tables between i and j, conditioned upon the interval being fully occupied and the rest being empty.
3. Add t virtual tables of capacity g, holding the people leaving restaurant.

E: English Restaurant

Solution, part 2

4. Dynamic programming, from smallest intervals to the longest. Pick the last table k occupied in an interval $[i, j]$. Intervals $[i, k-1]$ and $[k+1, j]$ have been occupied before. There are $\binom{j-i+1}{k-1}$ ways of interleaving these two parts.
5. Use consecutive occupancies to calculate non-consecutive occupancies: $F(k, \ell)$ is the average occupancy of the first k tables when ℓ of those tables are occupied.
6. F is calculated similarly to E.
7. Final answer is $F(n+t, t)$ – expected occupancy of the $n+t$ tables when t of them are occupied.

E: English Restaurant

Solution, part 2

4. Dynamic programming, from smallest intervals to the longest. Pick the last table k occupied in an interval $[i, j]$. Intervals $[i, k-1]$ and $[k+1, j]$ have been occupied before. There are $\binom{j-i+1}{k-1}$ ways of interleaving these two parts.
5. Use consecutive occupancies to calculate non-consecutive occupancies: $F(k, \ell)$ is the average occupancy of the first k tables when ℓ of those tables are occupied.
6. F is calculated similarly to E.
7. Final answer is $F(n+t, t)$ – expected occupancy of the $n+t$ tables when t of them are occupied.

Statistics: 8 submissions, ? accepted

Random numbers produced by the jury

836	number of posts made in the jury's forum. (NWERC 2016: 1081)
894	commits made to the problem set repository. (NWERC 2016: 964)
347	number of lines of code used in total by the shortest judge solutions to solve the entire problem set. (NWERC 2016: 370)
23.7	average number of jury solutions per problem, including incorrect ones. (NWERC 2016: 20.6)