ACM ICPC 2015–2016
Northeastern European Regional Contest
Problems Review

Roman Elizarov

December 6, 2015

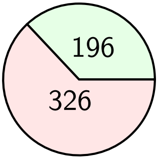
Problems summary

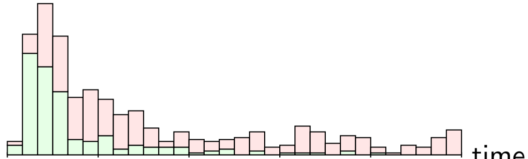
- Recap: 224 teams, 12 problems, 5 hours,
- 20-th NEERC — Jubilee
- This review assumes the knowledge of the problem statements (published separately on http://neerc.ifmo.ru/ web site)
- Summary table on the next slide lists problem name and stats
  - author — author of the original idea
  - acc — number 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
Adjustment Office	Vitaliy Aksenov	196 /522	37%
Binary vs Decimal	Mikhail Tikhomirov	12 /83	14%
Cactus Jubilee	Mikhail Tikhomirov	14 /28	50%
Distance on Triangulation	Gennady Korotkevich	9 /67	13%
Easy Problemset	Andrey Lopatin	208 /281	74%
Froggy Ford	Georgiy Korneev	101 /602	16%
Generators	Elena Andreeva	153 /533	28%
Hypercube	Oleg Khristenko	3 /15	20%
Iceberg Orders	Egor Kulikov	0 /2	0%
Jump	Maxim Buzdalov	52 /577	9%
King's Inspection	Mikhail Dvorkin	24 /253	9%
Landscape Improved	Georgiy Korneev	56 /213	26%

Problem A. Adjustment Office





	Java	C++	Total
Accepted	17	179	196
Rejected	53	273	326
Total	70	452	522

solution	team	att	time	size	lang
Fastest	SPbAU 1	1	7	1,051	C++
Shortest	NU 14	2	41	529	C++
Max atts.	UrSU 2	10	130	1,650	Java

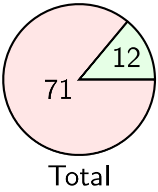
Problem A. Adjustment Office (1)

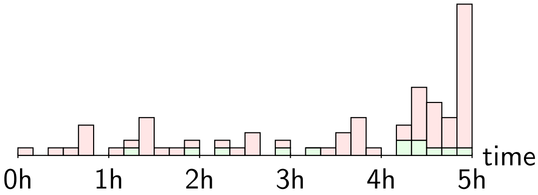
- Recap: $n \times n$ grid, each cell (x, y) has value $x + y$
- Initial sum at row or column k is equal to $nk + \frac{n(n+1)}{2}$
- Maintain two pieces of data for rows and columns:
  - The set of all zeroed out rows $\mathcal{S}_\mathrm{r}$ and columns $\mathcal{S}_\mathrm{c}$, initially both sets are empty
  - Two sums $s_{\mathrm{r},\mathrm{c}} = \sum_{i \in \{1 \ldots n\} \setminus \mathcal{S}_{\mathrm{r},\mathrm{c}}} i$, initially both sums are $\frac{n(n+1)}{2}$
- The result of row query "`R` r" is equal to:

$$\begin{cases} \left(n - |\mathcal{S}_\mathrm{c}|\right) r + s_\mathrm{c} & \text{if } r \notin \mathcal{S}_\mathrm{r} \\ 0 & \text{otherwise} \end{cases}$$

- If $r \notin \mathcal{S}_\mathrm{r}$, then $\mathcal{S}_\mathrm{r} \leftarrow \mathcal{S}_\mathrm{r} \cup \{r\}$ and $s_\mathrm{r} \leftarrow s_\mathrm{r} - r$
- Similarly for column queries

Problem B. Binary vs Decimal





	Java	C++	Total
Accepted	4	8	12
Rejected	4	67	71
Total	8	75	83

solution	team	att	time	size	lang
Fastest	SPb SU 1	2	73	1,914	Java
Shortest	ITMO 1	1	112	1,193	Java
Max atts.	MAI	5	269	256,031	C++

Problem B. Binary vs Decimal (1)

- It is easy to prove that ${10^k}_2$ has ${2^k}_2$ as a suffix, for example:

decimal	binary
1_{10}	1_2
10_{10}	1010_2
100_{10}	1100100_2
1000_{10}	1111101000_2

- Let $\mathcal{C}_k$ be a set of all numbers less than 10^k, whose decimal representation is equal to its k last binary digits
- Let $\mathcal{C}_k = \mathcal{A}_k \cup \mathcal{B}_k$, where all $x \in \mathcal{A}_k$ have k-th (counting from zero) digit of 0 and all $x \in \mathcal{B}_k$ have k-th digit of 1
- $\mathcal{B}_k$ is a set of *bindecimal* numbers of length k — one we want

k	$\mathcal{A}_k$	$\mathcal{B}_k$
0	$\{0\}$	$\{\}$
1	$\{0\}$	$\{1\}$
2	$\{0, 1\}$	$\{10, 11\}$
3	$\{0, 1, 10, 11\}$	$\{100, 101, 110, 111\}$
4	$\{0, 1, 100, 101\}$	$\{1000, 1001, 1100, 1101\}$

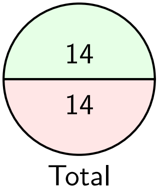
Problem B. Binary vs Decimal (2)

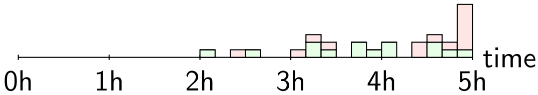
- Define a recursive rule to get $\mathcal{C}_k$ from $\mathcal{C}_{k-1}$

$$
\begin{aligned}
\mathcal{A}_k &= \{x && \mid x \in \mathcal{C}_{k-1} \text{ and } k\text{-th bit of } x \text{ is zero}\} \\
\mathcal{B}_k &= \{x + 10^k && \mid x \in \mathcal{C}_{k-1} \text{ and } k\text{-th bit of } x \text{ is zero}\} \\
\mathcal{C}_k &= \mathcal{A}_k \cup \mathcal{B}_k
\end{aligned}
$$

- Keep $\mathcal{C}_k$ as an ordered list, so when new $\mathcal{B}_k$ is computed as defined above, the bindecimal numbers in $\mathcal{B}_k$ are produced in ascending order; count them; stop when n-th bindecimal number is found
- Note, that the max answer for $n = 10\,000$ is around 10^{161}, so it will not fit into any standard data types; need long arithmetics to implement it

Problem C. Cactus Jubilee



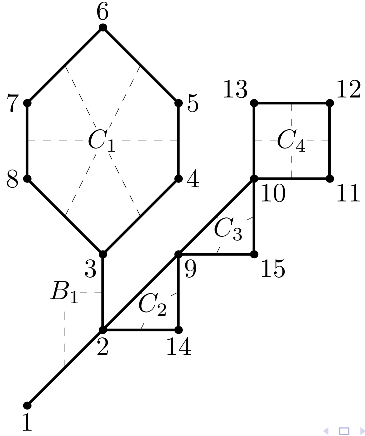


	Java	C++	Total
Accepted	0	14	14
Rejected	0	14	14
Total	0	28	28

solution	team	att	time	size	lang
Fastest	Saratov SU 4	1	124	3,673	C++
Shortest	Ural FU 1	2	151	2,735	C++
Max atts.	MIPT 2	4	279	3,487	C++

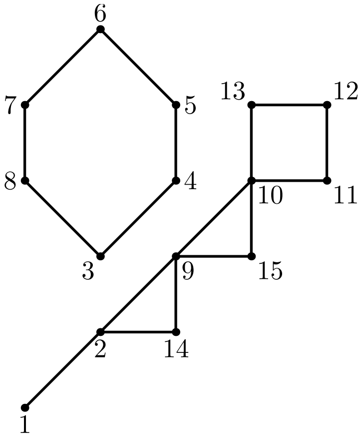
Problem C. Cactus Jubilee (1)

- Depth-first search (DFS) of the cactus to split the edges of the cactus into disjoint sets of *bridge-trees* B_i and *cycles* C_i
  - Each back edge found during DFS signals a cycle
  - Compute size of each B_i and a number of non-adjacent pairs of edges during the first DFS; do a second DFS to push it to all adjacent cycles



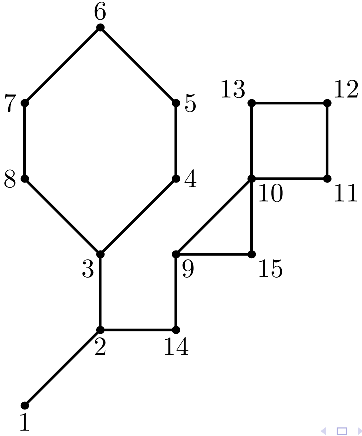
Problem C. Cactus Jubilee (2)

- When an edge from a bridge-tree B_i is removed, cactus splits into two connected components
  - any pair of vertices from these two components can be reconnected to get a cactus
  - number of ways can be counted during initial DFS

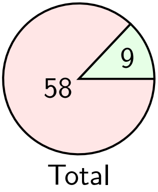


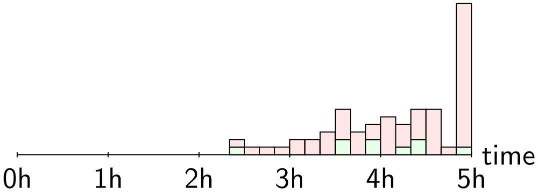
Problem C. Cactus Jubilee (3)

- When an edge from a cycle C_i is removed, cactus is still connected; bridge-trees adjacent to cycle merge
  - any pair of non-adjacent vertices from the same bridge-tree can be connected to get a cactus
  - Scan all cycles to figure out the the number of ways to add an edge for each cycle broken; multiple by the cycle size



Problem D. Distance on Triangulation



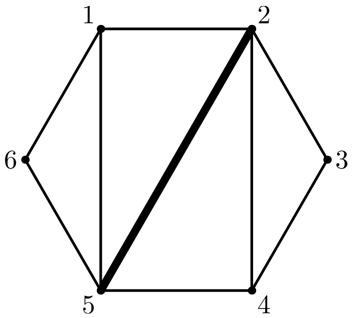


	Java	C++	Total
Accepted	0	9	9
Rejected	2	56	58
Total	2	65	67

solution	team	att	time	size	lang
Fastest	NNSU	1	145	4,602	C++
Shortest	SPbAU 1	1	235	3,579	C++
Max atts.	SPb SU 3	6	266	7,846	C++

Problem D. Distance on Triangulation (1)

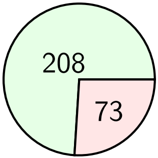
- Divide and conquer; prove that each triangulated polygon has a diagonal that cuts at least $n/3$ vertices
- Randomly picking a diagonal does not work — will time limit
- Recursively split polygons this way for a total depth of $O(\log n)$; get $O(n)$ subpolygons of total size $O(n \log n)$
- Terminal subpolygons for this problem are the ones that do not have have any diagonals to split them further — triangles



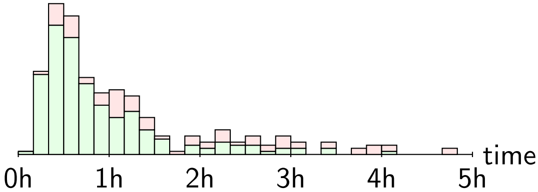
Problem D. Distance on Triangulation (2)

- For each subpolygon precompute the shortest distances from two ends of the diagonals of that was used to cut out this subpolygon from the large one
  - $O(n \log n)$ total memory to store the distances
  - Can be done in $O(n \log n)$ by doing breadth-first search in each subpolygon or in $O(n \log^2 n)$ by doing recursive queries (see below for query implementation)
- Each query can be answered in $O(\log n)$ recursively
  - Terminal subpolygon (triangle) — trivial
  - x and y in query are both on one side of splitting diagonal — recursive query into the corresponding subpolygon
  - x and y in query are on different sides — use precomputed distances to diagonal ends (diagonals do not intersect, so $x - y$ path goes through one of the ends)

Problem E. Easy Problemset



Total



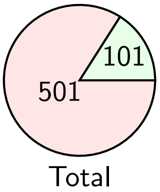
	Java	C++	Total
Accepted	17	191	208
Rejected	9	64	73
Total	26	255	281

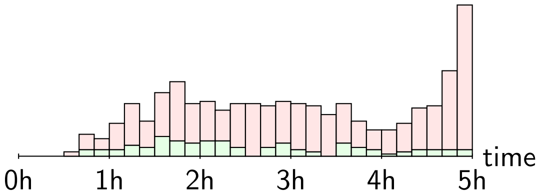
solution	team	att	time	size	lang
Fastest	Ural FU 1	1	8	1,011	C++
Shortest	NU 14	1	22	426	C++
Max atts.	Kyrgyz-Turkish U 1	5	188	781	C++

Problem E. Easy Problemset (1)

- The easiest problem
- Just implement what the problem statement says
- Pay attention to judges without remaining problems — don't forget to propose a hard problem
- Sample inputs and outputs were designed to expose all the tricky cases to make debugging easy

Problem F. Froggy Ford



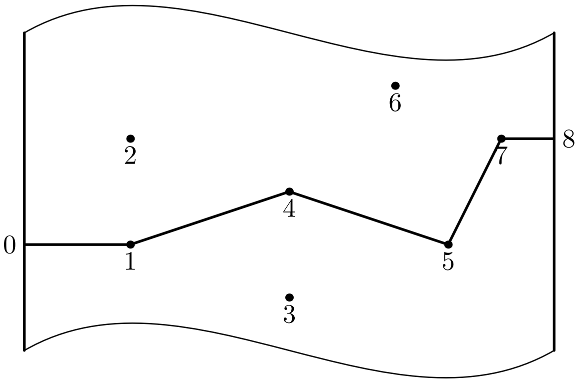


	Java	C++	Total
Accepted	5	96	101
Rejected	12	489	501
Total	17	585	602

solution	team	att	time	size	lang
Fastest	NNSU	1	41	2,281	C++
Shortest	ITMO 4	1	85	1,600	C++
Max atts.	Saratov SU 3	19	217	3,797	C++

Problem F. Froggy Ford (1)

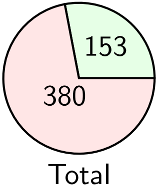
- Consider a graph with vertices $1, \ldots n$ corresponding to stones, 0 for the left shore, $n+1$ for the right one; obvious way to compute distances between vertices
- The problem of finding the optimal route from 0 to $n+1$ as defined in problem is called *minimax path problem*

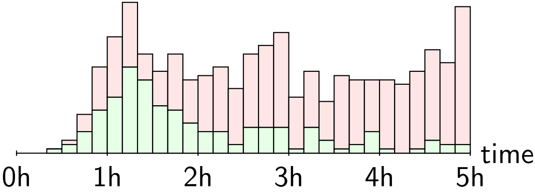


Problem F. Froggy Ford (2)

- Minimax path form 0 to all other vertices can be found by Djikstra algorithm with a corresponding minimax update rule; $O(n^2)$; no need to even have a heap in Djikstra
- Second invocation of the same algo to find minimax path from $n + 1$ to all others
- A new optimal stone can be only at the center between a pair of vertices (stone – stone, stone – shore, shore – shore)
  - Check all pairs of vertices; $O(n^2)$
  - Use precomputed distances to 0 and to $n + 1$ to find distance when new stone is placed; pick the optimal case
  - Make sure to correctly implement distances between shores (vertices 0 and $n + 1$); this case is not covered in sample input

Problem G. Generators





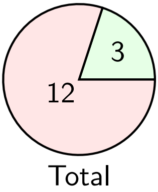
	Java	C++	Total
Accepted	9	144	153
Rejected	53	327	380
Total	62	471	533

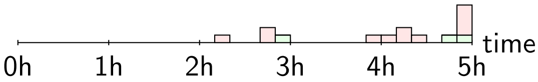
solution	team	att	time	size	lang
Fastest	MSU 3	1	29	3,169	C++
Shortest	Ural FU 4	2	86	1,136	C++
Max atts.	Far Eastern FU	19	262	1,699	C++

Problem G. Generators (1)

- Recap: $x_{i+1}^{(j)} = \left(a^{(j)} x_i^{(j)} + b^{(j)}\right) \mod c^{(j)}$
- Generate $c^{(j)}$ numbers for each LCG — produce all numbers this LCG can possibly generate; for each LCG find:
  - the maximum $x_{t_j}^{(j)}$; pay attention to $x_0^{(j)}$ (don't skip it)
  - the second maximum $x_{u_j}^{(j)}$, such that $\left(x_{t_j}^{(j)} - x_{u_j}^{(j)}\right) \mod k \neq 0$
  - Pay attention to cases when there is no second maximum, e.g. all generated numbers are the same or all differences between them are multiples of k
- When $\sum_{j=1}^{n} x_{t_j}^{(j)} \mod k \neq 0$ — that's the answer
- Otherwise, find j such that $\left(x_{t_j}^{(j)} - x_{u_j}^{(j)}\right)$ is maximized (if at least one second maximum u_j exists) and replace t_j with u_j
- Otherwise, there is no answer

Problem H. Hypercube





	Java	C++	Total
Accepted	0	3	3
Rejected	0	12	12
Total	0	15	15

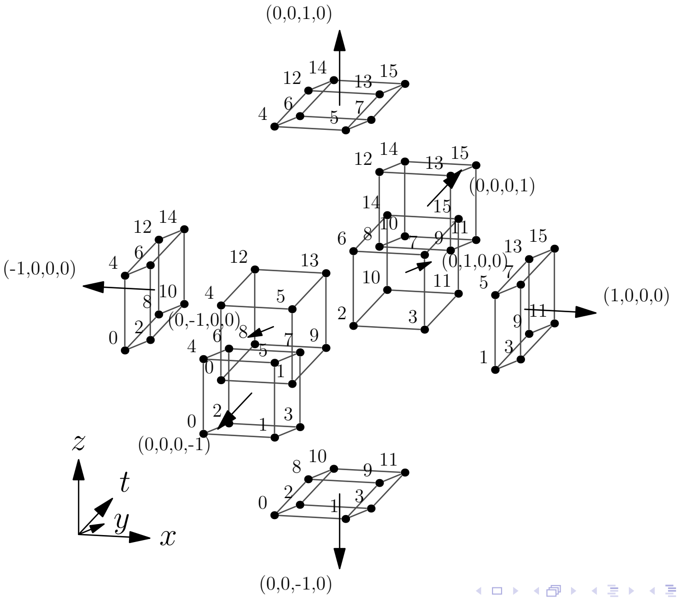
solution	team	att	time	size	lang
Fastest	SPb SU 1	1	175	2,241	C++
Shortest	SPb SU 1	1	175	2,241	C++
Max atts.	Ural FU 1	3	289	5,628	C++

Problem H. Hypercube (1)

- Disassemble tesseract into 8 cubic cells
- Start with an arbitrary cube of an octocube, assume it corresponds to an arbitrary cell of tesseract
- Visit all cubes of a given octocube via DFS
- Each time a cube is visited, see what cell it shall correspond to and if that cell was not used yet
- There are two conceptual ways to uniquely identify tesseract's cells and to traverse them
  - 3D geometry — represent each cell via numbering of its 8 vertices; no 4D vector manipulations required
  - 4D geomerty — represent each cell via a 4D vector normal; leads to simpler code

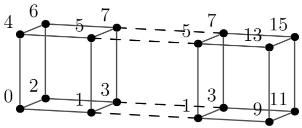
Problem H. Hypercube (2)

- Disassembled tesseract with numbered vertices and normals



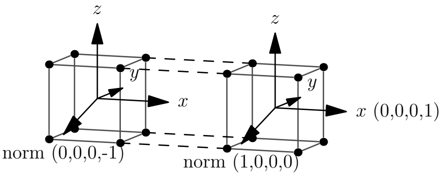
Problem H. Hypercube (3)

- Solution using 3D geometry
  - All vertices of a tesseract are numbered from 0 to 15
  - Cell in a tesseract are represented by cubes of 8 vertex indices
  - Moving in one of 6 directions in an given octocube, find 4 vertices in that direction on a corresponding face of the current cube
  - Among remaining cells find the one that can be rotated/flipped (in 3D) to get a match of vertices face-to-face — this is the next cell and its 3D rot'n/flip

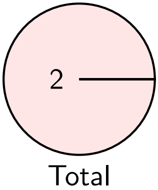


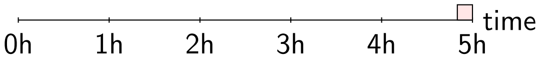
Problem H. Hypercube (4)

- Solution using 4D geometry
  - Keep track of a $4 \times$ 4D vectors: 3 vectors define a basis for current cell's hyperplane, 4th vector defines a normal
  - A normal also uniquely identifies a cell of a tesseract
  - Moving in one of 6 directions in an given octocube, the corresponding basis vector of the current hyperplane (multiplied by ± 1 depending on direction in the axis) becomes the normal of the next cell; the former normal replaces basis vector in that direction (multiplied by ∓ 1) — proper 4D rot'n



Problem I. Iceberg Orders





	Java	C++	Total
□ Accepted	0	0	0
□ Rejected	0	2	2
Total	0	2	2

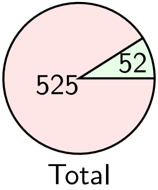
Problem I. Iceberg Orders (1)

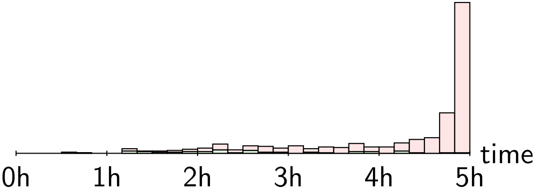
- The structure of the code is hinted in the statement:
  - keep buy and sell orders in the book separately in a sorted tree maps keyed by the price
  - at each price keep a list of orders ordered by priority
  - this way finding a set of orders to match with is efficient — $O(k)$, where k is the number of orders to match with
- The solution is then mostly boils down to implementing what the problem statement says, with one tricky case
- When big order (volume V_a is big) comes in, it produces a lot of trades with other orders that have small tip value TV_b; namely $O(V_a)$ trades — too many to simulate directly
- Recap: V_a is up to 10^9; while the total number of different matched order pairs is guaranteed not to exceed 10^5

Problem I. Iceberg Orders (2)

- The tricky case is addressed by figuring out how many times m an incoming order fully matches with all orders at a current price level
- m is found using binary search in $O(p \log V_a)$ operations, where p is the number of orders at a given price level
- Then, all the m matches can be simulated at one pass in $O(p)$; simulating remaining pass directly
- Care shall taken be in two additional cases
  - when $p \gg k$ make sure that $O(k)$ operations are performed — must do one direct order-by-order match at a given price level first
  - when incoming order volume V_a is so big that the whole price level with k orders is consumed, must do it in $O(k)$; can afford additional log in binary search only at the last matched price level

Problem J. Jump





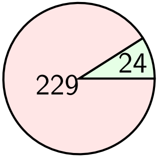
	Java	C++	Total
Accepted	1	51	52
Rejected	43	482	525
Total	44	533	577

solution	team	att	time	size	lang
Fastest	Ural FU 1	1	33	1,401	C++
Shortest	Ural FU 4	5	286	905	C++
Max atts.	Perm SU 1	7	251	1,323	C++

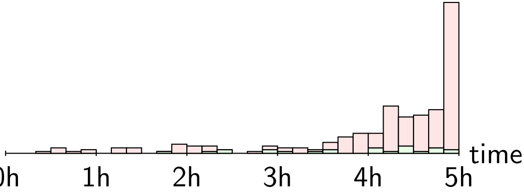
Problem J. Jump (1)

- Recap: must solve in $n + 500$ queries
- Solve the problem in two phases: Phase I with up to 499 queries and Phase II with up to $n + 1$ queries
- Phase I: find Q_{I} such that $\textsc{Jump}(Q_{\mathrm{I}}) = n/2$
  - do random queries in this phase
  - worst case when $n = 1000$, probability of guessing $n/2$ bits in a single random query is $\frac{\binom{1000}{500}}{2^{1000}} = 0.0252\ldots$
  - probability of **not** finding Q_{I} in 499 queries is 2.9×10^{-6}
  - Just quit if $\textsc{Jump}(Q_{\mathrm{I}}) = n$ is found
- Phase II: find solution Q_{II} such that $\textsc{Jump}(Q_{\mathrm{II}}) = n$
  - for $i = 2 \ldots n$ do queries with $Q_i = \{Q_{\mathrm{I}}$ bits 0 and i flipped$\}$
  - $\textsc{Jump}(Q_i) = n/2$ if bits 0 and i has the same "correctness"
  - assume 0 is correct bit in Q_{I}; make $Q_{\mathrm{II}} = \{Q_{\mathrm{I}}$ all bits j' flipped$\}$ where $\textsc{Jump}(Q_{j'}) \neq n/2$; try query Q_{II}; quit if got n
  - assume 0 is not correct; make $Q_{\mathrm{II}} = \{Q_{\mathrm{I}}$ all bits j'' flipped$\}$ where $j'' = 0$ or $\textsc{Jump}(Q_{j''}) = n/2$; must get $\textsc{Jump}(Q_{\mathrm{II}}) = n$

Problem K. King's Inspection



Total

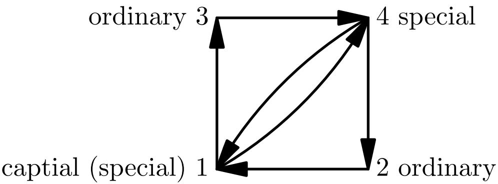


	Java	C++	Total
Accepted	0	24	24
Rejected	6	223	229
Total	6	247	253

solution	team	att	time	size	lang
Fastest	MIPT 5	2	102	2,920	C++
Shortest	ITMO 1	3	212	2,164	C++
Max atts.	NEFU 1	9	279	2,865	C++

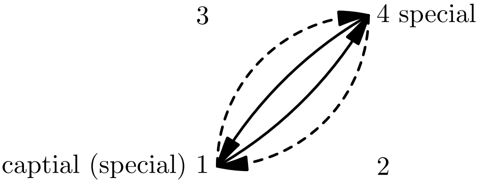
Problem K. King's Inspection (1)

- Count in-degree d_i^{in} and out-degree d_i^{out} for each city i; there is no route if either is zero for any city (important check!)
- Identify *special* cites i: capital ($i = 1$) and cities with $d_i^{\text{in}} > 1$ or $d_i^{\text{out}} > 1$; there are at most 41 special cities
- Other cities are *ordinary*



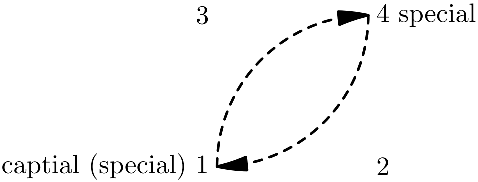
Problem K. King's Inspection (2)

- Merge all paths between special cities though ordinary cities.
- Ordinary cities: non-capital and $d_i^{\mathrm{in}} = 1$ and $d_i^{\mathrm{out}} = 1$
- For each special city create a list of outgoing paths to other special cities
  - there is no route if more than one outgoing path from a special city requires going through ordinary cities
  - if there is one outgoing path through ordinary cities, make it the only path in the outgoing list

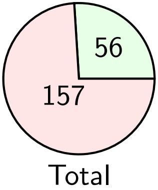


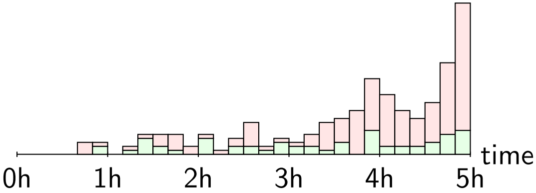
Problem K. King's Inspection (3)

- The picture shows properly reduced graph; but the list of cities on the reduced paths is still kept to print the answer
- Do exhaustive search (backtracking) for a path — at most 2^{20} operations
- There are at most 20 special cities with some choice (more that 1 outgoing path in list)



Problem L. Landscape Improved



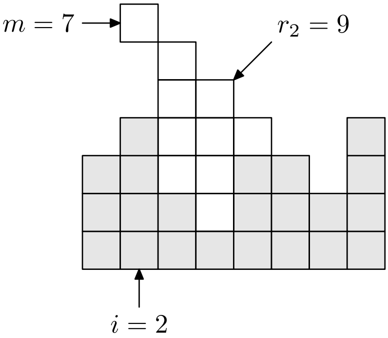


	Java	C++	Total
Accepted	2	54	56
Rejected	8	149	157
Total	10	203	213

solution	team	att	time	size	lang
Fastest	Saratov SU 4	1	53	2,969	C++
Shortest	Kazakh-British TU 2	5	289	1,465	C++
Max atts.	MIPT 3	9	298	2,880	C++

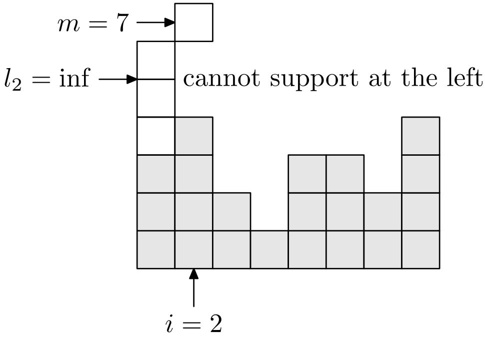
Problem L. Landscape Improved (1)

- Do binary search for the answer; try $O(\log n)$ guesses at the answer in the process
- For each guess m of the answer count the number of squares of stones required to build a mountain of height m, if it is possible; compare the result with n
- Let r_i be the number of squares of stones required to support the mountain of height m with a peak at i at the right



Problem L. Landscape Improved (2)

- Let l_i — the number to support at the left
- Total number of squares $t_i = l_i + r_i + m - h_i$
- The number of required squares is $\min t_i$ for all i
  - r_i is computed with a single pass for i from 1 to w in $O(w)$
  - l_i with a single pass for i from w to 1
  - overall time to find a solution is $O(w \log n)$



Credits

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

  Alexander Kaluzhin, Andrey Lopatin, Andrey Stankevich, Artem Vasilyev, Borys Minaiev, Demid Kucherenko, Dmitry Shtukenberg, Egor Kulikov, Elena Andreeva, Gennady Korotkevich, Georgiy Korneev, Gleb Evstropov, Grigoriy Shovkoplyas, Maxim Buzdalov, Mikhail Dvorkin, Mikhail Pyaderkin, Mikhail Tikhomirov, Nikita Kravtsov, Niyaz Nigmatullin, Oleg Hristenko, Pavel Krotkov, Pavel Kunyavsky, Pavel Mavrin, Petr Mitrichev, Viktor Omelyanenko, Vitaly Aksenov