ACM ICPC 2016–2017
Northeastern European Regional Contest
Problems Review

December 4, 2016

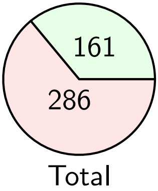
Problems summary

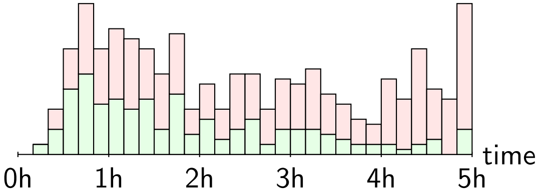
- Recap: 228 teams, 13 problems, 5 hours,
- This review assumes the knowledge of the problem statements (published separately on http://neerc.ifmo.ru/ web site)
- Summary table on the next slide lists problem name and stats
  - **author** — author of the original idea
  - **acc** — number of teams that had solved the problem (gray bar denotes a fraction of the teams that solved the problem)
  - **runs** — number of total attempts
  - **succ** — overall successful attempts rate (percent of accepted submissions to total, also shown as a bar)

Problems summary (2)

problem name	author	acc/runs	succ
Abbreviation	Roman Elizarov	161 /447	36%
Binary Code	Niyaz Nigmatullin	5 /76	6%
Cactus Construction	Pavel Kunyavsky	8 /20	40%
Delight for a Cat	Gennady Korotkevich	1 /2	50%
Expect to Wait	Vitaliy Aksenov	62 /208	29%
Foreign Postcards	Niyaz Nigmatullin	115 /261	44%
Game on Graph	Pavel Kunyavsky	3 /12	25%
Hard Refactoring	Elena Kryuchkova	168 /526	31%
Indiana Jones and the Uniform Cave	Georgiy Korneev	0 /13	0%
Jenga Boom	Georgiy Korneev	59 /396	14%
Kids Designing Kids	Pavel Mavrin	7 /38	18%
List of Primes	Mikhail Dvorkin	12 /28	42%
Mole Tunnels	Borys Minaiev	2 /4	50%

Problem A. Abbreviation





	Java	C++	Python	Total
Accepted	9	142	10	161
Rejected	39	230	17	286
Total	48	372	27	447

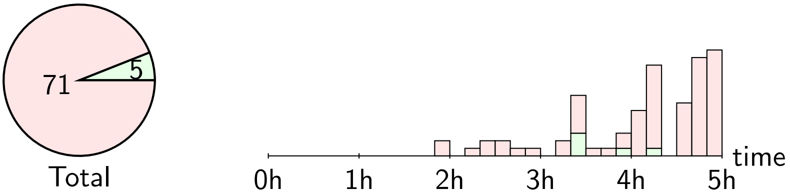
solution	team	att	time	size	lang
Fastest		1	16	2,015	C++
Shortest		1	19	386	Python
Max atts.		9	292	4,689	C++

Problem A. Abbreviation

- One of the two simple problems in the contest
- Many ways to write a solution
- One of the ways is this:
  - Split each line into words and separators
  - Identify *capitalized words* per problem statement
  - Find the longest sequences of two or more words separated by a single space
  - Perform replacements per the problem statement
- The shortest solution from Ural Federal University 4 (Ankudinov, Borzunov, Stikhin) uses a single regular expression with a replacement function:

```
re.sub(r'\b[A-Z][a-z]+( [A-Z][a-z]+\b)+', abbr, text)
```

Problem B. Binary Code



	Java	C++	Python	Total
Accepted	1	4	0	5
Rejected	1	70	0	71
Total	2	74	0	76

solution	team	att	time	size	lang
Fastest		2	204	4,918	Java
Shortest		2	204	4,918	Java
Max atts.		5	205	4,940	C++

Problem B. Binary Code

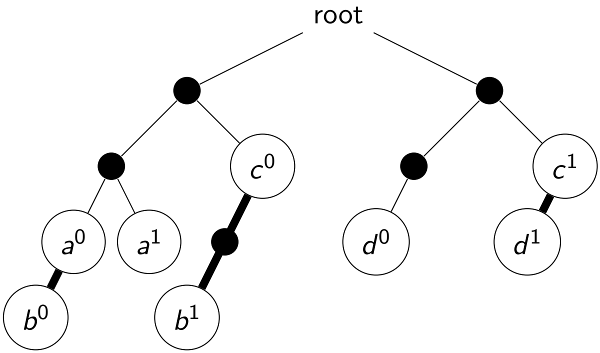
- Solution outline: solve the problem by converting it into an instance of 2-SAT problem
  1. Build a *trie* of the given strings
  2. Define two variables v_i^0 and $v_i^1 = \bar{v}_i^1$ for each word s_i that contains a "?"
     - v_i^0 is *true* and v_i^1 is *false* when "?" is replaced with "0" in s_i
     - v_i^0 is *false* and v_i^1 is *true* when "?" is replaced with "1" in s_i
  3. Create a graph with two nodes for each string. One node for v_i^0, the other for v_i^1
  4. Use the trie to convert binary code constraints into 2-SAT problem instance using *implications*
  5. Use the classical 2-SAT solution algorithm via the graph algorithm to find strongly connected components in *implications graph*

Problem B. Binary Code — Build a trie

- Follow the classic approach, build a *binary* trie
- For strings with "?" add *both* replacements for "?" into a trie
- At the terminal nodes for the string s with "?" put the corresponding variable (s^0 or s^1 depending on replacement)
- At the terminal nodes for the string s *without* "?" put the separate variable T that is always *true*
  - If more than one string without "?" ends at the same node of the trie, the answer is "`NO`"

Problem B. Binary Code — Trie example

- Trie for the first example



a: 00?
b: 0?00
c: ?1
d: 1?0

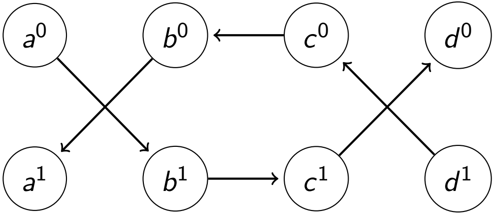
Implications

a^0 **nand** b^0:
- $a^0 \to b^1$
- $b^0 \to a^1$

c^0 **nand** b^1:
- $c^0 \to b^0$
- $b^1 \to c^1$

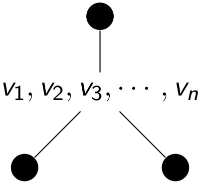
c^1 **nand** d^1:
- $c^1 \to d^0$
- $d^1 \to c^0$

Problem B. Binary Code — Implications graph example



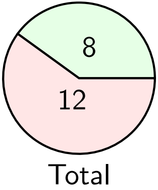
- Classic 2-SAT algorithm finds the answer or decides that it is impossible
- The sample output assigns *true* to a^0, b^1, c^1, d^0

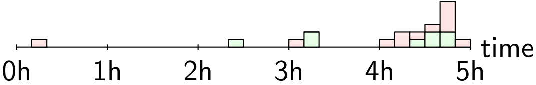
Problem B. Binary Code — Many terminals at node



- Node in a trie can have many terminals (variables) at one node
- At most one of them can be present in a binary code
- We can express this constraint in $O(n)$ implications using n additional variable pairs
  - Define additional variable r_i to be *true* if and only if at least one $v_j, j \geq i$ is *true*
- Or exclude all v_i and v_j pairs, but return "`NO`" answer when n is more than the depth of this node in a trie plus one

Problem C. Cactus Construction



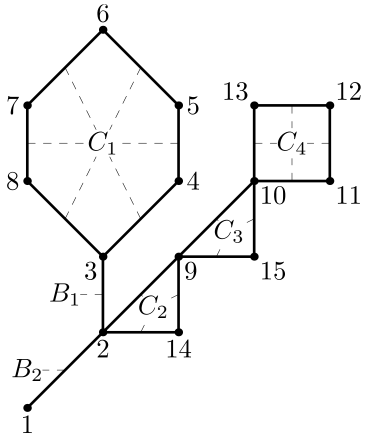


	Java	C++	Python	Total
Accepted	0	8	0	8
Rejected	0	12	0	12
Total	0	20	0	20

solution	team	att	time	size	lang
Fastest		1	142	4,953	C++
Shortest		1	261	2,512	C++
Max atts.		4	275	6,786	C++

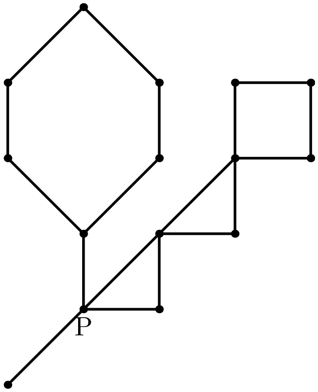
Problem C. Cactus Construction (1)

- Depth-first search (DFS) of the cactus to split the edges of the cactus into disjoint sets of *bridges* B_i and *cycles* C_i
  - Each back edge found during DFS signals a cycle
  - All edges that do not belong to any cycle are bridges



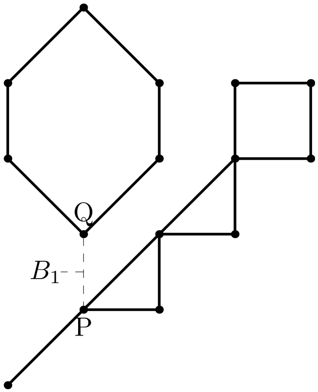
Problem C. Cactus Construction (2)

- Recursive procedure: given a cactus and a vertex P in it, construct the cactus in such a way that all vertices have color 2 except P, which must have color 1.
- Pick any vertex as P to start the procedure.



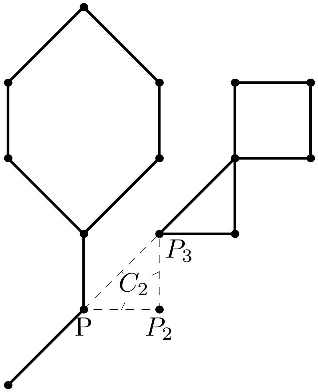
Problem C. Cactus Construction (3)

- If there's a bridge B_i connecting P with some other vertex Q:
  - Remove the bridge B_i.
  - Recursively construct two halves using P and Q as designated vertices (color 1).
  - Build edge B_i and color Q with 2 (more details later).



Problem C. Cactus Construction (4)

- If there's a cycle C_i passing through $P = P_1, P_2, \ldots, P_k$:
  - Remove all edges of C_i.
  - Graph splits into k components. Recursively construct them using P_j as designated vertices (color 1).
  - Build all edges of C_i and color $P_2, P_3, \ldots, P_k$ with 2 (more details later).

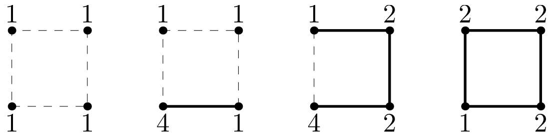


Problem C. Cactus Construction (5)

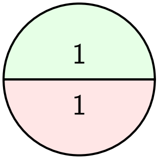
- How to build a new bridge connecting two components with designated vertices P and Q:
  - Initially P and Q are colored with 1, the rest with 2.
  - Recolor 1 to 3 in the component of Q.
  - Join components of P and Q.
  - Connect colors 1 an 3, connecting P and Q.
  - Recolor 3 to 2.

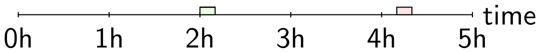
Problem C. Cactus Construction (6)

- How to build a new cycle connecting k components with designated vertices $P = P_1, P_2, \ldots P_k$:
  - Build edges of the cycle one by one, starting with the edge between P_1 and P_2.
  - Each edge except the last one is built as a new bridge.
  - But remember to recolor the first vertex P_1 to 4 instead of 2 after building the first edge.
  - This allows to close the cycle in the end by connecting colors 1 and 4.



Problem D. Delight for a Cat





	Java	C++	Python	Total
Accepted	0	1	0	1
Rejected	0	1	0	1
Total	0	2	0	2

solution	team	att	time	size	lang
Fastest		1	123	2,825	C++
Shortest		1	123	2,825	C++
Max atts.		1	123	2,825	C++

Problem D. Delight for a Cat

- Out of every k consecutive hours, the cat must sleep at least min_s hours and eat at least min_e hours
  - Let $max_e = k - min_s$, now the cat must eat between min_e and max_e hours out of every k
- Let's say that the cat is sleeping by default, and it gets $\delta_i = e_i - s_i$ delight for eating at hour i
  - In the end, add $\sum_{i=1}^{n} s_i$ to the answer

Problem D. Delight for a Cat (2)

- Let $min_e = 0$, $max_e = 1$
- Dynamic programming:
  - Let f_i be the maximum amount of delight for hours from i to n
  - $f_i = max(f_{i+1}, \delta_i + f_{i+k})$
  - Here, we define $f_i = 0$ for $i > n$
- Dynamic programming $\rightarrow$ shortest path:
  - Vertices S, T and $1, 2, \cdots, n$
  - Edge from vertex i to vertex $i + 1$ (or T, if $i + 1 > n$) with cost 0
  - Edge from vertex i to vertex $i + k$ (or T, if $i + k > n$) with cost $-\delta_i$
  - Edges from vertex S to vertices $1, 2, \cdots, k$ with cost 0
  - Negated length of the shortest path from S to T is the answer

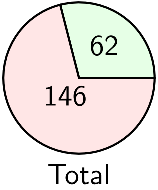
Problem D. Delight for a Cat (3)

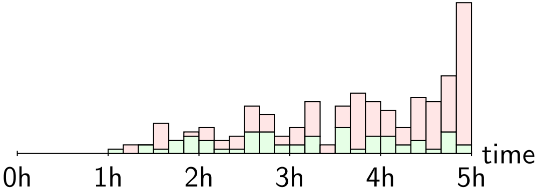
- Let $min_e = 0$, $max_e \geq 0$
- Graph for $max_e = 1 \rightarrow$ network for $max_e \geq 0$:
  - Vertices S, T and $1, 2, \cdots, n$
  - Edge from vertex i to vertex $i + 1$ (or T, if $i + 1 > n$) with cost 0 and capacity max_e
  - Edge from vertex i to vertex $i + k$ (or T, if $i + k > n$) with cost $-\delta_i$ and capacity 1
  - Edges from vertex S to vertices $1, 2, \cdots, k$ with cost 0 and capacity ∞
  - Negated minimum cost of flow of value max_e from S to T is the answer

Problem D. Delight for a Cat (4)

- Let $max_e \geq min_e \geq 0$
- Network for $min_e = 0 \rightarrow$ network for $min_e \geq 0$:
  - Vertices S, T and $1, 2, \cdots, n$
  - Edge from vertex i to vertex $i + 1$ (or T, if $i + 1 > n$) with cost 0 and capacity $max_e - min_e$
  - Edge from vertex i to vertex $i + k$ (or T, if $i + k > n$) with cost $-\delta_i$ and capacity 1
  - Edges from vertex S to vertices $1, 2, \cdots, k$ with cost 0 and capacity ∞
  - Negated minimum cost of flow of value max_e from S to T is the answer

Problem E. Expect to Wait

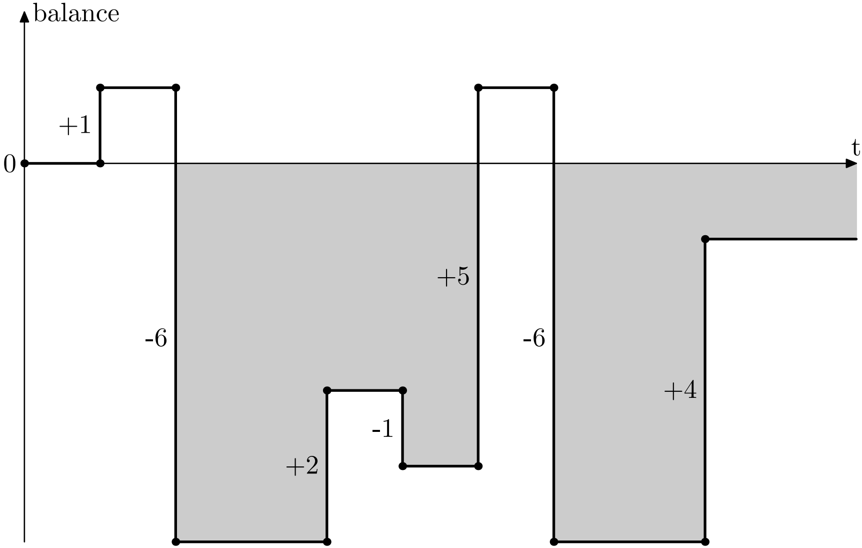




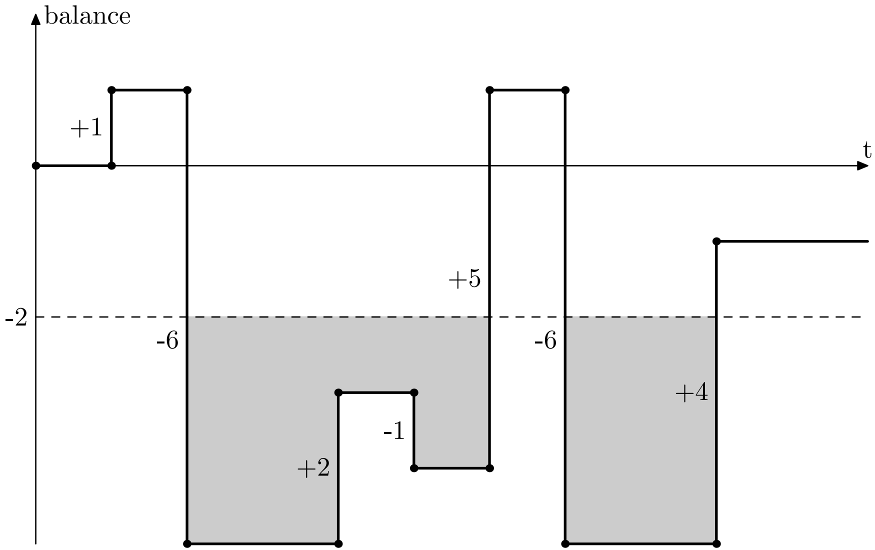
	Java	C++	Python	Total
Accepted	3	59	0	62
Rejected	3	143	0	146
Total	6	202	0	208

solution	team	att	time	size	lang
Fastest		1	61	2,093	C++
Shortest		1	115	1,274	C++
Max atts.		6	285	2,167	C++

Problem E. Expect to Wait



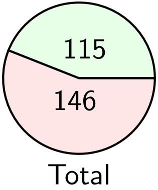
Problem E. Expect to Wait (2)

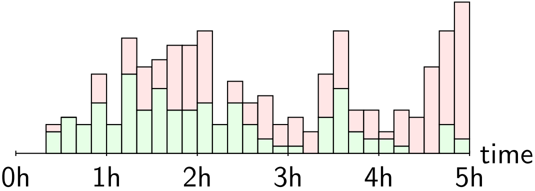


Problem E. Expect to Wait (3)

- For b_i, calculate square under line $balance = -b_i$
- Scan-line
- O(NlogN + QlogQ)

Problem F. Foreign Postcards





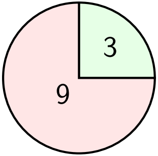
	Java	C++	Python	Total
Accepted	3	112	0	115
Rejected	17	126	3	146
Total	20	238	3	261

solution	team	att	time	size	lang
Fastest		1	23	1,774	C++
Shortest		1	68	483	C++
Max atts.		15	281	1,030	C++

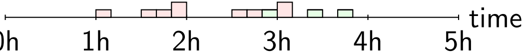
Problem F. Foreign Postcards

- A_i is the expected number of `W`'s starting from position i.
- $A_i = \frac{1}{n-i} \sum_{j=i+1}^{n} A_j + |\{k \mid k \in [i, j) \wedge S_i \neq S_k\}|$
- Straightforward calculation — $O(N^2)$ solution.
- The second sum is $\sum_{j=i+1}^{n} [S_i \neq S_j] \cdot (n - j)$
- It is equal to either $\sum_{S_j = C} (n - j)$ or $\sum_{S_j = W} (n - j)$
- Calculate everything in linear time, $O(N)$ solution.

Problem G. Game on Graph



Total



	Java	C++	Python	Total
Accepted	0	3	0	3
Rejected	0	9	0	9
Total	0	12	0	12

solution	team	att	time	size	lang
Fastest		7	174	3,103	C++
Shortest		2	221	2,023	C++
Max atts.		7	174	3,103	C++

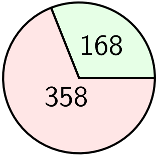
Problem G. Game on Graph

- Let's name a pair of vertex and player who has move now a *position* in game.
- If first player can enforce draw, regardless of second player moves, he will do it.
- If second player can enforce his winning, regardless of first player moves, he will do it.
- If first player can't enforce draw, second player will not allow draw happen, because it's worst result for him
- If second player can't enforce winning, first player will not allow him to win, because it's worst result for him.
- So, in all other positions, first player will win.

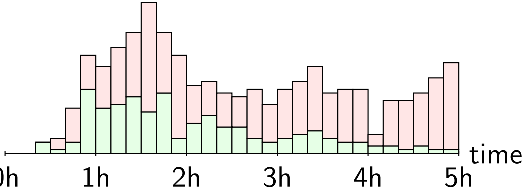
Problem G. Game on Graph (2)

- How to find all positions, where first player can enforce draw?
- Let's put in queue all positions, where player have no moves.
- While queue is not empty, get next position from queue.
- If it's first player turn, than mark all positions of second player with move to this posision and put them to queue.
- If it's second player turn, than mark all positions of first player, where it's last move to non-marked positions, and put them to queue.
- First player can enforce draw, iff position is not marked.
- Postions, where second player can enforce win, can be found in similar way.

Problem H. Hard Refactoring



Total



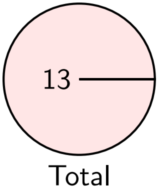
	Java	C++	Python	Total
□ Accepted	10	154	4	168
□ Rejected	29	318	11	358
Total	39	472	15	526

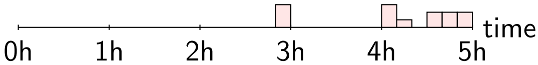
solution	team	att	time	size	lang
Fastest		1	20	1,982	C++
Shortest		2	109	1,303	Python
Max atts.		11	242	2,721	C++

Problem H. Hard Refactoring

- It is an easy problem. The hardest part is parsing
- Once input is parsed, it is Ok to simply fill a Boolean array of 2^{16} items, then print the answer
- The only tricky thing in this problem are the edges of the set of 16-bit integers and the corresponding samples are given in the problem statement

Problem I. Indiana Jones and the Uniform Cave



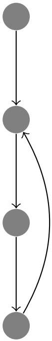


	Java	C++	Python	Total
Accepted	0	0	0	0
Rejected	0	13	0	13
Total	0	13	0	13

Problem I. Indiana Jones and the Uniform Cave

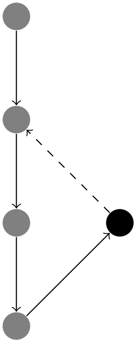
- Solution overview: use depth-first-search (DFS) to recursively traverse the graph
  - Mark with "`right`" chambers on the current DFS path to root. Let us call them *gray* chambers
  - Mark with "`left`" chambers that were already visited. Let us call them *black* chambers
- At each node do "`1 right 1`" m times to check all outgoing passages
- Keep the track of the highest gray ("`right`") chamber encountered down from the current chamber in dfs and the number of the corresponding passage in the chamber
- When all m passages out of the chamber are visited, follow to that chamber until gray ("`right`") chamber is encountered and follow down to the previous chamber

Problem I. Indiana Jones and the Uniform Cave (2)



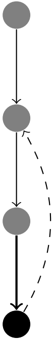
- When gray (“`right`”) chamber is encountered:
- Mark it with “`left`”
- Follow gray (“`right`”) chambers with “`0 right 0`” until the chamber previously marked with “`left`” is reached
- Count the number of passages visited
- Make another pass, taking one fewer passage to “backtrack” and put stones at “`right`” again
- Remember how many gray chambers up the path we’ve got to!

Problem I. Indiana Jones and the Uniform Cave (3)



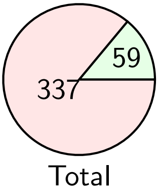
- When black (“`left`”) chamber is encountered:
- Follow black (“`left`”) chambers with “`0 left 0`”
- Follow gray (“`right`”) chambers with “`0 right 0`”
- Count the number of passages visited
- Make another pass, taking one fewer passage to “backtrack”
- Put stones at “`right`” or “`left`” as they were
- Remember how many gray chambers up the path we’ve got to!

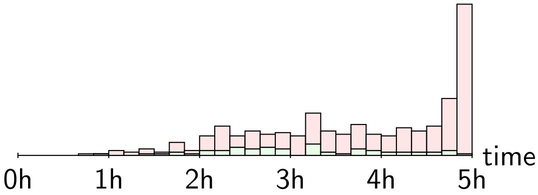
Problem I. Indiana Jones and the Uniform Cave (4)



- When leaving the chamber in dfs (after visiting all *m* passages), use the passage that is leading to the highest gray chamber
- Stones were properly left in place previously, just follow them
- Mark the chamber we are leaving as black (“`left`”)

Problem J. Jenga Boom





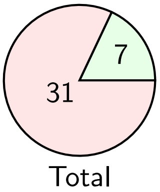
	Java	C++	Python	Total
Accepted	2	57	0	59
Rejected	21	316	0	337
Total	23	373	0	396

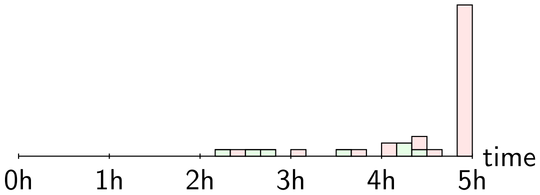
solution	team	att	time	size	lang
Fastest		1	55	1,919	C++
Shortest		1	138	1,707	C++
Max atts.		14	198	2,669	C++

Problem J. Jenga Boom

- The solution is straightforward
- Keep the sum of coordinates of centers of remaining blocks at each level
- Keep the set of remaining blocks (in a Boolean array)
- When a block is removed, update this information, then recheck stability condition for every level (top to bottom)
  - Sum centers of masses and count total number of blocks above the current cross-section
  - Use the set of remaining block at the level immediately below the current corss-section to compute its convex hull in a trivial way
- Some tips
  - The number w is irrelevant to the problem
  - It is easier to compute everything in 64-bit integer numbers
  - Do not do binary search on answer! Simulate every block removal

Problem K. Kids Designing Kids





	Java	C++	Python	Total
Accepted	0	7	0	7
Rejected	3	28	0	31
Total	3	35	0	38

solution	team	att	time	size	lang
Fastest		1	133	3,236	C++
Shortest		1	258	2,613	C++
Max atts.		4	252	3,682	C++

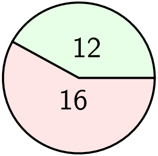
Problem K. Kids Designing Kids

- Find the top-left freckle in each of three given pictures.
- We'll prove that after moving the figures, some two of these three freckles must be in the same point.
- There are only three possible shifts, check them all.
- To check if two pictures are the same, again find top-left freckles in each of them. These freckles must be in the same point.

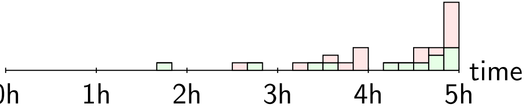
Problem K. Kids Designing Kids — Proof

- Problem is equal to the following: move figures A, B and C in such a way that $A \oplus B \oplus C = \emptyset$ (empty figure).
- Let's look on top-left freckle in each picture. Suppose that after moving, they are in different points.
- Now let's find the top-left freckle from these three.
- This freckle will be present in the final symmetrical difference $A \oplus B \oplus C$, because it cannot be denied by any other freckle.

Problem L. List of Primes



Total



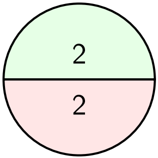
	Java	C++	Python	Total
Accepted	0	12	0	12
Rejected	0	16	0	16
Total	0	28	0	28

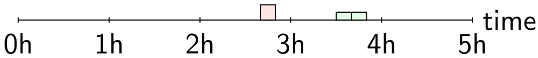
solution	team	att	time	size	lang
Fastest		1	108	3,093	C++
Shortest		1	108	3,093	C++
Max atts.		4	294	4,471	C++

Problem L. List of Primes

- Recursive procedure — output all sets with the following properties:
  - from the first x primes the set *prefix* is selected;
  - all the other selected primes add up to *sum*.
- $output(x, prefix, sum)$
  - $output(x + 1, prefix + [p_{x+1}], sum - p_{x+1})$
  - $output(x + 1, prefix, sum)$
- If the output is to left from the desired segment, skip it without going deeper in recursion.
- Precalculte number and total length of all sets in which:
  - the first x primes are not used;
  - selected primes add up to *sum*.
- Run $output(0, [], sum)$ for $sum = 2, 3, 4, \ldots$

Problem M. Mole Tunnels



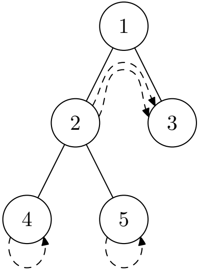


	Java	C++	Python	Total
Accepted	0	2	0	2
Rejected	0	2	0	2
Total	0	4	0	4

solution	team	att	time	size	lang
Fastest		3	215	3,397	C++
Shortest		1	227	2,185	C++
Max atts.		3	215	3,397	C++

Problem M. Mole Tunnels

- Cannot be solved with Minimum Cost Maximum Flow algorithm (too slow)
- We need to find shortest path in a faster way
- Calculate dynamic programming: shortest path in a subtree of vertex v
- Iterate over all possible LCA to find next shortest path
- After sending flow along augmenting path dynamic programming values changes only for at most 40 vertices



Credits

- Special thanks to all jury members and assistants (in alphabetic order):

Alexey Cherepanov, Andrey Lopatin, Andrey Stankevich, Artem Vasilyev, Borys Minaiev, Demid Kucherenko, Dmitry Shtukenberg, Egor Kulikov, Elena Andreeva, Elena Kryuchkova, Eugene Kurpiliansky, Gennady Korotkevich, Georgiy Korneev, Ilya Kornakov, Maxim Buzdalov, Mikhail Dvorkin, Mikhail Pyaderkin, Nikita Ioffe, Niyaz Nigmatullin, Oleg Davydov, Pavel Kunyavsky, Pavel Mavrin, Petr Mitrichev, Roman Elizarov, Sergey Kopeliovich, Sergey Melnikov, Vitaly Aksenov