T2613 Triple Keys

Isaac Chan {snowysecret}

2026-05-23

Table of Contents

Background

- Problem Idea & Preparation by `snowysecret`
- Special thanks: `QwertyPi` and `WongChun1234` for discussing improvements to the solutions with me!
- Special thanks: `microtony`, `ethening`, `gasbug` for testing!
- Inspired by *Squid Game: Season 3* (rated 15+, don't watch if you're too young!)

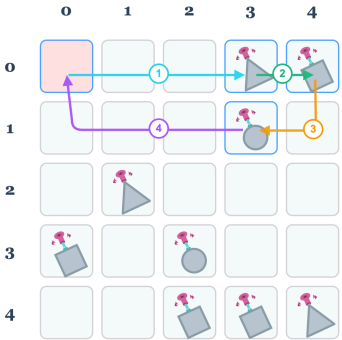
Abridged Statement

- You are given a N by N grid, where some cells have a key of one of the following shapes: C (Circle), S (Square) or T (Triangle).
- Answer Q queries. Each query is of the form: if you start at (R, C), what is the minimum steps needed to take one key of each shape and go back to the starting point?

- Tags: (Simple) Mathematics, Observations, Optimization, Data Structures

Example

```
triple_keys(5, ["...TS", "...C.", ".T...", "S.C..", "..SST"])
minimum_time(0, 0)
```



Subtasks

Subtask	Score	Constraints
1	3	$Q \leq 2500,\ N \leq 50$; for all queries, starting cell is `S` and surrounded by `T` neighbours
2	4	$Q \leq 2500,\ N \leq 50$; there is a single key of each shape, and they all occur in a 2×2 grid
3	14	$N \leq 50$
4	11	$N \leq 200$
5	23	$N \leq 500$
6	27	$N \leq 1000$
7	18	$N \leq 1700$

Statistics

- Maximum score = 32 :((Predicted was 55)

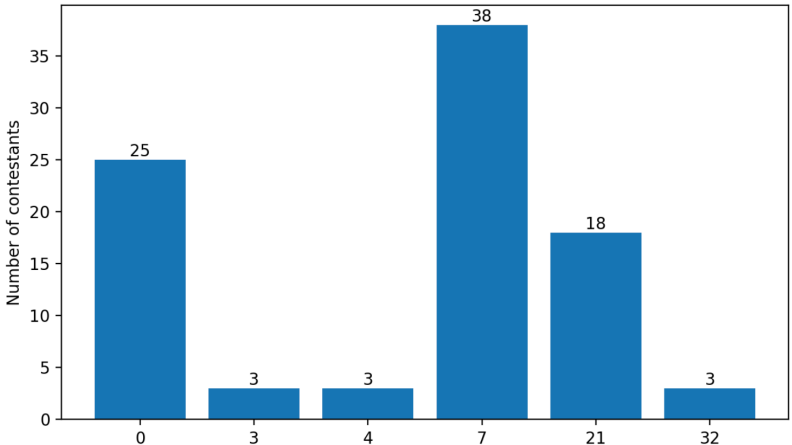


Table of Contents

Subtask 1 (3%): starting cell is `S` + surrounded by `T` neighbours

- In other words, you get a Square and a Triangle "for free" (by just walking at least 1 step – you need to anyway).
- You only need to care about finding a Circle key.
- Luckily, we have $Q \leq 2500, N \leq 50$, so it suffices to exhaust all possible cells, and check whether there's a Circle key there.
- Find the closest Circle key to the starting point.

Time complexity: $O(N^2)$ preprocessing, $O(N^2)$ per query.
Expected score: 3

Subtask 2 (4%): single key of each shape, all occur in 2×2 grid

- Initialise $\text{ans}[r][c] = \infty$ for $0 \le r, c \le N-1$, except the 2×2 grid which contains the three keys, where we initialise $\text{ans}[r][c] = 4$.
- Then do the following *propagation step*: For each pair of neighbouring cells (r_1, c_1) and (r_2, c_2), set $\text{ans}[r_1][c_1] := \min(\text{ans}[r_1][c_1], \text{ans}[r_2][c_2] + 2)$.
- You can first propagate each $\text{ans}[r_1][c_1]$ to $\text{ans}[r_1][c_1 + 1]$ from $c_1 = 0$ to $N-2$ in that order, then from each $\text{ans}[r_1][c_1]$ to $\text{ans}[r_1][c_1 - 1]$ from $c_1 = N-1$ to 1 in that order, and so on for the row dimension.
- The answer to query (r, c) is simply $\text{ans}[r][c]$.

Time complexity: $O(N^2)$ preprocessing, $O(1)$ per query.
Expected score: 4

Propagation Step

```
procedure propagate()
    // horizontal propagation
    for r = 0 to N - 1 do
        for c = N - 2 downto 0 do
            ans[r][c] = min(ans[r][c], ans[r][c + 1] + 2)
        for c = 1 to N - 1 do
            ans[r][c] = min(ans[r][c], ans[r][c - 1] + 2)
    // vertical propagation
    for c = 0 to N - 1 do
        for r = N - 2 downto 0 do
            ans[r][c] = min(ans[r][c], ans[r + 1][c] + 2)
        for r = 1 to N - 1 do
            ans[r][c] = min(ans[r][c], ans[r - 1][c] + 2)
```

So far so good! What's next?

Subtask	Score	Constraints
1	3	$Q \leq 2500,\ N \leq 50$; for all queries, starting cell is S and surrounded by T neighbours
2	4	$Q \leq 2500,\ N \leq 50$; there is a single key of each shape, and they all occur in a 2×2 grid
3	14	$N \leq 50$
4	11	$N \leq 200$
5	23	$N \leq 500$
6	27	$N \leq 1000$
7	18	$N \leq 1700$

- The rest of the subtasks require a **general solution** to the problem!

Table of Contents

Subtask i: $N, Q \leq 5$

- What is the most naive (polynomial time) general solution you can think of?

Subtask i: $N, Q \leq 5$

- What is the most naive (polynomial time) general solution you can think of?
- Recall: You have a starting cell (r, c), and you need to collect three keys one by one.
- We might consider exhausting the three cells to collect keys: $(r_1, c_1), (r_2, c_2), (r_3, c_3)$.
- The path we walk will be $(r, c) \to (r_1, c_1) \to (r_2, c_2) \to (r_3, c_3) \to (r, c)$.

Subtask i: $N, Q \leq 5$

- We might consider exhausting the three cells to collect keys: $(r_1, c_1), (r_2, c_2), (r_3, c_3)$.
- The path we walk will be $(r, c) \to (r_1, c_1) \to (r_2, c_2) \to (r_3, c_3) \to (r, c)$.

Fact

The cost of a path that starts and ends at (r, c) and passes through each of the three cells $(r_1, c_1), (r_2, c_2), (r_3, c_3)$ (in this order) is

$$|r - r_1| + |c - c_1| + |r_1 - r_2| + |c_1 - c_2| + |r_2 - r_3| + |c_2 - c_3| + |r_3 - r| + |c_3 - c|$$

- It remains to check if these three cells actually contain the three keys we want, each exactly once.

Subtask i: $N, Q \leq 5$

- Indeed, exhausting the three cells to collect keys: $(r_1, c_1), (r_2, c_2), (r_3, c_3)$ (followed by simple checking) is a valid algorithm.

Time complexity: $O(N^6)$ per query, $O(1)$ preprocessing.
Expected score: 0 :(
... but does this naive solution give us some insight?

Insight from the $O(N^6)$ algorithm

Fact

The cost of a path that starts and ends at (r, c) and passes through each of the three cells $(r_1, c_1), (r_2, c_2), (r_3, c_3)$ (in this order) is

$$|r - r_1| + |c - c_1| + |r_1 - r_2| + |c_1 - c_2| + |r_2 - r_3| + |c_2 - c_3| + |r_3 - r| + |c_3 - c|$$

- We can actually reduce the cost of this path just by redesigning the **ordering** of which you obtain the three keys.
- e.g. Maybe it is better for you to take the key from (r_2, c_2) first, then the key from (r_3, c_3), then finally the key from (r_1, c_1).
- What is the optimal ordering of obtaining the three keys though?

Insight from the $O(N^6)$ algorithm

Definition

Given the starting/ending cell (r, c) and the cells of which the three keys are located – $(r_1, c_1), (r_2, c_2), (r_3, c_3)$, we define

$$f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$$

to be the **minimum cost** of a path that starts and ends at (r, c), and passes through each of these three cells (**not necessarily** in this order).

- Our next goal: find a "useful" expression for $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$.

Finding an expression for $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$

$$|r - r_1| + |c - c_1| + |r_1 - r_2| + |c_1 - c_2| + |r_2 - r_3| + |c_2 - c_3| + |r_3 - r| + |c_3 - c|$$

$$= |r - r_1| + |r_1 - r_2| + |r_2 - r_3| + |r_3 - r| + |c - c_1| + |c_1 - c_2| + |c_2 - c_3| + |c_3 - c|$$

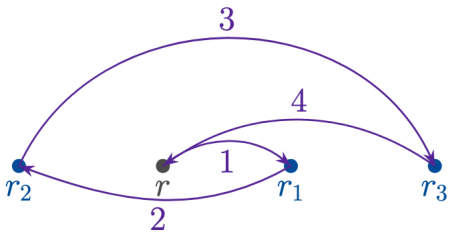
- After separating the r and c components, it is easy to see that they look basically the same.
- Let's focus on the r dimension to make things simple.

$$|r - r_1| + |r_1 - r_2| + |r_2 - r_3| + |r_3 - r|$$

What is the smallest value that this expression can take?

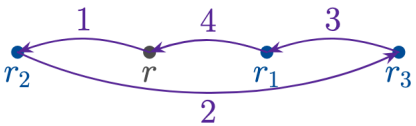
Finding an expression for $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$

Not minimum



$$2|r_2 - r| + 4|r - r_1| + 2|r_1 - r_3|$$

Minimum



$$\begin{aligned} &2|r_2 - r| + 2|r - r_1| + 2|r_1 - r_3| \\ &= 2|r_2 - r_3| \\ &= 2(\max(r) - \min(r)). \end{aligned}$$

Note: Numbers 1, 2, 3, 4 indicate the order of which the directed edge is taken

Finding an expression for $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$

- **Claim:** Under an appropriate rearrangement of r_1, r_2 and r_3, the expression

$$|r - r_1| + |r_1 - r_2| + |r_2 - r_3| + |r_3 - r|$$

has smallest value = $2(\max(r) - \min(r))$, where

$$\max(r) = \max(\{r, r_1, r_2, r_3\}) \qquad \min(r) = \min(\{r, r_1, r_2, r_3\})$$

- Similarly, under an appropriate rearrangement of c_1, c_2 and c_3, the expression

$$|c - c_1| + |c_1 - c_2| + |c_2 - c_3| + |c_3 - c|$$

has smallest value = $2(\max(c) - \min(c))$.

Finding an expression for $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3)$

- We claim the r-component is as small as $2(\max(r) - \min(r))$.
- Similarly, the c-component is as small as $2(\max(c) - \min(c))$.

Naturally...

Lemma

$f(r, c, r_1, c_1, r_2, c_2, r_3, c_3) = 2(\max(r) - \min(r) + \max(c) - \min(c))$.

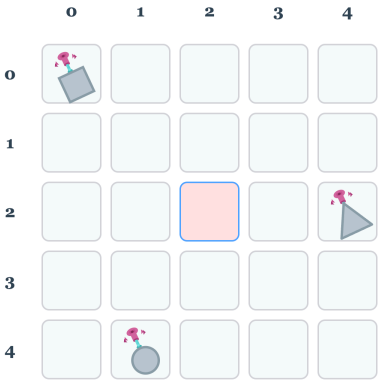
- What is the geometric meaning of this formula?

$$f(r, c, r_1, c_1, r_2, c_2, r_3, c_3) = 2(\max(r) - \min(r) + \max(c) - \min(c))$$

- What is the geometric meaning of $2(\max(r) - \min(r) + \max(c) - \min(c))$?
- It is the perimeter of a rectangle!
- More precisely, here it is the perimeter of the “tightest bounding box” that includes all four “important cells”.

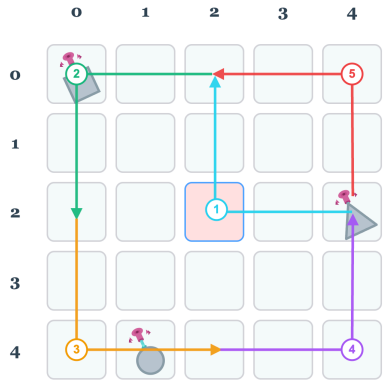
$$f(r, c, r_1, c_1, r_2, c_2, r_3, c_3) = 2(\max(r) - \min(r) + \max(c) - \min(c))$$

- Here, the “tightest bounding box” is the entire grid.



$$f(r, c, r_1, c_1, r_2, c_2, r_3, c_3) = 2(\max(r) - \min(r) + \max(c) - \min(c))$$

- Pick a quadrant Q (relative to the starting point) that has no key at all.
- We have 3 keys but 4 quadrants to choose from, so there must exist some valid quadrant Q.
- Then we **fold that quadrant's path inwards** so it touches the starting point.



The Story So Far

- $f(r, c, r_1, c_1, r_2, c_2, r_3, c_3) = 2(\max(r) - \min(r) + \max(c) - \min(c))$.
- That is: For a given choice of three keys and a starting point, the minimum cost to take exactly those three keys and go back to the starting point is the perimeter of the tightest bounding box that includes all the four cells.
- Next step: We drop the idea of "exhausting 3 cells for keys", and instead we focus on the *bounding boxes* of the 3 (+1) cells.
- This is because of dealing with subgrids of the grid is computationally a lot more efficient than dealing with triples of cells.

The Story So Far

- We will start from an algorithm with $O(N^5)$ preprocessing cost that exhausts bounding boxes (rectangles) and precomputes the array ans$[r][c]$ for all $0 \leq r, c \leq N-1$. We aim to just lookup the array ans$[r][c]$ for a query (r, c), achieving $O(1)$ time per query.
- Then, we will gradually optimize the preprocessing step to $O(N^3)$.

- For the rest of this slide deck, we use the notation (r_1, c_1, r_2, c_2) to denote the rectangle with top-left corner (r_1, c_1) and bottom-right corner (r_2, c_2).

Colourful Rectangles

Definition

A rectangle is **colourful** if it contains at least one `S`, one `C`, and one `T`.

- Build three 2D prefix sums: one for each of `S`, `C`, `T`.
- Then we can check whether any rectangle is colourful in $O(1)$.

If a rectangle with top-left corner (r_1, c_1) and bottom-right corner (r_2, c_2) is **colourful** then we can perform the following **update step**:

- For all (r, c) with $r_1 \leq r \leq r_2$ and $c_1 \leq c \leq c_2$,

$$\text{ans}[r][c] := \min(\text{ans}[r][c], 2(r_2 - r_1 + c_2 - c_1))$$

Subtask 3 (14%): $N \leq 50$

- Fix the top and bottom rows r_1, r_2.
- Reduce the problem to a 1D interval problem over columns.
- Use two pointers to find minimal colourful intervals.
- Apply the rectangle update.

```
1  for r1 = 0 to N - 1 do
2    for r2 = r1 to N - 1 do
3      c2 = -1
4      for c1 = 0 to N - 1 do
5        while c2 + 1 < N and
6              not colourful(r1,c1,r2,c2) do
7          c2 = c2 + 1
8        if colourful(r1,c1,r2,c2) then
9          for r = r1 to r2 do
10           for c = c1 to c2 do
11             ans[r][c] = min(ans[r][c],
12                             2(r2-r1+c2-c1))
```

Propagation Step

- So far, ans$[r][c]$ is good for cells whose answer is captured by some colourful rectangle.
- But this is not always the case for all cells (e.g. a whole column without a single key).
- In this case, we can walk to the rectangle, complete the route, and walk back.
- The propagation step is similar to that of subtask 2.

Subtask 3 (14%): $N \leq 50$

- Time complexity: $O(N^3)$ colourful rectangles, $O(N^2)$ update per rectangle.
- So $O(N^5)$ preprocessing, and $O(1)$ per query.
- Expected score: $3 + 4 + 14 = 21$.

Subtask 4 (11%): $N \leq 200$

- Fix a row band $[r_1, r_2]$ (as before).
- Use two pointers to find colourful column intervals (as before).
- For each column c, maintain `best[c]` – the minimum interval width covering c.

Subtask 4 (11%): $N \leq 200$

```
for r1 = 0 to N - 1 do
    for r2 = r1 to N - 1 do
        best[c] = INF for all columns c
        c2 = -1
        for c1 = 0 to N - 1 do
            while c2 + 1 < N and not colourful(r1, c1, r2, c2) do
                c2 = c2 + 1
            if colourful(r1, c1, r2, c2) then
                for c = c1 to c2 do
                    best[c] = min(best[c], c2-c1)   // Avoids naive update
        for r = r1 to r2 do
            for c = 0 to N - 1 do
                ans[r][c] = min(ans[r][c], 2(r2-r1+best[c]))
```

Subtask 4 (11%): $N \leq 200$

- We try at most $O(N^3)$ colourful rectangles. Each rectangle's update takes $O(N)$ in the worst case.
- This gives an overall complexity of $O(N^4)$.
- Expected score: $3 + 4 + 14 + 11 = 32$.

Subtask 5 (23%): $N \leq 500$

For a fixed row band $[r_1, r_2]$, the $O(N^4)$ bottlenecks are:

1. For each colourful rectangle (r_1, c_1, r_2, c_2), update $\texttt{best}[c]$ for all $c \in [c_1, c_2]$ naively.

$$\texttt{for } c = c_1 \ldots c_2 : \quad \texttt{best}[c] := \min(\texttt{best}[c], c_2 - c_1)$$

2. Updating all rows $r_1..r_2$ naively **for each column** c after two-pointers:

$$\texttt{for } r = r_1 \ldots r_2 : \quad \texttt{ans}[r][c] := \min(\texttt{ans}[r][c], \ldots)$$

We optimise each of the two bottlenecks in turn.

Optimization 1: Min-Queue over Active Intervals

Consider sweeping through the columns left to right, for a specific row band $[r_1, r_2]$. From the two-pointers procedure we obtain a list of *intervals* $[c_1, c_2]$, such that each (r_1, c_1, r_2, c_2) is colourful.

- At column c_1, the interval $[c_1, c_2]$ can be made *active*;
- at column $c_2 + 1$, the interval $[c_1, c_2]$ can be made *inactive*.
- While sweeping $c = 0..N-1$, we keep track of the minimum $c_2 - c_1$ among active intervals. This gives the value of $\texttt{best}[c]$.

This can be maintained using a queue supporting: `push`, `pop`, `getmin`. The standard solution to this problem uses two stacks (MinQueue).

So the whole $\texttt{best}[c]$ array can be computed in $O(N)$ **per row band**.

Optimization 2: Avoid Updating Every Row

Suppose the outer loop fixes r_1.

- For each r_2, after computing $\texttt{best}[c]$, update only:

$$\text{ans}[r_2][c] := \min(\text{ans}[r_2][c], 2((r_2 - r_1) + \texttt{best}[c])).$$

- After all r_2 are processed, propagate upwards:

$$\text{ans}[r][c] := \min(\text{ans}[r][c], \text{ans}[r+1][c])$$

for $r = N-2, N-3, \ldots, r_1$.

Why this works: if a rectangle covers row r, it also covers every row between r_1 and its bottom row.

Subtask 5 (23%): $N \leq 500$

Putting the two optimizations together:

- There are $O(N^2)$ row bands $[r_1, r_2]$.
- Each row band is processed in $O(N)$.
- Final propagation over the grid is $O(N^2)$.
- Time complexity: $O(N^3)$ preprocessing, $O(1)$ per query.
- Expected score: $3 + 4 + 14 + 11 + 23 = 55$.

Table of Contents

Key Idea: Only Minimal Rectangles Matter

A colourful rectangle is **minimal** if deleting any one of its four sides makes it no longer colourful.

- If a colourful rectangle is not minimal, shrink it until it is.
- A non-minimal rectangle never helps our final answer.
- That is because we have the final propagation step, which effectively says: "If the best path for (R, C) is not represented by a **minimal** colourful rectangle, just walk to one."
- Therefore, it is enough to enumerate all minimal colourful rectangles (how many are there?), initialise their interiors efficiently, then do the propagation step.

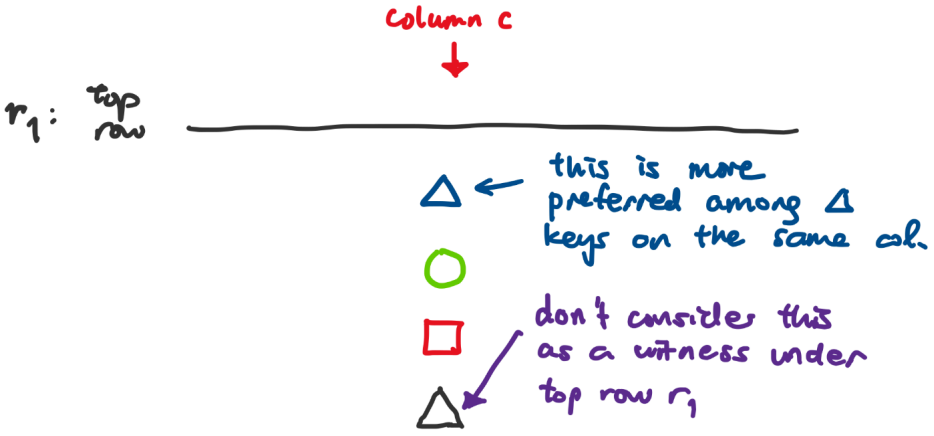
How Many Minimal Rectangles Are There?

Lemma

There are only $O(N^2)$ minimal colourful rectangles.

- Fix the top row r_1.
- Look for cells (r, c), where $r \geq r_1$ that may become the **bottom-side witness** of a minimal rectangle.
- For each column and each key type (square/triangle/circle), only the **first occurrence at or below** r_1 can be such a witness.
- Hence, for this fixed r_1, there are at most $3N$ witness cells.

$3N$ Witness Cells: Visual



$O(N^2)$ Minimal Rectangles

Fix a top row r_1 and a witness key at (r, c).

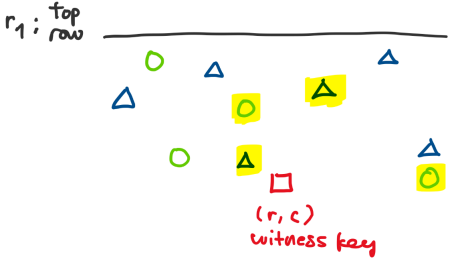
- The witness gives one of the three key types.
- The rectangle still needs the other two key types.
- Maintain the set of columns currently containing each key type in the row band $[r_1, r]$.
- For each of the other two key types, only the closest column on the left/right of c can define a minimal left/right edge.

So each witness generates only $O(1)$ candidate minimal rectangles.

$$N \text{ choices of } r_1 \times 3N \text{ witnesses} \times O(1) = O(N^2).$$

Naive analysis gives $12N^2$ minimal rectangles, closer analysis gives $6N^2$ (some of the candidates are strictly contained inside some others).

$O(N^2)$ Minimal Rectangles: Visual



Find: closest △ to the left/right
closest ○ to the left/right

For each r_1 : 3N witnesses × 2 choices for △
×2 choices for ○ = 12N minimal rects.

Total: $12N^2$ minimal rects (closer analysis gives $6N^2$)

Enumerating Minimal Rectangles

```
for r1 = 0 to N - 1 do
    clear sets col[3]
    prepare events by bottom row
    for r2 = r1 to N - 1 do
        add new witness columns into col[key]
        for each new witness (c, key) on row r2 do
            a = next/prev columns of key + 1 around c
            b = next/prev columns of key + 2 around c
            for O(1) pairs (a, b) do
                c1 = min(a, b, c)
                c2 = max(a, b, c)
                if rectangle is minimal then
                    add rectangle (r1, c1, r2, c2)
```

2D range chmin with $O(N^2)$ rectangles

We now have only $O(N^2)$ minimal rectangles to consider.

- Each rectangle gives a range update:

$$\text{ans}[r][c] := \min(\text{ans}[r][c], 2(r_2 - r_1 + c_2 - c_1))$$

  for all (r, c) satisfying $r_1 \le r \le r_2, c_1 \le c \le c_2$.
- This is a standard **2D "range chmin"** problem.
- A direct 2D segment tree approach gives about $O(N^2 \log^2 N + Q)$. Due to its large constant factor, unfortunately it does not give us any more points this time.

Solution 1: Segment Tree Approach

- Build a segment tree over the **columns**.
- Each segment tree node represents a range of columns $[c_1, c_2]$.
- Each segment tree node contains an array `best[r]`, which represents the best answer for cells (r, c) captured by this node, where the answer must be applicable to all of $c_1 \leq c \leq c_2$.

Solution 1: Segment Tree Approach

- We process the minimal rectangles in ascending $d = r_2 - r_1 + c_2 - c_1$.
- For each rectangle (r_1, c_1, r_2, c_2), for each of the $O(\log N)$ segment-tree nodes covering its column interval $[c_1, c_2]$, we perform an **update** that is equivalent to the following:

```
1 // Happens in a set of segment tree nodes that cover columns [c1, c2]
2 for r = r1 to r2 do
3   best[r] = min(best[r], 2 * d) // d = r2 - r1 + c2 - c1
```

- However, since we process all minimal rectangles in increasing d, the first value assigned to `best[r]` is always optimal!
- Fast-forwarding: instead of iterating over all the rows from r_1 to r_2 one-by-one, always jump to the next row that has not been updated yet.

Fast-Forwarding with DSU

```
1 procedure place_update(node, r1, r2, d)
2     r = node.next_unset(r1)
3     while r <= r2 do
4         node.val[r] = 2 * d
5         node.delete(r)       // skip this row next time
6         r = node.next_unset(r)
```

- `next_unset(r)` returns the first row at least r that has not yet been assigned in this node.
- `delete(r)` removes row r from the DSU linked list.
- Each row is deleted at most once per segment-tree node.

Solution 1: Segment Tree Approach

Time complexity analysis:

- Number of minimal rectangles: $O(N^2)$.
- Each rectangle triggers updates on $O(\log N)$ segment tree nodes (that collectively cover the columns $[c_1, c_2]$).
- For each update, we perform at least 1 DSU query (`next_unset(r)`).
- Hence, the total time complexity is $O(N^2 \log N\alpha(N))$.
- Expected score: $82 - 100$ (depends on constant factor, whether you used $12N^2$ or $6N^2$ minimal rectangles, etc).

Solution 2 by `QwertyPi` (Faster!)

- Same as before, we first find all $O(N^2)$ minimal rectangles, and aim to process them by increasing d.
- For each rectangle, we update its top edge (i.e. the cells (r_1, c_1) to (r_1, c_2)) and its left edge (i.e. the cells (r_1, c_1) to (r_2, c_1)), sped up by DSUs similar to Solution 1.
- Then, we perform the `propagate()` step once first.
- At this point, for each minimal rectangle (r_1, c_1, r_2, c_2), we have effectively assigned $\text{ans}[r][c] := \min(\text{ans}[r][c], 2(r_2 - r_1 + c_2 - c_1))$ for all cells on the top and/or left edges of the rectangle.
- But the remaining cells inside this rectangle might only see an answer that is propagated from these edges, rather than an answer that equals $2(r_2 - r_1 + c_2 - c_1)$ itself.

Solution 2 by `QwertyPi` (Faster!)

- Now let's say we sweep the cells once more in increasing r followed by increasing c.
- For a specific (r, c), say the answers $\text{ans}[r-1][c]$ and $\text{ans}[r][c-1]$ are already correct. Then even though $\text{ans}[r][c]$ might not be "correct" yet, it is restricted to the following values: $\text{ans}[r][c]$ itself, also $\text{ans}[r-1][c]+2, \text{ans}[r][c-1]+2, \text{ans}[r-1][c], \text{ans}[r][c-1]$.
- In other words, we perform $\text{ans}[r][c] := \min(\{\text{ans}[r][c], \text{ans}[r-1][c]+2, \text{ans}[r][c-1]+2\})$, then finally check explicitly if we can further decrease this answer by 2.
- It suffices to check whether there exists a minimal rectangle containing (r, c) with some perimeter $2k$. This can be done by maintaining a separate binary-indexed tree (BIT) for each value $k = 0, 1, \ldots, 2N$ (details left as exercise).

Solution 2 by `QwertyPi` (Faster!)

- In short, we update the top and left edges first, perform `propagate()` once, then perform a top-down left-to-right sweep again to update the top and left edge information to the rest of each subrectangle.
- In fact, this alternative solution achieves a complexity of $O(N^2 \log N + Q)$, since its DSU step is independent of any log-data structures.
- This solution has a better constant factor (avoids $\alpha(N)$ factor and allows use of BIT instead of segment tree) and is almost sure to obtain 100 points.

Table of Contents

Conclusion

1. Think about tightest **bounding boxes (rectangles)** rather than paths.
2. Only $O(N^2)$ **minimal rectangles** matter.
3. Update efficiently and propagate the answers to the whole grid.

Fun Facts

- The problem was originally proposed to HKOI 2025/26 Senior, with $O(N^3)$ as the full solution. (Would you rather this problem appear in HKOI Final or TFT?)
- Lots of effort was made (I mean it) to decide the optimal combination of TL, maximum N, and subtask constraints. Just a few days ago, we changed the constraints from $N \leq 1600$ to $N \leq 1700$ because there exists a $O(N^3)$ solution (which makes use of the final observation) that passes tests up to $N \leq 1600$...

Thank you!

Good luck in TFT Day 2!