# Contest 1

March 9, 2019

Mikhail Tikhomirov

Moscow ICPC Prefinals Workshop 2019

# A. A Two Floors Dungeon

There is grid dungeon, with each cell either on the first floor or the second floor. We can move between adjacent cells (provided they are on the same floor), or press switches in the current cell that flip floors for a subset of cells. Find the shortest time to get to the exit.

# A. A Two Floors Dungeon

Construct a graph with each vertex completely describing the state: our current coordinates, and the subset of switches currently on. Perform a BFS in this graph.

# A. A Two Floors Dungeon

Construct a graph with each vertex completely describing the state: our current coordinates, and the subset of switches currently on. Perform a BFS in this graph.

The number of vertices is $O(wh2^n)$, where $n$ is the number of switches; degree of each vertex is $O(1)$.

# A. A Two Floors Dungeon

Construct a graph with each vertex completely describing the state: our current coordinates, and the subset of switches currently on. Perform a BFS in this graph.

The number of vertices is $O(wh2^n)$, where $n$ is the number of switches; degree of each vertex is $O(1)$.

To move between cells we have to check if they are on the same floor with the current set of switches; can optimize the check to $O(1)$ with precomputation,

# A. A Two Floors Dungeon

Construct a graph with each vertex completely describing the state: our current coordinates, and the subset of switches currently on. Perform a BFS in this graph.

The number of vertices is $O(wh2^n)$, where $n$ is the number of switches; degree of each vertex is $O(1)$.

To move between cells we have to check if they are on the same floor with the current set of switches; can optimize the check to $O(1)$ with precomputation,

The total complexity is $O(wh2^n)$.

# B. Billiard

There are $n$ balls on a rectangular table. We hit ball 1 in a given direction; the ball reflects perfectly from the table sides. Determine the first collision, or that it doesn't happen for the first 10000 units travelled.

# B. Billiard

Can get away with a careful simulation: track reflections and check for collisions between them. There can be at most 10000 reflections.

# B. Billiard

Can get away with a careful simulation: track reflections and check for collisions between them. There can be at most 10000 reflections.

Can shrink ball 1 to a point and expand all others; need only the segment-circle intersection primitive.

# B. Billiard

Can get away with a careful simulation: track reflections and check for collisions between them. There can be at most 10000 reflections.

Can shrink ball 1 to a point and expand all others; need only the segment-circle intersection primitive.

Precision should not be an issue (can use rationals anyway).

# B. Billiard

Can get away with a careful simulation: track reflections and check for collisions between them. There can be at most 10000 reflections.

Can shrink ball 1 to a point and expand all others; need only the segment-circle intersection primitive.

Precision should not be an issue (can use rationals anyway).

Complexity is $O(nD)$, with $D = 10000$.

# C. Counting 1's

We have a sequence $k_1, \ldots, k_n$. Find such integers $A, B$ such that $1 \leqslant A \leqslant B \leqslant 10^{18}$ and for all $i$, there are exactly $k_i$ integers in the segment $[A, B]$ with $i$-th smallest bit set to 1.

# C. Counting 1's

Let $b$ be the highest bit such that $k_b \neq 0$. Note that in this case $B \in [2^b, 2^{b+1})$. There are two cases:

- $A \geqslant 2^b$: all numbers in $[A, B]$ have $b$-th bit set to 1; forget about $k_b$ and process the rest recursively.

# C. Counting 1's

Let $b$ be the highest bit such that $k_b \neq 0$. Note that in this case $B \in [2^b, 2^{b+1})$. There are two cases:

- $A \geqslant 2^b$: all numbers in $[A, B]$ have $b$-th bit set to 1; forget about $k_b$ and process the rest recursively.
- $A < 2^b$: $B = 2^b + k_b - 1$. Decrease $B$ to $2^b - 1$ while changing $k_i$ accordingly (can calculate changes in $O(n)$ time).

# C. Counting 1's

Let $b$ be the highest bit such that $k_b \neq 0$. Note that in this case $B \in [2^b, 2^{b+1})$. There are two cases:

- $A \geqslant 2^b$: all numbers in $[A, B]$ have $b$-th bit set to 1; forget about $k_b$ and process the rest recursively.
- $A < 2^b$: $B = 2^b + k_b - 1$. Decrease $B$ to $2^b - 1$ while changing $k_i$ accordingly (can calculate changes in $O(n)$ time). The length of the segment is either $2k_1$ or $2k_1 - 1$; check both options explicitly (again, in $O(n)$ time).

# C. Counting 1's

Let $b$ be the highest bit such that $k_b \neq 0$. Note that in this case $B \in [2^b, 2^{b+1})$. There are two cases:

- $A \geqslant 2^b$: all numbers in $[A, B]$ have $b$-th bit set to 1; forget about $k_b$ and process the rest recursively.
- $A < 2^b$: $B = 2^b + k_b - 1$. Decrease $B$ to $2^b - 1$ while changing $k_i$ accordingly (can calculate changes in $O(n)$ time). The length of the segment is either $2k_1$ or $2k_1 - 1$; check both options explicitly (again, in $O(n)$ time).

Don't forget to check if $A$ and $B$ are not larger than $10^{18}$.

# C. Counting 1's

Let $b$ be the highest bit such that $k_b \neq 0$. Note that in this case $B \in [2^b, 2^{b+1})$. There are two cases:

- $A \geqslant 2^b$: all numbers in $[A, B]$ have $b$-th bit set to 1; forget about $k_b$ and process the rest recursively.
- $A < 2^b$: $B = 2^b + k_b - 1$. Decrease $B$ to $2^b - 1$ while changing $k_i$ accordingly (can calculate changes in $O(n)$ time). The length of the segment is either $2k_1$ or $2k_1 - 1$; check both options explicitly (again, in $O(n)$ time).

Don't forget to check if $A$ and $B$ are not larger than $10^{18}$.

The total complexity is $O(n^2)$ per test case.

# D. Do Use Segment Tree

We are given a tree with weights in vertices. Process queries "assign weights on the simple path between vertices $a$ and $b$ to $c$", and "find the largest total weight of a consecutive subsegment of the simple path between two vertices".

# D. Do Use Segment Tree

Straightforward heavy-light decomposition application.

# D. Do Use Segment Tree

Straightforward heavy-light decomposition application.

HLD crash course: decompose the tree into vertical paths; assign a vertex $v$ to be in the same path with its parent $p$ if the subtree of $v$ contains at least half of vertices of the subtree of $p$.

# D. Do Use Segment Tree

Straightforward heavy-light decomposition application.

HLD crash course: decompose the tree into vertical paths; assign a vertex $v$ to be in the same path with its parent $p$ if the subtree of $v$ contains at least half of vertices of the subtree of $p$.

Construct a segment tree for each path in the decomposition that would solve the original problem for an array.

# D. Do Use Segment Tree

Straightforward heavy-light decomposition application.

HLD crash course: decompose the tree into vertical paths; assign a vertex $v$ to be in the same path with its parent $p$ if the subtree of $v$ contains at least half of vertices of the subtree of $p$.

Construct a segment tree for each path in the decomposition that would solve the original problem for an array.

Each simple path in the tree can be split into $O(\log n)$ segments each completely covered by a decomposition path; process each query in $O(\log^2 n)$ time.

# E. Eclipse

After careful reading, we need to find the smallest positive integer $x$ such that

$$\sum_{i=1}^{n} p_i O_i x^{O_i - 1} = 0.$$

$O_i = 0$ should be ignored.

# E. Eclipse

First of all, we can group all summands with equal $O_i$ together, cancelling out $x^j$ with zero coefficient.

# E. Eclipse

First of all, we can group all summands with equal $O_i$ together, cancelling out $x^j$ with zero coefficient.

If the result is non-zero, Bezout's theorem implies that $x$ must divide the non-zero coefficient $O_i p_i$ with the smallest power of $x$.

# E. Eclipse

First of all, we can group all summands with equal $O_i$ together, cancelling out $x^j$ with zero coefficient.

If the result is non-zero, Bezout's theorem implies that $x$ must divide the non-zero coefficient $O_i p_i$ with the smallest power of $x$.

Can factorize $O_i$ and $p_i$ separately with Pollard's rho algorithm.

# E. Eclipse

First of all, we can group all summands with equal $O_i$ together, cancelling out $x^j$ with zero coefficient.

If the result is non-zero, Bezout's theorem implies that $x$ must divide the non-zero coefficient $O_i p_i$ with the smallest power of $x$.

Can factorize $O_i$ and $p_i$ separately with Pollard's rho algorithm.

To check for equality, compute the polynomial modulo several random primes.

# F. Figures and Trees

We have to contruct a binary tree $T$ from an initial tree $T_s$ of three vertices. There is a copy buffer initially containing $T_s$. We can perform two operations:

- Store the current tree in the buffer.
- Paste the contents of the buffer to any leaf of the current tree (the root of the buffer merges with the leaf).

What's the smallest number of paste operations required?

# F. Figures and Trees

Consider the last tree $T'$ to be in the buffer; then $T$ must be constructable starting from $T'$. How do we check for this?

# F. Figures and Trees

Consider the last tree $T'$ to be in the buffer; then $T$ must be constructable starting from $T'$. How do we check for this?

Let $k$ be the number of *edges* of $T'$. Consider a vertex $v$ of $T$ containing at least $k$ edges in its subtree; also assume that $v$ is most distant from the root with this property.

# F. Figures and Trees

Consider the last tree $T'$ to be in the buffer; then $T$ must be constructable starting from $T'$. How do we check for this?

Let $k$ be the number of *edges* of $T'$. Consider a vertex $v$ of $T$ containing at least $k$ edges in its subtree; also assume that $v$ is most distant from the root with this property. One can see that the subtree of $v$ must be equal to $T'$, and must be obtained with a single paste operation.

# F. Figures and Trees

Consider the last tree $T'$ to be in the buffer; then $T$ must be constructable starting from $T'$. How do we check for this?

Let $k$ be the number of *edges* of $T'$. Consider a vertex $v$ of $T$ containing at least $k$ edges in its subtree; also assume that $v$ is most distant from the root with this property. One can see that the subtree of $v$ must be equal to $T'$, and must be obtained with a single paste operation.

Therefore, a greedy algorithm works: take the lowest vertex $v$ of $T$ with at least $k$ edges in its subtree, check if its subtree coincides with $T'$, remove the subtree of $v$ from $T$. Can do this in time linear in the size of $T$.

# F. Figures and Trees

Consider the last tree $T'$ to be in the buffer; then $T$ must be constructable starting from $T'$. How do we check for this?

Let $k$ be the number of *edges* of $T'$. Consider a vertex $v$ of $T$ containing at least $k$ edges in its subtree; also assume that $v$ is most distant from the root with this property. One can see that the subtree of $v$ must be equal to $T'$, and must be obtained with a single paste operation.

Therefore, a greedy algorithm works: take the lowest vertex $v$ of $T$ with at least $k$ edges in its subtree, check if its subtree coincides with $T'$, remove the subtree of $v$ from $T$. Can do this in time linear in the size of $T$.

The problem is now to construct an optimal sequence $T_0 = T, T_1, \ldots, T_m = T_s$ such that $T_i$ can be constructed from $T_{i+1}$.

# F. Figures and Trees

A few more observations:

- The number of edges of any $T_i$ must divide the number of edges of $T_{i-1}$ (and so, must divide the number of edges of $T$).

# F. Figures and Trees

A few more observations:

- The number of edges of any $T_i$ must divide the number of edges of $T_{i-1}$ (and so, must divide the number of edges of $T$).
- For any number of edges $k$, all trees with $k$ edges appearing in any sequence must be equal (and, indeed, equal to any of the lowest subtrees of $T$ with $k$ edges).

# F. Figures and Trees

A few more observations:

- The number of edges of any $T_i$ must divide the number of edges of $T_{i-1}$ (and so, must divide the number of edges of $T$).
- For any number of edges $k$, all trees with $k$ edges appearing in any sequence must be equal (and, indeed, equal to any of the lowest subtrees of $T$ with $k$ edges).

It now suffices to obtain all divisors of the number of edges of $T$, and for each pair of divisors check if respective trees are obtainable from each other.

# F. Figures and Trees

A few more observations:

- The number of edges of any $T_i$ must divide the number of edges of $T_{i-1}$ (and so, must divide the number of edges of $T$).
- For any number of edges $k$, all trees with $k$ edges appearing in any sequence must be equal (and, indeed, equal to any of the lowest subtrees of $T$ with $k$ edges).

It now suffices to obtain all divisors of the number of edges of $T$, and for each pair of divisors check if respective trees are obtainable from each other.

For a tree with $n$ edges, the complexity can be bounded as $\sum_{d|n} d \times \#(\text{divisors of } d)$, which is small enough.

# G. Game with Tiles

We have at most 15 tiles of twelve types, each containing roads connecting some of the tile borders. How many ways are there to fill an $R \times C$ grid with these tiles so that the opposite corners of the grid are connected with roads?

# G. Game with Tiles

Since $R \times C \leqslant 15$, then $\min(R, C) \leqslant 3$. We will use "curved profile" DP, with states storing all necessary information:

- multiplicities of all used tiles (note that this implies the number of processed cells);

# G. Game with Tiles

Since $R \times C \leqslant 15$, then $\min(R, C) \leqslant 3$. We will use "curved profile" DP, with states storing all necessary information:

- multiplicities of all used tiles (note that this implies the number of processed cells);
- partition of tiles in the profile into connected components;

# G. Game with Tiles

Since $R \times C \leqslant 15$, then $\min(R, C) \leqslant 3$. We will use "curved profile" DP, with states storing all necessary information:

- multiplicities of all used tiles (note that this implies the number of processed cells);
- partition of tiles in the profile into connected components;
- which of the components contains the top left corner cell.

# G. Game with Tiles

Since $R \times C \leqslant 15$, then $\min(R, C) \leqslant 3$. We will use "curved profile" DP, with states storing all necessary information:

- multiplicities of all used tiles (note that this implies the number of processed cells);
- partition of tiles in the profile into connected components;
- which of the components contains the top left corner cell.

The number of different multiplicities is at most 15 thousand, and the number of connectedness arrangements is as small as 10.

# G. Game with Tiles

Since $R \times C \leqslant 15$, then $\min(R, C) \leqslant 3$. We will use "curved profile" DP, with states storing all necessary information:

- multiplicities of all used tiles (note that this implies the number of processed cells);
- partition of tiles in the profile into connected components;
- which of the components contains the top left corner cell.

The number of different multiplicities is at most 15 thousand, and the number of connectedness arrangements is as small as 10.

To reduce constant factor, precompute transitions as much as possible.

# H. Hashigo Sama

Given a graph $G$ obtained by merging $n$ ladder graphs, find the number of ways to color it into two colors so that no connected component of the same color contains more than $k$ vertices.

# H. Hashigo Sama

An application of tree decomposition DP.

# H. Hashigo Sama

An application of tree decomposition DP.

Let us construct an auxiliary graph $T$ with each vertex corresponding to a $2 \times 2$ square of cells present in a ladder, and edges connecting squares sharing a side. Merging two ladders results in merging respective vertices in $T$.

# H. Hashigo Sama

An application of tree decomposition DP.

Let us construct an auxiliary graph $T$ with each vertex corresponding to a $2 \times 2$ square of cells present in a ladder, and edges connecting squares sharing a side. Merging two ladders results in merging respective vertices in $T$. The statement implies that $T$ is a tree.

# H. Hashigo Sama

An application of tree decomposition DP.

Let us construct an auxiliary graph $T$ with each vertex corresponding to a $2 \times 2$ square of cells present in a ladder, and edges connecting squares sharing a side. Merging two ladders results in merging respective vertices in $T$. The statement implies that $T$ is a tree.

Root $T$ at any vertex. For a vertex $v$ of $T$, we now want to compute $dp_{v,c}$ — the number of colorings of the part of $G$ induced by the subtree of $v$. Here parameter $c$ describes colors of four vertices of $G$ covered by $v$, as well as sizes of their components.

# H. Hashigo Sama

An application of tree decomposition DP.

Let us construct an auxiliary graph $T$ with each vertex corresponding to a $2 \times 2$ square of cells present in a ladder, and edges connecting squares sharing a side. Merging two ladders results in merging respective vertices in $T$. The statement implies that $T$ is a tree.

Root $T$ at any vertex. For a vertex $v$ of $T$, we now want to compute $dp_{v,c}$ — the number of colorings of the part of $G$ induced by the subtree of $v$. Here parameter $c$ describes colors of four vertices of $G$ covered by $v$, as well as sizes of their components.

The number of different values of $c$ can be bounded by $(2k)^4$, but is actually much smaller.

# H. Hashigo Sama

To make transitions, consider children of $v$ in $T$ one by one, and combine all pairs of parameters $c$ and $c'$ that agree on the colors of shared vertices of $G$; recompute $c$ accordingly (possibly merge some components and add their sizes; make sure that no component has more than $k$ vertices).

# H. Hashigo Sama

To make transitions, consider children of $v$ in $T$ one by one, and combine all pairs of parameters $c$ and $c'$ that agree on the colors of shared vertices of $G$; recompute $c$ accordingly (possibly merge some components and add their sizes; make sure that no component has more than $k$ vertices).

This should work fast enough; still lots of things to optimize (for instance, we only care about two vertices in $c'$ when introducing a subtree).

# H. Hashigo Sama

To make transitions, consider children of $v$ in $T$ one by one, and combine all pairs of parameters $c$ and $c'$ that agree on the colors of shared vertices of $G$; recompute $c$ accordingly (possibly merge some components and add their sizes; make sure that no component has more than $k$ vertices).

This should work fast enough; still lots of things to optimize (for instance, we only care about two vertices in $c'$ when introducing a subtree).

A *tree decomposition* is a structure similar to $T$ constructed for any given graph. A lot of problems can be solved fast when each vertex of $T$ is "responsible" for a small number of vertices of $G$.

# I. Interesting Game

We are given $r$ words $s_1, \ldots, s_r$ of size at most $c$. For each $i$, we need to write $s_i$ in some of the cells of the $i$-th row of a $r \times c$ grid without changing relative order of letters. Maximize the number of pairs of equal adjacent letters.

# I. Interesting Game

Suppose that we are placing letters in row-major order. For the current position $(i, j)$, we can either place the next letter of $s_i$ (if any) at $(i, j)$, or skip it.

# I. Interesting Game

Suppose that we are placing letters in row-major order. For the current position $(i, j)$, we can either place the next letter of $s_i$ (if any) at $(i, j)$, or skip it.

"Curved profile" DP $dp_{i,j,mask}$ — the largest score we can obtain up to the cell $(i, j)$ if the subset of columns with their lowest cells occupied is $mask$. $O(rc2^c)$ states, two transitions per state.

# I. Interesting Game

Suppose that we are placing letters in row-major order. For the current position $(i, j)$, we can either place the next letter of $s_i$ (if any) at $(i, j)$, or skip it.

"Curved profile" DP $dp_{i,j,mask}$ — the largest score we can obtain up to the cell $(i, j)$ if the subset of columns with their lowest cells occupied is $mask$. $O(rc2^c)$ states, two transitions per state.

Note that $mask$ contains enough information to determine which letter of $s_i$ is the next, and which exact letter is contained in any of the relevant cells.

# I. Interesting Game

Suppose that we are placing letters in row-major order. For the current position $(i, j)$, we can either place the next letter of $s_i$ (if any) at $(i, j)$, or skip it.

"Curved profile" DP $dp_{i,j,mask}$ — the largest score we can obtain up to the cell $(i, j)$ if the subset of columns with their lowest cells occupied is $mask$. $O(rc2^c)$ states, two transitions per state.

Note that $mask$ contains enough information to determine which letter of $s_i$ is the next, and which exact letter is contained in any of the relevant cells.

Can reduce memory consumption to $O(2^c)$ by only storing the values for the last two processed cells.

# I. Interesting Game

Suppose that we are placing letters in row-major order. For the current position $(i, j)$, we can either place the next letter of $s_i$ (if any) at $(i, j)$, or skip it.

"Curved profile" DP $dp_{i,j,mask}$ — the largest score we can obtain up to the cell $(i, j)$ if the subset of columns with their lowest cells occupied is $mask$. $O(rc2^c)$ states, two transitions per state.

Note that $mask$ contains enough information to determine which letter of $s_i$ is the next, and which exact letter is contained in any of the relevant cells.

Can reduce memory consumption to $O(2^c)$ by only storing the values for the last two processed cells.

With enough precomputation, can achieve complexity $O(rc2^c)$.

# J. Join Usoperanto Community

There are $n$ words in a phrase, each having a length $l_i$. Each word may also possibly modify another word, with modification structure forming a forest. The penalty for a modification in a particular ordering of words is the total length of words between the two words involved in modification. Determine the optimal order of words.

# J. Join Usoperanto Community

A simple exchange argument shows that any subtree in the forest should form a contiguous subsequence of words. We are only free to choose the order of subtrees for each vertex.

# J. Join Usoperanto Community

A simple exchange argument shows that any subtree in the forest should form a contiguous subsequence of words. We are only free to choose the order of subtrees for each vertex.

Observe that for $k$ subtrees with total length of words inside them equal to $s_1, \ldots, s_k$, the total penalty for this order is $s_{k-1} + 2s_{k-2} + \ldots$, hence it is optimal to sort the subtrees by increasing of their sizes.

# J. Join Usoperanto Community

A simple exchange argument shows that any subtree in the forest should form a contiguous subsequence of words. We are only free to choose the order of subtrees for each vertex.

Observe that for $k$ subtrees with total length of words inside them equal to $s_1, \ldots, s_k$, the total penalty for this order is $s_{k-1} + 2s_{k-2} + \ldots$, hence it is optimal to sort the subtrees by increasing of their sizes.

The complexity is $O(n \log n)$.

# K. King and Pipes

There is a water pipe network, represented as a set of segments in the plane (intersecting segments are communicating). A water supply, several valves, and a repair point are located. We have to disconnect the repair point from the supply by shutting some of the valves. What is the smallest total length of pipes needed to disconnect from the supply?

# K. King and Pipes

First, construct the actual graph of the network. No edges of this graph should contain segment endpoints, segment intersections or any important points (supply, repair point, valves) inside them.

# K. King and Pipes

First, construct the actual graph of the network. No edges of this graph should contain segment endpoints, segment intersections or any important points (supply, repair point, valves) inside them.

Starting from the repair point, run a DFS of this graph. Whenever we encounter a valve, we shut it and do not proceed in the search. If we find a way to the supply that does not contain any valves, the answer is clearly $-1$. Otherwise, the total length of traversed edges is the answer.

# K. King and Pipes

First, construct the actual graph of the network. No edges of this graph should contain segment endpoints, segment intersections or any important points (supply, repair point, valves) inside them.

Starting from the repair point, run a DFS of this graph. Whenever we encounter a valve, we shut it and do not proceed in the search. If we find a way to the supply that does not contain any valves, the answer is clearly $-1$. Otherwise, the total length of traversed edges is the answer.

Why is it optimal? If we do not shut any immediately reachable valve, we will still have to shut a valve further down the way, which will only make the answer worse.

# K. King and Pipes

First, construct the actual graph of the network. No edges of this graph should contain segment endpoints, segment intersections or any important points (supply, repair point, valves) inside them.

Starting from the repair point, run a DFS of this graph. Whenever we encounter a valve, we shut it and do not proceed in the search. If we find a way to the supply that does not contain any valves, the answer is clearly $-1$. Otherwise, the total length of traversed edges is the answer.

Why is it optimal? If we do not shut any immediately reachable valve, we will still have to shut a valve further down the way, which will only make the answer worse.

The complexity is linear in the size of the graph, which is $O(n^2 + m)$ ($n$ and $m$ are the numbers of segments and valves respectively).

# L. Let's Move On Dice

We have a six-faced die with a string written on each face. There is also a grid with a start and a goal cell. We want to roll the die from the start to the goal while respecting restrictions on the rolling directions in each cell. If we concatenate all strings visible on the top face after each roll, what is the lexicographically smallest string we can obtain? It could be that there is no finite smallest string.

# L. Let's Move On Dice

First, remove all cells that are not reachable from the start, or such that the goal is not reachable from them. The state of the process is described by the cell coordinates, as well as one of $S = 6 \times 4$ possible orientations of the cube.

# L. Let's Move On Dice

First, remove all cells that are not reachable from the start, or such that the goal is not reachable from them. The state of the process is described by the cell coordinates, as well as one of $S = 6 \times 4$ possible orientations of the cube.

We now try to repeatedly improve the answer for all states (much like in Bellman-Ford algorithm). If the answer is finite, then after *Shw* iterations there will be no improvements in any vertex that could possibly lead to improving the final answer.

# L. Let's Move On Dice

First, remove all cells that are not reachable from the start, or such that the goal is not reachable from them. The state of the process is described by the cell coordinates, as well as one of $S = 6 \times 4$ possible orientations of the cube.

We now try to repeatedly improve the answer for all states (much like in Bellman-Ford algorithm). If the answer is finite, then after *Shw* iterations there will be no improvements in any vertex that could possibly lead to improving the final answer.

To check for that, run Bellman-Ford for *Shw* more iterations and see any of the intermediate answers are lexicographically smaller than the provisional one.

# L. Let's Move On Dice

First, remove all cells that are not reachable from the start, or such that the goal is not reachable from them. The state of the process is described by the cell coordinates, as well as one of $S = 6 \times 4$ possible orientations of the cube.

We now try to repeatedly improve the answer for all states (much like in Bellman-Ford algorithm). If the answer is finite, then after *Shw* iterations there will be no improvements in any vertex that could possibly lead to improving the final answer.

To check for that, run Bellman-Ford for *Shw* more iterations and see any of the intermediate answers are lexicographically smaller than the provisional one.

Since we have to compare the answers explicitly, the complexity is $O((SHW)^2 L)$, where $L$ is the largest length among strings of die's faces. Can get rid of $O(L)$ factor with precomputation and two-pointers comparison.

# M. Monolith

You are given a set of glyphs as black and white pictures. There is also a picture of rectangles and glyphs, with each rectangle enclosing a phrase (possibly with more rectangles), and each glyph possibly reversed. According to transcription rules, recover the phrase.

# M. Monolith

First, parse the picture into a sequence of brackets and glyphs (possibly reversed). One can do that with a recursive descent function, parsing insides of a rectangle (note that each rectangle is a bounding box for its insides, and bottom pixels of all terms inside are aligned).

# M. Monolith

First, parse the picture into a sequence of brackets and glyphs (possibly reversed). One can do that with a recursive descent function, parsing insides of a rectangle (note that each rectangle is a bounding box for its insides, and bottom pixels of all terms inside are aligned).

Finally, apply the rules to obtain a canonical sequence. Pay attention to symmetrical glyphs, glyphs that are equal to another reversed glyph, and glyphs that are actually rectangles.