2025 South Conference Division 1 Solutions

The Judges

November 8, 2025

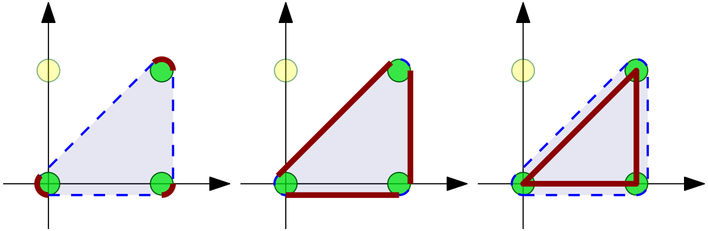
Orchard Fence

Problem

- Given *n* non-overlapping circles of diameter *d*, compute the expected perimeter of the convex hull of a random subset of *k* circles.

Observation

- The curved parts of the perimeter of the convex hull add up to one full circle.
- The straight part of the perimeter of the convex hull is equal to the length of the convex hull of the centers of the circles.



Orchard Fence

Solution

- Consider a fixed pair of points u and v. What is the probability uv is an edge of the convex hull?
- Consider the line ℓ_{uv} that goes through these two points, and partition the other points into one of these four sets:
  - S_{left} for the set of points to the left of ℓ_{uv}.
  - S_{right} for the set of points to the right of ℓ_{uv}.
  - S_{on} for the set of points on ℓ_{uv} between u and v.
  - S_{off} for the set of points on ℓ_{uv} not between u and v.
- Out of the $\binom{n}{k}$ ways to pick k points, the number of ways to pick k points such that the edge uv is an edge of the convex hull is:

$$\binom{|S_{\text{left}}| + |S_{\text{off}}|}{k-2} + \binom{|S_{\text{right}}| + |S_{\text{off}}|}{k-2}$$

- We can compute in $O(n^3)$ time.

Orchard Fence

Faster Solution

- Fix a point u, and sort all points in angular order.
- Consider rotating a line around u starting from a horizontal line. We can compute the points to the left/right/on/off of this horizontal line.
- Rotate this line to compute this for every other point v.
- Total time: $O(n^2 \log n)$

Implementation Details

- Multiply all coordinates by 100 to get integers.
- Careful when computing probabilities, there are $n - k - 1$ different values to compute, and can cut off computation of probabilities that are sufficiently small.
- Alternate way to avoid dealing with collinear points: perturb each coordinate by a small ε to avoid handling collinear points.

Blind Bottles

Problem

- There is a list of n unique bottles arranged in an unknown order. You make a guess of their order and receive the number of bottles that are correctly placed. Your task is to win this *Blind Bottles* game in 10^4 guesses.

Observation

- The limit of 10^4 guesses is large enough to make $O(n^2)$ guesses. If we can determine one bottle's correct position using $O(n)$ guesses, then we can solve the full problem by repeating this process n times.
- We can establish a baseline first by guessing an arbitrary ordering. Suppose the baseline has c correct positions. Then we can try determine which bottle is at the first position by swapping it with each of the other bottles, and checking if we get c, $c \pm 1$ or $c \pm 2$. Let's call this a *round*.

Blind Bottles

Solution

- We will name the positions $p_1, \ldots, p_n$, and the bottles $b_1, \ldots, b_n$. If all bottles are placed correctly, then b_i is at p_i for all i.
- In the current round, we are trying to find b_1 and move it to p_1. We swap the bottle currently at p_1 with every other bottle and record their number of correctly placed bottles.
- We have these cases :
    - b_1 is already at p_1: we will only get $c-1$ or $c-2$.
    - b_1 is not at p_1. Instead, b_1 is at p_k:
        - b_k happens to be at p_1: we will get $c+2$.
        - Some other bottle b_t is at p_1: we must get exactly two $c+1$ at two positions p_x and p_y, because one swap will get b_t into p_t, and another will get b_1 into p_1. Now what we do not know is which $c+1$ swap is the one that corrects b_1 (i.e. is $p_k = p_x$ or $p_k = p_y$?). We can resolve this with one extra guess – after exchanging the bottles at (p_1, p_x) and then at (p_x, p_y).

Blind Bottles

Solution (cont'd)

- After the first round, b_1 will be at p_1. We then proceed to the next round to place b_2 at p_2.
- Repeat the rounds until all bottles are placed correctly.
- It can be seen that this process uses one initial baseline guess, plus $(n-1)+(n-2)+\cdots+1 = n(n-1)/2$ guesses in the rounds. The extra guesses for the two $c+1$ case can at most double the guesses to $n(n-1)$. The total number of guesses is thus at most $1+n(n-1) < n^2$.

One Way Only

Problem

- You are given an r by c grid and a path starting at the top left, ending on the bottom right, and only consisting of downwards and rightwards steps.
- You wish to determine the minimum number of tiles you must mark such that your given path is the only path consisting of downwards and rightwards steps that goes through zero marked tiles.

Observation

- This path splits the grid into two regions (above and below), with each region being the transpose of the other. So it suffices to write a solution to block out all the paths that go above our given path, and repeat the same steps for paths that go below.

One Way Only

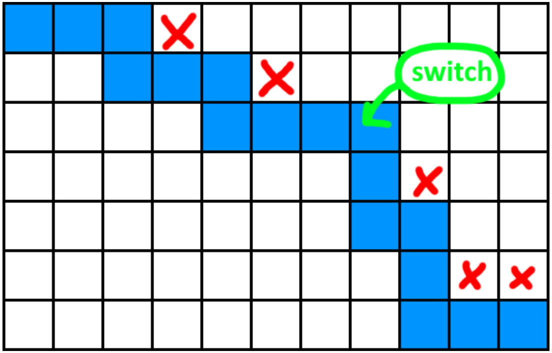
Observation

- We will focus on blocking off all paths that go above the given path.
- First, observe that it's never optimal to X out a tile that is not directly above or to the right of a tile on the input path.
- Furthermore, if and only if there exists two tiles *a* and *b* on our path such that *a* comes before *b* on the path, *a* has an empty tile to the right, and *b* has an empty tile above it, then there exists another path on our grid which does not go through a marked tile.
- So, our goal is to minimize the number of marked tiles while preserving the above condition.

One Way Only

Observation

- To do this, observe that we can iterate through the path, marking all the tiles immediately to the right of the path. At exactly one point, we can switch to marking out all the tiles immediately above the path instead.



One Way Only

Solution

- But where does this switch happen? To calculate this, store two arrays A and B.
- For A, for each tile on the input path, store the number of tiles that would get marked out if we were to mark out the tile immediately to the right of every tile that comes before this tile.
- For B, for each tile on the input path, store the number of tiles that would get marked out if we were to mark out the tile above every tile that comes after this tile.
- Then, our solution is the minimum of $A[i] + B[i]$, where i is a tile.

GUID Generator

Problem

- You are given a tree of n nodes in which each node has a hexadecimal character. Count the number of unique strings you can get by walking along any simple path on the tree and concatenating all the hexadecimal characters along the path.

Observation

- There are $O(n^2)$ different paths, and each path produces a string of length n. The total length of all these strings is $O(n^3)$. It would be too slow and take too much space to generate all these strings.
- Consider all the paths that start from the same node s, walking along those paths naturally corresponds to inserting those strings, starting from s, into a Trie (prefix tree).

GUID Generator

Observation (cont'd)

- The Trie also helps us deduplicate the strings. The size of the Trie gives us the count of unique strings.
- For a different starting node s', we can reuse the Trie that was built by starting from s earlier.

Solution

- For each node s, traverse the tree starting from s, while inserting every hexadecimal character encountered into a Trie. Each node in the Trie can have at most 16 children. The size of the Trie minus 1 is the final answer.
- Time and Space: $O(n^2)$.

GUID Generator

Alternative Solution

- We can also compute a hash value for the string we have while walking along the tree paths.
- A standard polynomial hash will work: $Hash(Sc) = (Hash(S) \cdot a + Hash(c)) \bmod p$.
  - S is a string, c is a character, and Sc is the result of concatenating S and c.
  - a and p are two constant integers that are coprime, with p being a prime.
- Counting the unique hash values at the end gives us the final answer.

Photo Encoding

Problem

- Given a list of integers, output the minimum n such that every integer x_i can be matched to a tile in an n by n image of which the Manhattan distance to the top-left tile is equal to x_i.

Observation

- For an n by n image, we notice that, for some distance d, there are exactly $(n - 1 - |n - 1 - d|)$ tiles in said image that have a distance of d to the top-left tile.

Solution

- We count the number of occurrences of each Manhattan distance, and use that to bound our n. If we have x occurrences of some distance d, then $(n - 1 - |n - 1 - d|) \geq x$.

Photo Encoding

Solution

- To solve this problem, first count the number of occurrences of each distance, and use the above equation to bound n. Once all inequalities are satisfied, we have our solution!
- The bounds are small enough to simply check every possible value of n against every inequality (we can show that n is always less than 2000). Alternatively, you can manipulate the inequality above to solve for n to solve the inequality instantly.

Meeting Free Fridays

Problem

- Given n intervals in the range $[0, t)$ and an integer k, find the size of the largest subset of non-intersecting intervals such that the length that is not covered by the intervals is at least k.

Meeting Free Fridays

Solution

- Sort all the left and right endpoints together. Let p_i denote the position of the ith endpoint.
- Let $\text{dp}[i][x]$ be the maximum uncovered length we can achieve if we take exactly x intervals with no interval extending past p_i.
- One transition is $\text{dp}[i][x] = \max(\text{dp}[i][x], \text{dp}[i-1][x] + p_i - p_{i-1})$.
- For each interval with a right endpoint at index i and left endpoint at index j, we can also transition $\text{dp}[i][x] = \max(\text{dp}[i][x], \text{dp}[j][x-1])$.
- This gives us $O(n^2)$ states and $O(n^2)$ transitions in total.
- The answer is the maximum x such that $\text{dp}[i][x] \geq k$.

Galaxy Governance

Problem

- The problem essentially boils down to finding a $k+1$ coloring of an undirected graph which was constructed from a DAG with the max indegree across all nodes being at most k which then had all of its edge directions erased.

Observation

- First think about how you would $k+1$ color a DAG with the properties described above. Here coloring refers to when the DAG had its edge directions ignored.
- Then think about how you would turn the undirected graph back into the directed graph.

Galaxy Governance

Solution

- To turn the undirected graph back into the directed graph, you can repeatedly pick a node with degree k or less and orient all of its edges to point into that node.
- This works because when we select some node with degree k or less, either
  - we have selected a node that corresponds to a node with 0 outdegree from the original graph. In this case our edge orientation exactly matches the original graph;
  - or we have selected some other node that had outgoing edges in the original graph. Such a node will no longer be contributing to the indegrees of its neighbors. Therefore, this does not negatively affect our construction.

Galaxy Governance

Solution (cont'd)

- Now that we have a DAG, we can simply repeatedly select a node with zero indegree, color it, disallow its color on all its neighbors, and erase it and all its edges. In other words, we can greedily color the nodes according to the DAG's topological order.

Troubleshooting

Here is a test case to help troubleshoot your solution:

```
6 7 2
1 3
2 3
3 4
4 5
5 6
```

Rows of Stars

Problem

- You are given an array of n integers, an integer k, and two integers p and q (d and s in the original statement).
- From this array, you generate a grid where row i (0-indexed) is the original array rotated counterclockwise by $k \cdot i$ elements.
- Your task is to compute the largest sum over all $p \times q$ subrectangles in this new generated grid.

Rows of Stars

Observation

- There will be at most n distinct $p \times q$ subrectangles in the generated grid.
- To show this, note that the elements of the array always appear in the same order in each row (modulo wraparound).
- Thus, if we know that an element a_i is the top-left element of a $p \times q$ subrectangle, we know that its first row is $[a_i, a_{i+1}, \ldots, a_{i+p}]$, its second row is $[a_{i+k}, a_{i+k+1}, \ldots, a_{i+p+k+1}]$, and so on $\pmod{n}$.
- Since there are n elements in the array, there are thus at most n such subrectangles.

Rows of Stars

Solution

- It is therefore possible to compute the sums of all distinct $p \times q$ subrectangles in this grid and choose the largest one.
- We can first use sliding window sums to compute the sum of all p-length subarrays of the initial row (with wraparound). Call these sums s_i.
- Then, note that the subrectangle with initial row having sum s_i will have its other rows with sums s_{i+k}, s_{i+2k}, etc. $\pmod n$.
- We can thus compute all these sums of all the subrectangles efficiently by partitioning these row sums into cycles and computing sliding window sums on those cycles.
- All this can be done in $O(n)$.

Rows of Stars

Solution (cont.)

- The last step is to determine which of these subrectangle sums actually appear in the original grid.
- However, because each subrectangle is defined by its top-left element, it suffices to enumerate the distinct elements in the top-left $n - p + 1 \times n - q + 1$ rectangle of the grid.
- This can be done efficiently in a variety of ways; for example, by treating each row as an interval over indices and using interval merging.
- This step is also $O(n)$, making the entire solution $O(n)$ as well.

Manhattan Interference

Problem

- Each company has exactly 2 cell towers.
- Phones with a SIM card from a company connect to the nearest tower by Manhattan distance.
- A SIM experiences **Manhattan interference** if a point is equidistant from both towers.
- Phones with 2 SIM cards explode if both SIM cards experience interference at the same point.
- **Goal**: Count pairs of companies which can cause an explosion.

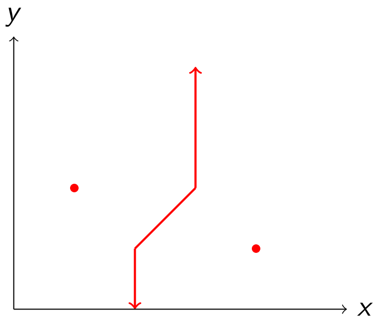
Manhattan Interference

The "Wiggle" Shape

For towers at (x_1, y_1) and (x_2, y_2), equidistant points form a "wiggle":

- Diagonal line segment with slope ± 1 through midpoint
- Two axis-aligned parallel rays extending to infinity
- Rays are either both horizontal or both vertical

Special case: If $|x_1 - x_2| = |y_1 - y_2|$, this will always intersect all other wiggles.



Manhattan Interference

Categorization

- **Vertical wiggles**: Rays extend up/down (parallel to y-axis)
- **Horizontal wiggles**: Rays extend left/right (parallel to x-axis)
- **Square wiggles**: If $|x_1 - x_2| = |y_1 - y_2|$, this will intersect all other wiggles.

Key observation: The only pairs that cannot intersect are two vertical wiggles or two horizontal wiggles.
Strategy: Count non-intersecting pairs within each type, subtract this value from $\binom{n}{2}$.

Manhattan Interference

Counting Non-Intersections

Consider vertical wiggles with endpoints (x_1, y_1) and (x_2, y_2):

- **Different slopes**: Wiggles intersect if x-ranges $[x_1, x_2]$ overlap. Can solve with a sweep line + Fenwick Tree.
- **Same slope**: Need 2D data structure to determine overlap, as overlapping x-ranges do not guarantee intersection!
- Handle horizontal wiggles symmetrically (reflect over line $y = x$).

Manhattan Interference

Implementation

Trick: Double all coordinates to make all points integers.
Algorithm:

1. Classify wiggles, discard $dx = dy$ cases.
2. Sort intervals and sweep.
3. Query intersections using interval/2D data structure.
4. Count non-intersecting pairs.
5. Answer $= \binom{n}{2}$ - (non-intersecting).

Complexity: $O(n \log^2 n)$ (from 2D range tree queries).

Game of Nines

Problem

- You have a list of n digits. You can add one digit to the other, and replace it with their sum mod 10. Any sum that is 9 is eliminated. Determine the maximum number of digits that can be eliminated.

Observation

- If we can obtain at least one digit 1, then we can use 1 to eliminate all the other digits by repeatedly adding 1 to the other digits until they become 9.
- It can be shown either by case work or number theory that you can obtain a digit 1 except for three cases:
  1. All digits are even.
  2. All digits are multiples of 5 (mod 10): i.e., 0 or 5.
  3. All digits are multiples of 3 (there exists at least one 3 or 6).

Game of Nines

Solution

- In cases 1 and 2, we cannot eliminate any digits. The answer is *n*.
- In case 3, we can add one 3 to all the other digits repeatedly until we eliminate all of them. The answer is 1.
- In all other scenarios, we can obtain a digit 1 and eliminate everything else. The answer is 1.

Swap for Palindrome

Problem

- You are given a string of $n \leq 5\,000$ characters. You are to make exactly one swap of two characters so that the resulting string has the longest palindromic substring.

Swap for Palindrome

Observation

- Since $n \leq 5\,000$, we can afford enumerating all palindromic substring centers and greedily fixing any mismatches going outward. This will take $O(n^2)$ time.
- Let `c` (or `cc`, when the substring length is even) be the center of the palindromic substring.
- (Case 1) The best mismatch we can resolve with one swap looks like this: `..[a..b..c..a..b]...` In this case we can swap `a` and `b`.
- (Case 2) If we have `..[a..c..b]..`, we can also try to find an `a` or `b` outside `[..]`, or from the center (don't forget `c` when `c` = `a` or `c` = `b`).
- After resolving the mismatch with one swap, we shall continue extending the palindrome until another mismatch, which we won't be able to fix.

Swap for Palindrome

Solution

- Enumerate the palindrome centers (separate odd/even lengths).
- Find the first two pairs of mismatches `..[s..p..c..q..t]..`, i.e. everything matches in `[..]` except that $\mathtt{p} \neq \mathtt{q}$ and $\mathtt{s} \neq \mathtt{t}$.
- Check if we can fix the mismatches using the cases discussed previously. After any fix, continue extending the palindrome until another mismatch.
- Two common pitfalls:
  - Must pay special attention to the center character (`c`) which can also participate in the swap.
  - The swapped letter in Case 2 to fix the (`p`, `q`) match could be between `s..p` or `q..t`, which changes where the palindrome can extend to.

Tree Racing

Problem

- You have m racers spawned at random nodes in a tree with n nodes. Each racer can traverse from one end of an edge to another in different amounts of time and they are all trying to reach a finish node at varying speeds.
- Some nodes in the tree are special, meaning that only the first k racers that reach the node can go through it. Anyone else following that is effectively eliminated from the race.
- Your task is to figure out who will be able to reach the finish node, and for those that are able to reach the finish node, how long it would take them to get there.

Tree Racing

Observation

- We can re-formulate the racetrack as a tree rooted at the finish node e and every racers trying to reach the root from arbitrary nodes in the racetrack.
- There are only one possible path from any node to the finish node, meaning that a racer will be eliminated if there exists a special node at which they are not one of the fastest k racers on the path from their spawned node to the finish node.
- If a racer is eliminated at a special node x, then there is no reason to consider them at some special node y such that y is an ancestor of x in the tree rooted at the finish node.

Tree Racing

Observation (cont'd)

- Notice that at any special node x, if we have a list of candidate racers that are trying to get through x, we can just take out the k fastest racers and move them to parent node of x in the rooted tree, thus allowing us to not consider the eliminated ones any further.
- Naively moving the racers upward towards the root will result in an $O(n^2)$ time complexity, which is not fast enough for the constraint of this problem. In other words, we need a more efficient way to construct the list of candidate racers for a special node.

Tree Racing

Solution

- Root the tree at the finish node e.
- Process each node in the rooted tree in a postorder traversal manner. At each special node, determine the list of candidate racers that can reach the node and only move the fastest k racers upward.
- To efficiently construct the list, one can utilize the *small-to-large merging* technique, also known as *DSU on tree*, to efficiently create list of candidate racers for a node based on that of its children.

Tree Racing

Solution (cont'd)

- Another way to construct the list is to have each node send its list of racers directly to its lowest ancestor that is a special node. To determine the closest special ancestor for each node, one can add additional information regarding most recently encountered special node within the DFS call and update it accordingly. Notice that in this way, keeping track of the depth of each node in the rooted tree will allow one to easily determine the distance from a node to its closest special ancestor.

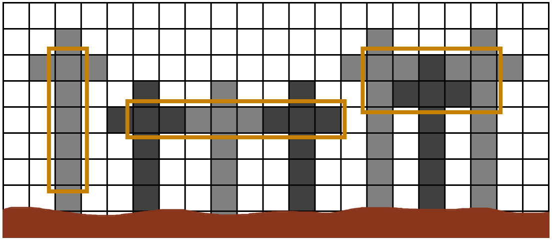
Much Room for Mushrooms

Problem

- You wish to determine if it is possible to completely fill an r by c region with specially defined “mushrooms”.

Observation

- We can observe three types of regions that we can fill with mushrooms, as shown in the diagram below.



Much Room for Mushrooms

Solution

- Check which of the above cases we have.
- For X by 1 regions, we can use a single tall mushroom.
- For 1 by X regions, we need $\lceil \frac{X}{3} \rceil$ mushrooms placed side by side at the same height.
- For 2 by X regions, we need $\lfloor \frac{X}{2} \rfloor + 1$ mushrooms placed at an alternating heights.
- All other cases are impossible.