2024 1st Half Chonnam Nat'l Univ PIMM Algorithm Party

Solutions Commentary Editorial

PIMM Algorithm Study

	Problem	Difficulty	Author
A	Clock Tower	**Bronze**	Mingyu Ko jjkmk1013
B	Guessing the Song using the Intro	**Bronze**	Yoonsu Lee lys9546
C	Rotating Sequence and Query	**Silver**	Jeonghwan Choi jh01533
D	Escaping Hexagontiles	**Silver**	Yoonsu Lee lys9546
E	Retiring from Retirement	**Gold**	Jonghyeon Park belline0124
F	Mighty Fine Morning	**Gold**	Geunseong Kim onsbtyd
G	Move or Block!	**Gold**	Jeonghwan Choi jh01533
H	Guardians of the Forest	**Gold**	Yeongdo Jeong 0do
I	Three Number XOR and Query	**Platinum**	Jeonghwan Choi jh01533

A. Clock Tower

`#implementation`

Difficulty — **Easy**

- Author: Mingyu Ko[jjkmk1013]
- 219 Submitted, 180 Passed
  AC Ratio: 82.648%
- First Solver: `ychangseok`, +1min

A. Clock Tower

- You can solve it by dividing the equation by 30 minutes of the changed clock tower, where the speed of the minute hand changes.
- For the correct time minutes $f(x)$, the minute the minute the changed clock tower points to x is as below.
- $f(x) = \begin{cases} \frac{1}{2}x & (x \leq 30) \\ \frac{3}{2}x + 15 & (x > 30) \end{cases}$
- Simulations of calculating the actual time are also possible, adding 0 minutes to 1 for the time of the changed clock tower.

B. Guessing the Song using the Intro

`#implementation`, `#brute_force`

Difficulty — **Bronze**

- Author: Yoonsu Lee`lys9546`
- 192 Submitted, 158 Passed
  AC Ratio: 83.333%
- First Solver: `nflight11`, +2min

B. Guessing the Song using the Intro

- You can solve the equation by dividing it based on the 30−minute point of the changed clock tower, where the speed of the minute hand changes.
- For the correct minute $f(x)$, if you set up the equation with the minute x that the minute hand of the changed clock tower points to, it is as follows.
- $f(x) = \begin{cases} \frac{1}{2}x & (x \leq 30) \\ \frac{3}{2}x + 15 & (x > 30) \end{cases}$
- You can also simulate calculating the actual time by adding 1 from 0 minutes for the time of the changed clock tower.

C. Rotating Sequence and Query

`#prefix_sum`, `#implementation`

Difficulty — **Silver**

- Author: Jeongwhan Choi `jh01533`
- 288 Submitted, 74 Passed
  AC Ratio: 29.514%
- First Solver: `mj1000j`, +3min

C. Rotating Sequence and Query

- Let’s think about how to rotating the sequence to the right by k.
- If we simply use the deque to remove the last element and put it in the front by repeating the task k times, the time complexity will be $O(NQ)$, and the time will exceed.
- Is there a way to achieve the same effect as rotating without actually rotating it?

C. Rotating Sequence and Query

- Let's think of the sequence as a raw sequence. When the raw sequence is rotated, the distance moved by all elements is the same and the left-right relationship between elements is always constant.
- In other words, if we only know the unrotated sequence and the distance moved by one element, we can know the current sequence.
- Let's say the moved distance is D. (Initially, $D = 0$)
- When query 1 comes in, it rotates to the right by k, so it's the same as doing $D += k$.
- When query 2 comes in, it rotates to the left by k, so it's the same as doing $D -= k$.

C. Rotating Sequence and Query

- Let's process query 3 using the above information.
- Recalculate input a, b using D. For convenience, we use `0-based` (indexing from 0).
- $x = (a - 1 - D) \bmod N$ ($\bmod$ is remainder operation, be careful to handle negative $\bmod$)
- $y = (b - 1 - D) \bmod N$

(It would be good to draw the original sequence yourself and see how it works.)

C. Rotating Sequence and Query

- In this case, the problem is to find the sum from the x index to the y index in the unrotated sequence, and this can be found using the cumulative sum technique.
- Note that if $y < x$, we need to process it as (interval sum from x to $(N-1)) +$ (interval sum from 0 to y).
- The total time complexity is $O(N+Q)$.

D. Escaping Hexagontiles

`#graph`, `#bfs`

Difficulty — **Silver**

- Author: Yoonsu Lee`lys9546`
- 88 Submitted, 60 Passed
  AC Ratio: 69.318%
- First Solver: `kyo20111`, +11min

D. Escaping Hexagontiles

- You can model a graph by treating the odd and even rows of adjacent hexagonal tiles differently.
- Then, you can find the shortest path tiles from the starting point $(0, 0)$ to $(N-1, M-1)$ through BFS.

E. Retiring from Retirement

`#dp, #knapsack`

Difficulty — **Gold**

- Author: Jonghyeon Park`belline0124`
- 66 Submitted, 25 Passed
  AC Ratio: 37.879%
- First Solver: `dabbler1`, +25min

E. Retiring from Retirement

» All three queries given bring relatively forward a retirement of Yeongdo.

» Since this problem asks about the relative difference in retirement dates between Jonghyeon and Yeongdo, the three queries can be processed as if they bring forward a retirement of Yeongdo.

» Early retirement: Bring forward a retirement of Yeongdo by D.

» Military training unit: Bring forward a retirement of Yeongdo by D.

» Temporary non-commissioned officer: Bring forward a retirement of Yeongdo by $M \times 30$.

E. Retiring from Retirement

- If you unify the target of the queries to one person, this problem can be solved with 0-1 knapsack.
- Since the relative difference in retirement dates between the two must be processed, it can be processed as if Jonghyeon's retirement is delayed.

F. Mighty Fine Morning

`#floyd_warshall, #dijkstra`

Difficulty — Gold

- Author: Geunseong Kim onsbtyd
- 75 Submitted, 22 Passed
  AC Ratio: 29.333%
- First Solver: `kyo20111`, +21min

F. Mighty Fine Morning

» The summary of this problem is "Find the minimum cost of going from node S to node E without going through node K times Q."

» Representative algorithms for finding this shortest path include Bellman–Ford, Floyd–Warshall, and Dijkstra.

F. Mighty Fine Morning

- **Bellman–Ford**

  It computes with a time complexity of $O(N^3M)$. It will time out approximately 10 billion in the given constraint.

- **Floyd–Warshall**

  It computes with a time complexity of $O(N^4)$. It will pass within the given constraint with approximately 1 billion in the given constraint.

- **Heap Dijkstra**

  It computes with a time complexity of $O(N^2(N+M)\log N)$. It will time out approximately 7 billion in the given constraint.

F. Mighty Fine Morning

» N^2 **Dijkstra**

It computes with a time complexity of $O(N^4)$. Given the constraint, it passes within the time limit of approximately 1 billion.

F. Mighty Fine Morning

- The Solution is to run $O(N^2)$ Dijkstra N^2 times, or $O(N^3)$ Floyd Warshall N times.
- **Dijkstra**

  Declare an array in the form `D[N][N][N]`.

  If you start at node i and do not visit node j, run Dijkstra N^2 times in advance, filling in `D[j][i][1~N]`.

F. Mighty Fine Morning

» **Floyd Warshall:**

Declare an $N \times N$ array with N layers in the form `D[N][N][N]`.

It uses the fact that the outermost `k` loop of the Floyd Warshall loop connects the path entering the kth node and the path leaving the kth node.

`D[1~k-1][1~N][1~N], D[k+1 ~ N][1~N][1~N]`, that is, it updates the rest except for the kth layer.

F. Mighty Fine Morning

You can calculate the above process and answer each query. The total time complexity is $O(N^4 + Q)$.

» Key

1. 1 billion operations can be completed within 1 second.
2. Heap Dijkstra is slower than N^2 Dijkstra in a complete graph.
3. Floyd's outermost k loop connects the incoming and outgoing paths.

G. Move or Block!

`#game_theory`

Difficulty — Gold

- Author: Jeonghwan Choi `jh01533`
- 74 Submitted, 9 Passed
  AC Ratio: 12.162%
- First Solver: `fermion5`, +49min

G. Move or Block!

- For convenience, we will express the action of moving as a `move`, building a wall as a `block`, and assume that there is a wall outside the game board.
- Let’s handle the trivial case first.
- `~~XOX~~` – Second turn win (not given as input)
- `~~XO.~~`, `~~.OX~~` – First turn win
- The following explanation does not consider the above cases.

G. Move or Block!

To solve this problem, we need to combine several observations.

1. Except in cases where it is possible to end the game or it is unavoidable, you should not `block` adjacent squares or `move` to a place that is a wall.
   - » This is a move that leads directly to defeat.
   - » Other observations are established under the assumption that this rule is followed.

G. Move or Block!

2. It is best for a player with a sure-win strategy to `block` every turn.

   » Let's assume that only `move` is the best.

   If the next person does `move` back to the original position, the pattern repeats and the game never ends. In other words, you can't win.

   » Therefore, if you have a surefire strategy, you can see that `block` every turn is the best.

G. Move or Block!

3. When the words are blocked with `~~X.O.X~~`
   - » If the number of remaining `.` is odd, the first turn wins, and if it is even, the second turn wins.
4. If the game ends, you must go through the case of statement 3.
   - » It is always established by statement 1.
5. If it is not the current case of $3.$, you can make the number of `.`s as biased as you want after your turn ends while satisfying 1.
   - » If you want to maintain the current bias, you can `block` it if it is not a `move`.
   - » Players win by making the case of statement 3. and making the number of `.`s even.

G. Move or Block!

- When making statement 3., you must `block`, so to win, the number of `.` must be odd on your turn.
- However, if it is not statement 3. at the current point, based on statement 5.,

  you can always hand over a situation where the number of `.` is even to the opponent.

  According to statement 4., this means you can make it so that the opponent can never win.

G. Move or Block!

In other words, if you fail to make statement 3. on the first turn of the game, you will always end up in a draw. Therefore, we can divide the whole thing into 5 cases.

» `~~XOX~~`: After-turn win (does not need to be processed because it is not given as input.)

» `~~XO.~~`, `~~.OX~~`: First-turn win

» `~~X.O.X~~`: If the number of `.` is odd, the first turn wins, if it is even, the second turn wins

» `~~X.O..~~`, `~~..O.X~~`: If the number of `.` is odd, first turn wins, if it is even, then a draw

» `~~..O..~~`: Draw

H. Guardians of the Forest

`#tree, #graph_theory, #graph_traversal, #greedy`

Difficulty — Gold

- Author: Yeongdo Jeong `0do`
- 16 Submitted, 4 Passed
  AC Ratio: 25%
- First Solver: `dabbler1`, +117min

H. Guardians of the Forest

The output of this problem is the maximum number of times you can use replication.

» You can see that replication is most efficient when it is executed at every vertex you pass, and it is also most efficient when you do not use wait.

» In other words, it can be simplified to “How many vertices are visited when the path passes through as many vertices as possible?”

H. Guardians of the Forest

$O(N)$ solution

- It is inefficient to use one edge more than 3 times.
- It is an unnecessary back–and–forth between two vertices.
- In other words, all edges should be used less than 2 times.

H. Guardians of the Forest

- The number of visited vertices increases by one every time the number of edges used 2 times increases by one.
- Similarly, the number of visited vertices also increases by one when the number of edges used 1 times increases by one.
- The optimal solution is guaranteed when the sum of the edges used 2 times and the edges used 1 times is the maximum.
- Assuming that a random path is created for the root to move, the number of edges used 1 times is the same as the number of edges between the b vertex and the destination vertex of the path.

H. Guardians of the Forest

» We can deduce deductively that “the optimal solution is guaranteed when the root moves as much as possible in the direction of the a vertex.”

» There is more than one optimal path.

» Let’s assume that the arbitrary vertex on the tree that becomes the destination of the optimal path is the c vertex.

H. Guardians of the Forest

» When the a vertex is set as the root node of the tree, if the c vertex is a descendant node of the b vertex,

» It is guaranteed that the root moves as much as possible in the direction of the fire and then moves to the c vertex is one of the optimal paths.

» This is because the number of edges used 1 times remains unchanged, and the number of edges used 2 times is maximized.

H. Guardians of the Forest

- When vertex a is set as the root node of the tree, if vertex c is not a descendant of vertex b,
- In order to reach vertex c from vertex b, the root must move in the direction of vertex a.
- If the descendant node of the vertex where the root is located has moved until it becomes vertex c,
- It is guaranteed that the root moves in the direction of the maximum fire and then moves to vertex c is one of the optimal paths.

H. Guardians of the Forest

A solution that operates in time complexity $O(N)$ can be derived.

1. Find the midpoint between vertex a and vertex b.

» This midpoint is the last vertex where the root can stand when moving from vertex b to vertex a as much as possible.

» Let's assume this vertex is vertex d.

2. After selecting a random vertex on the tree, assume this vertex is the c vertex.
3. Find the optimal path when the c vertex is the destination and the number of vertices passed on the optimal path.

H. Guardians of the Forest

- [(the number of vertices between the b vertex and the d vertex) + the number of vertices between the d vertex and the c vertex) − (the number of vertices in the overlapping part of the previous two paths)] = the number of vertices passed

4. Repeat steps $2. \sim 3.$ for all vertices. The maximum number of vertices from step $3.$ for all vertices is the answer to the problem.

- The values required to perform step $3.$ can be calculated in advance in $O(N)$ time before performing step $4.$.

I. Three Number XOR and Query

`#bitmask`, `#lazyprop`

Difficulty — **Platinum**

- Author: Jeonghwan Choi `jh01533`
- 16 Submitted, 1 Passed
  AC Ratio: 12.5%
- First Solver: `kyo20111`, +63min

I. Three Number XOR and Query

- Let’s process query 1 first.
- Since A_i is small, we can preprocess all the three pairs that can make a number.
- The number of three pairs that can make x is 715, so if we only know the number of numbers in the interval $[l, r]$, we can process query 1.
- We can find the number of numbers in the interval using a segment tree, but it seems difficult to return within $O(Q64 \log N + 715Qc)$ time.

I. Three Number XOR and Query

» Let's take a closer look at 'The number of three pairs that can make x'.

» If 3 numbers are the same among the three pairs, it is in the form (x, x, x), and if 2 numbers are the same, it is in the form (y, y, x).

» Does this mean that x exists in more than 3 cases? Which number exists more than 2 and x exists? It can be simplified to.

» Therefore, if we process the form separately, we only need to find the set of numbers, not the number of numbers, and this can be quickly found using a segment tree using a bit set.

I. Three Number XOR and Query

» Based on the above information, let’s construct a segment tree using a bit set.

» Using the `unsigned long long` (or `bitset`) data type, we store whether the number i exists in the current section in the ith bit.

» The operation to merge nodes can be quickly calculated using the `OR` operation of two nodes.

» In this way, we quickly found the set of numbers in the section in $O(\log N)$.

I. Three Number XOR and Query

» The method for processing the (x, x, x) and (y, y, x) forms will be described later, and we need to check 651 cases excluding these.

» Even though the amount of operations seems large, since the set of numbers in the interval is bits, one case can be processed with a simple `&` operation, so it can be processed quickly.

» If you do not check all 651 cases, but only check the numbers included in the set, the number of operations is reduced by more than 2 times due to the characteristics of `XOR`.

I. Three Number XOR and Query

Let's process query 2.

» It can be processed with a slowly updated segment tree. This can also be accelerated with bit operations.

» Doing $+x \bmod 64$ on a node is the same as rotating the bits x spaces to the right.

» `node = ((node << (64-x)) | (node >> x)) & (1<<64-1` If you use a 64−bit data type, it is okay to omit it, be careful of overflow)

I. Three Number XOR and Query

» 'Are there 3 or more x? There are several ways to process 'Which number exists more than 2 and x exists'.

» Add a set of numbers that exist more than 2 and a set of numbers that exist more than 3 to the node. This can also be easily processed with bit operations.

» The set of numbers that exist more than 3 can be processed using the pigeonhole principle without inserting them.

I. Three Number XOR and Query

- » The overall time complexity is $O(N \log N + Q \log N + 651Q)$.
- » The $651Q$ part may look large, but as described above, it is a simple operation that can be processed with a single `&` operation, and if you only search the numbers in the interval, the amount of operation is reduced by half according to the characteristics of `XOR`. It can be processed in about 0.1 seconds without any special optimization.