2026 WYIOI Editorial

T261 Quiz Score

Rock Lee, Cheung Hui Yat

2026-09-26

Solution

Output $\max(2 \times C - (N - C), 0)$.

You can either use `std::max` or if conditions.

T262 The Perfect Dive

Rock Lee, Cheung Hui Yat

2026-09-26

Solution

Loop through the array to compute the sum, the maximum and minimum element of $[s_1, \ldots, s_N]$. Output the sum minus the maximum and minimum.

Alternatively, you can use `std::accumulate` for the sum, `std::max_element` for the maximum and `std::min_element` for the minimum.

T263 Evil Snakes

Siu Lok Yin

2026-09-26

Problem Statement

Given a board from cell 1 to $N(\geq 8)$. Rock can move 1 to 6 cells forward each turn. There are K ladders where the ladder i sends you forward from cell A_i to B_i. Add at most 6 snakes that can send you back so that Rock cannot reach the cell N.

Problem Statement

You should be aware that there are many constraints in the problem statement.

- Ladders do not intersect with each other (but can be connected)
  - $1 < A_1 < B_1 \leq A_2 < B_2 \leq \cdots \leq A_K < B_K < N$
- You cannot place the head of a snake in cell N
  - $1 \leq Y < X < N$
- You cannot place the head of a snake in the same cell as the bottom of a ladder
  - $X \neq A_i$ for all $1 \leq i \leq K$

Observation

You should be aware that there are many constraints in the problem statement.
Extract useful information from it.

- Ladders cannot start at cell 1 or end at cell N
  - $1 < A_1 < B_1 \leq A_2 < B_2 \leq \cdots \leq A_K < B_K < N$
- We can always set the tail of a snake to cell 1
  - $1 \leq Y < X < N$
- You cannot place the head of a snake in the same cell as the bottom of a ladder
  - $X \neq A_i$ for all $1 \leq i \leq K$

Solution

Since ladders cannot end at cell N, you must rely on a die move (1 to 6 cells) to reach cell N. Thus, Rock can reach cell N if and only Rock can stop at one of the cells $N-6$ to $N-1$ for a die roll.

Therefore, we can just put a snake head at each of those 6 cells and lead them back to cell 1. Be aware that you cannot place the head of a snake at the same cell as the bottom of a ladder.

T264 Squares

Siu Lok Yin

2026-09-26

Problem Statement

Given an infinite grid of square cells where each of them is initially white. If you choose a positive integer $K(\leq 10^9)$, you toggle the colour (white to black and black to white) of cells (x, y) where $0 \leq x, y < K$.

Given $T(\leq 2 \times 10^5)$ queries, determine whether you can choose some positive integers so that there are exactly $Q_t(\leq 10^9)$ cells that are black.

Subtask 1 (15%): Q_t is a perfect square

Choose $K_1 = \sqrt{Q_t}$.

Subtask 2 (25%): $Q_t + 1$ is a perfect square

Choose $K_1 = 1$ and $K_2 = \sqrt{Q_t + 1}$

Subtask 3 (35%): Q_t is odd

If we choose K and $K+1$ only, we will have exactly $(K+1)^2 - K^2 = 2K+1$ black cells.

This allows us to deal with all odd numbers in the form $2K+1$ where $K \geq 0$, i.e. all odd numbers except 1.

1 is a special case where we can simply choose $K_1 = 1$.

Solution

As hinted in Subtask 2 and combined with Subtask 3, if we choose 1, K and $K+1$, we will have exactly $(K+1)^2 - K^2 + 1 = 2(K+1)$ black cells.

This allows us to deal with all even numbers in the form $2(K+1)$ where $K \geq 0$, i.e. all even numbers except 2 and 4.

It is impossible to construct 2 and we can choose $K_1 = 2$ for 4.

T265 Growing Vegetables is Boring

Siu Lok Yin

2026-09-26

Problem Statement

Vegetable i has height

$$\max(0, V_i - |d - V_i|)$$

on day d.

In other words, its height increases by 1 every day until day V_i, then decreases by 1 every day until it reaches 0.

Choose one day to harvest all vegetables and maximise their total height.

Subtask 1 (21%): $N, V_i \leq 1000$

The last day where any vegetable has positive height is at most

$$2 \max V_i.$$

Therefore, we can try every possible day from 0 to $2 \max V_i$, and calculate the total height of all vegetables directly.

Time complexity: $\mathcal{O}(N \max V_i)$

Observation

Consider the height of a single vegetable.

- Before day V_i, its height increases by 1 per day.
- From day V_i to day $2V_i$, its height decreases by 1 per day.
- After day $2V_i$, its height stays at 0.

Therefore, the behaviour of the total harvest only changes on days V_i and $2V_i$.

Between two such days, the total harvest changes linearly.

Subtask 2 (23%): $N \leq 2000$

Since the total harvest is linear between two consecutive days where its behaviour changes, a maximum can always be found at one of these endpoints.

Therefore, it is sufficient to try all days

$$V_i \quad \text{and} \quad 2V_i.$$

There are only $2N$ candidate days. For each candidate day, loop through all vegetables and calculate the total harvest.

Time complexity: $\mathcal{O}(N^2)$

Subtask 3 (25%): $\max V_i \leq 2 \times \min V_i$

Let

$$L = \min V_i, \qquad R = \max V_i.$$

Before day L, every vegetable is growing, while after day R, every vegetable is decreasing or already at 0. Therefore, an optimal day lies in $[L, R]$.

Since $R \leq 2L \leq 2V_i$, no vegetable reaches 0 inside this range.

Subtask 3 (25%): $\max V_i \leq 2 \times \min V_i$

When the day increases by 1:

- a vegetable not yet at V_i contributes $+1$;
- a vegetable already past V_i contributes -1.

Therefore, after sorting V, the change in total harvest decreases by 2 whenever we pass some V_i.

The maximum is reached at a median of V, where the number of positive contributions is at least the number of negative contributions.

Time complexity: $\mathcal{O}(N \log N)$

Observation

The full solution uses the same idea: maintain how much the total harvest changes when the chosen day increases by 1.

For vegetable i, its contribution to this change is

$$+1 \quad \longrightarrow \quad -1 \quad \longrightarrow \quad 0.$$

Therefore:

- on day V_i, the change decreases by 2;
- on day $2V_i$, the change increases by 1.

Solution

Initially, on day 0, the total harvest is 0 and its change per day is N.

Create two events for every vegetable:

$$(V_i, -2), \qquad (2V_i, +1).$$

Sort all events by day.

Between two consecutive event days, the change is constant, so if the previous event is at day x and the next one is at day y,

$$\text{harvest}(y) = \text{harvest}(x) + \text{change} \times (y - x).$$

Solution

Sweep through the events in increasing order of day:

- update the total harvest using the current change;
- update the maximum answer;
- apply all events on the current day.

There are $2N$ events.

Time complexity: $\mathcal{O}(N \log N)$

T266 Treasure Hunting

Lau Pok Man

2026-09-26

Problem Statement

Find up to K treasures on an integer array of size N, where A_i is the minimum distance from the i-th cell to the closest treasure for $A_i \neq -1$.

Place at most K treasures on the N cells, such that all A_i where $A_i \neq -1$ are satisfied.

The following slides will denote the minimum number of treasure required as $T_{\min}$ (if a valid solution exists).

Subtask 1 (7%): $K = 1$

Find a A_i such that $A_i \neq -1$, the only treasure is at $i - A_i$ or $i + A_i$, if not out of bounds.

Crosscheck with other A_i, if they do not point to the same position output `-1`.

Time complexity: $\mathcal{O}(N)$

Subtask 2 (14%): $N \leq 16$

Exhaust all possible solutions, and check each solution to see if it works.

You may do so by keeping track of the current closest treasure and update accordingly when the next treasure is closer.

Time complexity: $\mathcal{O}(N \times 2^N)$

Observation

The location of the treasure closest to A_i is in position $i - A_i$ or $i + A_i$, additionally, the range $i - A_i + 1$ to $i + A_i - 1$(inclusive) cannot contain treasure. Each $A_i \neq -1$ demands at least one treasure at $i - A_i$ or $i + A_i$.

Subtask 3 (23%): $N \leq 5000,\ K \geq 2 \times \mathrm{T}_{\min}$

For each $A_i \neq -1$, mark cells $i - A_i + 1$ to $i + A_i - 1$ as impossible to contain treasure. Check for each $A_i \neq -1$ whether $i - A_i$ and $i + A_i$ are marked, if so, output -1, else just take whichever is not marked.

Note that you don't have to place another treasure if there is already one that satisfies A_i, and we do not have to optimise the number of treasures placed for now, so do not need to consider which position is better.

Time complexity: $\mathcal{O}(N^2)$

Subtask 4 (19%): $K \geq 2 \times \mathsf{T}_{\min}$

Same idea as Subtask 3, but we need to optimise the marking process.

Use a difference array instead of plain looping.

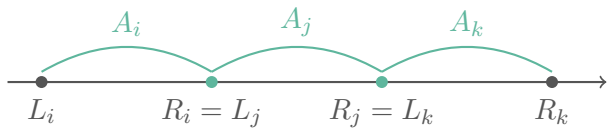
Time complexity: $\mathcal{O}(N)$

Observation

For each $A_i \neq -1$, let

$$L_i = i - A_i, \qquad R_i = i + A_i.$$

After marking, L_i and R_i are the only possible treasure positions.

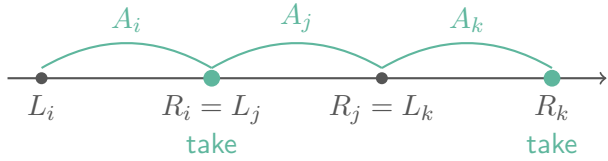


The remaining intervals cannot overlap in their interiors, since all cells strictly inside each interval are marked.

Therefore, two intervals are either disjoint or share exactly one endpoint. Hence, at most two A_i can share the same possible treasure position, and the remaining intervals form independent single intervals or chains.

Solution

We only need to find the minimum number of treasures for each chain.



Process A_i from left to right.

- If L_i has already been chosen, A_i is already satisfied.
- Otherwise, choose R_i.

Choosing R_i may also satisfy the next constraint in the chain, while choosing L_i cannot help any later constraint.

Time complexity: $\mathcal{O}(N)$

T267 Landscapes

Siu Lok Yin

2026-09-26

Problem Statement

Given an odd-length array A which forms a strict mountain with peak at P.

In one operation, increase any A_i by 1.

Find the minimum number of operations such that the resulting array:

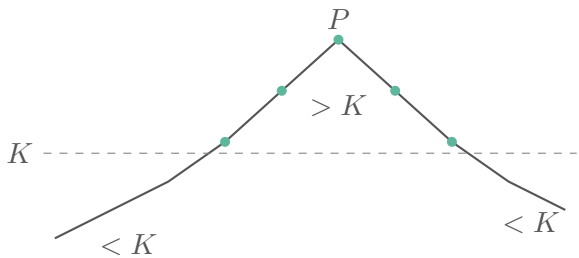
- is still a strict mountain with peak at P;
- has median exactly K;
- has exactly one element equal to K.

Observation

Let $h = \frac{N-1}{2}$. Since K is the unique median, there must be exactly

$$h \text{ values} < K, \qquad 1 \text{ value} = K, \qquad h \text{ values} > K.$$

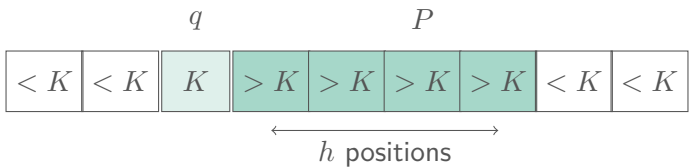
Since B is a strict mountain, the positions with value $> K$ must form one consecutive interval containing the peak P.



Observation

Suppose position q is the unique position with value K.

If $q < P$, the h positions with value $> K$ must be directly to its right.



Therefore, if $q < P$, the positions with value at least K are

$$[q, q + h].$$

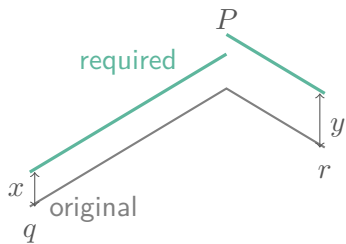
The case $q > P$ is symmetric.

Subtask 1 (13%): $|A_i - A_{i+1}| = 1$

Since A is a strict mountain,

$$A_i = A_P - |i - P|.$$

Consider $q < P$ and let $r = q + h$.



Raising one endpoint forces the whole slope towards the peak to increase accordingly.

So:

- the left side is raised by x,
- the right side is raised by y,
- the peak must satisfy both.

Subtask 1 (13%): $|A_i - A_{i+1}| = 1$

Let

$$x = K - A_q, \qquad y = \max(0, K + 1 - A_r).$$

The left slope requires an increase of x, while the right slope requires an increase of y.

The peak must satisfy both sides, so it is increased by $\max(x, y)$.

Therefore, the additional cost is

$$x(P - q) + y(r - P) + \max(x, y).$$

Try every possible q and handle $q > P$ symmetrically.

Time complexity: $\mathcal{O}(N)$

Subtask 2 (17%): $K > \max A_i$, $P = \frac{N+1}{2}$

Since $K > \max A_i$, every original height is below K.

For a fixed position q of the unique K, every required height inside the interval is greater than its original height.

Therefore, every required increase is forced.

The required heights on each side of the peak form arithmetic sequences.

Subtask 2 (17%): $K > \max A_i$, $P = \frac{N+1}{2}$

Use prefix sums of A.

For each possible q:

- calculate the sum of required heights using arithmetic progression formulae;
- subtract the sum of the original heights using prefix sums.

Each q can be checked in $\mathcal{O}(1)$.

Time complexity: $\mathcal{O}(N)$

Subtask 3 (19%): $K > \max A_i$

The same idea as Subtask 2 works even when P is not in the middle.

We only need to check whether the interval corresponding to q contains P and stays inside the array.

For $q < P$, let $r = q + h$. We require

$$q < P \leq r \leq N.$$

The case $q > P$ is symmetric.

Time complexity: $\mathcal{O}(N)$

Subtask 4 (23%): $N \leq 3000$

For the general case, try every possible position q of the unique K.

For example, if $q < P$ and $r = q + h$, the minimum required heights are

$$B_i \geq K + i - q \qquad (q \leq i < P),$$

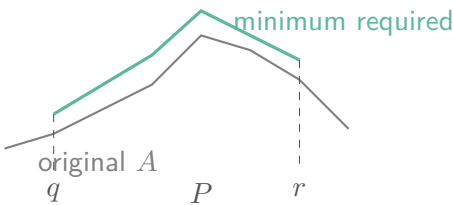
$$B_i \geq K + 1 + r - i \qquad (P < i \leq r),$$

and

$$B_P \geq \max(K + P - q,\ K + 1 + r - P).$$

Subtask 4 (23%): $N \leq 3000$

For a fixed q, raise every position only as much as necessary.



Positions outside $[q, r]$ must already be below K.

For each position inside the interval, add

$$\max(0, \text{required height} - A_i).$$

Time complexity: $\mathcal{O}(N^2)$

Observation

To optimise Subtask 4, consider a term on the increasing side:

$$\max(0, C + i - A_i).$$

Rewrite it as

$$\max(0, C - (A_i - i)).$$

Let

$$D_i = A_i - i.$$

Since the left side of A is strictly increasing, D_i is non-decreasing.

Observation

Therefore, for

$$\max(0, C - D_i),$$

all positions with positive contribution form a prefix of the range.

Binary search for the last position where

$$D_i < C.$$

If there are m such positions, their total cost is

$$mC - \sum D_i.$$

Prefix sums of D allow this sum to be calculated in $\mathcal{O}(1)$.

Solution

The decreasing side is similar.

Let

$$E_i = A_i + i.$$

On the right side of the peak, E_i is non-increasing.

A term of the form

$$\max(0, C - i - A_i) = \max(0, C - E_i)$$

can again be calculated using binary search and prefix sums.

The peak itself is handled separately in $\mathcal{O}(1)$.

Solution

Try every possible position q of the unique K.

For each q:

- determine the corresponding interval of values at least K;
- check that all positions outside the interval are already below K;
- calculate the cost of the increasing side using $A_i - i$;
- calculate the cost of the decreasing side using $A_i + i$;
- calculate the required height of the peak.

Each candidate takes $\mathcal{O}(\log N)$.

Time complexity: $\mathcal{O}(N \log N)$

T268 Medalist

Git Lun Mak

2026-09-26

Background

This task is inspired by the manga/anime series *Medalist*, which is about figure skating.

I highly recommend you watch the anime (too bad it is exclusive on Disney+).

Problem Statement

Construct a permutation of numbers from 1 to N, so that when treated as a circular array, the sum of distance between each adjacent numbers is exactly M, or deduce that there is no such permutation.

Subtask 1 (5%): $N \leq 4$

The following are all of the possible cases and one of the solutions:

N	M	**Solution**
1	0	1
2	1	1, 2
3	2	1, 2, 3
4	3	1, 2, 3, 4
4	4	1, 2, 4, 3
4	5	1, 3, 2, 4

Hard code all possible cases. Anything else are impossible cases.

Subtask 2 (8%): $N \leq 10$

Exhaust all permutations. You can calculate the sum of distance using this function:

```
long long sum(int perm[]) {
    long long res = 0;
    for (int i = 1; i <= N; i++)
        pos[perm[i]] = i;
    for (int i = 2; i <= N; i++) {
        int dif = abs(pos[i] - pos[i - 1]);
        res += min(dif, N - dif);
    }
    return res;
}
```

Time complexity: $\mathcal{O}(N \times N!)$

Observation 1

From the scoring and subtasks, it is (pretty obviously) hinted that you should separate the cases by the parity of N and find a way to determine whether it is possible for the given N and M before constructing the answer.

An educated guess is that for a fixed N, the range of M for the possible cases should be an interval (some $[L, R]$), so we should try to find the lower and upper bounds of M in terms of N.

The lower bound of M is clearly $N - 1$, since the distance is at least 1 when moving to a different position.

Observation 2

The upper bound for M is $\begin{cases} N^2/2 - N + 1 & \text{if } N \text{ is even,} \\ (N-1)^2/2 & \text{otherwise} \end{cases}$

Proof:

Let $K = \lfloor N/2 \rfloor$, which is the maximum distance in a move.

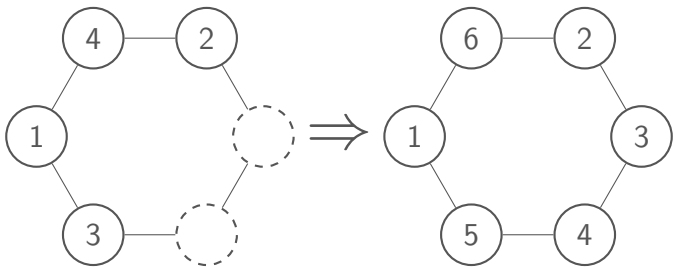
For the odd N cases, since $\gcd(K, N) = 1$, it is possible that all $N - 1$ moves to have a distance of K.

For the even N cases, we cannot have two consecutive moves with a distance of K since we would return to the same position. Thus, we can at most do $K, K-1, K, K-1, \ldots, K$.

Towards construction

Since trying to take care of movements to both directions is really hard, we should try to limit ourselves to moving in a single direction. This turns out to work about 90% of the time.

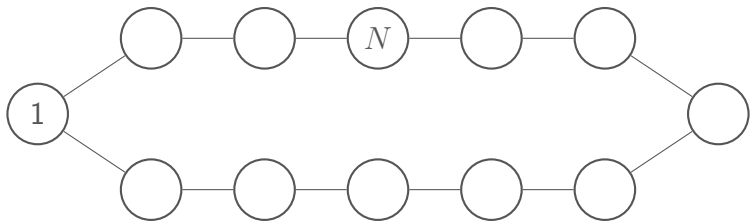
Another important way to simplify construction is to notice that if we can achieve M in less than $N-1$ moves, we can leave gaps (placeholders) in our array and always extend it to a valid solution later without adding any extra distance. (why?)



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

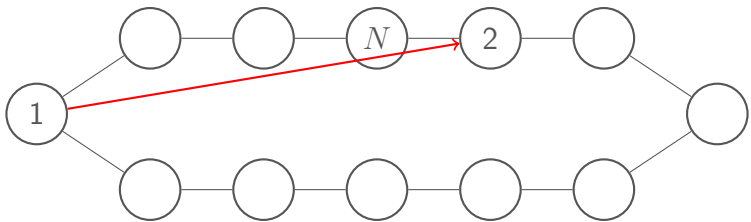
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

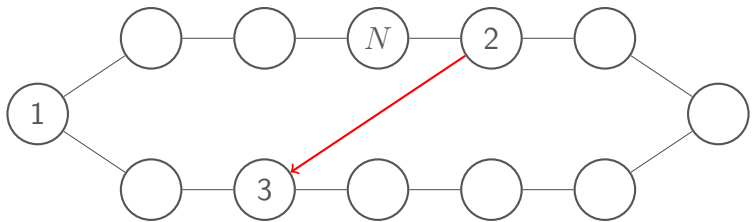
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

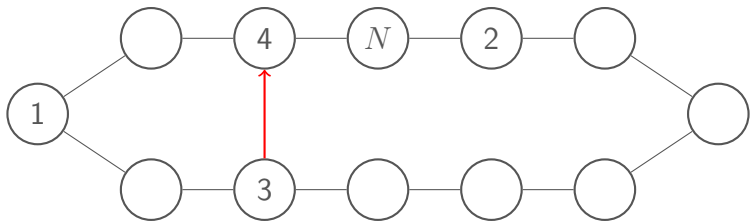
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

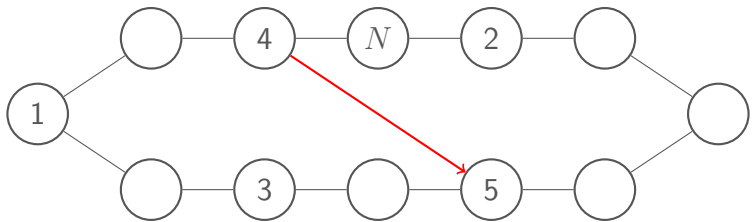
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

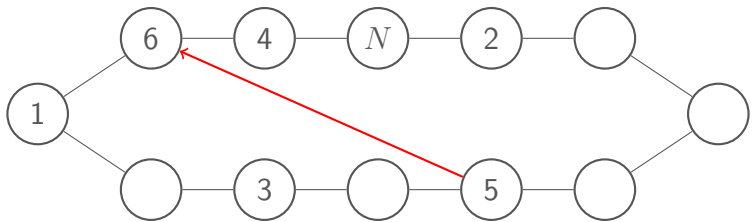
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

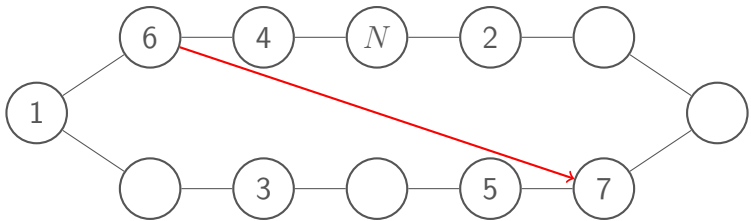
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

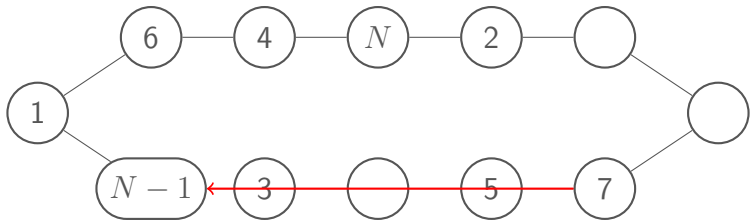
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

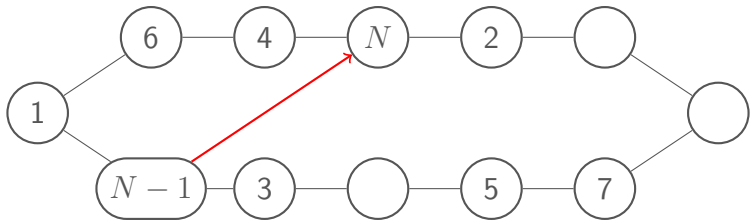
Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

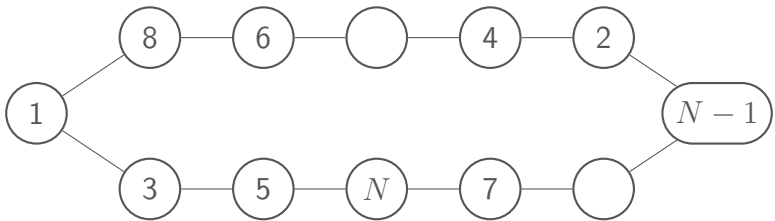
With the limitation of moving one direction only, the end position will be fixed. This changes the problem to looping around $Q = \lfloor M/N \rfloor$ times. We branch our logic based on the remainder $R = M \bmod N$.

Case: $1 \leq R < N/2$



Subtask 3 (10%): N is even, $M \leq N^2/4$

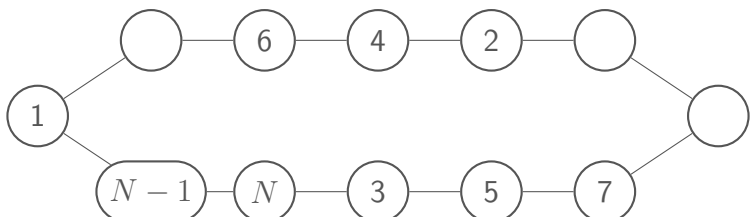
Case: $N/2 \leq R < N$



Subtask 4 (10%): N is even, $M \leq N^2/4$

Case: $R = 0$ (M is a multiple of N)

If M is a multiple of N, it would be impossible to limit to only one direction, since that requires that the starting and ending positions be the same. However, there is a rather simple way to solve it by placing our anchor jumps at the edges.



Subtask 5 (14%): N is even

Notice that the method for subtasks 3 and 4 might fail if $Q = \lfloor M/N \rfloor$ is too large (approaching the maximum bound). Luckily, there should be only 2 edge cases you need to handle and their solutions can be found by looking for patterns in small cases.

M	**Solution**
$N^2/2 - N + 1$	1, 3, 5, ..., $N-1$, 2, 4, 6, ..., N
$N^2/2 - N$	1, 3, 5, ..., $N-1$, 2, 4, ..., $N-4, N, N-2$

Subtasks 6 & 7 (33%): N is odd, $M \leq N^2/4$

The logic for odd N closely mirrors the even N cases. By moving in a single direction, we can wrap around the rink $Q = \lfloor M/N \rfloor$ times.

We then divide the construction into cases based on the remainder $R = M \bmod N$. Because N is odd, the maximum distance in a single jump is $\lfloor N/2 \rfloor$.

For each case, we only need to place 3 to 4 "anchor" jumps in our array at specific distances to secure the target distance M. The remaining empty positions will be handled by our extension technique.

Subtask 8 (20%): N is odd, Edge Cases

Just like the even N constraints, when M is extremely large and approaches the absolute maximum bound of $(N-1)^2/2$, the standard winding strategy fails because Q becomes too large.

We can solve these specific edge cases by finding patterns for the maximums:

M	**Solution**
$(N-1)^2/2$	$1, 3, 5, \ldots, N, 2, 4, 6, \ldots, N-1$
$(N-1)^2/2-1$	The above, but swap the middle two elements

Alternative Solution (by ChatGPT)

Instead of analyzing the remainder M mod N, we can reframe the problem as a **greedy subset-sum problem**.

Core Algorithmic Intuition:

- Every arrangement can be seen as a base sequence that has been modified by "flipping" the placement of certain elements across the circle.
- Each flip alters the total distance by a pre-calculable, deterministic "weight."
- By calculating the exact difference between our base sequence's distance and our target M, we can greedily select a subset of these weights to perfectly bridge the gap.

Alternative Solution: $M \leq \lfloor N^2/4 \rfloor$

1. The Base State: A simple sequential placement $(1, 2, 3, \ldots, N)$ yields the absolute minimum distance of $N - 1$. Our target excess is $X = M - (N - 1)$.

2. The Greedy Choice:

- When placing the r-th element, choosing to put it on the opposite side of our current sequence (instead of right next to it) increases the total distance by exactly $W_r = \min(r, N - r) - 1$.
- We sort all possible W_r values in descending order.
- We iterate through the weights, greedily "taking" a flip if $W_r \leq X$, and update $X \leftarrow X - W_r$.

3. Construction: Using a left and right pointer, we populate the sequence. If a step was "taken", we alternate the pointer we draw from, effectively creating the zigzag pattern that generates the required extra distance.

Alternative Solution: Odd N, Large M

When M is large, starting from the minimum distance and adding up is no longer viable because we run out of valid disjoint weights.

Reverse Greedy Strategy:

- We invert the logic: start from the **maximum** possible distance.
- For an odd N, the absolute maximum distance is achieved by constantly jumping by the maximum stride $K = \lfloor N/2 \rfloor$. Since $\gcd(K, N) = 1$, this naturally visits every position.
- Our target deficit is now $X = M_{\text{max}} - M$.
- We apply the exact same greedy subset-sum algorithm, but this time we select weights that represent the *reduction* in distance when we deviate from the maximal stride.

Alternative Solution: Even N, Large M

Antipodal Pairing:

- We mathematically link opposite nodes on the circle into pairs $(x, x + K)$.
- The distance traversed depends purely on the relative orientation of these pairs (whether we traverse them "forward" or "backward").
- By systematically toggling the Boolean orientations of these pairs, we can reliably step the total sum up or down to hit high values of M.

The Middle Gap: For the narrow band of values between the "Low" maximum and "High" minimum ($K^2 < M < K^2 + K - 1$), the greedy strategy leaves a tiny gap. We bridge this by simply checking all single-element shifts or prefix-reversals on our base sequence, one of which is mathematically guaranteed to hit M.

Loose Ends

There are definitely (better) alternative solutions to the given ones.

For your reference, my solution code has about 160 lines and ChatGPT's has more than 200 lines. However, I think you would have a worse time debugging my solution (more constructive) than ChatGPT's solution (more algorithmic).

Takeaway

There are a few tips for solving constructive algorithm / ad-hoc tasks:

- Solve some small cases manually / with the aid of programs
  - Secure points for first few subtasks and gain observations
- Observe patterns / relations between them
  - And distinguish edge cases that might not be easy to solve
- Making some "reasonable" guesses
  - Lower and upper bounds for M
- Convince yourself that the guess is correct (or incorrect?)
  - Maths is pretty useful in this case
  - Submission has no penalty (in IOI-style competition), so make hypothesis and verify them through submission (proof by AC)
- Create functions for repetitive code

These would be very useful if you want to gain score from senior Q4.