2018 ACM-ICPC North America Qualification Contest Solution Outlines

The Judges

October 6, 2018

icpc International Collegiate Programming Contest

programming tools sponsor

2018 ICPC North America Qualifier Contest

event sponsor

B – Das Blinkenlights – First solved at 0:01

Description

Will two lights that blink at regular intervals (p and q seconds) blink at the same time in the next s seconds?

That is, is there an integer $1 \leq t \leq s$ such that there are two other integers m and n where $n \cdot q = m \cdot p = t$?

B – Das Blinkenlights – First solved at 0:01

Description

Will two lights that blink at regular intervals (p and q seconds) blink at the same time in the next s seconds?

That is, is there an integer $1 \leq t \leq s$ such that there are two other integers m and n where $n \cdot q = m \cdot p = t$?

Solution

Simple – because $1 \leq s \leq 10\,000$, we can solve this by brute force.

We can also be more clever and use the least common multiple.

B – Das Blinkenlights – First solved at 0:01

Solution Strategy

Brute force approach:

- Try all values of $1 \leq t \leq s$
- For each t, test whether $(t \bmod p) = (t \bmod q) = 0$.
- If any t works, the answer is yes.

Runs in $O(s)$.

Faster alternative: find the least common multiple (LCM) of p and q:

- If $\text{LCM}(p, q) \leq s$, then the answer is yes.
- $\text{LCM}(p, q) = p \cdot q / \text{GCD}(p, q)$
- GCD is the greatest common divisor, via Euclid's algorithm.

Runs in $O(1)$.

K – Run-Length Encoding, Run! – First solved at 0:03

Description

Encode: Count consecutive, repeated characters.

Decode: Duplicate characters based on the number of repetitions.

K – Run-Length Encoding, Run! – First solved at 0:03

Description

Encode: Count consecutive, repeated characters.
Decode: Duplicate characters based on the number of repetitions.

Solution

Encode:

- Increment a counter for each repeated character encountered.
- On a new character, print the character and count.

Decode:

- For each character-digit pair cd, output d copies of c.

G – Left and Right – First solved at 0:15

Description

Find the lexicographically earliest n-permutation $p[1..n]$ so that each adjacent pair satisfies the L/R (smaller/greater) conditions.

G – Left and Right – First solved at 0:15

Description

Find the lexicographically earliest n-permutation $p[1..n]$ so that each adjacent pair satisfies the L/R (smaller/greater) conditions.

Solution

Consider consecutive L's and R's as groups. Greedily look ahead and make decisions for the first two groups, trying to keep the permutation lexicographically earliest.

G – Left and Right – First solved at 0:15

Solution Strategy

- Group each sequence of consecutive L's and R's. For convenience we can prepend a single R to the beginning and assume the robot starts at a virtual leftmost house 0, so that we are always facing an R group before an L group.
- If the next R group has more than one R, move one step to the right.
- If the next R group has only a single R, look at the L group following it (If there is no more L group then just walk right). Suppose there are k L's in that group. Let the robot move to the next $(k+1)$-th position first (the single R). Then take k single steps to the left (the k L's).
- Repeat the above until all directions are processed.

The solution takes linear time.

L – Superdoku – First solved at 0:20

Description

Given a partially filled matrix, fill out the full matrix so that each number from 1 to n appears in each row and column exactly once.

L – Superdoku – First solved at 0:20

Description

Given a partially filled matrix, fill out the full matrix so that each number from 1 to n appears in each row and column exactly once.

Solution Sketch

One observation is that rows are quite independent. We should be able to fill in each row independently.

L – Superdoku – First solved at 0:20

Solution Strategy

The only way that will result in a NO is that the given partially filled matrix is invalid. Otherwise there is always a solution. This can be proved using Hall's marriage theorem, and the solution can then be found by a bipartite graph matching for each row.

L – Superdoku – First solved at 0:20

Proof

Suppose that the previous k rows are valid, and now we are going to fill the $(k+1)$st row. We use a set $C = \{c_1, \ldots, c_n\}$ to represent n columns, and $f(c_i)$ to represent the set of valid numbers that can be filled in the ith column.

We know that $|f(c_i)| = n - k$ for all i. Therefore, if we allow the same number to be counted multiple times, then $|\bigcup_{c_i \in C'} f(c_i)| = (n-k)|C'|$ for any subset C' of C.

In order to show that the Hall's condition holds, we need to prove that the number of distinct elements in $\bigcup_{c_i \in C'} f(c_i)$ is greater than or equal to $|C'|$ for all C'.

We know that each element will appear in $\bigcup_{c_i \in C'} f(c_i)$ for at most $n-k$ times, because the previous k rows are valid.

Thus, for $(n-k)|C'|$ given elements, there are at least $(n-k)|C'|/(n-k) = |C'|$ of them are distinct. Thus Hall's condition holds for all C'. Therefore, a perfect matching always exists.

L – Superdoku – First solved at 0:20

Solution

First we need to check if the given partially filled matrix is valid. If so, the answer is always YES.

After that, go through each row in sequential order, and run a bipartite graph matching on each row.

- The matching is between the n numbers and n columns, allowing number i to match column j only if it has not been used previously in that column.
- Find any matching for that row, and update the matrix.
- Remove each number from the matched column's set of possible numbers.
- Do this until the whole matrix is filled.

The total time complexity is $O(n^4)$.

A – Bingo Ties – First solved at 0:27

Description

Given a set of n bingo cards, is there any sequence of numbers called that will result in a tie (in a row) between two cards?

A – Bingo Ties – First solved at 0:27

Description

Given a set of n bingo cards, is there any sequence of numbers called that will result in a tie (in a row) between two cards?

Solution

- For each possible bingo between two cards, consider shortest sequence s of numbers that results in that bingo.
- Check if any proper prefix of s results in a bingo on another card.
- If not, report the tie.

A – Bingo Ties – First solved at 0:27

Solution Strategy

1. Find set S of all numbers that appear on more than one card.
2. For each $s \in S$:
   1. Find all pairs of cards c_1, c_2 that both contain s.
   2. For each pair, construct list L of numbers that appear in the same row as s in either c_1 or c_2.
   3. For all c cards other than c_1, c_2, determine if $L \setminus \{s\}$ results in a bingo on card c.
   4. If no card results in a bingo, then c_1, c_2 is a tie.
3. Of all pairs of cards that generate a tie, report the pair that is the smallest lexicographically.
4. If no pair is a tie, report this.

Runtime: $O(|S|n^3)$ where n is the number of cards.

E – Fruit Slicer – First solved at 0:29

Description

Determine how many of the $n \leq 100$ unit circles can a single infinite line intersect or touch.

E – Fruit Slicer – First solved at 0:29

Description

Determine how many of the $n \leq 100$ unit circles can a single infinite line intersect or touch.

Solution

There always exists an optimal line that is externally tangent to two circles. Enumerate $O(n^2)$ candidate lines, and for each of them linearly check how many circles it intersects or touches.

E – Fruit Slicer – First solved at 0:29

Solution Strategy

- Enumerate an *ordered* pair of circles (i, j) and identify their external tangent line on the right side of the ray from the center of circle i to the center of circle j.
- For each external tangent line, linearly enumerate every circle, and check if the line intersects or touches this circle.

The solution takes $O(n^3)$ time.

E – Fruit Slicer – First solved at 0:29

Work in Integer Space

- We can multiply every coordinate by 100 and handle everything with integer.
- The actual tangent line may have irrational coefficients. To have the solution work in integer space, we can use the line $l : ax + by + c = 0$ that passes the centers of two circles to check intersection.
- A circle intersects or touches the external tangent if the distance from its center (x_c, y_c) to l is between $[0, 2]$. We can thus check if $|ax_c + by_c + c|/\sqrt{a^2 + b^2} \leq 2$, which is equivalent to $(ax_c + by_c + c)^2 \leq 4(a^2 + b^2)$.
- Be careful with integer overflow when numbers are multiplied by 100 and then squared.

D – Froggie – First solved at 0:33

Description

Given a road description, starting position, and sequence of hops (Up, Down, Left, Right), determine if Froggie safely crosses the road or is squished by a moving car.

D – Froggie – First solved at 0:33

Description

Given a road description, starting position, and sequence of hops (Up, Down, Left, Right), determine if Froggie safely crosses the road or is squished by a moving car.

Solution

Simulate Froggie's hops and the moving cars to determine whether Froggie enters the path of a moving car.

D – Froggie – First solved at 0:33

Solution Strategy

- During one time step, a car at column c moving right at speed s makes columns $[c+1, c+s]$ *unsafe* for Froggie.
  - For cars moving left, similar logic applies.
- Froggie starts at column P, and makes hops: $h_1, h_2, \ldots, h_M$.
- At time i, Froggie hops in the grid according to hop h_i.
- Froggie is safe in column c of lane ℓ if no car in the lane is unsafe for her. That is, no car passes through that location during that time step.
- Froggie is "safe" at completion if she exits the top of the table.

D – Froggie – First solved at 0:33

Solution Strategy

- During one time step, a car at column c moving right at speed s makes columns $[c+1, c+s]$ *unsafe* for Froggie.
  - For cars moving left, similar logic applies.
- Froggie starts at column P, and makes hops: $h_1, h_2, \ldots, h_M$.
- At time i, Froggie hops in the grid according to hop h_i.
- Froggie is safe in column c of lane ℓ if no car in the lane is unsafe for her. That is, no car passes through that location during that time step.
- Froggie is "safe" at completion if she exits the top of the table.

Implementation

The simulation can be conducted by either:

- maintaining a 2D array of symbols (positions of all cars), or
- determining unsafe columns just for Froggie's current lane.

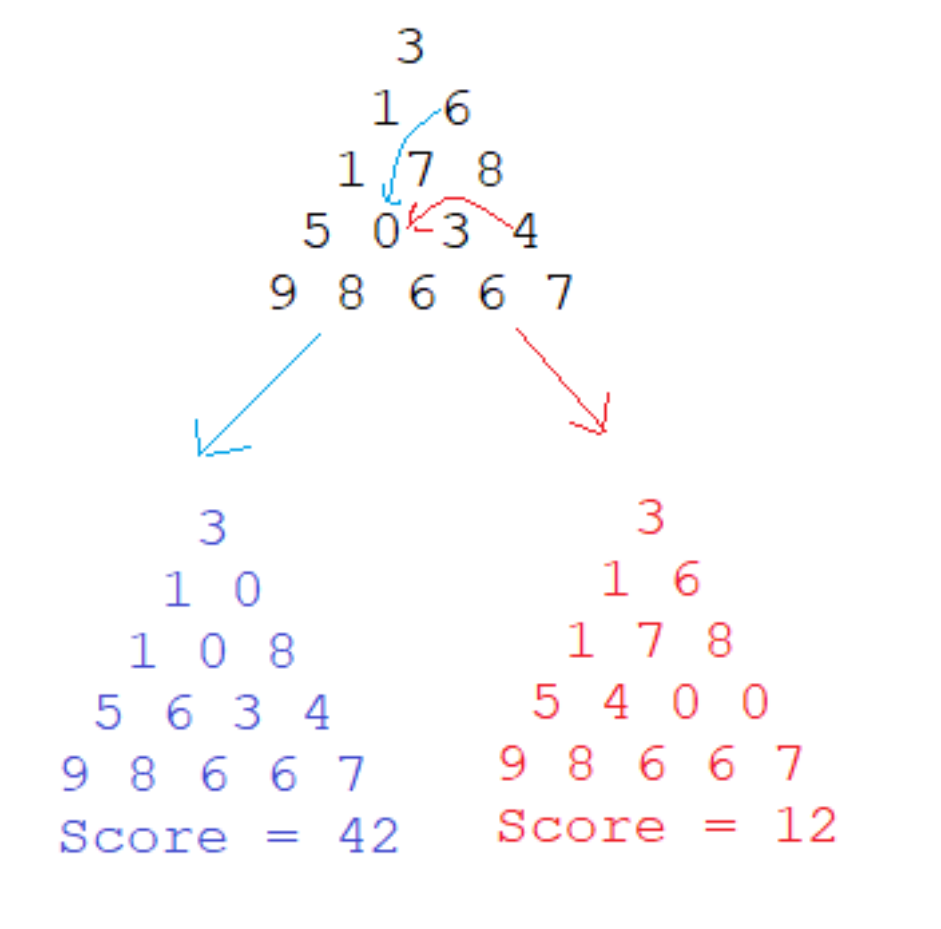
J – Peg Game for Two – First solved at 0:42

Description

- Two player game on a 5-row triangular game board.
- Game starts with one open hole and 14 pegs assigned numeric values.
- Players take turns making valid jumps on straight lines.
- In order to jump, peg A must be adjacent to peg B, and the location in the same direction from peg A to peg B adjacent to peg B must be a hole.
- After the jump, peg B is removed, and peg A goes to the location that was previously a hole.
- The player making the jump adds the product of peg A and peg B to his/her score.
- Goal: To maximize the difference of score between you and your opponent.

J – Peg Game for Two – First solved at 0:42

Example of Move Choices



J – Peg Game for Two – First solved at 0:42

Solution

- Key realization is that the actual number of game paths is small enough to allow a brute force solution.
- The other helpful idea in simplifying the solution is that instead of a min-max tree, since we are trying to maximize difference in score, at each level of the game tree, we can perform the same maximization function, instead of keeping track of what level we are on and alternately minimizing or maximizing.

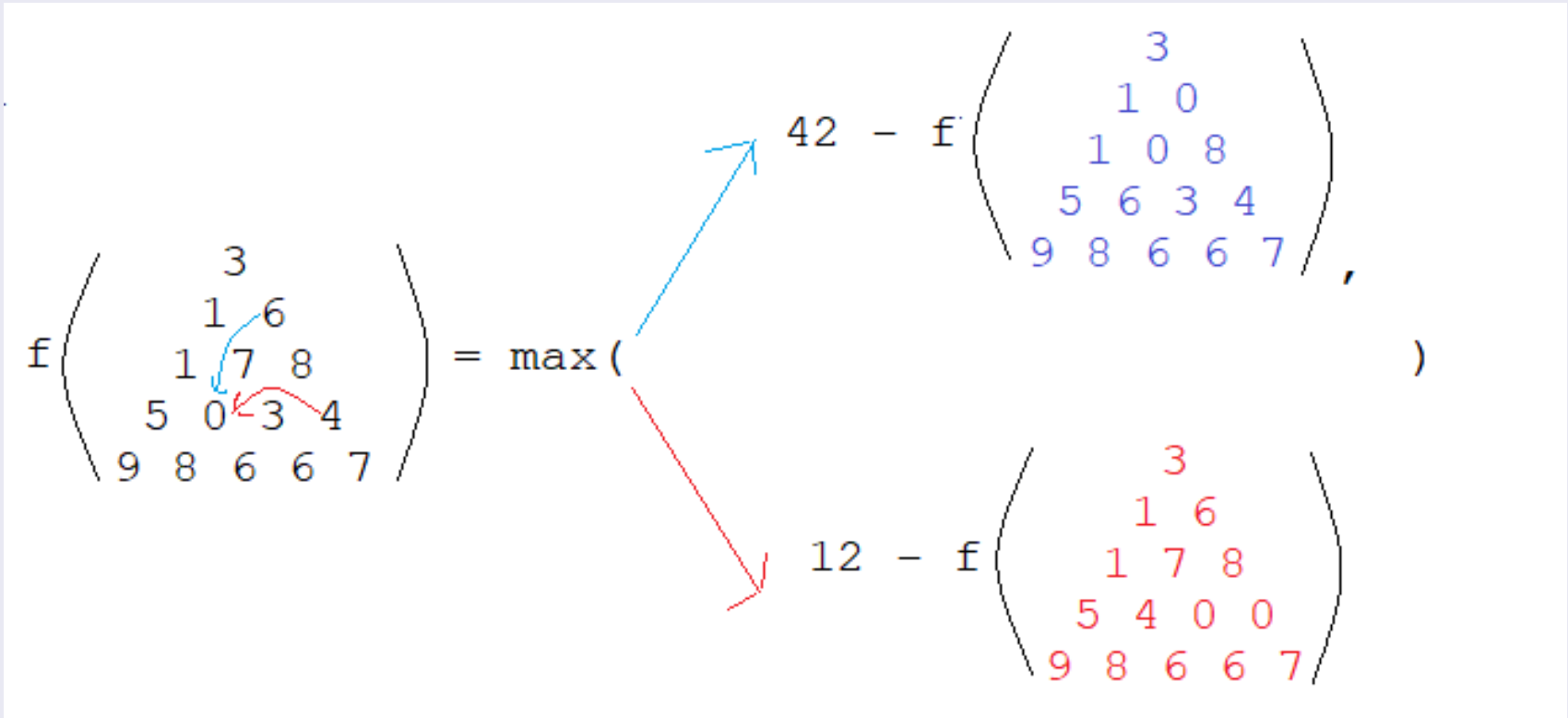
J – Peg Game for Two – First solved at 0:42

Solution Strategy

- Brute Force Idea: Try each possible move, and take the best.
- Let $f(g)$ be your score on board g. For each possible jump, j_i, let the product of the pegs involved in the jump be p_i and the ensuing board position be g_i.
- Then, for move i, our potential difference in score is $p_i - f(g_i)$.
- You score p_i points; then your opponent also tries to maximize their score. Keep track of their best difference during recursion. From your viewpoint, this value should be *subtracted* from your score, not added.
- Calculate each of these options and over all i, maximize $p_i - f(g_i)$.
- For a single starting board position, there are relatively few games that can be played.
- Memoization helps, but is not necessary with the small board size.

J – Peg Game for Two – First solved at 0:42

Maximizing Over Multiple Options



I – Monitoring Ski Paths – First solved at 0:55

Description

Given: Paths in a forest of rooted trees, each path having the property that one endpoint is a descendant of the other.

Goal: Find the fewest nodes X to cover all paths.
i.e. $P \cap X \neq \emptyset$ for each path P.

I – Monitoring Ski Paths – First solved at 0:55

Description

Given: Paths in a forest of rooted trees, each path having the property that one endpoint is a descendant of the other.

Goal: Find the fewest nodes X to cover all paths.
i.e. $P \cap X \neq \emptyset$ for each path P.

Solution

Greedy

Repeat until all paths are covered:

- Find an uncovered path P where no other path ends below $top(P)$.
- Add $top(P)$ to X.

Must also implement this efficiently.

I – Monitoring Ski Paths – First solved at 0:55

Solution Strategy

Claim: If v is the top of a path P and no other path ends below v, then v lies in some optimal solution.

Proof: Some node x of P must be covered in the optimum. But v covers all the paths that x covered too! So we might as well use v instead of x.

I – Monitoring Ski Paths – First solved at 0:55

Algorithm: Consider the nodes of a tree T in post-order traversal order (*eg.* using a DFS). If some uncovered path P has $top(P) = v$, then use v.

To quickly check if such a path is not covered, every time we pick some node w we **mark** all nodes below w. Then a path P is not covered at v if and only if $bottom(P)$ is not marked.

When we choose v, mark all nodes below it using a DFS, making sure to not recurse on already-marked nodes.

Can be implemented to run in linear time.

C – Ebony and Ivory – First solved at 1:06

Description

Given a right-hand music piece that needs to be played with 5 fingers, find the cost of an ergonomically optimal assignment of keys to fingers. Ergonomics is determined by how difficult it is to play a particular combination of keys with a particular combination of fingers. The cost of playing a combination depends on the number of half steps. Furthermore, a distinction is made between black and white keys, thus requiring 4 tables with the costs for each combination of black and white to be given.

C – Ebony and Ivory – First solved at 1:06

Description

Given a right-hand music piece that needs to be played with 5 fingers, find the cost of an ergonomically optimal assignment of keys to fingers. Ergonomics is determined by how difficult it is to play a particular combination of keys with a particular combination of fingers. The cost of playing a combination depends on the number of half steps. Furthermore, a distinction is made between black and white keys, thus requiring 4 tables with the costs for each combination of black and white to be given.

Solution

Standard dynamic programming. Dynamic programming state is $D_{i,j}$ - minimum cost of playing sequence of keys $a_0 \ldots a_i$ such that a_i is played with finger j.

C – Ebony and Ivory – First solved at 1:06

Solution Strategy

DP recurrence is:

$$D_{i,j} = \begin{cases} \min_{k \in [1,5]} D_{i-1,k} + c_{k,j}(a_j - a_{j-1}) & \text{if} \quad i > 0 \wedge a_j \neq a_{j-1} \\ D_{i-1,j} & \text{if} \quad i > 0 \wedge a_j = a_{j-1} \\ 0 & \text{if} \quad i = 0 \end{cases}$$

where $c_{k,j}(s)$ is cost read from appropriate table for s half steps, skipping pairs (k, j) for which no entry exists in the respective table. The table chosen depends on whether (a_{j-1}, a_j) is a white/white, black/white, white/black, or black/black key pair. If $s < 0$, use $c_{j,k}(-s)$ instead. Final answer is $\min_j D_{L-1,j}$ and complexity is $O(L)$ where L is the number of keys. For more details, see Hart, Bosch, and Tsai [1].

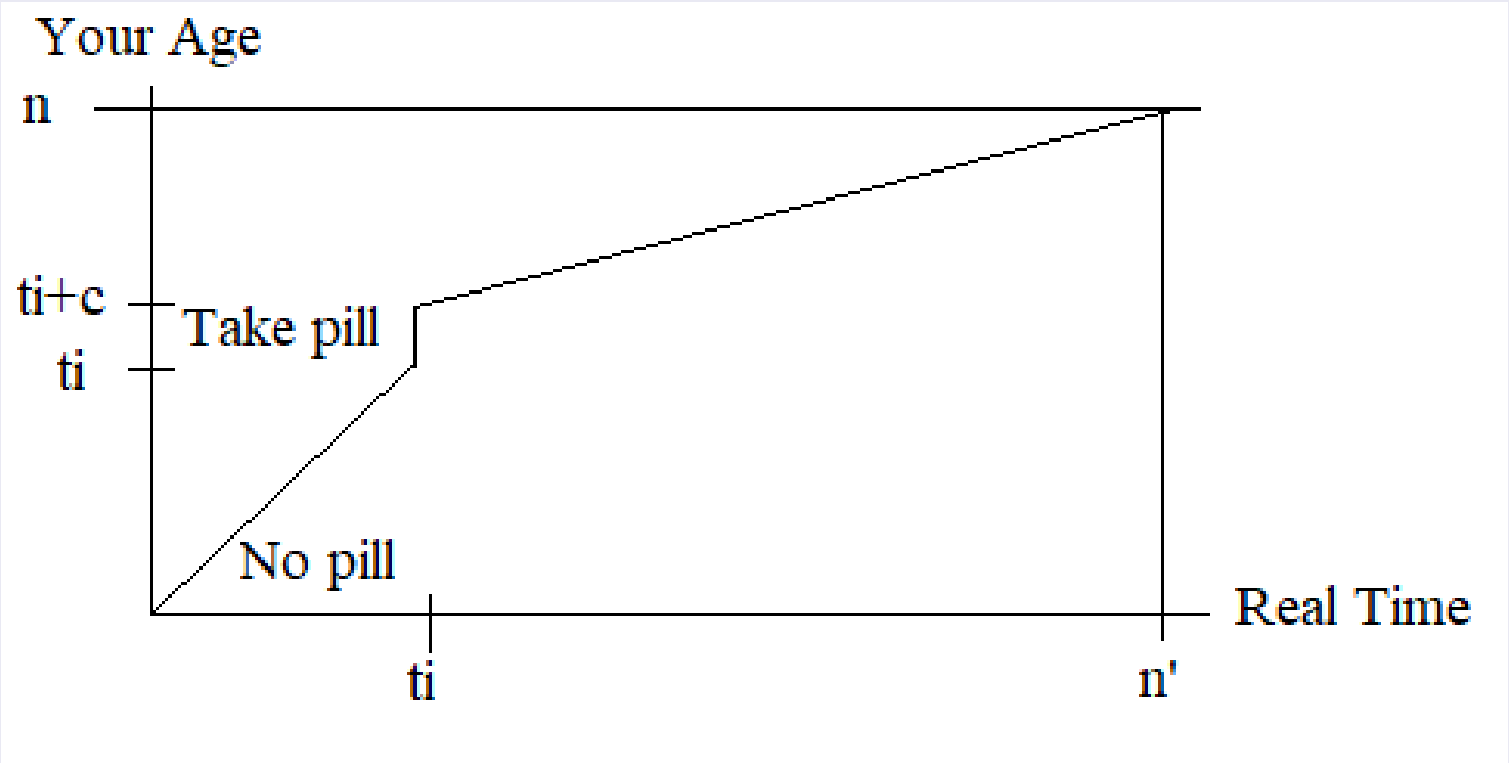
H – Longest Life – First solved at 1:53

Description

- Normally you will live to be n seconds old.
- But there are p pills that come to market. Pill p_i comes to market at time t_i and allows you to age y_i seconds over the real span of x_i.
- You can be on one pill at a time and the only drawback to switching to a pill from no pill or another pill is that you automatically age c seconds when you switch.
- Goal: Maximize how long you live.

H – Longest Life – First solved at 1:53

Visualization of Taking a Pill



H – Longest Life – First solved at 1:53

Initial Solution Ideas

- Complete Brute Force – Try all combinations of pills.
  - With $p \leq 100\,000$ this creates an absurd $2^{100\,000}$ number of combinations to check. So this isn't viable.
- Straight Forward Dynamic Programming: For each pill i, we want to calculate the longest we'd live if it were the last pill we changed to.
  - To do this, we know that we'd have to switch from another pill j to it, where $j < i$.
  - We could simply try each possible $j < i$, and determine which gave us the maximal answer for taking pill i.
  - Issue: with the double loop structure, the run time is $O(p^2)$.
  - For our upper bound on p, this is still too slow to run in time.

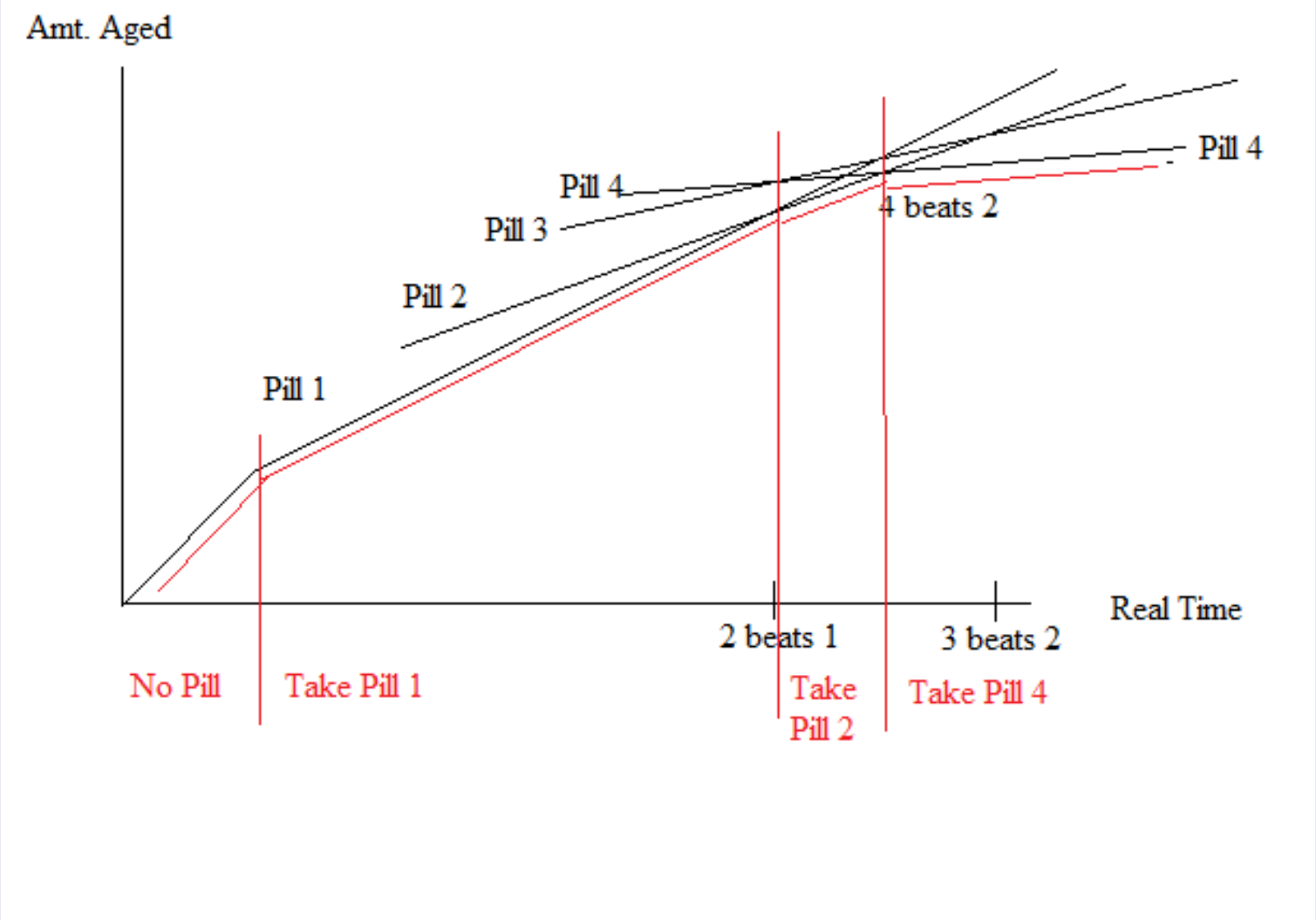
H – Longest Life – First solved at 1:53

Solution Speed Up Ideas

- The effect of each pill can be visualized as a line.
- We only care to build off of the "best" lines.
- Key realization: If we are only comparing two lines, eventually the one with the better (in this case lower) slope will overtake the other line at some cross over point. From that point on, that pill will be better.
- Thus, there's a fixed window of contiguous time where any individual pill will be optimal, and each of these windows form a list of time periods where different pills with decreasing slopes will take over.

H – Longest Life – First solved at 1:53

Visualization of Solution Sketch



H – Longest Life – First solved at 1:53

Solution Strategy

- Maintain a deque which stores which pills are optimal for which ranges of time.
- For each pill b (in increasing time order):
  - If b's slope isn't lower than all the previous ones, skip it.
  - Otherwise, remove from the front all pills whose optimal period has passed.
  - Then, calculate when b overtakes the pill e at the end of the deque.
    - If b overtakes e before e is optimal, remove e from the deque.
    - Continue this process until we get to a pill in the deque for which this isn't the case.
  - Let s be the time when b overtakes e.
  - Adjust the range of e to end at s.
  - Add b to the back of the deque, with range $[s, \infty)$.

H – Longest Life – First solved at 1:53

Run Time Analysis

- Outer loop runs through p pills.
- Two inner loops potentially remove 0 or more items from the front or back of the deque at each pass.
- Each pill gets pushed onto the deque at most once.
- Since there are at most p pushes, there can be at most p pops.
- Thus, the total runtime is $O(p)$.

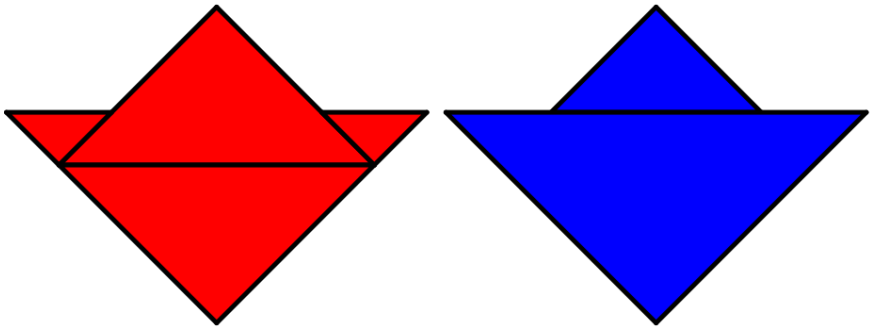
M – Triangular Clouds – First solved at 3:49

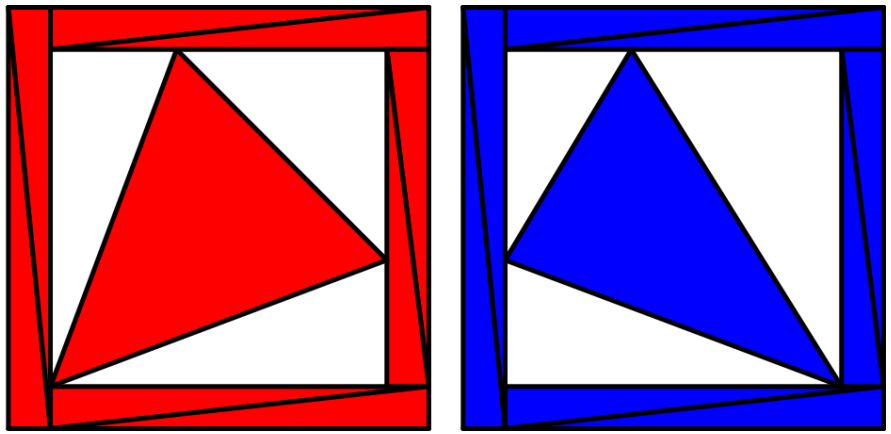
Description

Is the union of a set of triangles the same as the union of another set of triangles? Complications include the large magnitude of coordinates (10^9) and large number of triangles (10^5).

Tricky Cases

Adjacent triangles may not share vertices.
Region could have holes.





M – Triangular Clouds – First solved at 3:49

Description

Is the union of a set of triangles the same as the union of another set of triangles? Complications include the large magnitude of coordinates (10^9) and large number of triangles (10^5).

Solution

Solution 1: Trace triangles of set A in counterclockwise direction, and triangles of set B in clockwise direction. Ensure all line segments cancel.
Solution 2: Hash triangles by taking the double integral of hash(x,y) across a triangle. Sum the hashes to get the hash of the entire set, and compare the resulting hashes.

M – Triangular Clouds – First solved at 3:49

Solution 1 Strategy:

As a thought exercise, consider the problem in one instead of two dimensions.

- For example, $[1, 3) \cup [3, 5) = [1, 5)$.
- We can make 'start' and 'end' events, such as (1, start), (3, end), (3, start), and (5, end). Then, the (3, start) and (3, end) events will cancel.
- To check that both sets are equal, add $[1, 5)$ with negative orientation:

$$[1, 3) \cup [3, 5) \setminus [1, 5) = \emptyset.$$

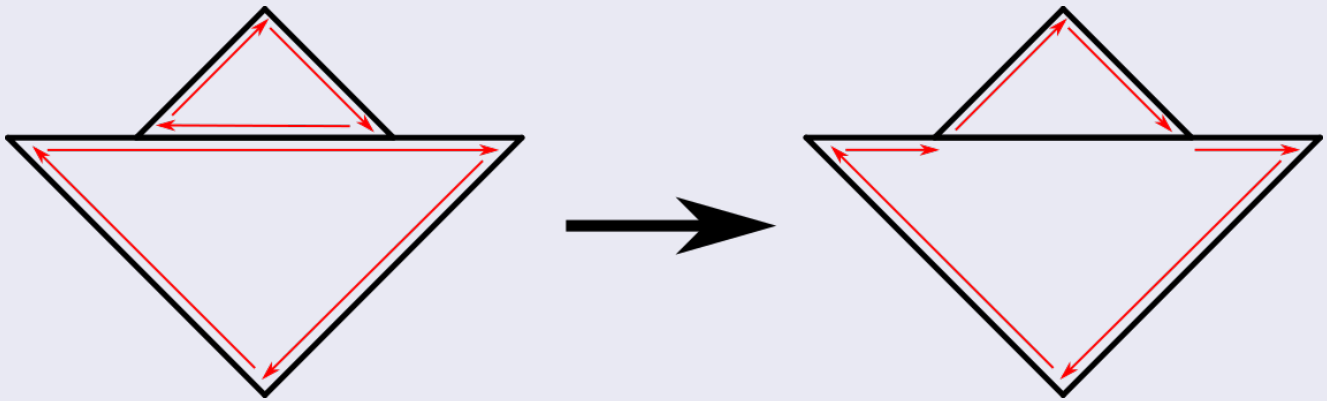
- (1, start), (3, end), (3, start), (5, end), (5, start), (1, end) completely cancel.

M – Triangular Clouds – First solved at 3:49

Solution 1 Strategy:

Now consider 2D.

- By tracing each triangle clockwise, we can assign each line segment a direction. Thus, when two triangles both share an edge, the line segment that is shared would cancel.
- After canceling all the shared line segments, left with canonical oriented boundary edges:



M – Triangular Clouds – First solved at 3:49

Solution 1 Implementation:

Efficiency is very important.

- Place line segments into buckets based on line parameters ($ax + by = c$), $O(N)$ time total.
- For each bucket, create start and end event for each segment, and sort the events. Do linear sweep to cancel events. Ensure that the remaining line segments after canceling are equal for Jerry and Garry. $O(N \log N)$ total.
- Most elegant solution: Overlay both sets of triangles, with Garry clockwise and Jerry counterclockwise. Ensure there are no segments left after canceling.

M – Triangular Clouds – First solved at 3:49

Solution 2 Strategy:

Goal: Hash the sets of triangles in such a way that if $A \cup B = C$ then $Hash(A) + Hash(B) = Hash(C)$.

- In order to make this true, we make our hash function the double-integral across the entire area, since addition is associative/commutative.
- Example: Let $hash(X, Y) = 1$. The double integral across the triangle is the area of the triangle. Thus, if $A \cup B = C$, then $hash(A) + hash(B) = hash(C)$ since A and B are disjoint.
- Issue: Unfortunately the converse, $Hash(A) + Hash(B) = Hash(C)$ does not imply $A \cup B = C$, which is what we need.
- So an intellegent hash function needs to be used, to get as close to this statement as possible.

M – Triangular Clouds – First solved at 3:49

Solution 2 Picking the hash:

If the hash is too simple, it will have collisions.

- $hash(X, Y) = 1$ is vulnerable to sliding a triangle over by 1.
- $hash(X, Y) = X$ is vulnerable because sliding the triangle up 1 breaks it, and all triangles symmetric over the X axis hash to 0.
- If the hash is too complex, it is too hard to double-integrate during the contest.
- To make a simpler hash work, apply linear transformations / translations to the judge data to reduce the chance of collision.
- $Hash(X^n + X^{(n-1)} + \ldots + 1)$ is used by the judge solution, but it requires random rotation to avoid vertical translation.

M – Triangular Clouds – First solved at 3:49

Solution 2 Complexity/Correctness:

- Complexity is $O(N)$, because we hash each triangle once and add the hashes.
- Hash collisions based on sheer number of comparisons is very unlikely; even a 32-bit hash should be sufficient if well distributed.

M – Triangular Clouds – First solved at 3:49

General Pitfalls

- Avoid monte-carlo, doubles, and sampling techniques, these will be hard to work with 10^{18} possible points.
- $O(N^2)$ solutions will run out of time, so geometry solutions involving merging triangles are quite painful and unlikely to pass.
- Consider edge cases where many triangles meet at the same point, separate components, and more.
- Don't let this problem take up several hours during the contest!

F – LCM Tree – Not solved during the contest!

Description

Given $n \leq 25$ nodes with positive integer numbers, count the arrangements of these nodes in a binary tree, so that the LCM of the numbers on every pair of children equals their parent's number.

F – LCM Tree – Not solved during the contest!

Description

Given $n \leq 25$ nodes with positive integer numbers, count the arrangements of these nodes in a binary tree, so that the LCM of the numbers on every pair of children equals their parent's number.

Solution

Use dynamic programming to keep track of which set of nodes are yet to be added to the tree. Choose one pair of nodes at a time to append to one of the tree leaves.

F – LCM Tree – Not solved during the contest!

Solution Strategy

- Create a bitmask to represent the nodes. A node's bit is on means the node has been added to the tree, but we have NOT yet decided its two children. If we have appended two children to a node, then that node's bit is turned off. Initially, only the largest node has its bit on.
- In each step of the dynamic programming, we find the a node z that has its bit on. We enumerate two children x and y that satisfy $LCM(x, y) = z$ and append them to z. After that, z will have its bit off, and x and y will have their bits on. Alternatively, we can decide not to give z any children.
- Repeat the step above until all nodes have their bits off.

F – LCM Tree – Not solved during the contest!

Optimizations

A naive implementation of the above strategy will timeout, as $O(2^n)$ DP states plus enumerating two children for every state takes $O(n^2 2^n)$ time. Several optimizations can be applied to significantly speed up the DP:

- The largest node among a subtree must be the root of the subtree. We can order the nodes decreasingly, and always choose the largest node that has its bit on to be z.
- z can be taken out from the bitmask after we decide its children. The size of the bitmask is reduced by one after each DP transition. The length of the bitmask can be kept in the bitmask using a highest bit.
- If the largest node remaining in the bitmask has its bit off, we stop immediately because there is no chance to add this node to the tree.
- Group all nodes by their numbers. When there are multiple nodes of a same number, always take the first node, but multiply the DP count by the number of nodes in the group.

F – LCM Tree – Not solved during the contest!

Running Time Analysis

- Initially the bitmask has size 25, and one of the bits is on.
- After each transition, one bit is turned off, and zero or two bits are turned on. The size of the bitmask is reduced by one. The number of bits that are on can either increase by one, or decrease by one.
- If there have been i nodes removed from the bitmask, then the bitmask has size $25 - i$, and at most $i + 1$ bits among them are on. Therefore the total number of states that can be visited is $\sum_{i=0}^{25} \sum_{j=0}^{i+1} C(25 - i, j)$. Additionally, we are enumerating two children for each state, which has a cost of $(25 - i - j)^2$.
- The total cost is $\sum_{i=0}^{25} \sum_{j=0}^{i+1} C(25 - i, j)(25 - i - j)^2 \approx 6.888 \times 10^7$. The algorithm actually runs faster because many states are invalid by not satisfying the LCM condition.

References

[1] Melanie Hart, Robert Bosch, and Elbert Tsai.
Finding optimal piano fingerings.
The UMAP Journal, 21(2):167–177, 2000.