SOLUTION

An approach for 40 points asks a query for each rectangle with an upper-left corner located in cell $(1,1)$. Now, using the inclusion-exclusion principle, we can easily calculate whether a cell (r,c) contains treasure (see Figure 1 below):

$$treasure[r][c] = response(1,1,r,c) - response(1,1,r-1,c) - response(1,1,r,c-1) + response(1,1,r-1,c-1)$$

The special case, when the cell (r,c) is in the first row or first column is even easier.

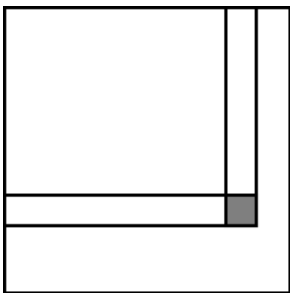


Figure 1

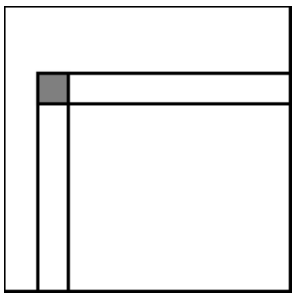


Figure 2

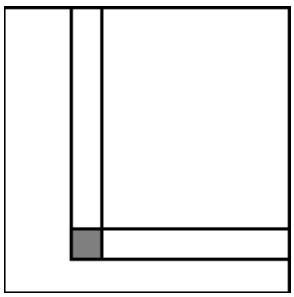


Figure 3

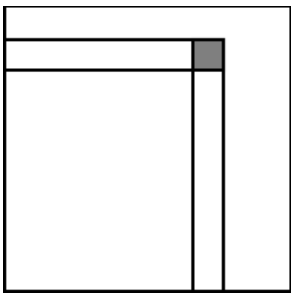


Figure 4

For the cells (r,c) belonging to the upper-left quarter of the island, that approach asks queries for some very small rectangles, which is expensive. For these cells, instead of asking for the rectangles with a corner located in $(1,1)$, we can ask for the rectangles with a corner located in (N,N) and then use a similar formula (see Figure 2 above):

$$treasure[r][c] = response(r,c,N,N) - response(r,c+1,N,N) - response(r+1,c,N,N) + response(r+1,c+1,N,N).$$

Analogously, for (r,c) in the upper-right quarter of the island, we can ask for the rectangles with a corner located in $(N,1)$. For (r,c) in the lower-left quarter of the island, we can ask for the rectangles with a corner located in $(1,N)$ (see Figures 3 and 4 above).

The algorithm, therefore, does the following: for every cell (r,c), find the largest among four rectangles beginning at (r,c) and ending in a corner of the island, and ask the oracle for this rectangle. Finally, calculate the values of all cells using the formulas as above.

You might notice that, in this approach, tricky situations appear for the cells in the middle row/column if N is odd or two middle rows/columns if N is even. In these situations, we do not have responses for some of the rectangles appearing in the formulas above (because they are a bit smaller), but we can calculate their values using other, larger rectangles that we do have responses for. These observations make a difference between 80% and 100% of the points.

M1842 - Another RMQ

Alex Tung
alex20030190@yahoo.com.hk

28 April 2018

Solutions

There are many solutions to this problem.

- Solution 1: Mo's algorithm
  - $O(Q \log Q + N\sqrt{N} \log RANGE + Q \log RANGE)$, or
  - $O(Q \log Q + N\sqrt{N} + Q\sqrt{RANGE})$

Solutions

There are many solutions to this problem.

- Solution 1: Mo's algorithm
  - $O(Q \log Q + N\sqrt{N} \log RANGE + Q \log RANGE)$, or
  - $O(Q \log Q + N\sqrt{N} + Q\sqrt{RANGE})$
- Solution 2: Partition tree (Link; content in Chinese)
  - $O((N + Q) \log N)$

Solutions

There are many solutions to this problem.

- Solution 1: Mo's algorithm
  - $O(Q \log Q + N\sqrt{N} \log RANGE + Q \log RANGE)$, or
  - $O(Q \log Q + N\sqrt{N} + Q\sqrt{RANGE})$
- Solution 2: Partition tree (Link; content in Chinese)
  - $O((N + Q) \log N)$
- Below we will describe two solutions with two logs in time complexity.

Offline solution: Parallel Binary Search

- Suppose all queries are of the form (L_i, R_i, V_i): Is the median of $A[L_i], \ldots, A[R_i] \leq V_i$?
- Parallel binary search: If we can handle **all** (L_i, R_i, V_i) queries in $O(X)$ time, then we can answer the original queries in $O(X \log RANGE)$ time, where $RANGE$ is the range of answer.

Offline solution: Parallel Binary Search

- Suppose all queries are of the form (L_i, R_i, V_i): Is the median of $A[L_i], \ldots, A[R_i] \leq V_i$?
- Parallel binary search: If we can handle **all** (L_i, R_i, V_i) queries in $O(X)$ time, then we can answer the original queries in $O(X \log RANGE)$ time, where $RANGE$ is the range of answer.
- Target: answer all (L_i, R_i, V_i) queries quickly.

Offline solution: Parallel Binary Search

- To do so, process updates $(UPDATE, i, A[i])$ and queries $(QUERY, L_i, R_i, V_i)$ together.
- Sort them in ascending order of $A[i]$ or V_i, then process one by one.

Offline solution: Parallel Binary Search

- To do so, process updates $(UPDATE, i, A[i])$ and queries $(QUERY, L_i, R_i, V_i)$ together.
- Sort them in ascending order of $A[i]$ or V_i, then process one by one.
- Maintain a segment tree or binary-indexed tree.
  - $(UPDATE, i, A[i])$: add 1 to position i.
  - $(QUERY, L_i, R_i, V_i)$: query range sum in $[L_i, R_i]$; check if sum $\geq \frac{R_i - L_i}{2} + 1$.

Offline solution: Parallel Binary Search

- To do so, process updates $(UPDATE, i, A[i])$ and queries $(QUERY, L_i, R_i, V_i)$ together.
- Sort them in ascending order of $A[i]$ or V_i, then process one by one.
- Maintain a segment tree or binary-indexed tree.
  - $(UPDATE, i, A[i])$: add 1 to position i.
  - $(QUERY, L_i, R_i, V_i)$: query range sum in $[L_i, R_i]$; check if sum $\geq \frac{R_i - L_i}{2} + 1$.
- Time complexity for this subroutine: $O((N + Q)\log(N + Q))$.

Offline solution: Parallel Binary Search

- To complete the solution, for each query we maintain $(ansL_i, ansR_i)$: answer for the i-th query must be in range $[ansL_i, ansR_i]$.
- Set $V_i := \frac{ansL_i + ansR_i}{2}$, then run the subroutine.
- Afterwards, update the bounds accordingly.

Offline solution: Parallel Binary Search

- To complete the solution, for each query we maintain $(ansL_i, ansR_i)$: answer for the i-th query must be in range $[ansL_i, ansR_i]$.
- Set $V_i := \frac{ansL_i + ansR_i}{2}$, then run the subroutine.
- Afterwards, update the bounds accordingly.
- Overall time complexity: $O((N + Q) \log(N + Q) \log RANGE)$.

Online solution: Segment Tree of Indices

- Build a segment tree on $[1, RANGE]$. Each node is an array (`std::vector` is useful).
- For $1 \leq i \leq N$, add index i to the $O(\log RANGE)$ nodes whose range contains $A[i]$.
- For example, $RANGE = 8$, $A[2] = 5$, then add 2 to $[1, 8], [5, 8], [5, 6], [5, 5]$.

Online solution: Segment Tree of Indices

- Build a segment tree on $[1, RANGE]$. Each node is an array (`std::vector` is useful).
- For $1 \leq i \leq N$, add index i to the $O(\log RANGE)$ nodes whose range contains $A[i]$.
- For example, $RANGE = 8$, $A[2] = 5$, then add 2 to $[1, 8], [5, 8], [5, 6], [5, 5]$.
- Note that all arrays in the segment tree are sorted.

Online solution: Segment Tree of Indices

- Build a segment tree on $[1, RANGE]$. Each node is an array (`std::vector` is useful).
- For $1 \leq i \leq N$, add index i to the $O(\log RANGE)$ nodes whose range contains $A[i]$.
- For example, $RANGE = 8$, $A[2] = 5$, then add 2 to $[1, 8], [5, 8], [5, 6], [5, 5]$.
- Note that all arrays in the segment tree are sorted.
- This tree can help us do *binary search on answer* in $O(\log N \log RANGE)$ per query.
- Overall time complexity: $O(N \log RANGE + Q \log N \log RANGE)$.

M1843 Team Contest

Alex Tung
alex20030190@yahoo.com.hk

28 April 2018

A Brief Description of Partial Solutions

- $N = 4$: exhaust order of sending programmers to the first four judges
- All t_j equal: pick the four best programmers; send in the order $1, 2, 3, 4, 1, \ldots$
- $M \leq 8$: pick the eight best programmers; try all possibilities

Observation

To solve the problem in full, one needs a critical observation.

Intuition

We only need to consider the top X programmers for some *small* X.

Observation

To solve the problem in full, one needs a critical observation.

Intuition

We only need to consider the top X programmers for some *small* X.

Observation

$X = 7$.

Observation

To solve the problem in full, one needs a critical observation.

Intuition

We only need to consider the top X programmers for some *small* X.

Observation

$X = 7$.

Proof

Suppose a programmer other than the top seven is sent for round j.
We can replace him/her by one of the top seven, without affecting other rounds.
For instance, just pick one who is not sent for rounds $j-3$, $j-2$, $j-1$, $j+1$, $j+2$, $j+3$.

Observation

To solve the problem in full, one needs a critical observation.

Intuition

We only need to consider the top X programmers for some *small* X.

Observation

$X = 7$.

Proof

Suppose a programmer other than the top seven is sent for round j.
We can replace him/her by one of the top seven, without affecting other rounds.
For instance, just pick one who is not sent for rounds $j-3$, $j-2$, $j-1$, $j+1$, $j+2$, $j+3$.

Exercise: find a case to show that $X = 6$ does not work.

Solution

- DP, of course.
- For simplicity, suppose that only the top $N' := min(N, 7)$ programmers remain.
- Let $dp[j][x][y][z] :=$ max. score after round j, if:
  - Programmer x is sent for round j
  - Programmer y is sent for round $j-1$
  - Programmer z is sent for round $j-2$
- Skip if $x = y$ or $y = z$ or $x = z$.
- Transition formula: $dp[j][x][y][z] = max_{p \neq x,y,z}(dp[j-1][y][z][p] +$ score of sending programmer x to judge $j)$.
- To track the optimal sequence of sending programmers, store also the index p that maximizes R.H.S..
- Time complexity: $O(MN'^4)$, where $N' \leq 7$.

Spiderman and lamp posts

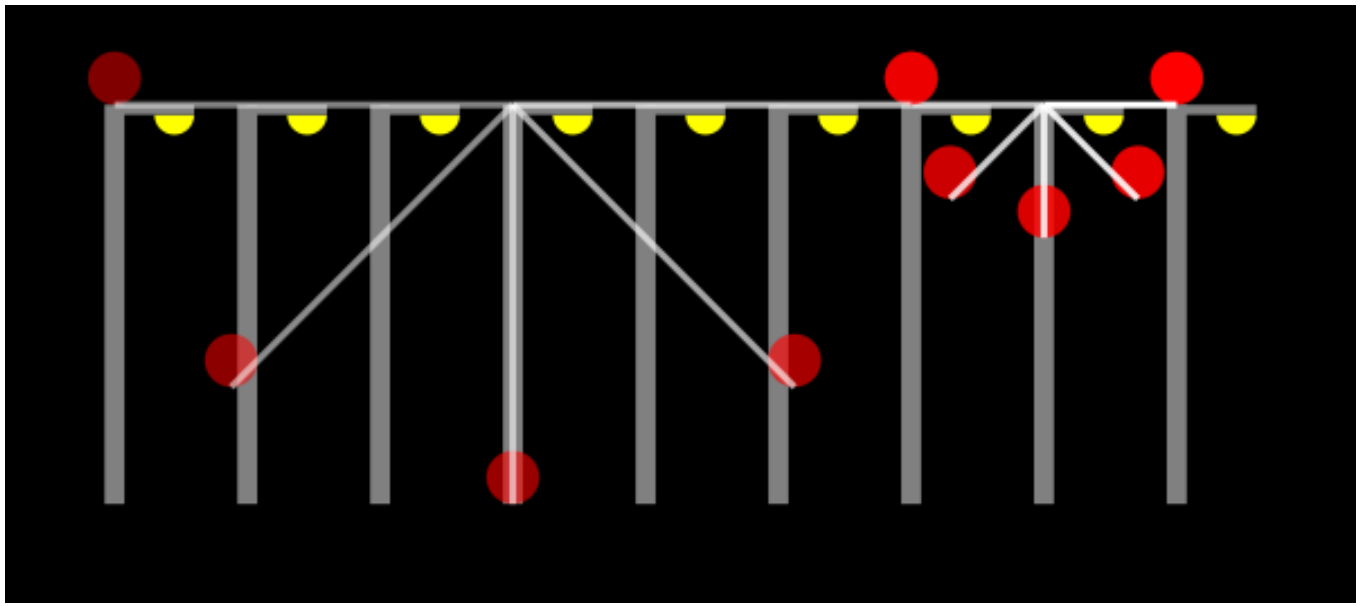
solution

Lau Chi Yung

2018/04/28

Observation

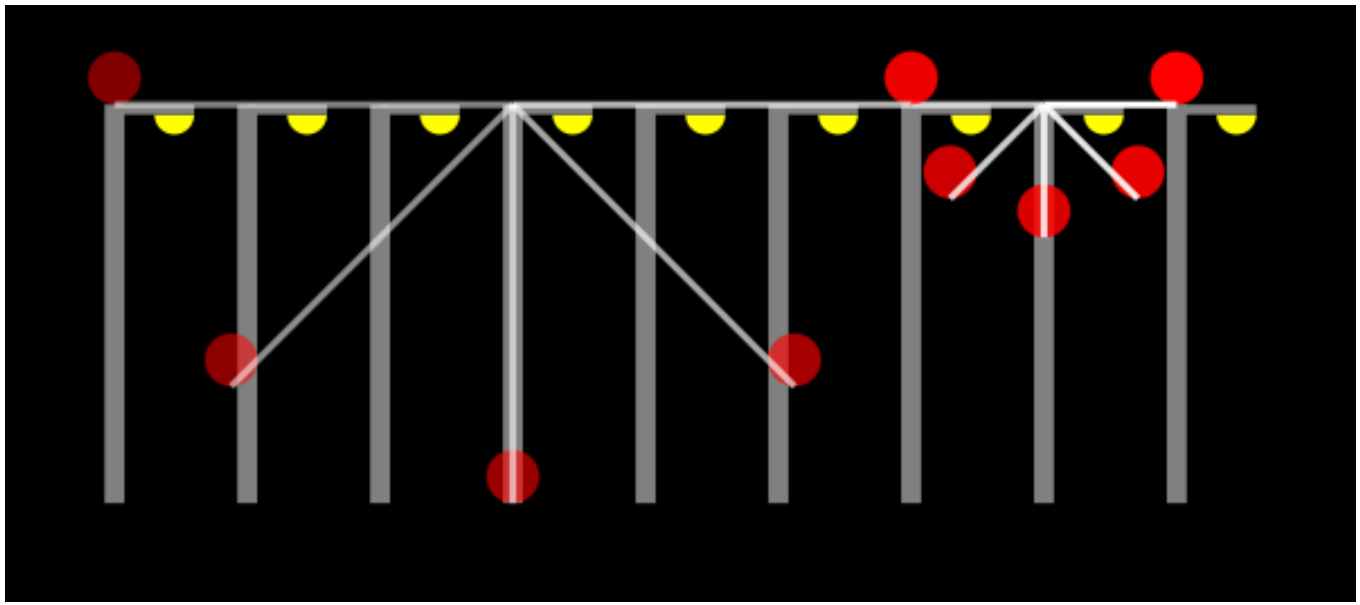
- Spiderman always swings at H_1m above ground
- Spiderman can never reach the n^{th} lamp post if n is even



Subtask 1

- All lamp posts are of the same heights
- Spiderman can fly 2 to $2H_1$m forward in one swing
- Spiderman needs to fly $(N-1)$m forward
- If $N-1$ is odd, output -1;
- otherwise output

$$\lceil \frac{N-1}{2H_1} \rceil$$



Subtask 2,3,4

- Should always attach spider silk to the tip of a lamp post
- Reachable from lamp post $i-j-j$ to i?

$$\text{reachable}(i-j-j,i) = \begin{cases} \text{true} & \text{if } j^2 + |H_1 - H_{i-j}|^2 \leq H_{i-j}^2 \\ \text{false} & \text{otherwise} \end{cases}$$

- dp_i = minimum number of swings to reach lamp post i
- $$dp_i = \min_{\substack{j\in[1,100] \\ \text{reachable}(i-j-j,i)}} \{dp_{i-j-j}+1\}$$
- Time complexity: $O(100N)$
- No need to check dp states > 200m backward because they must not be reachable
- Subtask 2, 3 are for those who do not know this fact

Subtask 2,3,4 (alternative)

- Breath-first search
- Each lamp post is a node
- Each lamp post may swing to 100 other lamp posts
- N nodes, $100N$ edges
- Time complexity: $O(100N)$

Mistake

- A trainee pointed out a mistake in this task after the contest
- If Spiderman is allowed to swing backward,
  - the dp solution is no longer correct
  - with slight modification, BFS can still work
- The problem statement is now updated to restrict swinging backward