Special School Seminar
How to Solve the 2018 Staff-Student Programming Contest Problems

Bobby McCann, Vlad-Andrei Munteanu, Sean Parker, Ivan Donat Pupovac, Patrick Sava, Ian Pratt-Hartmann

2nd May, 2018
A: Building Capacity (54 submissions; 2 correct solutions; credit—Ivan D. Pupovac)

- We only need to consider interval endpoints.
- Each interval has two points: \((\text{opening\_time}, \text{start})\) and \((\text{ending\_time}, \text{end})\).
- Sort these by timestamp and then go through them, increasing count at each start and decreasing at each end of interval, keeping in mind the maximum count.
- Complexity is \(O(n \log n)\), where \(n\) is the number of intervals.
- Common pitfall: Loop through all time units (note: time goes up to \(10^{18}\)).
B: Dice (18 submissions; 11 correct solutions; credit—Artem Ripatti)

• We need to determine if we can restore the numbers for all the die in the stack.
• Iterate over all the die from the top of the stack to the bottom
• For each die, check that:

\[ x \neq a \land x \neq b \land x \neq 7 - a \land x \neq 7 - b \]
C: Ice cream (0 submissions; credit—Ian P-H)

- Represent the problem as a flow network with nodes representing *boys* and *flavours* plus source and sink:

- We want to compute the value of the maximum (integral) flow through this network.
- This can be done using the *Ford-Fulkerson* algorithm, which relies on the remarkable *min-cut/max-flow theorem*.
- Complexity is $O(C \cdot (n + m)^3)$, where $C$ is maximum numerical value.
D: Path (40 submissions; 13 correct solutions; credit—Ivan D. Pupovac)

- This is a simple modular arithmetic problem.
- Solution:

\[ \text{result} = n \mod w \]

where: \( n \) number of tiles, \( w \) width of path

- Complexity is \( O(1) \).

- Common pitfall: Not using a data type which allows for big enough integers.
E: Wibbles (25 submissions; 0 correct solutions; credit—Ian P-H)

- Most submitters realized this concerns the Fibonacci sequence:
  \[
  f_1 = f_2 = 1, \quad f_{n+2} = f_{n+1} + f_n \quad (n \geq 1)
  \]

- The number of wibbles (including wiblets) on Wibbleheim at time \( t \) (first wibble arrives at \( t = 0 \)) is \( f_{t+2} \) for all \( t \geq 0 \).
- Find smallest \( t \) s.t. \( f_{t+2} \) has at least \( d \) digits. (\( d \leq \frac{2^{32}}{5} \)).
- We use the Binet formula:
  \[
  f_n = \frac{((1 + \sqrt{5})/2)^n - ((1 - \sqrt{5})/2)^n}{\sqrt{5}}.
  \]

- The second term is, in effect, the ‘rounding error’.
- Hence, in (at least) “almost all” cases, the number of digits in \( f_n \) is number of digits to the left of the decimal point in the first term.
E: Wibbles (25 submissions; 0 correct solutions; credit—Ian P-H)

- Most submitters realized this concerns the Fibonacci sequence:

\[ f_1 = f_2 = 1, \quad f_{n+2} = f_{n+1} + f_n \quad (n \geq 1) \]

- The number of wibbles (including wiblets) on Wibbleheim at time \( t \) (first wibble arrives at \( t = 0 \)) is \( f_{t+2} \) for all \( t \geq 0 \).
- Find smallest \( t \) s.t. \( f_{t+2} \) has at least \( d \) digits. \( (d \leq 2^{32}/5) \).
- We use the Binet formula:

\[ f_n = \frac{(1 + \sqrt{5})/2)^n - (1 - \sqrt{5}/2)^n}{\sqrt{5}}. \]

- The second term is, in effect, the ‘rounding error’.
- Hence, in (at least) “almost all” cases, the number of digits in \( f_n \) is number of digits to the left of the decimal point in the first term.
• Writing $\alpha = (1 + \sqrt{5})/2$, number of digits in $f_n$ is therefore

$$\lfloor \log(\alpha^n/\sqrt{5}) \rfloor + 1 = \lfloor n \log \alpha - \log \sqrt{5} \rfloor + 1.$$  

• We want the smallest $t$ such that

$$\lfloor (t + 2) \log \alpha - \log \sqrt{5} \rfloor + 1 \geq d,$$

i.e. the smallest $t$ such that

$$(t + 2) \log \alpha - \log \sqrt{5} + 1 \geq d,$$

i.e. such that

$$t \geq (d + \log \sqrt{5} - 1)/\log \alpha - 2$$

i.e.

$$\lceil (d + \log \sqrt{5} - 1)/\log \alpha \rceil - 2.$$
F: Fractal (0 submissions; problem from NWERC 2009)

- Polyline made from \( n \) points, find the point \( f \) along the polyline on the fractal of depth \( d \).

- We don’t have to calculate all the points of the fractal!
- Just need to transform the original polyline \( d - 1 \) times.
- How can we implement this?
Complex Numbers!

- Multiplication for scaling and rotating, addition for translation.

- $p_1 \ldots p_n$, the points of the polyline as complex numbers,
- $c_1 \ldots c_n$, the normalised representation
  \[ c_j = \frac{(p_j - p_1)}{(p_n - p_1)} \]
- Do $d - 1$ times:
  - Traverse $f$ along the polyline, we end up somewhere between $p_a$ and $p_{a+1}$. $f$ becomes the fraction of the distance we are between these points.
  - Transform $p_1 \ldots p_n$ to map to the line segment that connects $p_a$ to $p_{a+1}$. $p_j = c_j(p_{a+1} - p_a) + p_a$
  - Traverse $f$ along the line to the final point.
G: Virus (1 submission; 0 correct solutions; credit—Andrei Grigorean)

- We need to save \( n - k \) nodes by cutting the minimum number of edges.
- Observe that an edge is relevant iff no edge can be removed from it to the root.
- Use dynamic programming and DFS to find the size of each subtree then reverse sort.
- Finally, iterate over sorted subtrees, count number of cuts until \( totalNodesSaved + k \geq n \)
H: Banknotes (69 submissions; 6 correct solutions; credit—CSAcademy)

- Let \( x \) be number of notes worth £\( a \), \( n - x \) be number of notes worth £\( b \), we form the equation:

\[
s = x \cdot a + (n - x) \cdot b
\]

We can solve for \( x \):

\[
x = \frac{(s - n \cdot b)}{(a - b)}
\]

**Pitfalls:**

- Not checking that \( a - b \) divides \( s - n \cdot b \)
- Not checking \( 0 \leq \frac{(s - n \cdot b)}{(a - b)} \leq n \)
We want to find the Minimum Spanning Tree.

The weight of an edge is the Euclidean distance between the corresponding islands.

Use Prim’s Algorithm or Kruskal’s Algorithm; both of which are greedy algorithms, to find the MST.

Complexity is $O(|E| \log |E|) = O(|E| \log |V|)$, where $|E|$ is the number of edges, $|V|$ the number of vertices.
• Build buckets of different dishes—starting with one dish in each bucket—and sort them by size.
• Sort the customers according to the number of dishes they can afford.
• For each person, starting with the one who can afford the most dishes, try to find the best bucket.
• The best bucket for a person is the smallest bucket whose size is greater or equal to the number of dishes that person can afford.
• If there are any dishes left in that bucket, move them to the next smaller bucket. (Notice that, if we do this, the buckets remain disjoint.)
• If there is no bucket with this property, just give the biggest bucket to the current person.