香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

M2651 Introvert Leaving

Nicholas Ko {__jk__}
2026-06-30

Background

- Problem idea by `__jk__`
- Preparation by `__jk__`, `WongChun1234`, `QwertyPi`, `VCLH`

The Problem

You are given a N × M grid of nonnegative integers A[0 .. N - 1][0 .. M - 1].

For each cell (i, j) in the grid, we define the “leaving cost” of the cell (i, j) as the minimum possible value of A[][] seen when moving from (i, j) to anywhere outside the grid by orthogonal moves.

Answer Q queries: each query consists of two cells (r0, c0) and (r1, c1) — on top of the orthogonal moves, you can also teleport between the two given cells. For each query, find the sum of the leaving costs of all N × M cells.

Subtask Overview

Subtask	Score	Constraints
1	12	$N \times M \leq 1000$ $Q \leq 30$
2	17	$N \times M \leq 2 \times 10^5$ $Q \leq 30$
3	15	$A[i][j] \leq 20$
4	19	(r0, c0) is a cell on the boundary
5	19	$N \times M \leq 2 \times 10^5$ $Q \leq 2 \times 10^5$
6	18	No additional constraints

Statistics

- First solved by wy_24215 (Yang Chun Kit) @ 2h 43min
- Last solved by WYK23F32 (Xu Adam) @4h 28min

- 2x 100
- 1x 63
- 2x 48
- Handful of 12s and 29s

Subtask 1 (12%): N × M ≤ 1000, Q ≤ 30

- We can interpret the grid as a graph, where
  - the nodes are the N × M cells, as well as one extra node representing “outside”
  - the edges are between neighbouring cells in the grid, as well as one extra node connecting between (r0, c0) and (r1, c1)

- What is the leaving cost of a cell (i, j)?
- It is the **smallest** nonnegative integer x such that there exists a path from the node representing cell (i, j) to the node representing outside.

Subtask 1 (12%): $N \times M \leq 1000$, $Q \leq 30$

- For each query and for each separate cell, we use a DSU to maintain the connectivity of each node.
- We sweep in increasing order of x, uniting the new relevant edges (the edges with max(A value of the two ends) = x).
- The leaving cost is the smallest such x for which (i, j) and outside are connected, which can be checked in $\alpha(NM)$ time.

- Time Complexity: $O(NM \log NM + Q \times (NM \times \alpha(NM)))$
- Expected Score: 12

Subtask 2 (17%): $N \times M \leq 2 \times 10^5$, $Q \leq 30$

- We want to optimise our approach in Subtask 1.
- It is inefficient to compute the leaving costs of every cell separately.

- Notice that the edges are undirected.
- So, instead of computing the minimax distance of every cell to the outside node, it is equivalent to computing the minimax distance **from** the outside node **to** each of the $N \times M$ cells.

Subtask 2 (17%): $N \times M \leq 2 \times 10^5$, $Q \leq 30$

- We can use dijkstra to find the single source (in this case, it is the outside node) shortest path of all nodes.

- Time Complexity: O(Q * (NM log NM))
- Expected Score: 29

Subtask 3 (15%): A[i][j] ≤ 20

- In Subtask 1, we did a DSU in increasing order of edge weights.
- We try to optimise this part.

- Suppose first that no teleporters exist.

- The DSU process can be thought of having 21 “tick”s of time.
  - Before the 0th tick, there are N × M + 1 singleton nodes.
  - After the 20th tick, all the nodes are united into one component.

Subtask 3 (15%): A[i][j] ≤ 20

- Informally, after the i-th tick, the DSU unites all edges with max(A value on both sides) ≤ i.
- We compute size[i], the number of nodes with leaving cost ≤ i (i.e. the number of cells in the same component as outside after the i-th tick).
- For each query, we want to compute size'[i], the number of nodes with leaving cost ≤ i after considering the teleporter (r0, c0) to (r1, c1).
  - Then the answer is just ∑ i × (size'[i] - size'[i - 1])

Subtask 3 (15%): A[i][j] ≤ 20

- Obviously, if max(A[r0][c0], A[r1][c1]) > i, then size[i] == size'[i].
- If max(A[r0][c0], A[r1][c1]) ≤ i, how many new cells have leaving cost ≤ i?
- If (r0, c0) is connected to the outside after the i-th tick but (r1, c1) is not, then the number of such new cells is the component size of (r1, c1) after the i-th tick.
- Similarly for (r1, c1) connected to outside but (r0, c0) is not.
- If both are connected or both not connected to outside, then no change.

- Time Complexity: O(max A * NM × α(NM) + Q * max A)
- Expected Score: 15 (Cumulative: 44)

Subtask 4 (19%): (r0, c0) is a boundary cell

- The solution to Subtask 2 involved finding some minimax distance.
- What is a common technique that allows you to find the minimax distance between pairs of nodes efficiently?

Kruskal Reconstruction Tree

- We build the Kruskal Reconstruction Tree for the grid graph, where for every newly created node w as a consequence of the edge u – v, we assign the weight of node w as max(A[u], A[v]).

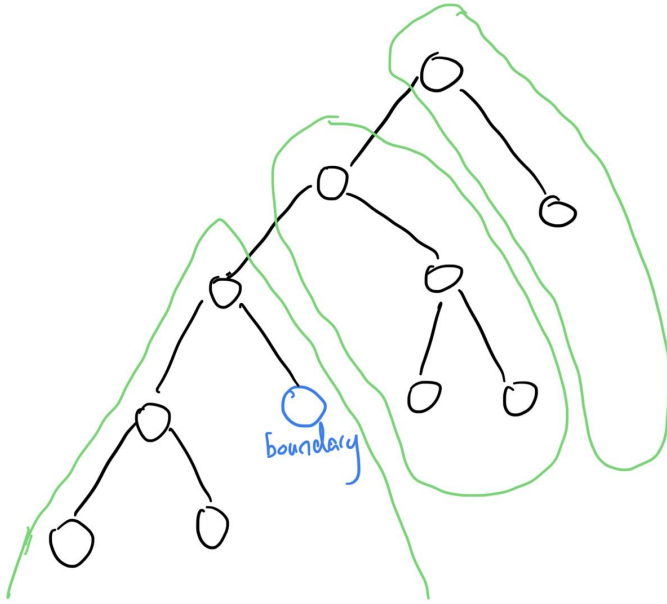
Subtask 4 (19%): (r0, c0) is a boundary cell

- If there were no teleporters, then the answer is just sum {over all N × M cells C} of weight[LCA(C, outside node)].

- Now that there is a teleporter between (r0, c0) and (r1, c1), there is now a shortcut from (r1, c1) to the boundary with cost max(A[r0][c0], A[r1][c1]).
- So, every cell C has an alternative option with cost
  max(weight[LCA(C, (r1, c1))], A[r0][c0], A[r1][c1]).
- When might this alternative option be better than the original option, which has cost weight[LCA(C, outside node)]?

Subtask 4 (19%): (r0, c0) is a boundary cell

- Consider the path from the boundary cell to the root in the Kruskal Reconstruction Tree.
- This path partitions the tree into several subtrees, where all the nodes in a subtree have the same original cost.
- If a cell C is not in the same subtree as (r1, c1), then the alternative route definitely won't be better.



Subtask 4 (19%): (r0, c0) is a boundary cell

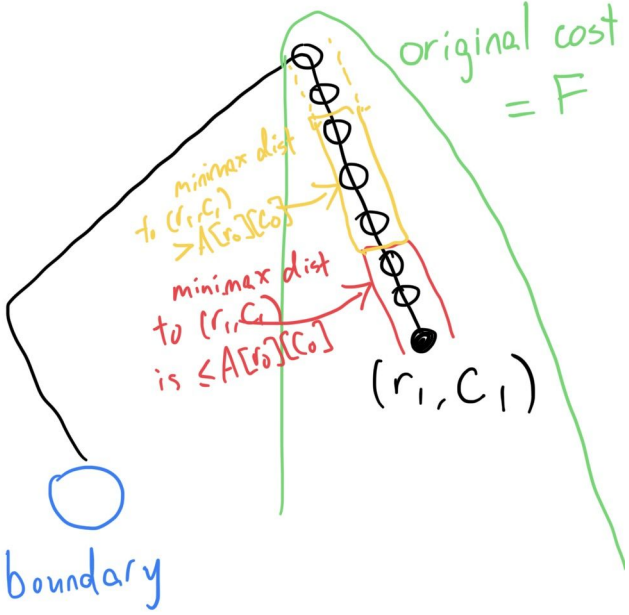
- If a cell C is in the same subtree as (r1, c1), in which all the nodes in the subtree has original cost = F, then you save

  max(0, F - max(weight[LCA(C, (r1, c1))], A[r0][c0]))
- If we can sum the expression over all such cells C, then we are done.

- Consider the ancestors of (r1, c1) until it meets the path between the outside cell and the root.
- Some (possibly empty) prefix of it has A[r0][c0] as the limiting term, followed by another interval with weight[LCA(c, (r1, c1))] as the limiting term, then by a suffix with 0 as the limiting term in the expression above.

Subtask 4 (19%): (r0, c0) is a boundary cell

- For the first part (A[r0][c0] is limiting), the contribution is simply the expression multiplied by the number of cell nodes in the shallowest subtree for which the first part is limiting.
- For the second part (weight[LCA(C, (r1,c1))] is limiting), calculating the sums directly is inconvenient. Instead, we do a difference array transformation to see that we can rewrite it as ∑ (weight[pa[i]] - weight[i]) × cnt_cell_nodes[pa[i]] over all i along that path.



Subtask 4 (19%): (r0, c0) is a boundary cell

- This can be precomputed in O(NM log NM) time.
- Answering queries can be done by binary lifting, which is O(log NM) time per query.
- Total time complexity: O(NM log NM + Q log NM)
- Expected Score: 19 (Cumulative: 63)

Subtasks 5 & 6 (19% + 18%): No additional constraints

- WLOG assume the original leaving cost from (r0, c0) is smaller than that from (r1, c1).

- We observe that useful teleportations can only happen in the direction (r1, c1) → (r0, c0).
- The alternative option of a cell C now has cost
  max(weight[LCA(C, (r1, c1))], leaving cost from (r0, c0)).
- Since the second quantity is still a constant, we can just use the solution from Subtask 4 with appropriate minor modifications.

Subtasks 5 & 6 (19% + 18%): No additional constraints

- The intention of Subtask 5 is to allow slightly inefficient ways to calculate the contributions. Instead of difference arrays and then binary lifting, you might directly do a heavy light decomposition.

- Time Complexity: $O(NM \log NM + Q \log^2 NM)$ / $O(NM \log NM + Q \log NM)$
- Expected Score: 82 / 100

Takeaways

- Familiarise yourself with the standard techniques! Whenever you see something like “minimum possible value of the max-s”, it is instructive to realise that Kruskal Reconstruction Tree is the type of data structure that suits well for handling such queries.

- After reducing the problem onto a tree, think carefully whether you can use techniques that are easier to code, such as partial sums, differences arrays, and binary lifting, instead of directly hammering the problem with complicated-to-write decompositions.

M2652 Subgrid Message

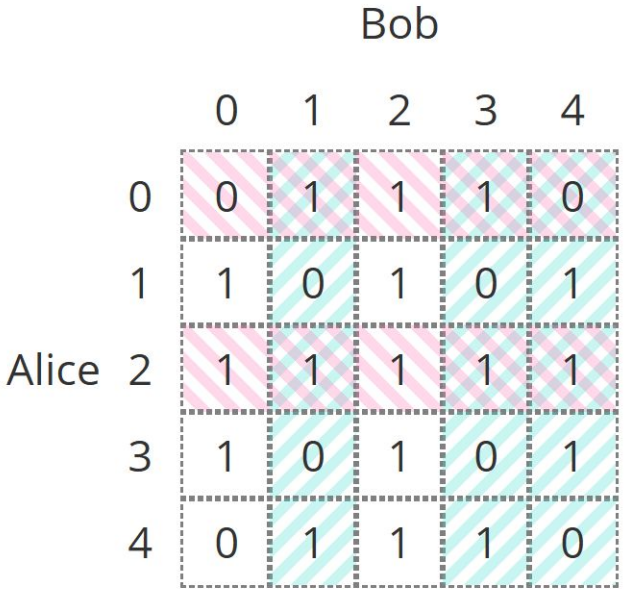
Daniel Hsieh {QwertyPi}

2026-06-30

Background

Problem Idea by QwertyPi

Preparation by QwertyPi and ITO

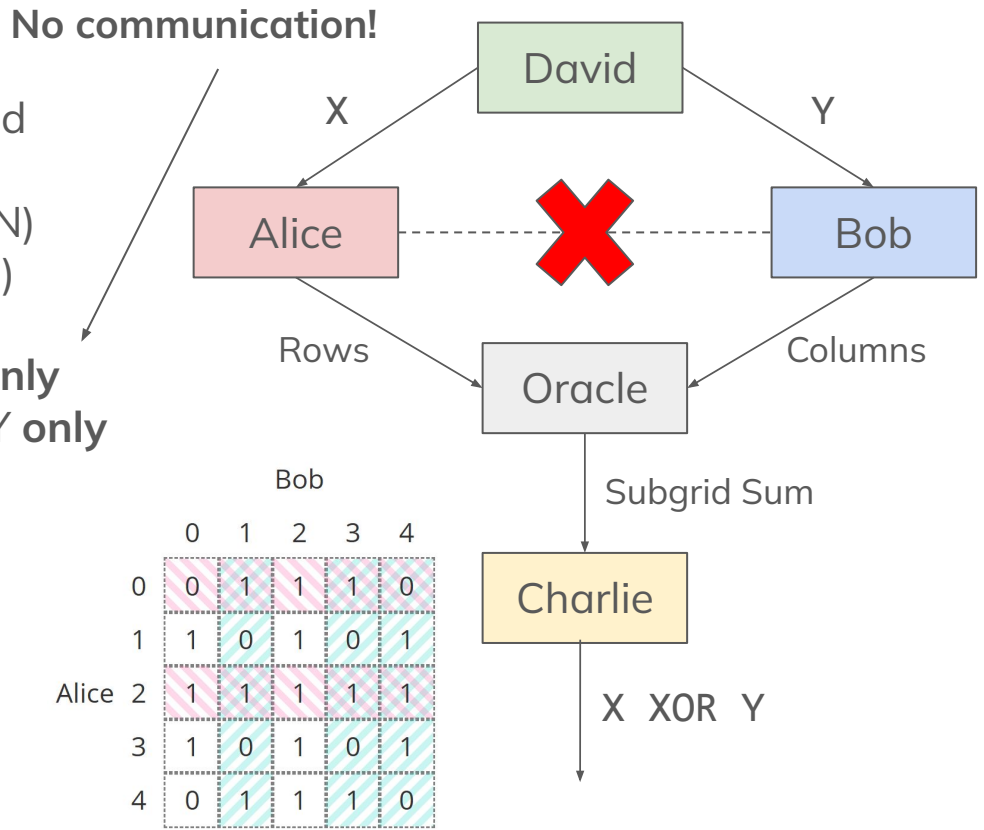


Inspired by the CHSH game in quantum mechanics!

TL;DR: Correlation may arise from measurements without communication

Task Setup

1. Alice and Bob decides K × K 01-grid
2. (a) Alice receives secret X ($0 \le X < N$)
   (b) Bob receives secret Y ($0 \le Y < N$)
3. (a) Alice selects rows based on X **only**
   (b) Bob selects columns based on Y **only**
4. Subgrid sum given to Charlie
5. Charlie returns X XOR Y



Subtask Scoring

Hard limit on grid size: K = 512

Subtask 1 (20%): N ≤ 64, all or nothing

Subtask 2 (80%): N ≤ 256, multiplier = 51 / K

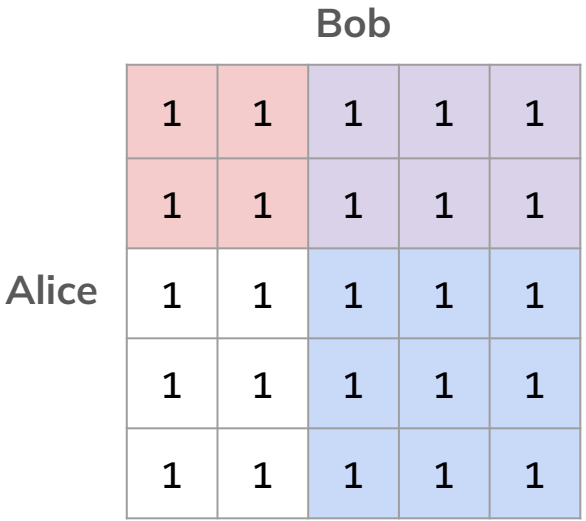
Statistics

M2652 - Subgrid Message	27	75.135	58.434	22.52	20: 27	80: 0 55.135: 4 54.4: 6 53.684: 5 52.308: 2 49.157: 1 15.814: 1 14.624: 1 14.166: 1 8: 2 7.984: 1

Simple Solutions

Idea 1. All Ones Grid

- Consider the simplest case: suppose the K × K grid are all 1s.
- Now, suppose Alice selects A rows and Bob selects B columns.
- Charlie receives the subgrid sum C = A * B.




```
# of rows = 2
# of columns = 3
C = 2 × 3 = 6
```

Solution 1. Prime Factorisation (K = 311 / 1619)

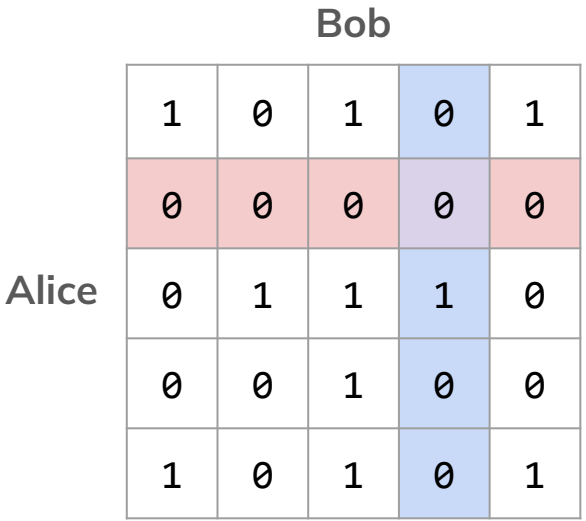
- Issue: when say C = 4, we have no way to tell either (A, B) = (4, 1) or (2, 2)!
- Workaround: suppose Pr[k] is the (k + 1)-th prime number, then alice selects Pr[X] rows and bob selects Pr[Y] columns.

- Since C = Pr[X] × Pr[Y] can be factorised uniquely (up to order), we can obtain both X and Y and hence the value of X XOR Y.
- Fact: Pr[63] = 311, Pr[255] = 1619

- Expected Score: 20

Idea 2. Grid as Boolean Array

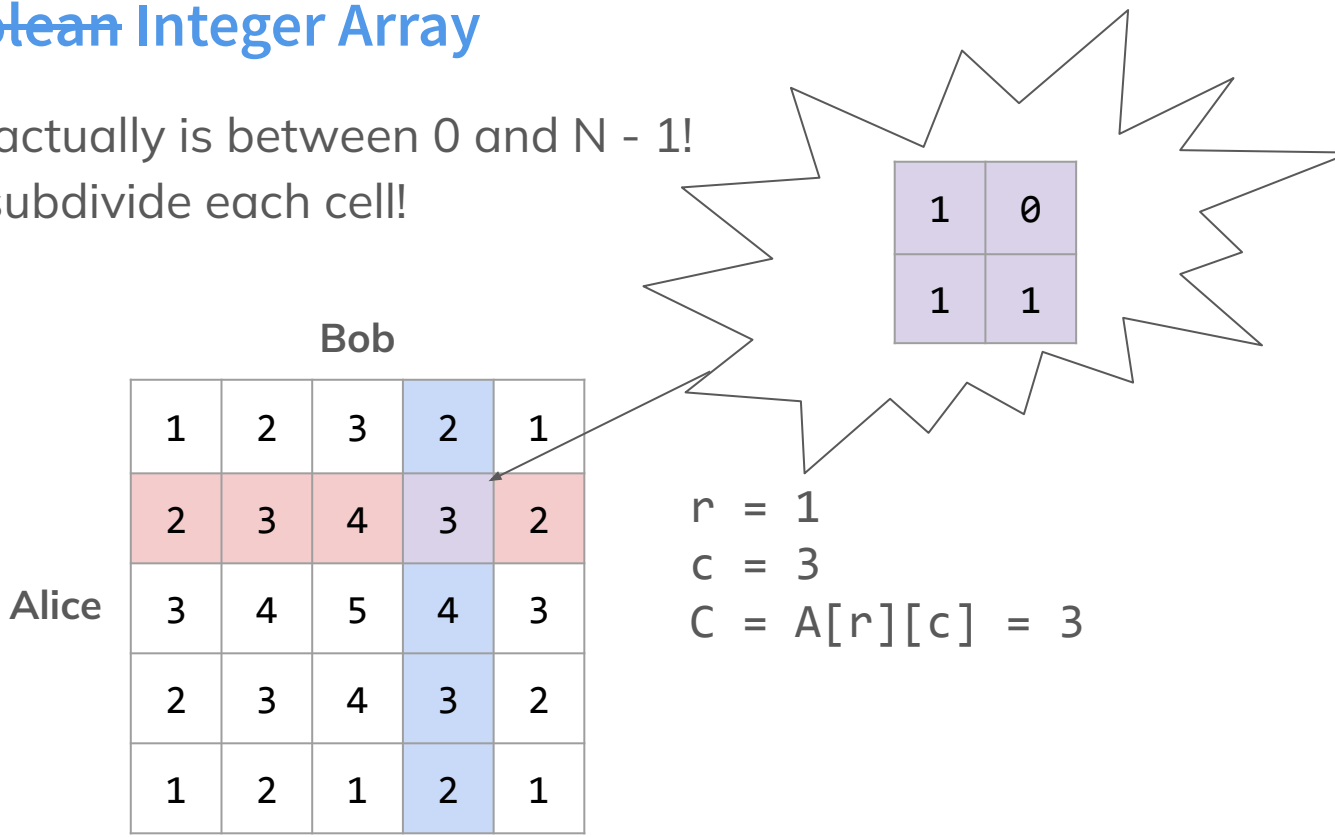
- Another simple idea: What if Alice just sends the X-th row, and Bob just sends the Y-th column?
- Effectively, just treat the grid as a “2D boolean array”:




```
r = 1
c = 3
C = A[r][c] = 0
```

Idea 2+. Grid as ~~Boolean~~ Integer Array

- Issue: Our answer actually is between 0 and N - 1!
- Workaround: Just subdivide each cell!



Solution 2. Integer Array (K = 256 / 4096)

- To send: A[X][Y] = X XOR Y.
- Since the value in each cell is between 0 and N - 1, we can take the side length for each “cell” as sqrt(N).
- This leads to a grid size of K = N sqrt(N).
- Expected Score: 20

Bob

Alice

0	0	1	0	1	1	1	1
0	0	0	0	0	0	1	0
1	0	0	0	1	1	1	1
0	0	0	0	1	0	0	0
1	1	1	1	0	0	1	0
0	0	1	0	0	0	0	0
1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0

Example for N = 4

Solution 3. Encode (X, Y) (K = 128 / 512)

- How about we send (X, Y) directly?
- By sending Z = NX + Y to Charlie, he can recover (X, Y) directly!

- Grid: Like the one shown on the right
- Alice: row 0 to X - 1 and 2N - 1
- Bob: column 0 to Y - 1 and N to 2N - 1

- Grid Size: K = 2N
- Expected Score: 27.969

Bob

Alice

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0

Example for N = 4

(Not-that) Simple Solutions

Entropy Analysis

Directions for improving our solutions:

1. How can we **reduce information to be sent**?
2. How can we **send information more efficiently**?

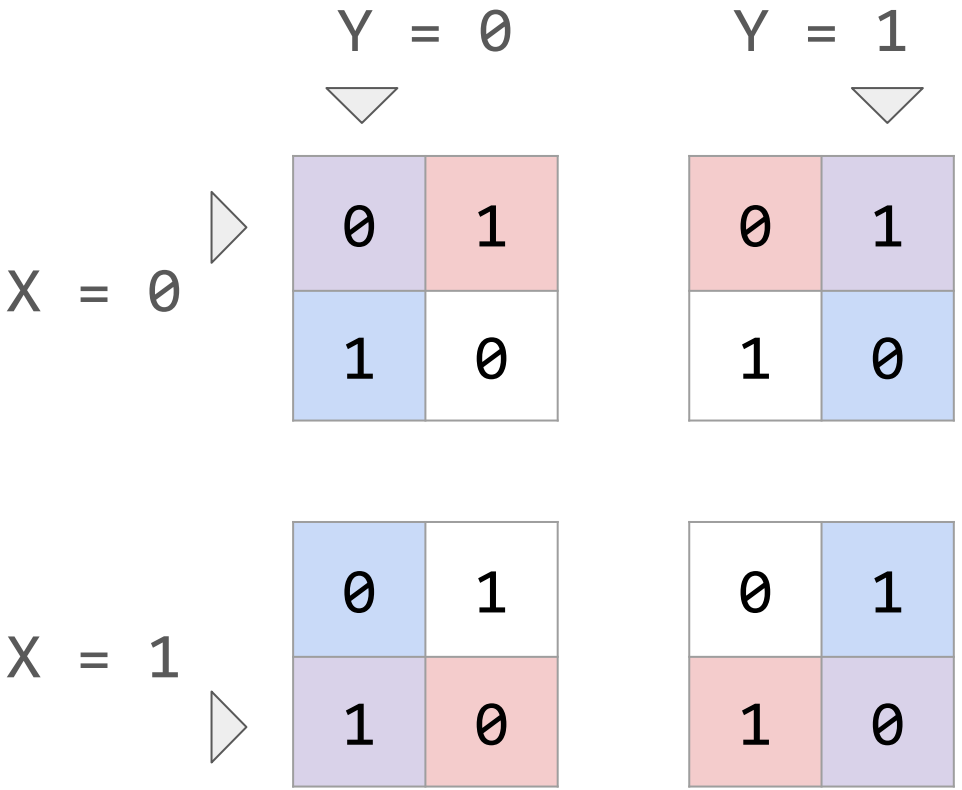
For (1), we want to encode stuff of **less entropy** - say X XOR Y instead of (X, Y), which only has N possibilities instead of N^2.

For (2), we want to **maximise number of cells involved**, that is, we want to use more space in the grid.

Anyways - we should start to solve the task bit-by-bit, since we are dealing with bitwise XOR!

Bitwise XOR

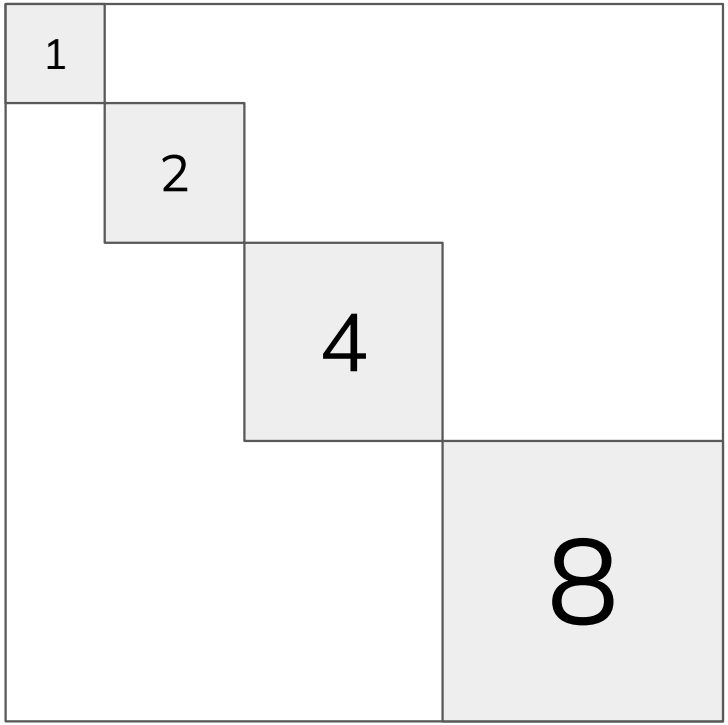
$X \oplus Y$



Solution 4.1. Bitwise Subgrid (K = 76 for N = 256)

- Simply send the bits along diagonals!
- For each diagonal subgrid, encode the XOR for that bit.
- The max number of black cells sent to Charlie is simply N - 1, so entropy wise is theoretical best.

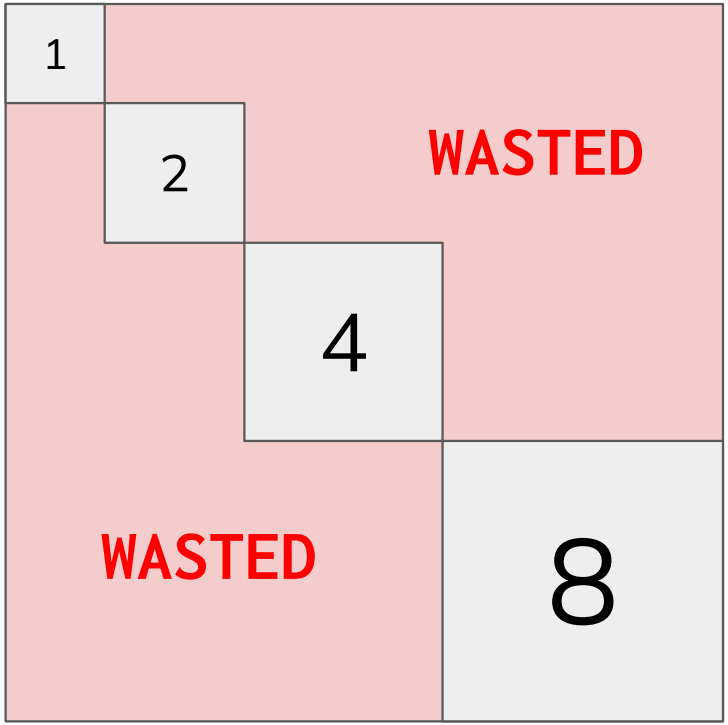
- When N = 256, K = (1 + 2 + 2 + 3 + 4 + 6 + 8 + 12) × 2 = 76
- Expected Score: 73.684



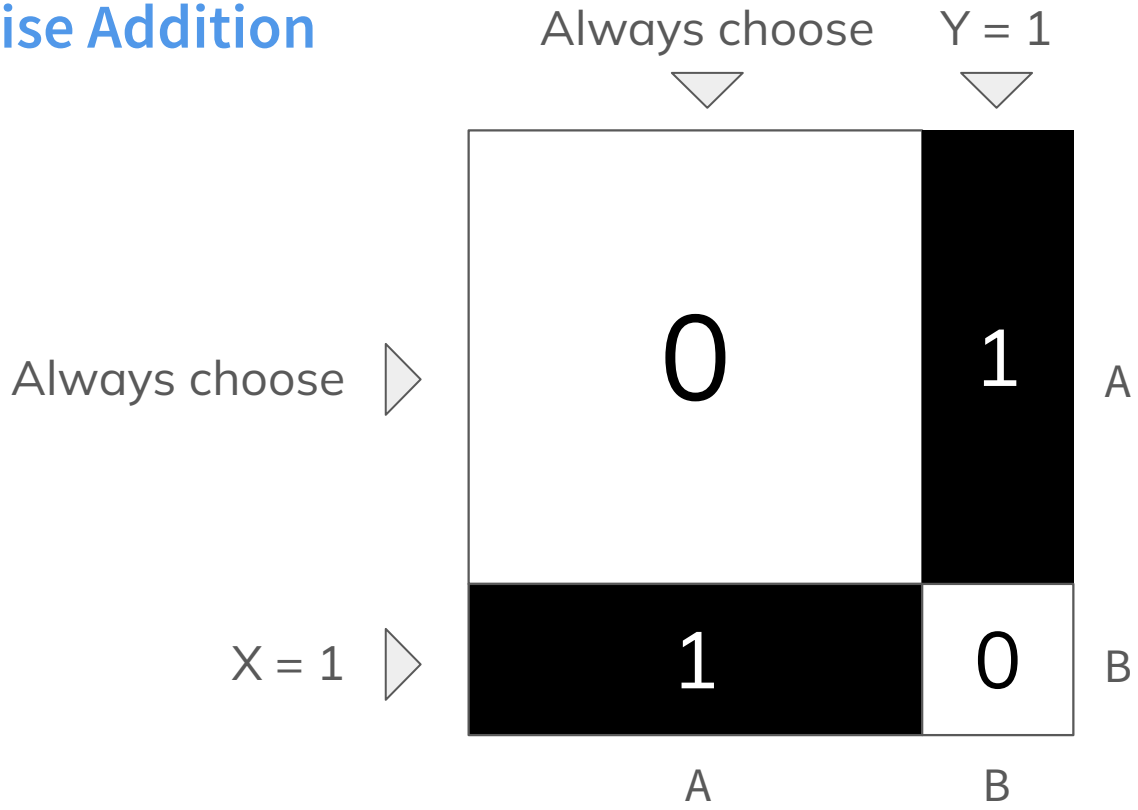
"Space" Complexity

- ~~Here, the number of possibilities sent to Charlie is simply N, which means we are not wasting anything.~~

- Unfortunately, spaces are wasted :(
- The grid size here is limited by space instead of entropy of the output!
- How to use space more efficiently?



Bitwise Addition



Effectively sending

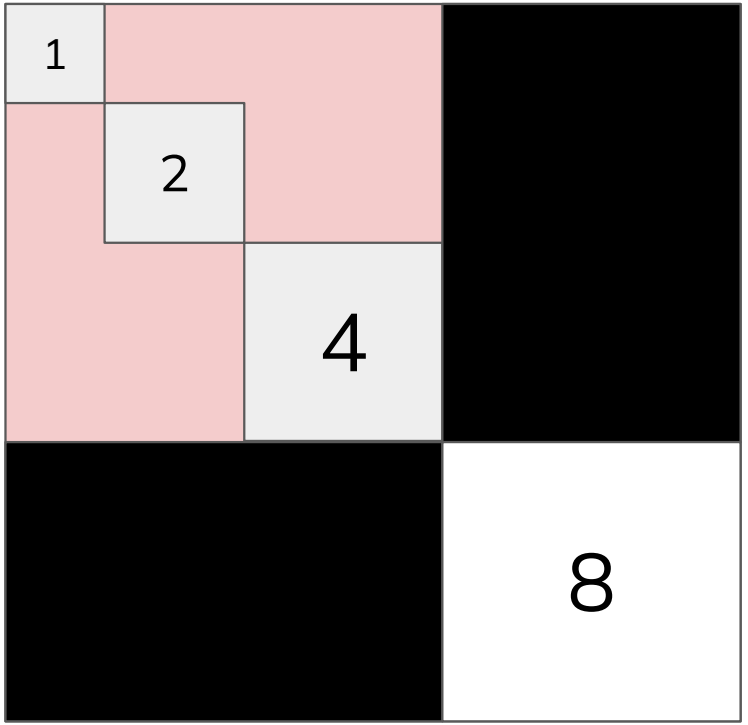
$$(X + Y)\ AB$$

- Base-3 :(
- But use the space wasted before :)

Solution 4.2. Bitwise Subgrid Pro (K = 57 for N = 256)

- Use base-3 for the last bit!
- Key observation: For each bit encode by XOR, for certain the number of rows / columns picked are constant by construction!

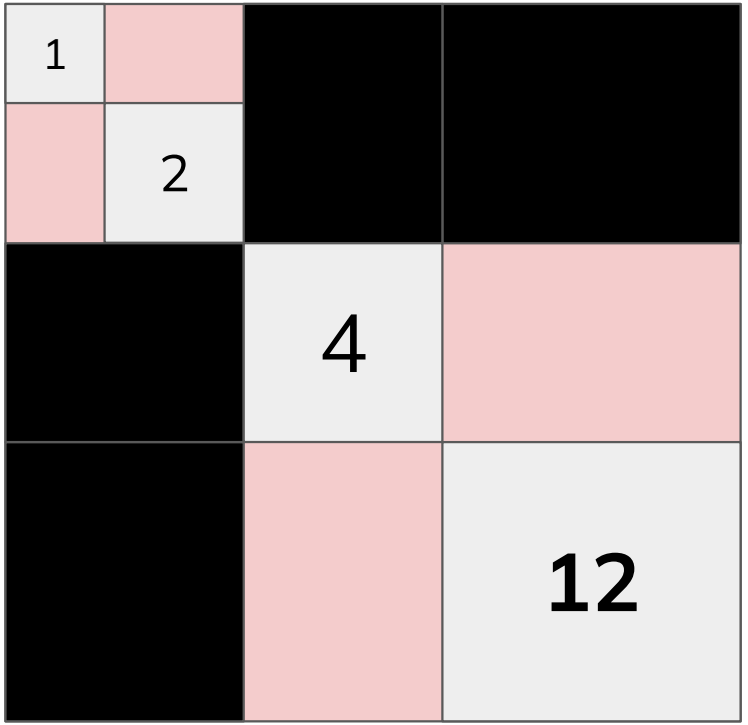
- When N = 256, K = (1 + 2 + 2 + 3 + 4 + 6 + 8) × 2 + ceil(128 / 26) = 57
- Expected Score: 91.578



Solution 4.3. Bitwise Subgrid Pro Max (K = 51 for N = 256)

- Use base-3 for the last **two** bits!
- Key observation: For each bit encode by XOR, for certain the number of rows / columns picked are constant by construction!

- When N = 256, K = (1 + 2 + 2 + 3 + 4 + 6) × 2 + ceil(64 / 18) + ceil(**192** / 18) = 51
- Expected Score: **100**



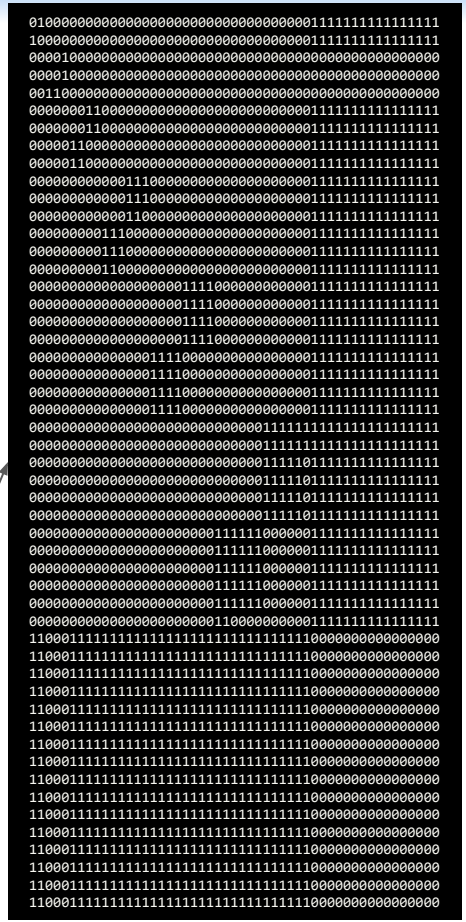
Takeaways

- Think: What are the challenge for this task?
  - Usable Space
  - Output Entropy

  Try to justify and come up with different solutions inspired by these challenges!

- Come up with more “simple” components, and try to mix and match them
  - You need both base-2 and base-3 ideas for this task

- Try to illustrate your program output for debugging!



M2653 - Tree Communication

Yu San Cheng {yellowtoad}

2026-06-30

Background

Problem Idea by `yellowtoad`

Preparation by `yellowtoad`, `firewater`, `gasbug`

Presented by `yellowtoad`

The Problem

Alice and Bob are intially in distinct leaves of a binary tree. They have no information about the structure of the binary tree as well as the location of the other person.

They take turns to move alternately. In each move, the person can either move to an adjacent node or stay at the node where he/she is currently at (for full score the player must move). Then, the other person knows whether the distance between the two people increases, decreases, or remains unchanged after the move.

The goal is to implement a strategy such that Alice and Bob can eventually meet in a node using minimum number of moves.

Statistics

M2653 - Tree Communication	34	100	31.763	36.659	100: 6 97: 1 25: 13 19.321: 3

First solved by `dbsculver0412` at `1h 31m 16s`.

$4N$ Solution, Allow Not Moving

Alice: Never moves.
Bob: Performs DFS on the tree, only moving to unvisited nodes. If there are no unvisited adjacent nodes, go back to the node where he first arrived the current node from.

After moving $2N$ times, Bob will have completed the DFS on the binary tree, visiting every node in it. Since Alice never moves, it is guaranteed that Bob can meet her at some point.

Total number of moves: $4N$
Expected score: 6.25

$4N$ Solution

Both Alice and Bob performs DFS simultaneously.

It is guaranteed that Alice and Bob can meet at some node. Proof is left as exercise for readers.

It requires $2N$ moves for each person to complete their DFS. So, $4N$ moves in total.

Total number of moves: $4N$
Expected score: 25

$6D$ Solution, Allow Not Moving

Consider a solution where Bob helps Alice find her path to meet him.

Let $u_1, ..., u_k$ be the simple path from Alice to Bob, where u_1 is Alice's initial position and u_k is Bob's initial position.

Assume that Alice has already discovered the path from u_1 to u_i and is currently at u_i.

Since the tree is a binary tree, there are at most 3 neighbours for each node, and only one of them decreases the distance between Alice and Bob (u_{i+1}). Alice further knows that u_{i-1} increases the distance. So, there are at most two potential candidates for u_{i+1}.

$6D$ Solution, Allow Not Moving

The following shows how Bob can help Alice find u_{i+1} within 6 moves, with Alice moving first:

1. Alice moves to a potential candidate of u_{i+1}.
2. If the distance decreased, Alice has chosen the correct node and Bob tells Alice by not moving. Otherwise, Bob move to any adjacent node.
3. If the distance remains unchanged (Bob did not move), Alice knows that she chose the correct node, then she proceeds to find u_{i+2} by repeating from step 1. Otherwise, she knows she has chosen the wrong candidate. She moves back to u_i.

$6D$ Solution, Allow Not Moving

The following shows how Bob can help Alice find u_{i+1} within 6 moves, with Alice moving first:

4. Bob moves back to his original position.
5. Alice moves to the other candidate, which must be u_{i+1}.
6. Bob does not move. Then, repeat from step 1 to find u_{i+2}.

6 moves are used at worst case to decrease the distance by 1 between Alice and Bob.

Total number of moves: $6D$
Expected score: 25

$13D$ Solution

In the previous solution, Bob indicates whether Alice moved closer by either moving or not moving. Can we still find a way to indicate this when Alice and Bob are forced to move?

Note that if a player is not at a leaf, then there are at least two neighbours, one of them must decrease the distance while others increase the distance.

How can we use this fact to communicate with Alice?

$13D$ Solution

First, Alice moves to a potential candidate for the next node. **(1 move)**

If Alice moved correctly, Bob moves to all of its neighbours at least once. Otherwise, Bob moves to the same neighbour three times. **(9 move)**

Alice then observes Bob response for the previous moves. If all three responses are the same, she knows that Bob moved to the same neighbour three times, meaning that she chose the correct node. Otherwise, one of Bob's neighbour must give a different response than the others, meaning that she chose the wrong node. **(3 moves)**

$13D$ Solution

After 13 moves, Alice is at u_{i+1} and Bob is at his original position, with Bob moving first. So, it's Alice turn to help Bob find his next node.

By alternately helping each other find their next nodes with the above strategy, they will ultimately meet on the simple path between them.

Total number of moves: $13D$
Expected score: 50.5

$9D$ Solution

The $13D$ solution gave us insights on how Alice and Bob alternately help each other to find their next node.

How can Bob indicate whether Alice moved correctly using less moves?

Let $u_1, ..., u_k$ be the simple path from Alice to Bob, where u_1 is Alice's initial position and u_k is Bob's initial position.

Assume that Alice has already discovered the path from u_1 to u_i and is currently at u_i, and Bob has already discovered the path from u_k to u_j and is currently at u_j.

$9D$ Solution

Case 1: u_j has two neighbours

Denote u_{j+1} as v and the other neighbour as w.

Case 2: u_j has three neighbours

Denote the two neighbours that is not u_{j+1} as v and w.

Observation: v and w always give different responses in the above two cases.

So, Bob can first send the response of node v to Alice, then indicate whether Alice moved correctly by moving to v or w.

$9D$ Solution

$9D$ solution:

1. Alice moves to a potential candidate for u_{i+1}.
2. Bob moves to v.
3. Alice remembers the response of v, then moves back to u_i.
4. Bob moves back to u_j.
5. Alice moves to the same potential candidate for u_{i+1}.
6. If Alice moved closer, Bob moves to v again. Otherwise, Bob moves to w.

$9D$ Solution

$9D$ solution:

7. If Bob gave the same response as v, Alice knew she moved correctly. Otherwise, she moved to the wrong candidate. For simplicity, she moves back to u_i no matter what.
8. Bob moves back to u_j.
9. Alice moves to u_{i+1}.

Then, Alice can help Bob to find his next node with the same strategy.

Total number of moves: $9D$
Expected score: 69.25

$7D$ Solution

The flaw of the $9D$ solution is that we have to spend 2 moves to send the response of v to Alice. Can we optimize this?

Let $u_1, ..., u_k$ be the simple path from Alice to Bob, where u_1 is Alice's initial position and u_k is Bob's initial position.

Assume that Alice has already discovered the path from u_1 to u_i and is currently at u_i, and Bob has already discovered the path from u_k to u_j and is currently at u_{j+1}.

Then, our v and w is simply u_j and u_{j+2}.

$7D$ Solution

$7D$ solution:

1. Alice moves to a potential candidate for u_{i+1}.
2. If Alice moved closer, Bob moves to u_j. Otherwise, Bob moves to u_{j+2}.
3. If Bob moved closer, it means that Bob moved to u_j, Alice knows she chose the correct candidate. Otherwise, Bob moved to u_{j+2}, Alice knows she chose the wrong candidate. For simplicity, she moves back to u_i no matter what.

$7D$ Solution

$7D$ solution:

4. Bob moves back to u_{j+1}.
5. Alice moves to u_{i+1}.
6. Bob moves to u_j.
7. Alice moves to u_i.

The last two moves are for 'resetting' the positions of Alice and Bob so that Alice can continue to help Bob find his next node.

$7D$ Solution

Note that for this solution to work, Alice and Bob must discover at least three nodes each (They will have to discover at least u_3 and u_{k-2} respectively).

- u_2 and u_{k-1} can be trivially found since the starting positions are leaves. **(2 moves)**.
- u_3 and u_{k-2} can be found using the $9D$ solution. **(18 moves)**.
- Alice moves one step backward to set up for $7D$ solution **(1 move)**.

21 moves are used to set up the $7D$ solution while decreasing D by 3. So, the solution still fits in the $7D$ constraint.

Total number of moves: $7D$
Expected score: 88

$6(D+1)$ Solution

In the previous solution, we used some moves to 'reset' the positions of Alice and Bob so that they can alternately help each other find their next positions. Is it possible to use less?

Let $u_1, ..., u_k$ be the simple path from Alice to Bob, where u_1 is Alice's initial position and u_k is Bob's initial position.

Assume that Alice has already discovered the path from u_1 to u_i and is currently at u_i, and Bob has already discovered the path from u_k to u_j and is currently at u_{j+1}.

$6(D+1)$ Solution

$6(D+1)$ solution:

1. Alice moves to a potential candidate for u_{i+1}.
2. If Alice moved closer, Bob moves to u_j. Otherwise, Bob moves to u_{j+2}.
3. If Bob moved closer, it means that Bob moved to u_j, Alice knows she chose the correct candidate. Otherwise, Bob moved to u_{j+2}, Alice knows she chose the wrong candidate. For simplicity, she moves back to u_i no matter what.

$6(D+1)$ Solution

$6(D+1)$ solution:

4. Bob moves back to u_{j+1}.
5. Alice moves to u_{i+1}.
6. Bob moves to u_{j+2}.

In the last move, Bob moves to u_{j+2} to set up for the next 6 moves to help Alice find u_{i+2}. In the next 6 moves:

$6(D+1)$ Solution

$6(D+1)$ solution:

7. Alice moves to a potential candidate for u_{i+2}.
8. If Alice moved closer, Bob moves to u_{j+1}. Otherwise, Bob moves to u_{j+3}.
9. If Bob moved closer, it means that Bob moved to u_{j+1}, Alice knows she chose the correct candidate. Otherwise, Bob moved to u_{j+3}, Alice knows she chose the wrong candidate. For simplicity, she moves back to u_{i+1} no matter what.

$6(D+1)$ Solution

$6(D+1)$ solution:

10. Bob moves back to u_{j+2}.
11. Alice moves to u_{i+3}.
12. Bob moves to u_{j+1}.

Then, we can repeat from the start, with Bob always helping Alice to find her way to him.

We used 12 moves to help Alice find 2 nodes. So, 6 moves per node.

$6(D+1)$ Solution

Note that the solution requires both Alice and Bob to have found at least 4 nodes (u_4 and u_{k-3} must be found before we adapt to the solution).

- u_2 and u_{k-1} can be trivially found since the starting positions are leaves. **(2 moves)**.
- u_3 and u_{k-2} can be found using the $9D$ solution. **(18 moves)**.
- Alice moves one step backward to set up for $7D$ solution **(1 move)**.
- u_4 and u_{k-3} can be found using the $7D$ solution. **(14 moves)**.

Then, we start the $6(D+1)$ solution, with Alice helping Bob to find his way to her.

$6(D+1)$ Solution

Note that Alice is one step behind the furthest node she discovered when the $6(D+1)$ solution is being set up. So, 35 moves are used to decrease D by 5. The solution fits in the $6(D+1)$ constraint.

Total number of moves: $6(D+1)$
Expected score: 97

$6D$ Solution

Just like the $6(D+1)$ solution, we can try to have one person always leading another person, instead of having them alternately helping each other.

Assume that Alice has already discovered the path from u_1 to u_i and is currently at u_i, and Bob has already discovered u_{k-1} and is currently at it.

Consider the observation from the $9D$ solution: It is always possible to find two neighbours of u_{k-1}, denoted v and w, such that they give different responses.

$6D$ Solution

Setting up the $6D$ solution:

1. Alice moves to any neighbour.
2. Bob moves to v.
3. Alice remembers the response of v, then moves back to u_i.
4. Bob moves back to u_{k-1}.

$6D$ Solution

$6D$ solution:

1. Alice moves to a potential candidate for u_{i+1}.
2. If Alice chose the correct candidate, Bob moves to v. Otherwise, he moves to w.
3. If Bob gives the same response as v, Alice knows she chose the correct node. Otherwise, she chose the wrong one. For simplicity, she moves back to u_i anyways.

$6D$ Solution

$6D$ solution:

4. Bob moves back to u_{k-1}.
5. Alice moves to u_{i+1}.
6. Bob moves to any neighbour of u_{k-1}.
7. Alice moves to a potential candidate for u_{i+2}.
8. Bob remembers the response from Alice, then moves back to u_{k-1}.
9. Alice moves back to u_{i+1}.

$6D$ Solution

$6D$ solution:

10. If Alice is closer when she moved to u_{i+2}, Bob moves to v. Otherwise, Bob moves to w.
11. If Bob gives the same response as v, Alice knows she chose the correct node. Otherwise, she chose the wrong one. She then moves to the correct u_{i+2}.
12. Bob moves back to u_{k-1}.

Then, we can repeat from step 1.

$6D$ Solution

The advantage of this solution is that it works as long as u_{k-1} is found. So, we use 2 moves to find u_2 and u_{k-1} trivially. Then, another 4 moves to set up for the $6D$ solution. Then, every 12 moves allow Alice to find 2 nodes. The solution fits well in the $6D$ constraint.

Total number of moves: $6D$
Expected score: 100

Bonus

Question for Chief Judge	M2651 - Introvert Leaving	To whoever presenting edi: my sol uses 5D moves, maybe add that to edi

$5D$ Solution

We have tried:

- Alice and Bob alternately helping each other.
- Alice helping Bob to find his way to her.

Is there any other possible ways for them to communicate? Yes!!!

Let's try to have them **simultaneously help each other**!

$5D$ Solution

Let's see how Alice and Bob can help each other find their next nodes in 10 moves:

1. In each person's first move, move to one of the possible candidates for their next nodes, denote this as v.
2. Each person remembers the response from the other person, then moves back to their original positions.
3. If the response from the last move is closer, the person moves to v again. Otherwise, there is a node w that gives a different response from v (same as the $9D$ solution), move to w.

$5D$ Solution

Let's see how Alice and Bob can help each other find their next nodes in 10 moves:

4. Each person compares the newest response with the response obtained previously. If the responses are equal, it means that they have chosen the correct candidate in their first step. Otherwise, they have chosen the wrong one. They then move back to their original positions.
5. They move to the correct candidate.

Alice and Bob can then use the same strategy to find their next nodes.

$5D$ Solution

5 moves are used for each person to find their next nodes. In total, 10 moves are used to decrease D by 2.

Total number of moves: $5D$
Expected score: 100

Takeaways

- **There are a lot of ways to attempt a task**, some can lead to unexpected breakthrough and ultimately to the full solution. For example, in this task, we can have Alice and Bob alternately helping each other, or have a person guiding another towards the destination, or even simultaneously help each other out.
- **Build on what you have.** Think about what is useful and what can be optimized. For example, the $9D$ solution tells us that if we can find two node v and w such that they give diffferent responses, then we can complete the task. Then the problem becomes 'how to find v and w using as few moves as possible'.