ICPC 2018–2019
NEERC – Northern Eurasia Finals
Problems Review

December 2, 2018

Problems summary

- Recap: 299 teams, 13 problems, 5 hours

Problems summary

- Recap: 299 teams, 13 problems, 5 hours
- Full text analysis and problem statements are published at http://neerc.ifmo.ru/

Problems summary

- Recap: 299 teams, 13 problems, 5 hours
- Full text analysis and problem statements are published at http://neerc.ifmo.ru/
- These slides give only **brief idea** of general solution direction

Problems summary

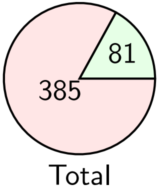
- Recap: 299 teams, 13 problems, 5 hours
- Full text analysis and problem statements are published at http://neerc.ifmo.ru/
- These slides give only **brief idea** of general solution direction
- Summary table on the next slide lists problem name and stats
  - **acc** — number of teams that had solved 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)

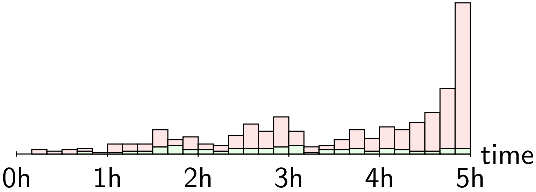
Problems summary

problem name	acc/runs	succ
Alice the Fan	81 /466	17%
Bimatching	0 /53	0%
Cactus Search	26 /121	21%
Distance Sum	0 /19	0%
Easy Chess	249 /565	44%
Fractions	148 /677	21%
Guest Student	225 /589	38%
Harder Satisfiability	1 /11	9%
Interval-Free Permutations	2 /5	40%
JS Minification	5 /50	10%
King Kog's Reception	20 /65	30%
Lazyland	247 /490	50%
Minegraphed	66 /278	23%

Problem A. Alice the Fan

Author: Oleg Hristenko   Statements and tests: Niyaz Nigmatullin





	Java	Kotlin	C++	Python	Total
Accepted	1	0	80	0	81
Rejected	2	0	377	6	385
Total	3	0	457	6	466

solution	team	att	time	size	lang
Fastest	ITMO 4	3	42	2819	C++
Shortest	BelarusianSU 5	2	157	1907	C++
Max atts.	MISiS 4	10	294	5557	C++

Problem A. Alice the Fan

- Denote as C maximum possible value of a and b.

Problem A. Alice the Fan

- Denote as C maximum possible value of a and b.
- Compute dynamic programming that answers all possible test cases.

Problem A. Alice the Fan

- Denote as C maximum possible value of a and b.
- Compute dynamic programming that answers all possible test cases.
- $d_{r,i,j}$ equals to maximum possible number of sets "Team A" can win if r sets were played, "Team A" scored i points and their opponents scored j points.

Problem A. Alice the Fan

- Denote as C maximum possible value of a and b.
- Compute dynamic programming that answers all possible test cases.
- $d_{r,i,j}$ equals to maximum possible number of sets "Team A" can win if r sets were played, "Team A" scored i points and their opponents scored j points.
- There are $O(C^2)$ states $(5 \cdot C \cdot C)$.

Problem A. Alice the Fan

- Denote as C maximum possible value of a and b.
- Compute dynamic programming that answers all possible test cases.
- $d_{r,i,j}$ equals to maximum possible number of sets "Team A" can win if r sets were played, "Team A" scored i points and their opponents scored j points.
- There are $O(C^2)$ states $(5 \cdot C \cdot C)$.
- Each set has $O(C)$ possible outcomes.

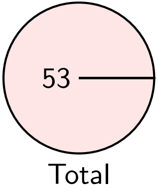
Problem A. Alice the Fan

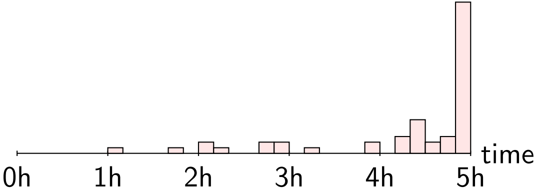
- Denote as C maximum possible value of a and b.
- Compute dynamic programming that answers all possible test cases.
- $d_{r,i,j}$ equals to maximum possible number of sets "Team A" can win if r sets were played, "Team A" scored i points and their opponents scored j points.
- There are $O(C^2)$ states $(5 \cdot C \cdot C)$.
- Each set has $O(C)$ possible outcomes.
- To restore the answer compute $f_{r,i,j}$ — the outcome of the r-th set that results in optimal answer for $d_{r,i,j}$.

Problem B. Bimatching

Author: Pavel Irzhavski

Statements and tests: Pavel Irzhavski





	Java	Kotlin	C++	Python	Total
Accepted	0	0	0	0	0
Rejected	0	0	53	0	53
Total	0	0	53	0	53

Problem B. Bimatching

- First we observe that bimatching as not easier than maximum matching in general graphs.

Problem B. Bimatching

- First we observe that bimatching as not easier than maximum matching in general graphs.
- Indeed, consider some graph. You can split each of its edges in the middle and add extra node there.

Problem B. Bimatching

- First we observe that bimatching as not easier than maximum matching in general graphs.
- Indeed, consider some graph. You can split each of its edges in the middle and add extra node there.
- Put all nodes of original graph in a right part and middle points in a left part.

Problem B. Bimatching

- First we observe that bimatching as not easier than maximum matching in general graphs.
- Indeed, consider some graph. You can split each of its edges in the middle and add extra node there.
- Put all nodes of original graph in a right part and middle points in a left part.
- Now, solving bimatching problem solves maximum matching, thus it is not easier.

Problem B. Bimatching

- First we observe that bimatching as not easier than maximum matching in general graphs.
- Indeed, consider some graph. You can split each of its edges in the middle and add extra node there.
- Put all nodes of original graph in a right part and middle points in a left part.
- Now, solving bimatching problem solves maximum matching, thus it is not easier.
- As an experienced participant you should stop to solve the problem with maximum flow or minimum cost maximum flow and think whether you can convert this two problems in the other direction and solve bimatching with maximum matching.

Problem B. Bimatching

- For each node v in the left part create its clone v'.

Problem B. Bimatching

- For each node v in the left part create its clone v'.
- Connect v' to each node in the right part that is connected to v.

Problem B. Bimatching

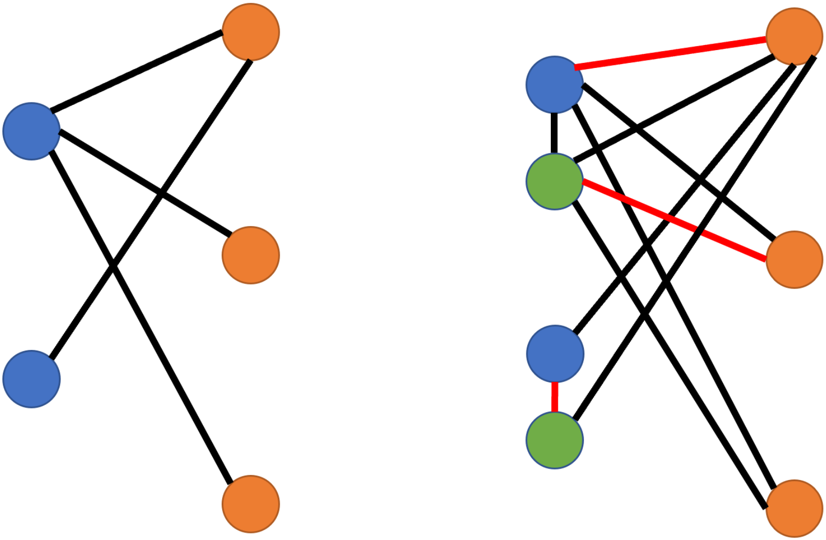
- For each node v in the left part create its clone v'.
- Connect v' to each node in the right part that is connected to v.
- Now, add an edge between v and its clone v'.

Problem B. Bimatching

- For each node v in the left part create its clone v'.
- Connect v' to each node in the right part that is connected to v.
- Now, add an edge between v and its clone v'.
- Find maximum matching in this new graph. Its size is at least n (size of the left part).

Problem B. Bimatching

- For each node v in the left part create its clone v'.
- Connect v' to each node in the right part that is connected to v.
- Now, add an edge between v and its clone v'.
- Find maximum matching in this new graph. Its size is at least n (size of the left part).
- However, switching v and v' to match with some nodes in the right part increases the answer by 1.



Problem B. Bimatching

- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.

Problem B. Bimatching

- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.
- However, assuming the given constraints there is no sense to aim for $O(VE)$ and one can go with easy to implement $O(V^2E)$ version.

Problem B. Bimatching

- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.
- However, assuming the given constraints there is no sense to aim for $O(VE)$ and one can go with easy to implement $O(V^2E)$ version.
- Another method to approach maximum matching problem is to consider Tutte matrix M of this graph.

Problem B. Bimatching

- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.
- However, assuming the given constraints there is no sense to aim for $O(VE)$ and one can go with easy to implement $O(V^2E)$ version.
- Another method to approach maximum matching problem is to consider Tutte matrix M of this graph.
- To find whether there exists a perfect matching in the graph one checks whether $|M| \neq 0$.

Problem B. Bimatching

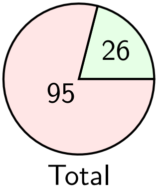
- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.
- However, assuming the given constraints there is no sense to aim for $O(VE)$ and one can go with easy to implement $O(V^2E)$ version.
- Another method to approach maximum matching problem is to consider Tutte matrix M of this graph.
- To find whether there exists a perfect matching in the graph one checks whether $|M| \neq 0$.
- To find the size of the maximum matching we apply Gauss elimination to find rank of Tutte matrix.

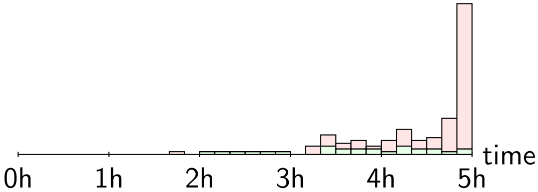
Problem B. Bimatching

- We can find maximum matching with Edmonds' blossom shrinking algorithm in $O(VE)$ time.
- However, assuming the given constraints there is no sense to aim for $O(VE)$ and one can go with easy to implement $O(V^2E)$ version.
- Another method to approach maximum matching problem is to consider Tutte matrix M of this graph.
- To find whether there exists a perfect matching in the graph one checks whether $|M| \neq 0$.
- To find the size of the maximum matching we apply Gauss elimination to find rank of Tutte matrix.
- FYI, the best known result to find maximum matching is $O(\frac{VE}{\log V})$.

Problem C. Cactus Search

Author: Borys Minaiev   Statements and tests: Borys Minaiev





	Java	Kotlin	C++	Python	Total
Accepted	0	0	26	0	26
Rejected	0	0	95	0	95
Total	0	0	121	0	121

solution	team	att	time	size	lang
Fastest	ITMO 2	2	127	2417	C++
Shortest	BelarusianSU 5	1	269	1230	C++
Max atts.	KBTU 1	8	248	2372	C++

Problem C. Cactus Search

- First consider this problem on a tree.

Problem C. Cactus Search

- First consider this problem on a tree.
- Each tree has at least one centroid, such node that its removal splits the tree in connected components of size no more than $\frac{n}{2}$.

Problem C. Cactus Search

- First consider this problem on a tree.
- Each tree has at least one centroid, such node that its removal splits the tree in connected components of size no more than $\frac{n}{2}$.
- Thus, we can pick this centroid and the jury pics one the components.

Problem C. Cactus Search

- First consider this problem on a tree.
- Each tree has at least one centroid, such node that its removal splits the tree in connected components of size no more than $\frac{n}{2}$.
- Thus, we can pick this centroid and the jury pics one the components.
- The process will finish in no more than $\log n + 1$ steps.

Problem C. Cactus Search

- Now we consider cactus case.

Problem C. Cactus Search

- Now we consider cactus case.
- Actually, ignore the fact we are given a cactus and solve the problem for any connected graph.

Problem C. Cactus Search

- Now we consider cactus case.
- Actually, ignore the fact we are given a cactus and solve the problem for any connected graph.
- Apply Floyd algorithm to compute a matrix of distances $\rho(u, v)$.

Problem C. Cactus Search

- Now we consider cactus case.
- Actually, ignore the fact we are given a cactus and solve the problem for any connected graph.
- Apply Floyd algorithm to compute a matrix of distances $\rho(u, v)$.
- We maintain a set of candidates A. Initially it contains all nodes of given graph G.

Problem C. Cactus Search

- Now we consider cactus case.
- Actually, ignore the fact we are given a cactus and solve the problem for any connected graph.
- Apply Floyd algorithm to compute a matrix of distances $\rho(u, v)$.
- We maintain a set of candidates A. Initially it contains all nodes of given graph G.
- If we ask about node u and node v is given as a reply we eliminate from set A all nodes $w \in A$ such that $\rho(v, w) < \rho(u, w)$.

Problem C. Cactus Search

- We aim to eliminate as many nodes from A as possible.

Problem C. Cactus Search

- We aim to eliminate as many nodes from A as possible.
- Consider every node v (not necessarily in A). For each u neighbor of v compute the number of nodes w in A that are closer to u.

Problem C. Cactus Search

- We aim to eliminate as many nodes from A as possible.
- Consider every node v (not necessarily in A). For each u neighbor of v compute the number of nodes w in A that are closer to u.
- Pick v that has minimum maximum number of such w's among all u's.

Problem C. Cactus Search

- We aim to eliminate as many nodes from A as possible.
- Consider every node v (not necessarily in A). For each u neighbor of v compute the number of nodes w in A that are closer to u.
- Pick v that has minimum maximum number of such w's among all u's.
- How many steps it will take to reduce A down to a single node?

Problem C. Cactus Search

- We aim to eliminate as many nodes from A as possible.
- Consider every node v (not necessarily in A). For each u neighbor of v compute the number of nodes w in A that are closer to u.
- Pick v that has minimum maximum number of such w's among all u's.
- How many steps it will take to reduce A down to a single node?
- We claim there always exists such v that the size of A will reduce at least twice.

Problem C. Cactus Search

- Let $P(v) = \sum_{u \in A} \rho(v, u)$. If more than a half of nodes $w \in A$ are closer to some u (neighbor of v), then $P(u) < P(v)$.

Problem C. Cactus Search

- Let $P(v) = \sum_{u \in A} \rho(v, u)$. If more than a half of nodes $w \in A$ are closer to some u (neighbor of v), then $P(u) < P(v)$.
- If we pick the node that minimizes $P(V)$ it will eliminate at least a half of A.

Problem C. Cactus Search

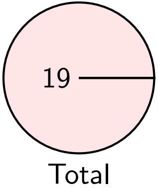
- Let $P(v) = \sum_{u \in A} \rho(v, u)$. If more than a half of nodes $w \in A$ are closer to some u (neighbor of v), then $P(u) < P(v)$.
- If we pick the node that minimizes $P(V)$ it will eliminate at least a half of A.
- The number of steps is $\log n + 1$ and the running time is $O(n^2)$ per single query.

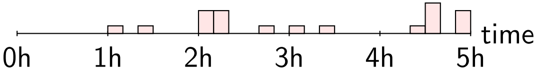
Problem C. Cactus Search

- Let $P(v) = \sum_{u \in A} \rho(v, u)$. If more than a half of nodes $w \in A$ are closer to some u (neighbor of v), then $P(u) < P(v)$.
- If we pick the node that minimizes $P(V)$ it will eliminate at least a half of A.
- The number of steps is $\log n + 1$ and the running time is $O(n^2)$ per single query.
- That results in $O(n^3)$ per test case.

Problem D. Distance Sum

Author: Gennady Korotkevich   Statements and tests: Gennady Korotkevich





	Java	Kotlin	C++	Python	Total
Accepted	0	0	0	0	0
Rejected	2	0	8	9	19
Total	2	0	8	9	19

Problem D. Distance Sum

- Need: $\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} d(u, v)$

Problem D. Distance Sum

- Need: $\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} d(u, v)$
- Generalize! Vertex $i \to$ weight w_i
- Need: $\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} w_u w_v d(u, v)$

Problem D. Distance Sum

- While there is a vertex of degree 1:
  - Pick any such vertex v, let its only neighbor be u
  - Add $w_v \cdot (n - w_v)$ to the answer
  - Increase w_u by w_v
  - Remove v

Problem D. Distance Sum

- While there is a vertex of degree 1:
  - Pick any such vertex v, let its only neighbor be u
  - Add $w_v \cdot (n - w_v)$ to the answer
  - Increase w_u by w_v
  - Remove v
- If there is a single vertex of degree 0, exit

Problem D. Distance Sum

- While there is a vertex of degree 1:
  - Pick any such vertex v, let its only neighbor be u
  - Add $w_v \cdot (n - w_v)$ to the answer
  - Increase w_u by w_v
  - Remove v
- If there is a single vertex of degree 0, exit
- Otherwise all vertices have degree at least 2
- $\deg(v) > 2 \implies v$ is special
- $m \leq n + 42 \implies$ there are at most 84 special vertices

Problem D. Distance Sum

- 84 special vertices connected by paths of non-special vertices
- Find the distance from every special vertex to all other vertices using BFS

Problem D. Distance Sum

- 84 special vertices connected by paths of non-special vertices
- Find the distance from every special vertex to all other vertices using BFS
- $s(v) = \sum\limits_{u \neq v} w_u d(u, v)$
- The answer is $\frac{1}{2} \sum\limits_{v} s(v)$

Problem D. Distance Sum

- How to calculate $s(v)$?

Problem D. Distance Sum

- How to calculate $s(v)$?
- For special vertices u, $d(u, v)$ is already known

Problem D. Distance Sum

- How to calculate $s(v)$?
- For special vertices u, $d(u, v)$ is already known
- Consider a non-special vertex u lying on a path between special vertices u_1 and u_2:
  - $u_1, x_1, x_2, \ldots, x_k, u_2$

Problem D. Distance Sum

- How to calculate $s(v)$?
- For special vertices u, $d(u, v)$ is already known
- Consider a non-special vertex u lying on a path between special vertices u_1 and u_2:
  - $u_1, x_1, x_2, \ldots, x_k, u_2$
- The shortest path from v to u visits either u_1 or u_2
  - $\exists t \in [0; k]$: the shortest path from v to $x_1, x_2, \ldots x_t$ visits u_1, and the shortest path from v to $x_{t+1}, x_{t+2}, \ldots, x_k$ visits u_2

Problem D. Distance Sum

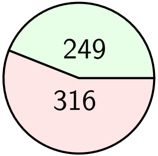
- How to calculate $s(v)$?
- For special vertices u, $d(u, v)$ is already known
- Consider a non-special vertex u lying on a path between special vertices u_1 and u_2:
  - $u_1, x_1, x_2, \ldots, x_k, u_2$
- The shortest path from v to u visits either u_1 or u_2
  - $\exists t \in [0; k]$: the shortest path from v to $x_1, x_2, \ldots x_t$ visits u_1, and the shortest path from v to $x_{t+1}, x_{t+2}, \ldots, x_k$ visits u_2
- t can be calculated based on $d(v, u_1)$, $d(v, u_2)$, and k

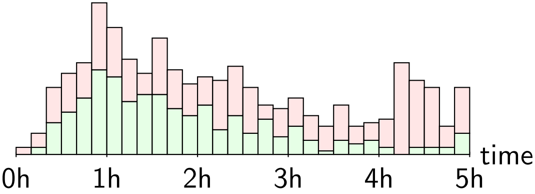
Problem D. Distance Sum

- How to calculate $s(v)$?
- For special vertices u, $d(u, v)$ is already known
- Consider a non-special vertex u lying on a path between special vertices u_1 and u_2:
  - $u_1, x_1, x_2, \ldots, x_k, u_2$
- The shortest path from v to u visits either u_1 or u_2
  - $\exists t \in [0; k]$: the shortest path from v to $x_1, x_2, \ldots x_t$ visits u_1, and the shortest path from v to $x_{t+1}, x_{t+2}, \ldots, x_k$ visits u_2
- t can be calculated based on $d(v, u_1)$, $d(v, u_2)$, and k
- The part of $s(v)$ dependent on $x_1, x_2, \ldots, x_k$ can be calculated based on prefix sums of $w_{x_1}, w_{x_2}, \ldots, w_{x_k}$ and $1 \cdot w_{x_1}, 2 \cdot w_{x_2}, \ldots, k \cdot w_{x_k}$

Problem E. Easy Chess

Author: Mikhail Dvorkin Statements and tests: Mikhail Dvorkin





	Java	Kotlin	C++	Python	Total
□ Accepted	8	1	219	21	249
□ Rejected	67	1	198	50	316
Total	75	2	417	71	565

solution	team	att	time	size	lang
Fastest	MIPT 1	1	17	2237	C++
Shortest	Ataturk-Alatoo 2	2	167	417	Python
Max atts.	IrkutskNRTU 1	54	293	3542	Java

Problem E. Easy Chess

- Visited cells per column, $n = 2$: [1, 0, 0, 0, 0, 0, 0, 2].
- Add $+1$ arbitrarily $n - 2$ times, keeping ≤ 8.

Problem E. Easy Chess

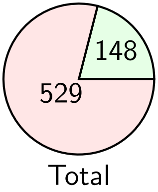
- Visited cells per column, $n = 2$: [1, 0, 0, 0, 0, 0, 0, 2].
- Add $+1$ arbitrarily $n - 2$ times, keeping ≤ 8.
- Process columns left to right. If visited[i] is 0: skip.
- Otherwise: enter at current row, arbitrarily visit some cells.

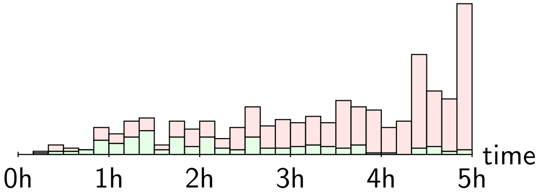
Problem E. Easy Chess

- Visited cells per column, $n = 2$: [1, 0, 0, 0, 0, 0, 0, 2].
- Add $+1$ arbitrarily $n - 2$ times, keeping ≤ 8.
- Process columns left to right. If visited[i] is 0: skip.
- Otherwise: enter at current row, arbitrarily visit some cells.
- Constraints:
  - in column a: start in row 1
  - in columns a—g: don't finish in row 8
  - in column h: finish in row 8

Problem F. Fractions

Author: Dmitry Yakutov   Statements and tests: Dmitry Yakutov





	Java	Kotlin	C++	Python	Total
Accepted	0	0	143	5	148
Rejected	4	2	512	11	529
Total	4	2	655	16	677

solution	team	att	time	size	lang
Fastest	SPbSU 3	1	14	1224	C++
Shortest	CrimeanFU 1	1	123	381	Python
Max atts.	RybinskSATU	10	298	2515	C++

Problem F. Fractions

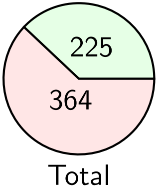
- If $n = p^k$, where $k > 0$ and p is prime, then print “NO”.
- Otherwise there exist such integers a, b that $n = a \cdot b$, $1 < a, b < n$, $\gcd(a, b) = 1$, $a \le \sqrt{n}$. You can do it in $O(\sqrt{n})$.
- Let's try to represent $\frac{n-1}{n}$ as a sum of two fractions $\frac{n-1}{n} = \frac{x}{a} + \frac{y}{b}$.
- It is always possible to find such x and y $(1 \le x < a, 1 \le y < b)$ that $\frac{n-1}{n} = \frac{x}{a} + \frac{y}{b}$.
- Just iterate over all possible values x between 1 and $a - 1$ and check that $y = \frac{n-1-x \cdot b}{a}$ is integer. It requires $O(\sqrt{n})$ steps.
- Total time complexity is $O(\sqrt{n})$.

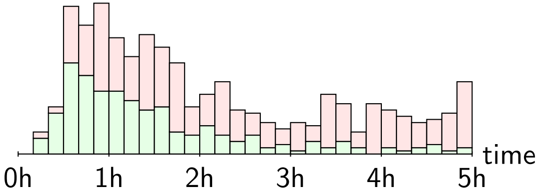
Problem F. Fractions: Proof

- For each x between 1 and $a-1$ the value $y = \frac{n-1-x\cdot b}{a} > 0$ (since $a \cdot b = n$).
- Show that $y = \frac{n-1-x\cdot b}{a}$ is integer for some x between 1 and $a-1$.
- Let's try all $x = 0 \ldots a-1$ and look on $(n-1-x\cdot b) \mod a$.
- For $x = 0$ $(n-1-0\cdot b) \mod a \neq 0$.
- If for all $x = 1 \ldots a-1$ all values $(n-1-x\cdot b) \mod a \neq 0$, then there are two equal remainder.
- Say, $(n-1-x_1\cdot b) \mod a = (n-1-x_2\cdot b) \mod a$.
- It means $(x_1-x_2)\cdot b \mod a = 0$, but it is imposible since $|x_1-x_2| < a$.

Problem G. Guest Student

Author: Mikhail Mirzayanov   Statements and tests: Mikhail Mirzayanov





	Java	Kotlin	C++	Python	Total
Accepted	4	1	208	12	225
Rejected	25	0	324	15	364
Total	29	1	532	27	589

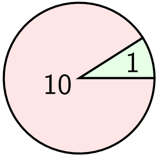
solution	team	att	time	size	lang
Fastest	IvanovoSPowU	1	14	810	C++
Shortest	YerevanSU 2	2	64	443	Python
Max atts.	IrkutskSU 3	9	242	1456	C++

Problem G. Guest Student

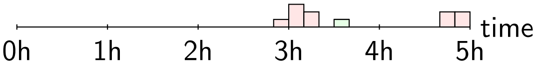
- Try each day of a week to start education.
- For each fixed starting day of week solve the problem independently.
- Calculate number of whole weeks. Roughly speaking, number of whole weeks is $\approx k/(a_1 + a_2 + \cdots + a_7)$. Process the remainder day by day.
- Return the best result over all possible 7 starting days of a week.

Problem H. Harder Satisfiability

Author: Andrey Stankevich Statements and tests: Artem Vasilyev



Total



	Java	Kotlin	C++	Python	Total
Accepted	0	0	1	0	1
Rejected	0	0	10	0	10
Total	0	0	11	0	11

solution	team	att	time	size	lang
The only	MSU 3	3	210	2318	C++

Problem H. Harder Satisfiability

Recall a solution for classic 2-SAT problem:

- Convert formula to implication graph with vertices x_i, $\overline{x_i}$ for all variables: $x \vee y \Rightarrow (\overline{x} \rightarrow y), (\overline{y} \rightarrow x)$
- Important property: if x is reachable from y, than $\overline{y}$ is reachable from $\overline{x}$.
- Find strong connected components (SCC) in this graph.
- If for any vertex x and $\overline{x}$ in same component, no solution exists.
- Otherwise, assign values to variables using the topological order.

Problem H. Harder Satisfiability

Necessary conditions:

- For all i: x_i and $\overline{x_i}$ are not in same SCC.
- For all $i < j$ with $\exists$ quantifier near x_i and $\forall$ near x_j: none of x_i, $\overline{x_i}$ is in one SCC with x_j, $\overline{x_j}$.
- For all i, j with $\forall$ quantifiers: none of x_i, $\overline{x_i}$, x_j, $\overline{x_j}$ are not reachable from each other.

Problem H. Harder Satisfiability

And they are sufficient!

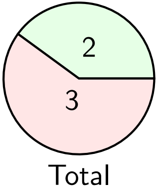
- For all i: x_i and $\overline{x_i}$ not in same SCC.
- For all $i < j$ with $\exists$ quantifier near x_i and $\forall$ near x_j: none of x_i, $\overline{x_i}$ is in one SCC with x_j, $\overline{x_j}$.
- For all i, j with $\forall$ quantifiers: none of x_i, $\overline{x_i}$, x_j, $\overline{x_j}$ are not reachable from each other.

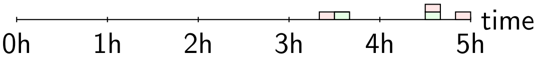
Assign algorithm:

- Condition 2 allows us to mark SCC as "any", if it have a $\forall$ vertex inside.
- Let's make all components reachable from "any" as true, and their negations as false.
- This doesn't lead to a contradiction, because if some vertex is reachable from one *forall* and some *forall* is reachable from it, then the third condition fails.
- Other components can be decided the same way as classic 2-SAT

Problem I. Interval-Free Permutations

Author: Andrey Stankevich Statements and tests: Pavel Kunyavsky





	Java	Kotlin	C++	Python	Total
Accepted	0	0	2	0	2
Rejected	0	0	3	0	3
Total	0	0	5	0	5

solution	team	att	time	size	lang
Fastest	MIPT 6	2	216	2063	C++
Shortest	MSU 3	2	279	1468	C++
Max atts.	MSU 3	2	279	1468	C++

Problem I. Interval-Free Permutations

- Use dynamic programming. Count all permutations and subtract bad ones
- Let's call an interval a block, if it's not inside any other interval, except for an entire permutation.
- Two blocks either non-intersecting or cover full permutation

Problem I. Interval-Free Permutations

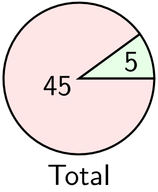
- Permutation is either split by at least three blocks or covered by two.
- In the first case, splitting is unique. We can choose any permutation in each block, while permutation of blocks should be good.
- In the second case, there are several ways to split. To make it unique, we need to force left permutation have no prefixes, which are permutation.

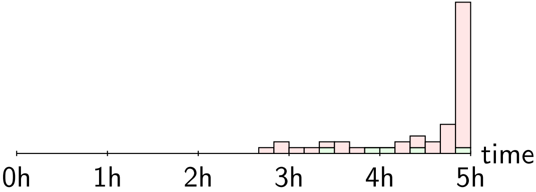
Problem I. Interval-Free Permutations

- Number of permutation, none of prefixes of which is permutation.
  $$I_n = n! - \sum_{k=1}^{n-1} I_k \cdot (n-k)!$$
- Number of ways to choose k permutations of total length n.
  $$B_{n,k} = \sum_{t=1}^{n} B_{n-t,k-1} \cdot t!$$
- The answer
  $$A_n = n! - 2 \cdot \sum_{k=1}^{n-1} I_k \cdot (n-k)! - \sum_{k=3}^{n-1} B_{n,k} \cdot A_k$$

Problem J. JS Minification

Author: Roman Elizarov Statements and tests: Roman Elizarov





	Java	Kotlin	C++	Python	Total
Accepted	0	0	5	0	5
Rejected	0	0	44	1	45
Total	0	0	49	1	50

solution	team	att	time	size	lang
Fastest	MIPT 6	2	203	7724	C++
Shortest	MSU 3	1	242	4493	C++
Max atts.	ITMO 4	5	261	6512	C++

Problem J. JS Minification

- Parse input
  - Either longest word/number or a reserved token

Problem J. JS Minification

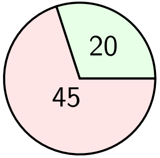
- Parse input
  - Either longest word/number or a reserved token
- Rename words
  - Remember to skip reserved

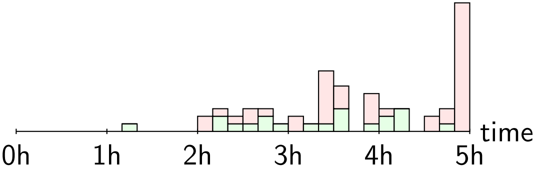
Problem J. JS Minification

- Parse input
  - Either longest word/number or a reserved token
- Rename words
  - Remember to skip reserved
- Write output
  - Greedily insert spaces when needed
  - Keep a list of tokens written since the last space

Problem K. King Kog's Reception

Author: Vitaliy Aksenov   Statements and tests: Vitaliy Aksenov





	Java	Kotlin	C++	Python	Total
Accepted	0	0	20	0	20
Rejected	0	0	45	0	45
Total	0	0	65	0	65

solution	team	att	time	size	lang
Fastest	MIPT 6	1	79	3191	C++
Shortest	SPbHSE 1	2	250	2168	C++
Max atts.	UofLatvia 1	5	219	3819	C++

Problem K. King Kog's Reception

- Let t_i = arrival time, d_i = duration time.

Problem K. King Kog's Reception

- Let t_i = arrival time, d_i = duration time.
- For query with time T the answer is:

$$\max_{i:t_i \le T} (t_i + \sum_{j:t_i \le t_j \le T} d_j)$$

- (Knight i comes in his time, all next knights wait consecutively.)

Problem K. King Kog's Reception

- Let t_i = arrival time, d_i = duration time.
- For query with time T the answer is:

$$\max_{i: t_i \leq T} (t_i + \sum_{j: t_i \leq t_j \leq T} d_j)$$

- (Knight i comes in his time, all next knights wait consecutively.)
- Make the maximized formula independent of T:

$$\max_{i: t_i \leq T} (t_i + \sum_{j: t_i \leq t_j} d_j) - \sum_{j: T < t_j} d_j$$

Problem K. King Kog's Reception

- Let t_i = arrival time, d_i = duration time.
- For query with time T the answer is:

$$\max_{i:t_i \le T}(t_i + \sum_{j:t_i \le t_j \le T} d_j)$$

- (Knight i comes in his time, all next knights wait consecutively.)
- Make the maximized formula independent of T:

$$\max_{i:t_i \le T}(t_i + \sum_{j:t_i \le t_j} d_j) - \sum_{j:T < t_j} d_j$$

- Right part: interval tree with sum

Problem K. King Kog's Reception

- Let t_i = arrival time, d_i = duration time.
- For query with time T the answer is:

$$\max_{i:t_i \leq T} (t_i + \sum_{j:t_i \leq t_j \leq T} d_j)$$

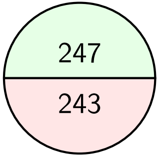
- (Knight i comes in his time, all next knights wait consecutively.)
- Make the maximized formula independent of T:

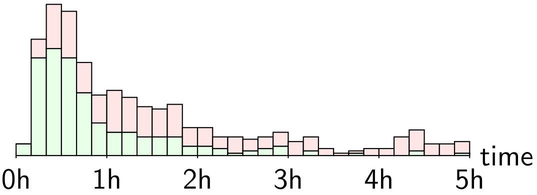
$$\max_{i:t_i \leq T} (t_i + \sum_{j:t_i \leq t_j} d_j) - \sum_{j:T < t_j} d_j$$

- Right part: interval tree with sum
- Left part: adding/removing a knight affect a segment.
- So, interval tree with addition on a segment.

Problem L. Lazyland

Author: Pavel Mavrin   Statements and tests: Pavel Mavrin





	Java	Kotlin	C++	Python	Total
Accepted	7	1	226	13	247
Rejected	14	0	204	25	243
Total	21	1	430	38	490

solution	team	att	time	size	lang
Fastest	BelarusianSU 1	1	7	965	C++
Shortest	SUrSU 3	1	30	367	Python
Max atts.	RybinskSATU	8	121	955	C++

Problem L. Lazyland

- For each job, find the idler with the maximum value of b_i.
- Assign this job to this idler.

Problem L. Lazyland

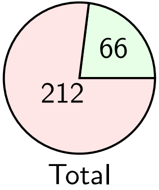
- For each job, find the idler with the maximum value of b_i.
- Assign this job to this idler.
- Put all remaining idlers in the array, sort them by the value of b_i.

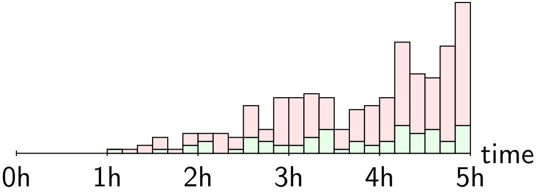
Problem L. Lazyland

- For each job, find the idler with the maximum value of b_i.
- Assign this job to this idler.
- Put all remaining idlers in the array, sort them by the value of b_i.
- Assign idlers with minimum values to remaining jobs.

Problem M. Minegraphed

Author: Mikhail Dvorkin   Statements and tests: Mikhail Dvorkin





	Java	Kotlin	C++	Python	Total
Accepted	1	0	65	0	66
Rejected	6	0	206	0	212
Total	7	0	271	0	278

solution	team	att	time	size	lang
Fastest	ITMO 1	1	67	2304	C++
Shortest	UrFU 7	3	254	1844	C++
Max atts.	TbilisiIBSU 1	9	255	2286	C++

Problem M. Minegraphed

- Possible construnction: $3n \times 3n \times 3$ parallelepiped.

Problem M. Minegraphed

- Possible construnction: $3n \times 3n \times 3$ parallelepiped.
- Vertex i: north-south tunnel in bottom layer + west-east tunnel in top layer.
- Full-length tunnels with 2 walls between: $\{x = 3i \wedge z = 0\} \cup \{y = 3i \wedge z = 2\}$.

Problem M. Minegraphed

- Possible construnction: $3n \times 3n \times 3$ parallelepiped.
- Vertex i: north-south tunnel in bottom layer + west-east tunnel in top layer.
- Full-length tunnels with 2 walls between: $\{x = 3i \wedge z = 0\} \cup \{y = 3i \wedge z = 2\}$.
- For each i, make 2-step "staircase" from bottom to top.

Problem M. Minegraphed

- Possible construnction: $3n \times 3n \times 3$ parallelepiped.
- Vertex i: north-south tunnel in bottom layer + west-east tunnel in top layer.
- Full-length tunnels with 2 walls between: $\{x = 3i \wedge z = 0\} \cup \{y = 3i \wedge z = 2\}$.
- For each i, make 2-step "staircase" from bottom to top.
- For each edge $i \to j$, make vertical hole to fall from i-th top tunnel to j-th bottom tunnel.

Credits

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

Andery Halyavin, Andrey Stankevich, Artem Vasilyev, Borys Minaiev, Dmitry Yakutov, Evgeniy Kuprilyanskiy, Gennady Korotkevich, Gleb Evstropov, Ilya Zban, Ivan Belonogov, Ivan Kazmenko, Mikhail Dvorkin, Mikhail Mirzayanov, Mikhail Tikhomirov, Niyaz Nigmatullin, Oleg Hristenko, Pavel Irzhavski, Pavel Kunyavskiy, Pavel Mavrin, Roman Elizarov, Vitaly Aksenov