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

M2551 Tree Programming

Hsieh Chong Ho {QwertyPi}

2025-07-01

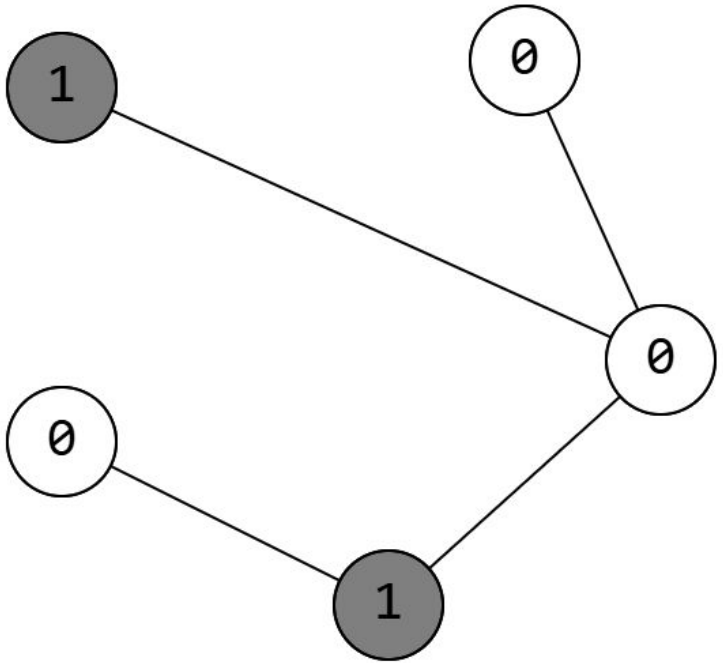
The Problem

Tree Machine consists of **N (2 ≤ N ≤ 100) nodes** and **C (= 1000) colours**.

Initial Tree is coloured with **White (= 0)** and **Black (= 1)**.
Repeat M (= 1000) rounds:
    Update node colour base on its own and its neighbour, **simultaneously**.
Finally, report the answer based on the colour only, for each node.

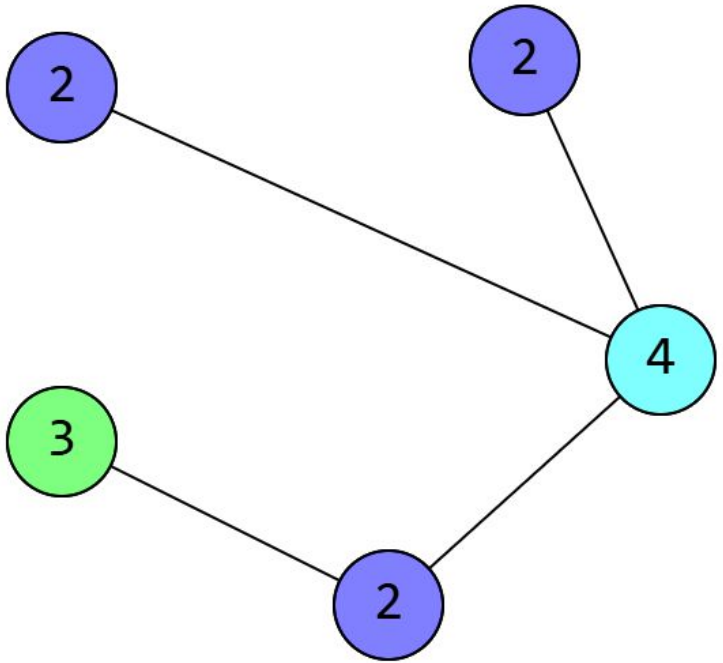
Some tree tasks are given to you to solve.

Example (Number of Black Neighbours)



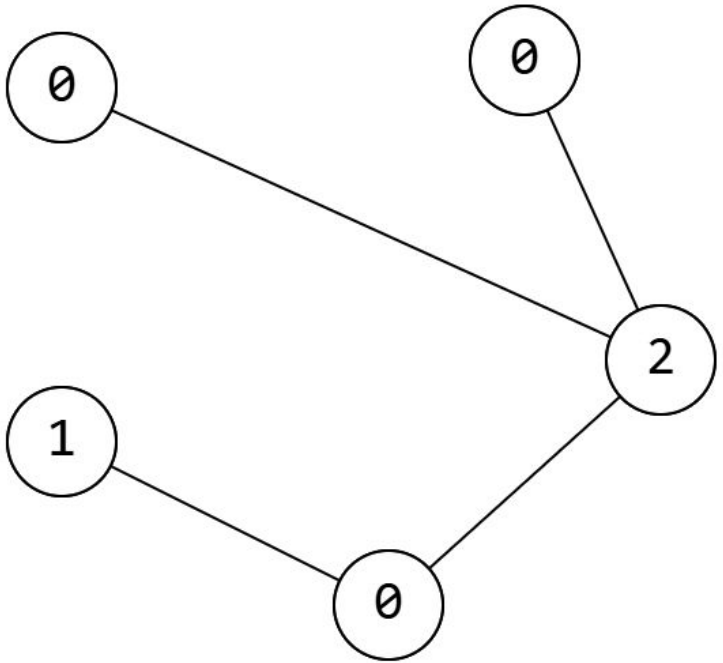
T = 0

Example (Number of Black Neighbours)



T = 1

Example (Number of Black Neighbours)

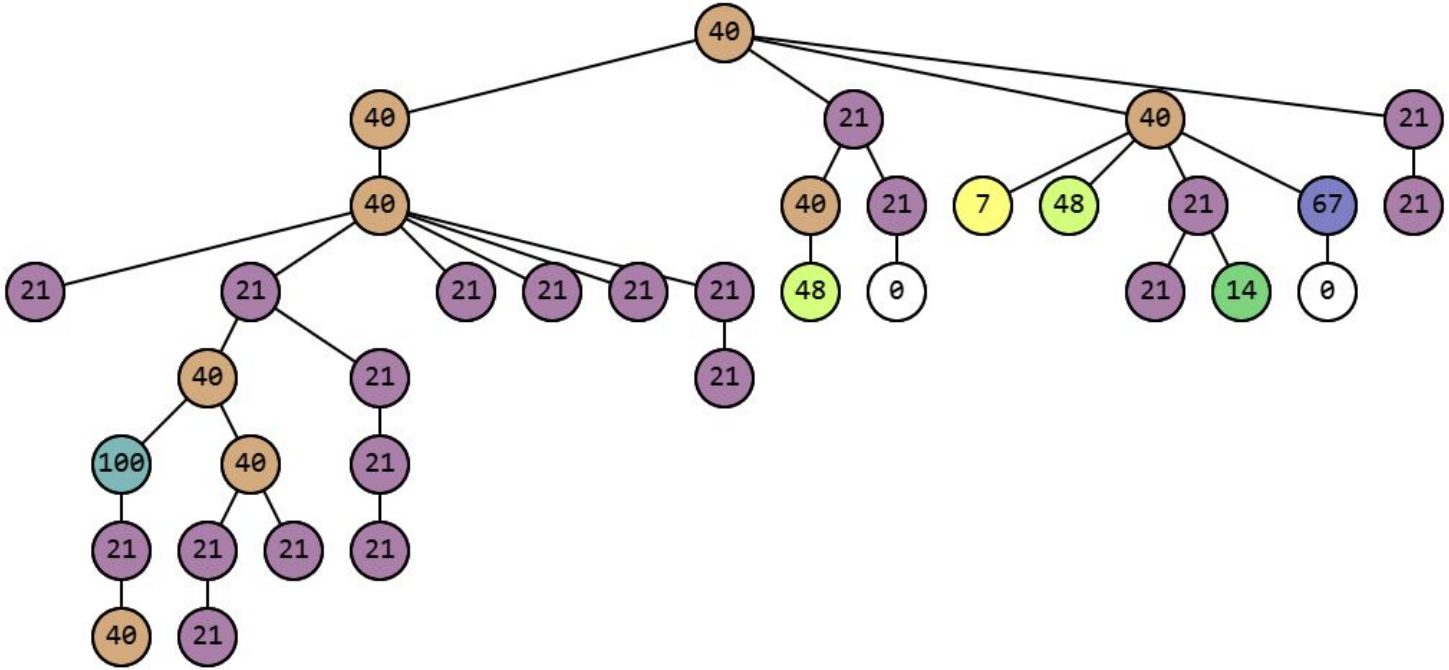


Final Answer

Subtask Overview

Subtask	Score	Tree Task
1	7	All Black
2	14	Multi-source BFS
3	27	Tree Diameter
4	19	Single Connected Component
5	33	Count Connected Components

Statistics


40
40
21
40
21
40
40
21
7
48
21
67
21
21
21
21
21
21
21
21
48
0
21
14
0
21
40
21
100
40
21
21
21
21
21
40
21

Subtask 1 (7%, All Black)

If **all the nodes are black**, the final answer is 1. Otherwise, final answer is 0.

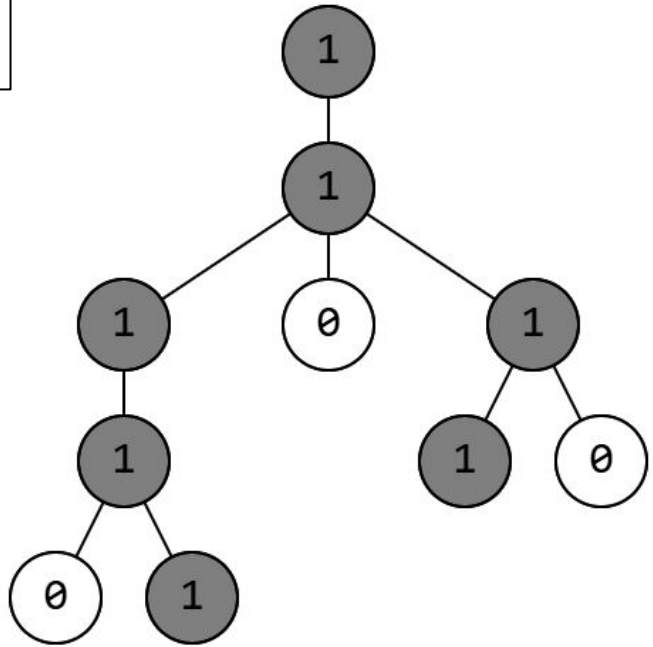
Free points for you to test the machine!

The solution is quite simple:

- If current node and all adjacent nodes is black, then keep it black.
- Otherwise, repaint it white.

Subtask 1 (7%, All Black)

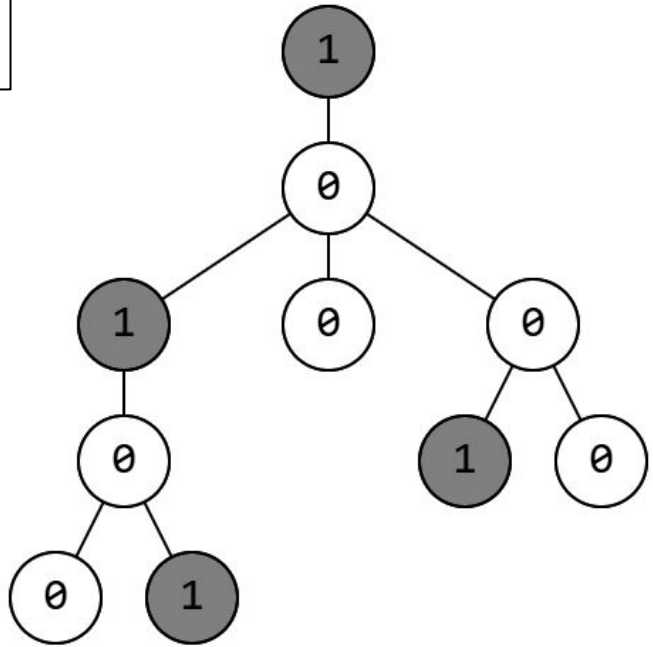
Case I. Exists White



T = 0

Subtask 1 (7%, All Black)

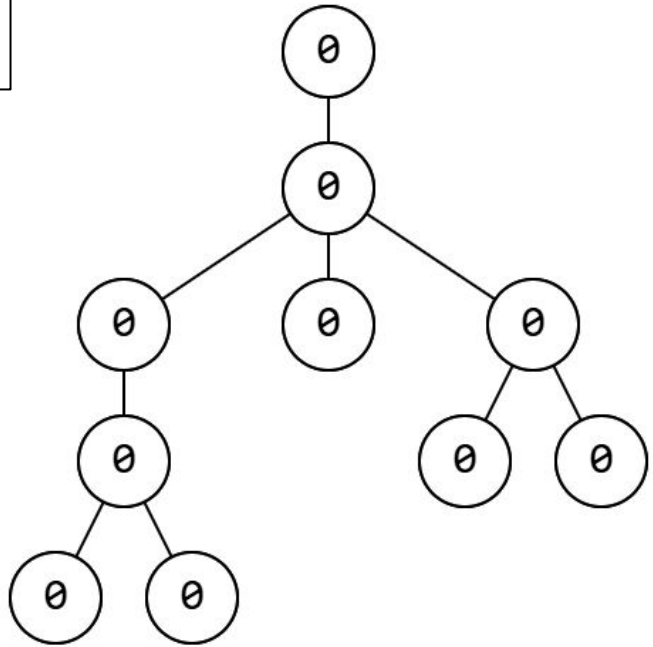
Case I. Exists White



T = 1

Subtask 1 (7%, All Black)

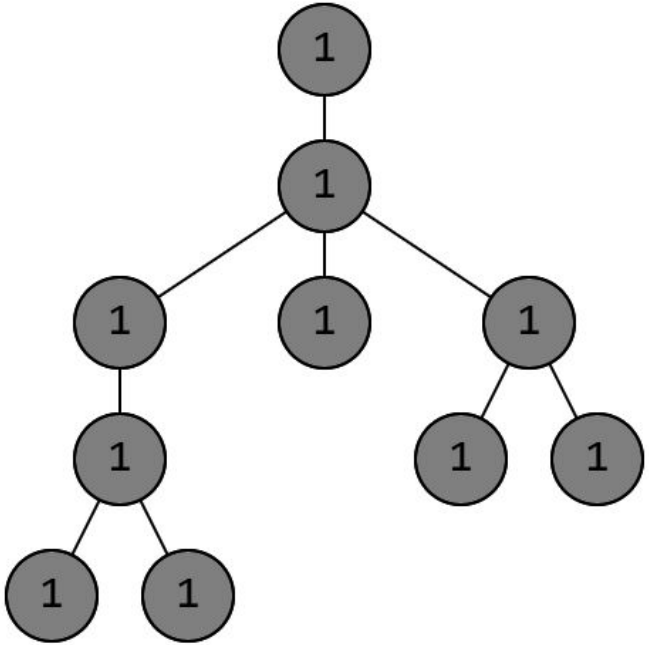
Case I. Exists White



T = 2

Subtask 1 (7%, All Black)

Case II. All Black



T = 0

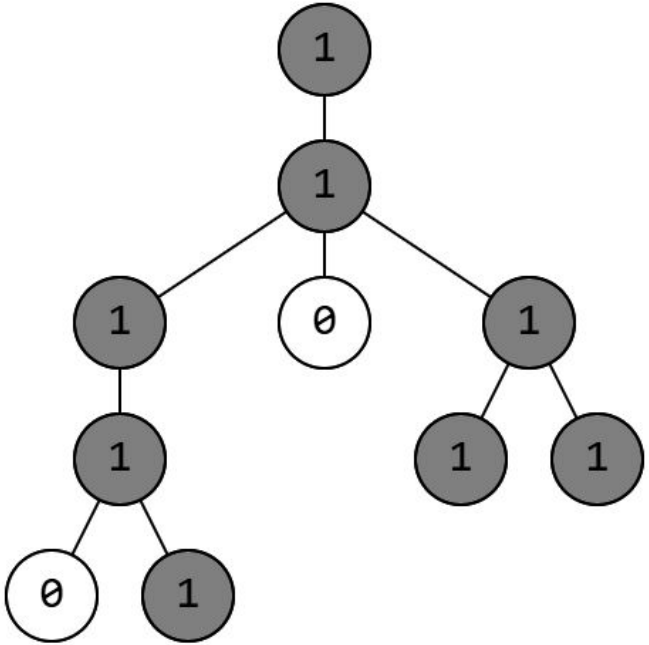
Subtask 2 (14%, Multi-source BFS)

For each node, the final answer is the **shortest distance to any white nodes**.

Again the solution is quite simple:

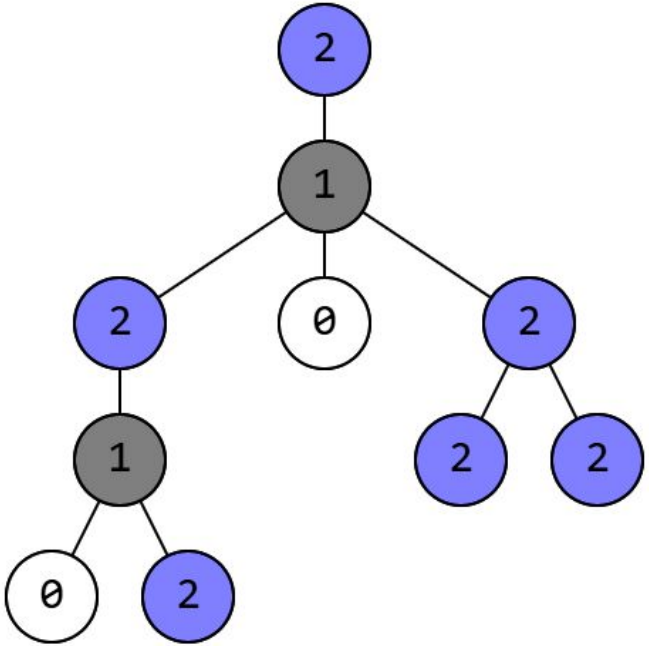
- Let's say `adj[v]` be the adjacent nodes of node v
- `A[v] := min{A[u] | u in adj[v]} + 1`

Subtask 2 (14%, Multi-source BFS)



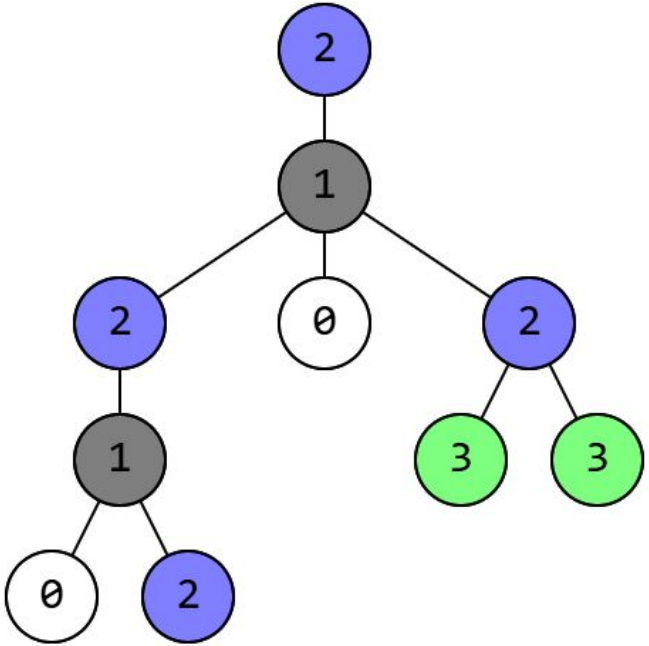
T = 0

Subtask 2 (14%, Multi-source BFS)



T = 1

Subtask 2 (14%, Multi-source BFS)



T = 2

Subtask 3 (27%, Tree Diameter)

Final answer is the **tree diameter**, the maximum distance between two nodes.

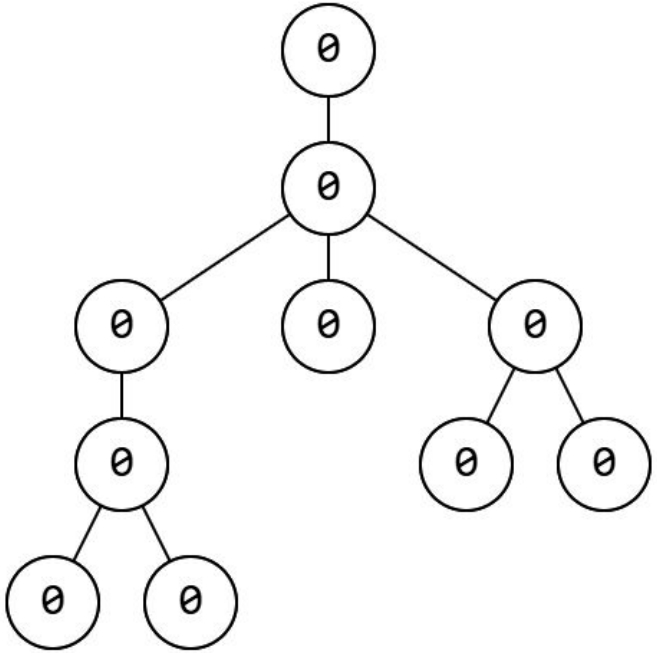
Note that you can paint the leaf nodes with colour 1 (black) and “propagate” the distances to every other nodes.

When “the distance meets”, we can assign a colour of `(100 + d)` where `d` is the answer. Slightly different handling for odd / even diameter length.

We can then spread this specific colour to everywhere, and simply answer `C[v] - 100` in `answer()` function.

Subtask 3 (27%, Tree Diameter)

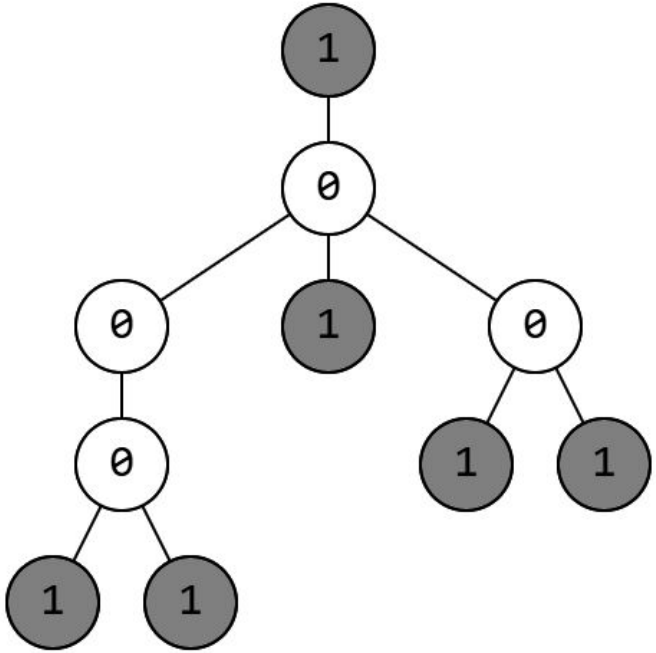
Odd Diameter



T = 0

Subtask 3 (27%, Tree Diameter)

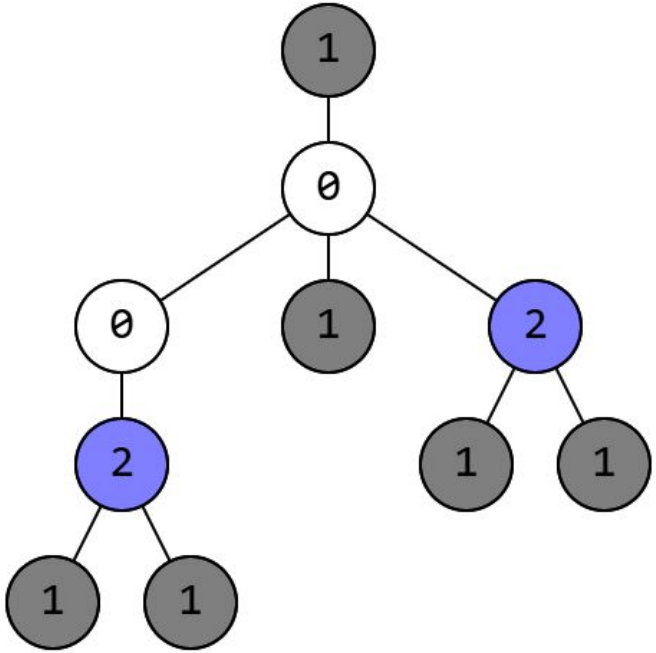
Odd Diameter



T = 1

Subtask 3 (27%, Tree Diameter)

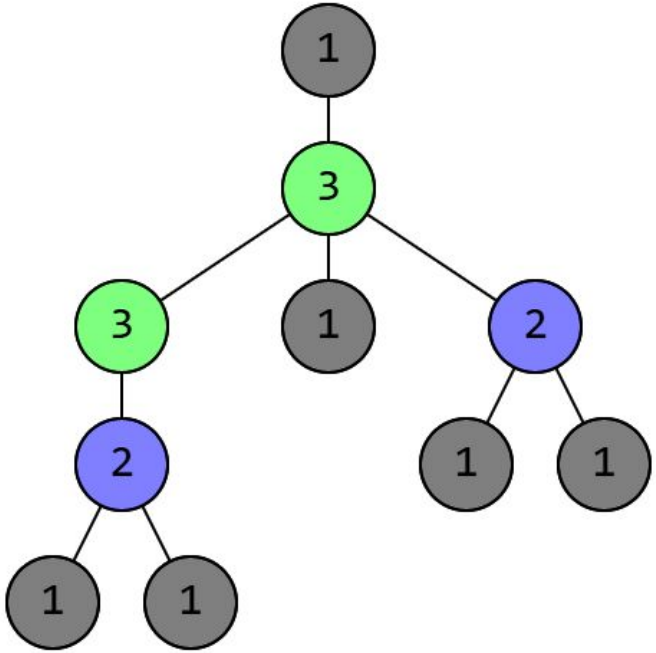
Odd Diameter



T = 2

Subtask 3 (27%, Tree Diameter)

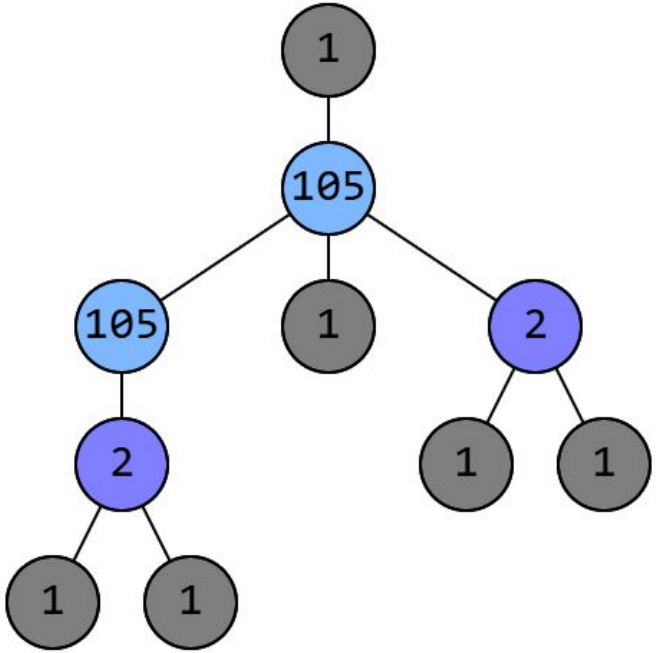
Odd Diameter



T = 3

Subtask 3 (27%, Tree Diameter)

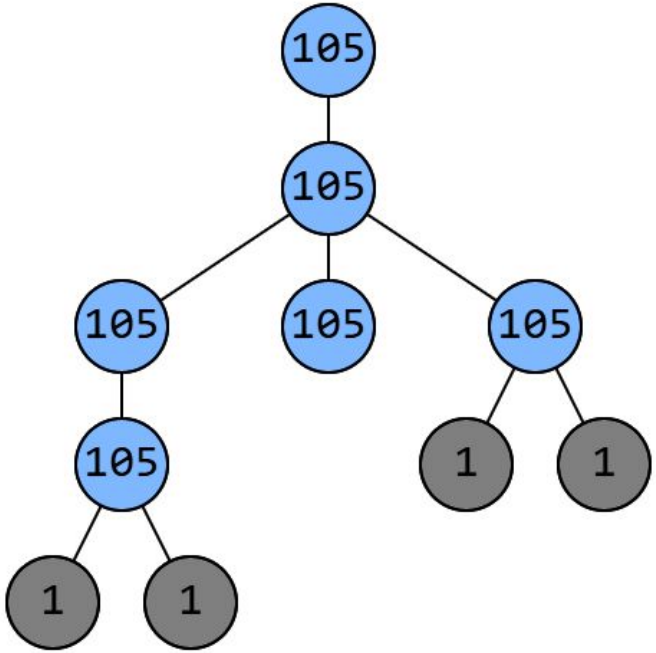
Odd Diameter



T = 4

Subtask 3 (27%, Tree Diameter)

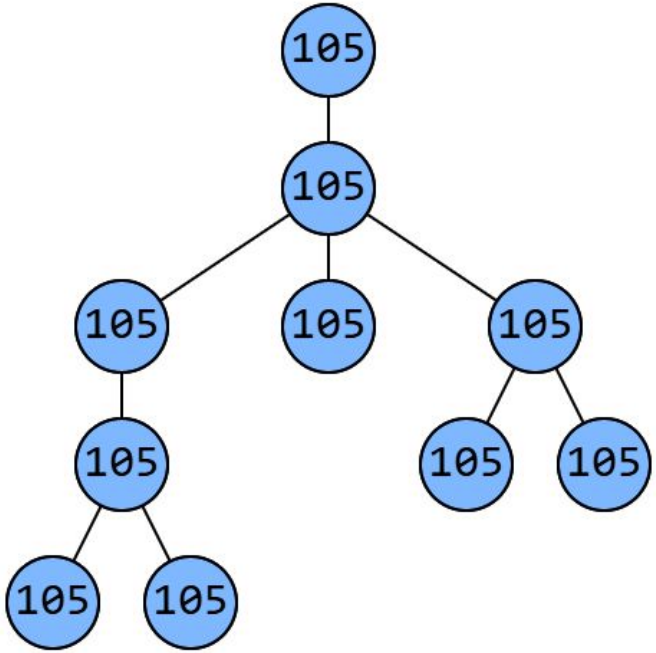
Odd Diameter



T = 5

Subtask 3 (27%, Tree Diameter)

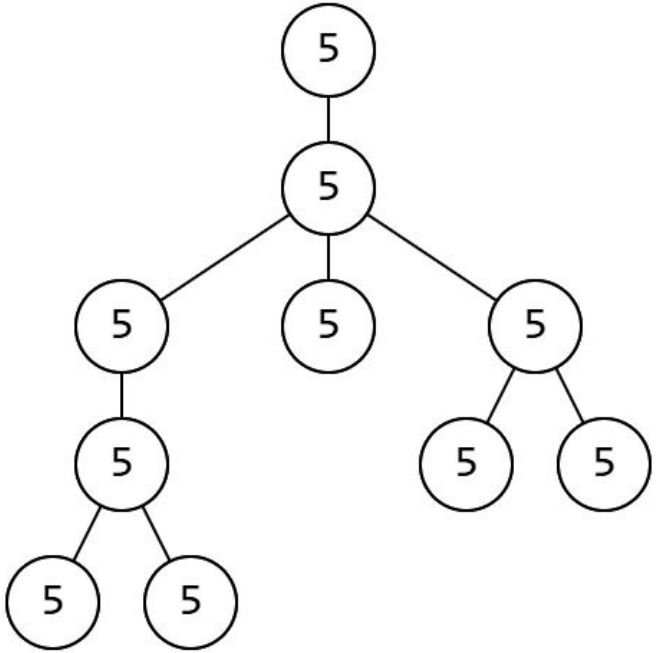
Odd Diameter



T = 6

Subtask 3 (27%, Tree Diameter)

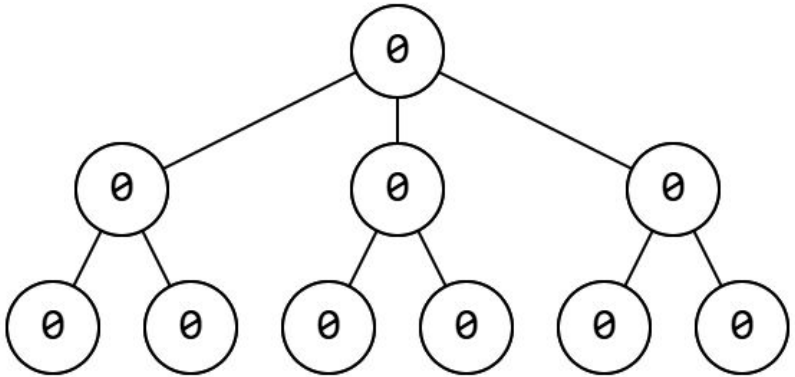
Odd Diameter



Final Answer

Subtask 3 (27%, Tree Diameter)

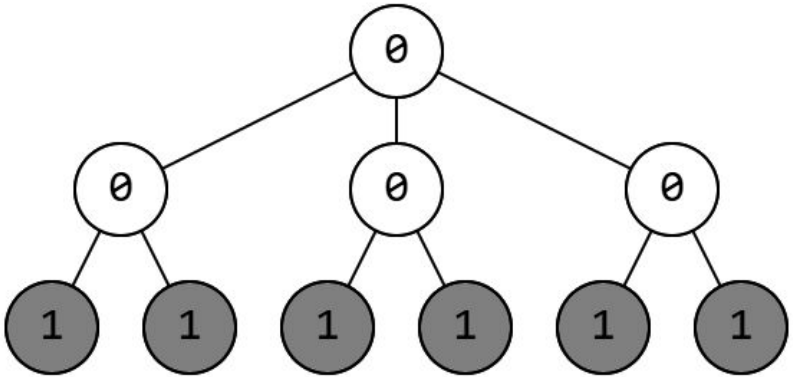
Even Diameter



T = 0

Subtask 3 (27%, Tree Diameter)

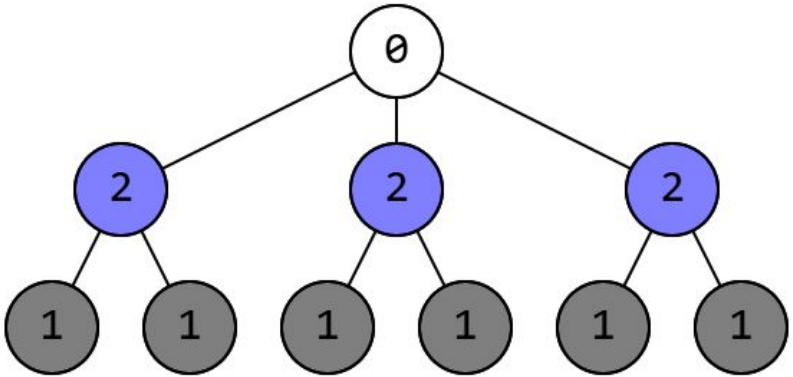
Even Diameter



T = 1

Subtask 3 (27%, Tree Diameter)

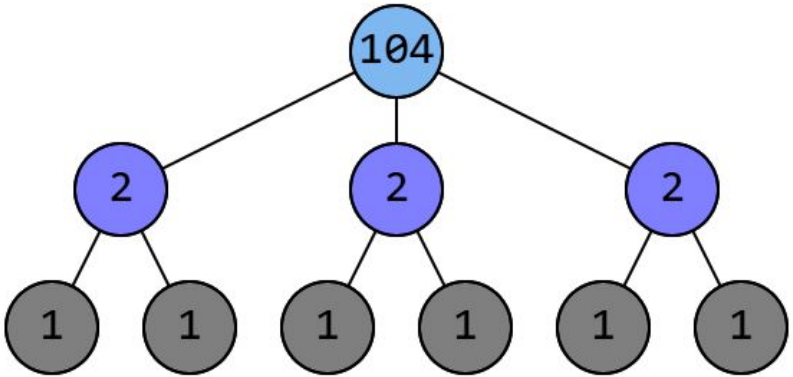
Even Diameter



T = 2

Subtask 3 (27%, Tree Diameter)

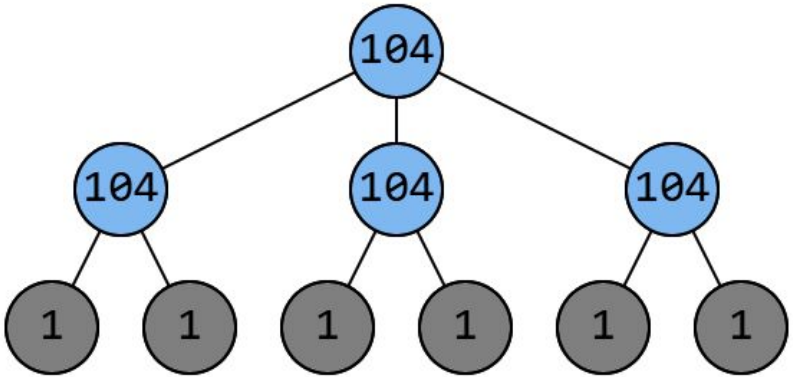
Even Diameter



T = 3

Subtask 3 (27%, Tree Diameter)

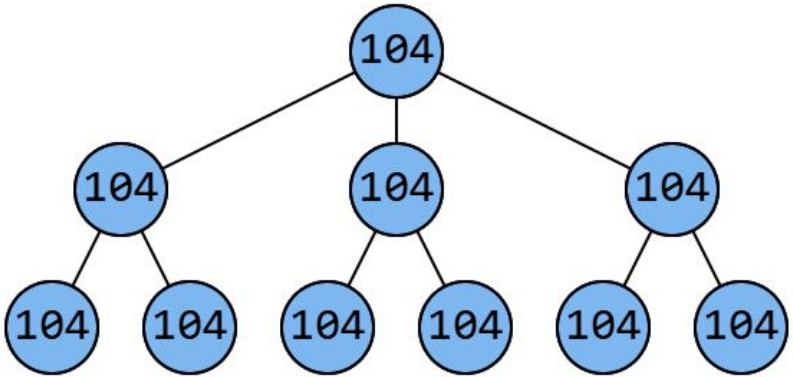
Even Diameter



T = 4

Subtask 3 (27%, Tree Diameter)

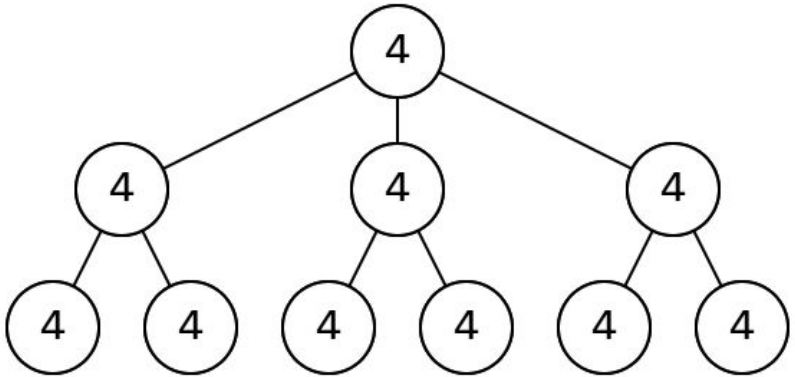
Even Diameter



T = 5

Subtask 3 (27%, Tree Diameter)

Even Diameter



Final Answer

Subtask 4 (19%, Single Connected Component)

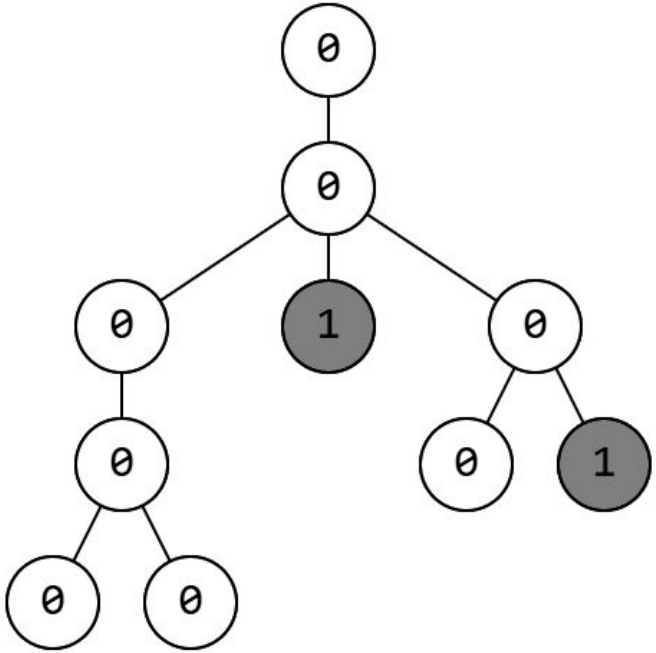
Final answer is whether the tree has a **single black connected component**.

Once again, you can spread the coloured black colours to adjacent nodes.

If we detect “merging” of two connected components, then assign and spread a special colour that denotes an answer of NO.

Subtask 4 (19%, Single Connected Component)

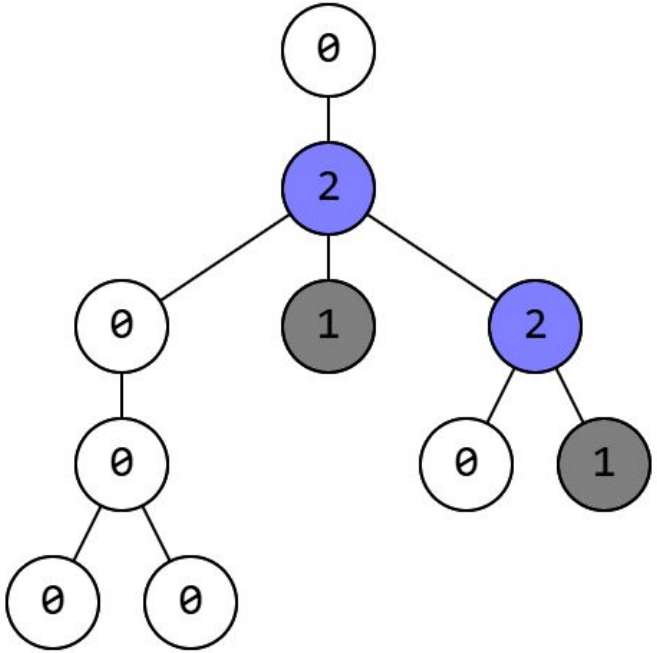
Answer: NO



T = 0

Subtask 4 (19%, Single Connected Component)

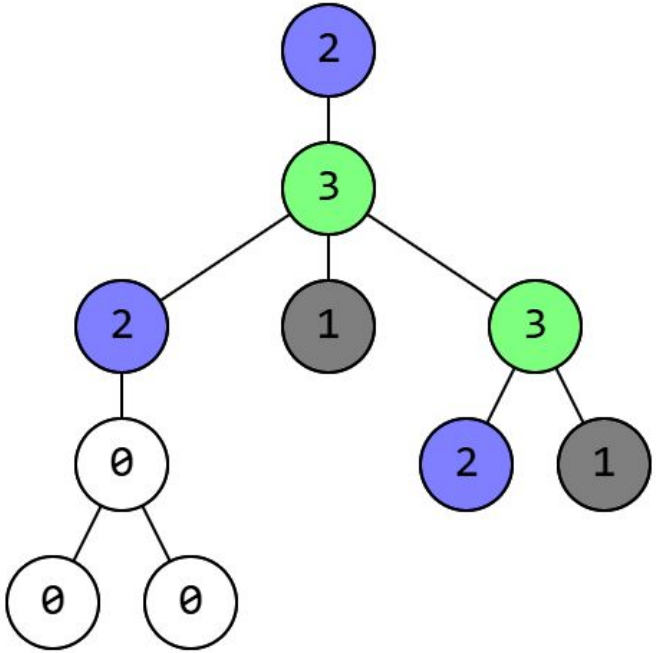
Answer: NO



T = 1

Subtask 4 (19%, Single Connected Component)

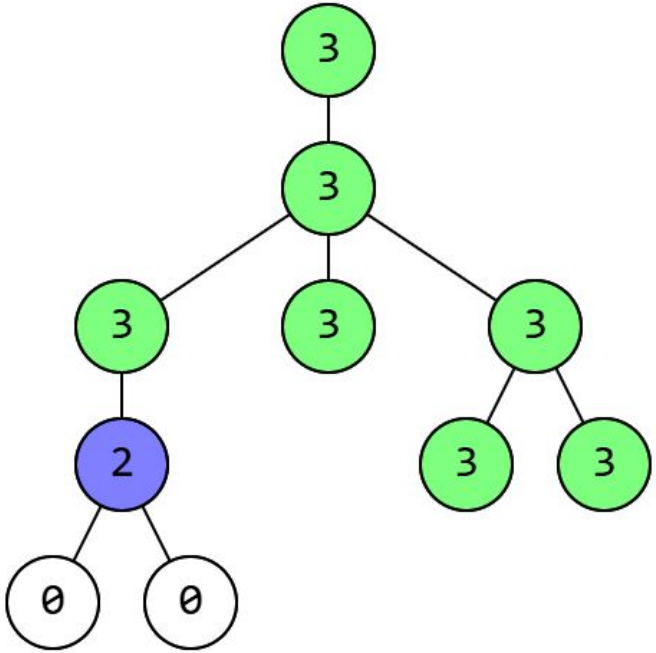
Answer: NO



T = 2

Subtask 4 (19%, Single Connected Component)

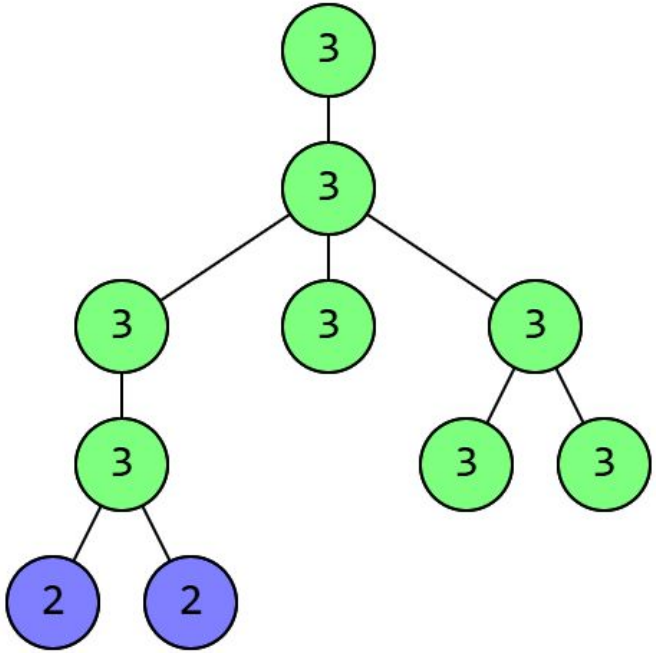
Answer: NO



T = 3

Subtask 4 (19%, Single Connected Component)

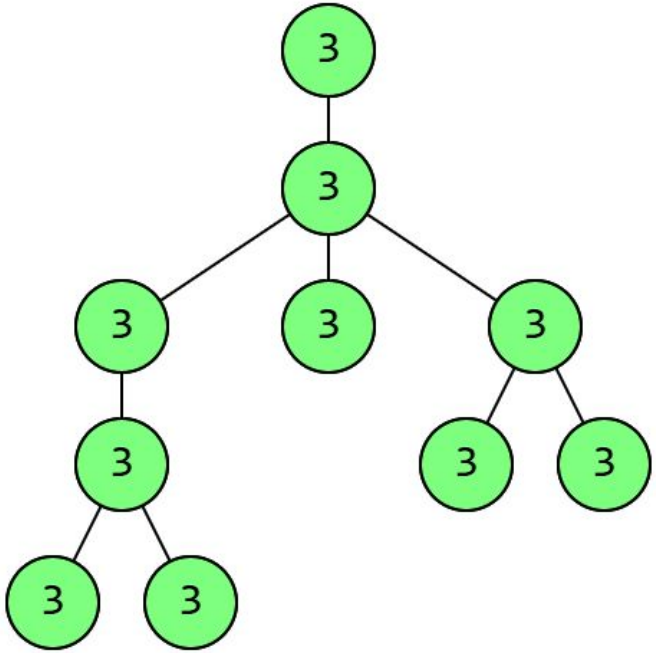
Answer: NO



T = 4

Subtask 4 (19%, Single Connected Component)

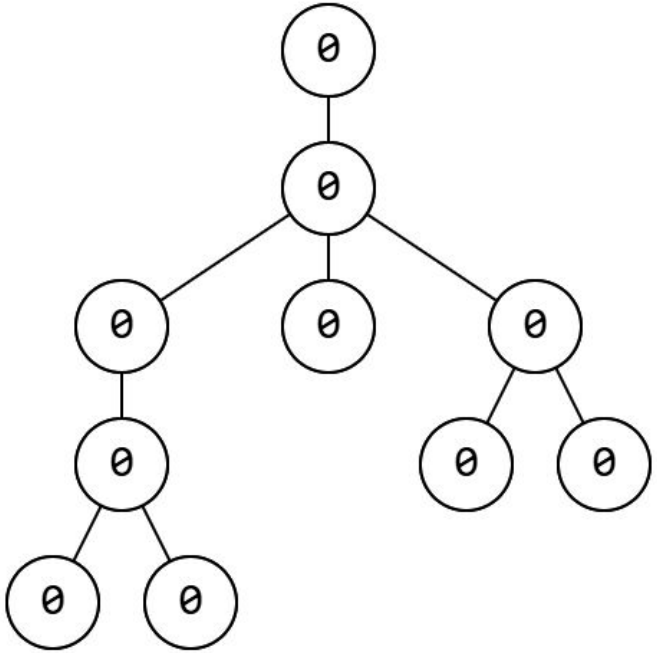
Answer: NO



T = 5

Subtask 4 (19%, Single Connected Component)

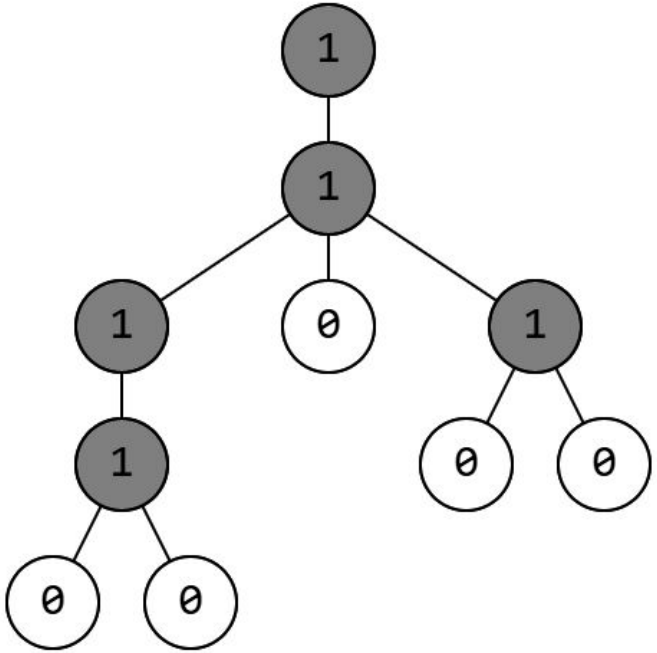
Answer: NO



Final Answer

Subtask 4 (19%, Single Connected Component)

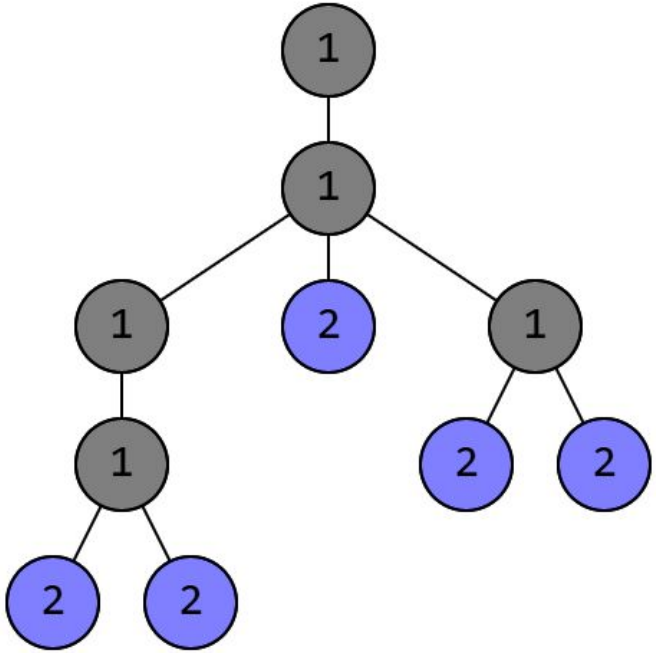
Answer: YES



T = 0

Subtask 4 (19%, Single Connected Component)

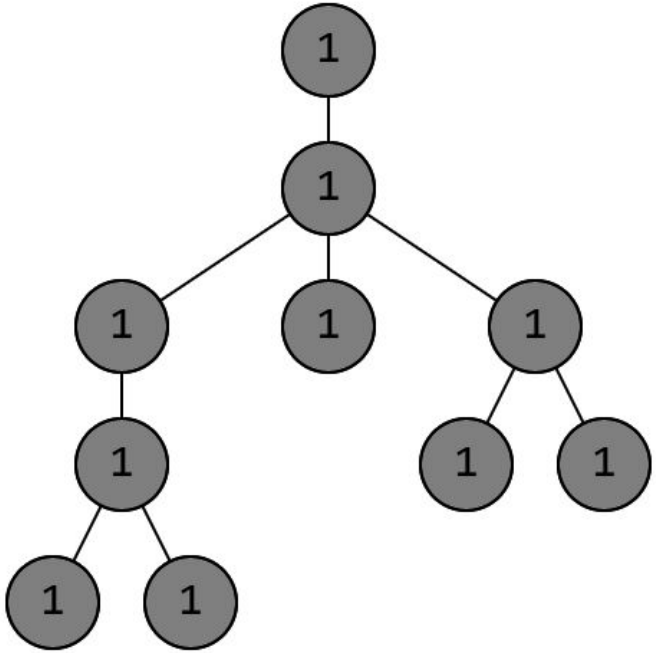
Answer: YES



T = 1

Subtask 4 (19%, Single Connected Component)

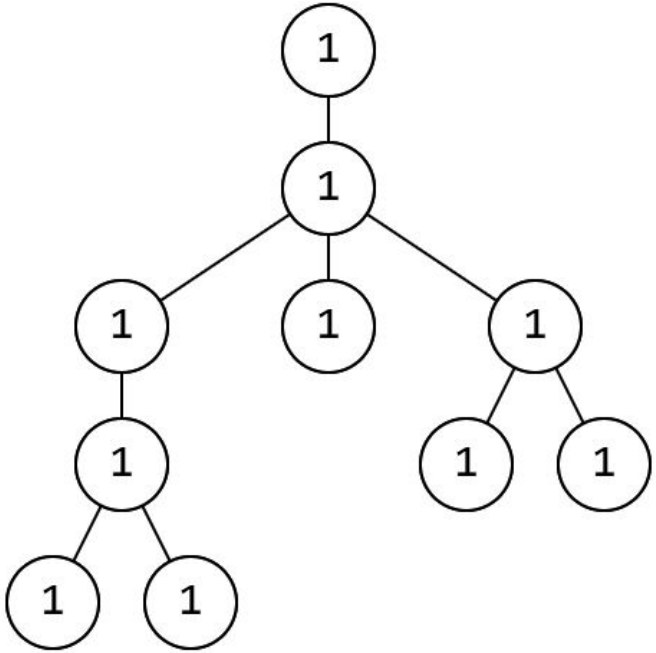
Answer: YES



T = 2

Subtask 4 (19%, Single Connected Component)

Answer: YES



Final Answer

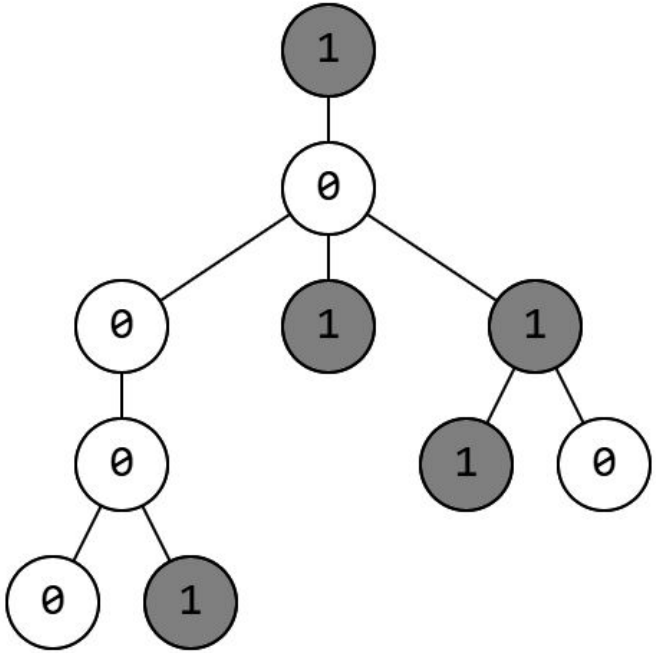
Subtask 5 (33%, Count Connected Components)

Count the **number of connected components** consisting of black nodes.

Let's say we encode the states in form of `(ANSWER, IS_BLACK, STATUS)`.

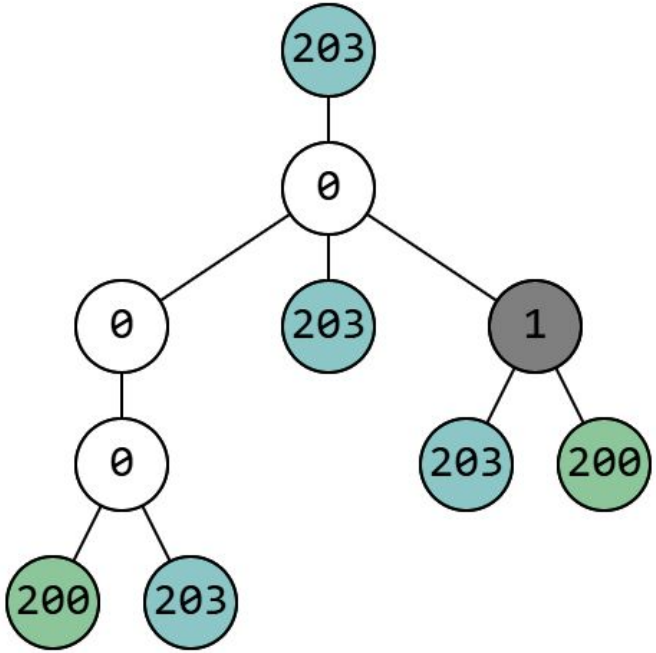
Then, notice that we can in fact do "dynamic programming" on the tree by propagating upwards just like calculating tree diameter, then propagate downwards the actual answer.

Subtask 5 (33%, Count Connected Components)



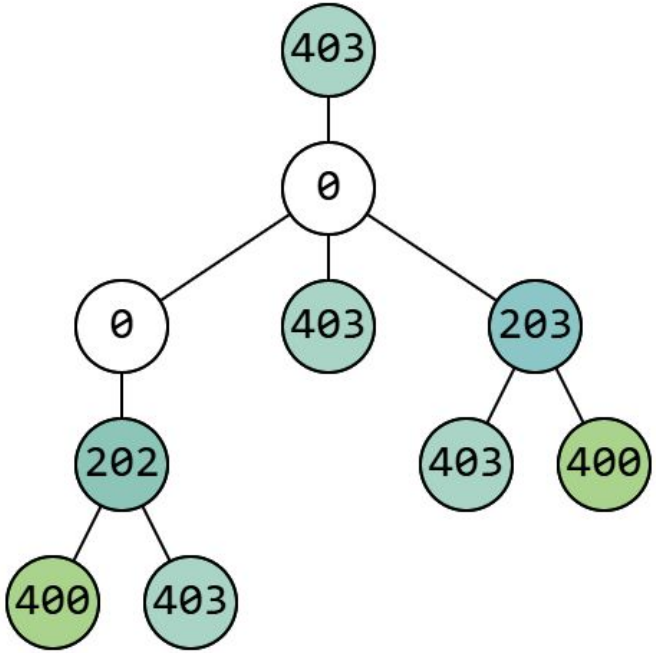
T = 0

Subtask 5 (33%, Count Connected Components)



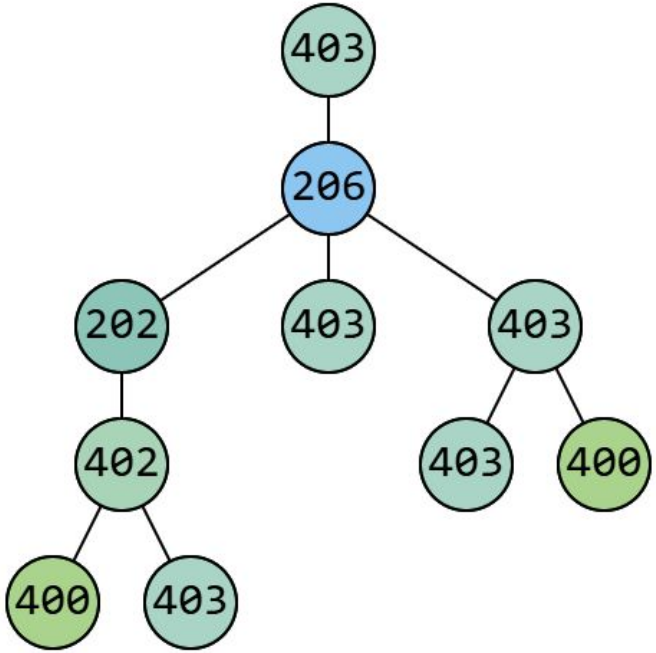
T = 1

Subtask 5 (33%, Count Connected Components)



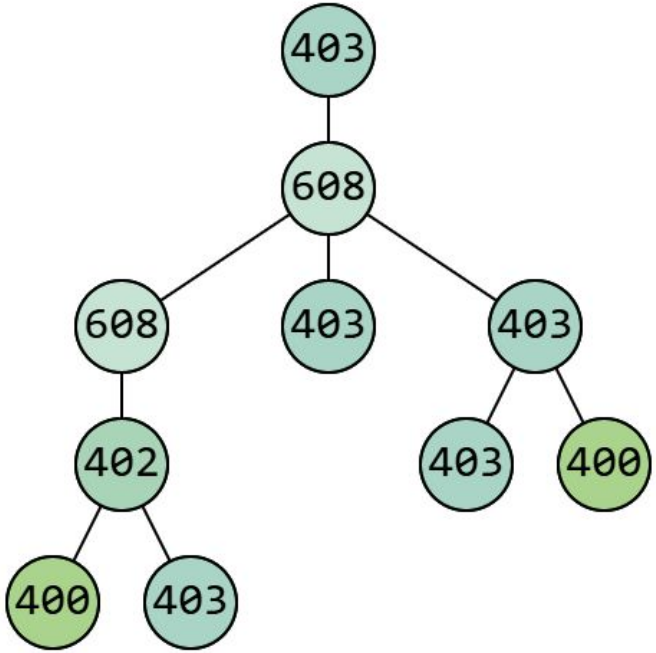
T = 2

Subtask 5 (33%, Count Connected Components)



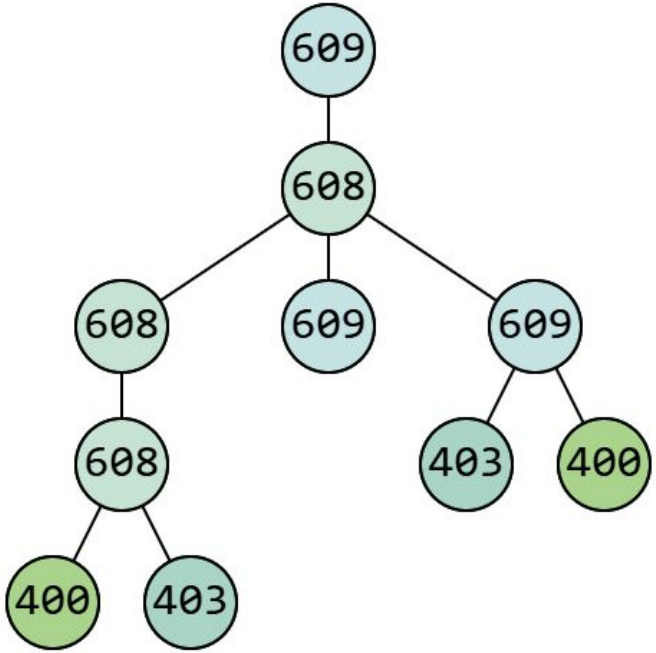
T = 3

Subtask 5 (33%, Count Connected Components)



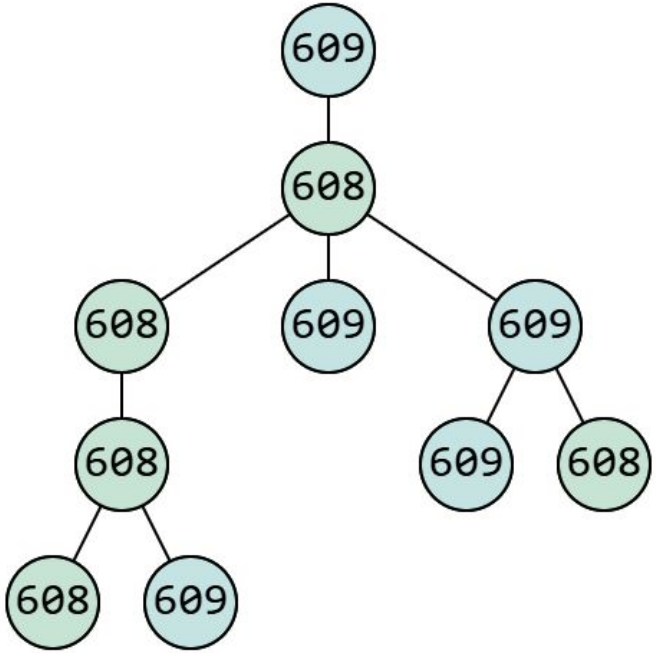
T = 4

Subtask 5 (33%, Count Connected Components)



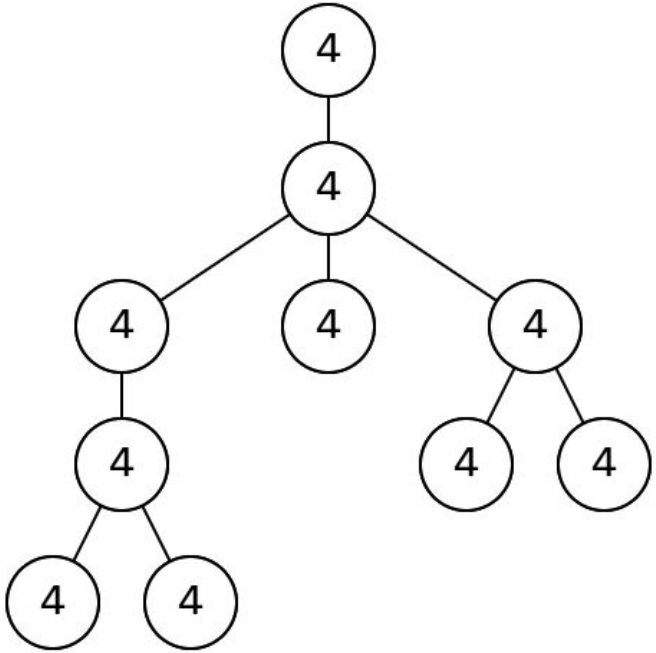
T = 5

Subtask 5 (33%, Count Connected Components)



T = 6

Subtask 5 (33%, Count Connected Components)



Final Answer

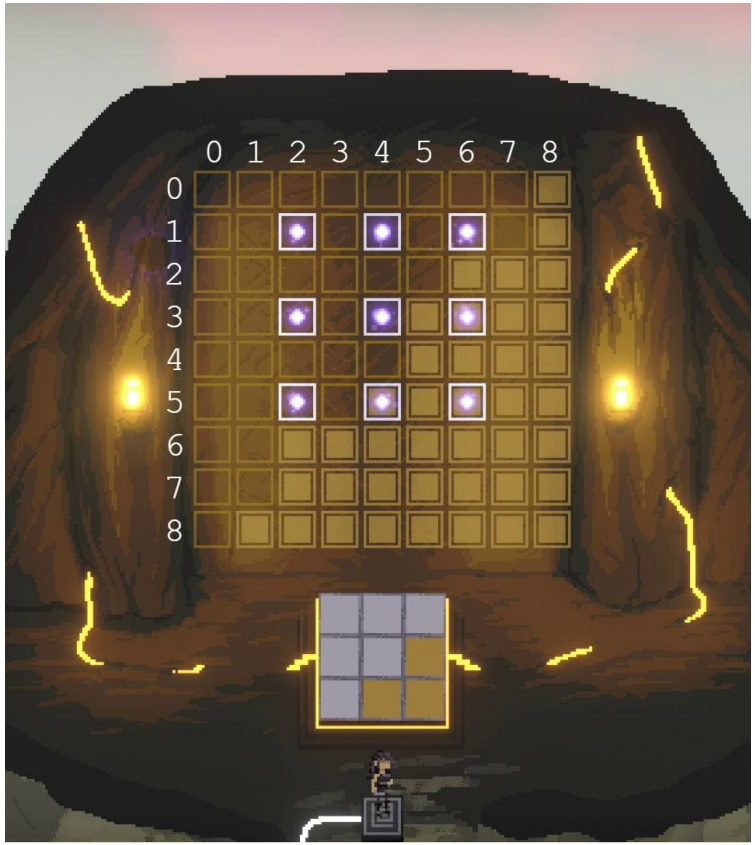
香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

M2552 The Last Puzzle

Bosco Wang {happypotato}
2025-07-01

The Problem

Given a 01 matrix of size N * N, and A[x][y] >= max(A[x][y-1], A[x-1][y])
Process Q queries: given a K * K 01 matrix, check if it appears as a "submatrix", i.e. there exists $(x_1, x_2, \ldots x_k)$, $(y_1, y_2, \ldots, y_k)$ such that $A[x_i][y_j] == query[i][j]$



Subtask Overview

Subtask	Score	Constraints
1	5	$K = 2$
2	5	$N = 3$
3	10	$N \le 7, Q \le 1000$
4	7	$A_0 = A_1 = \ldots = A_{N-1}$
5	8	$A_i = i$ for all $0 \le i < N$
6	20	$N, S, Q \le 100$
7	15	$N, S, Q \le 1000$
8	30	No additional constraints

Statistics

M2552 - The Last Puzzle	28	100	35.892	40.676	5: 11	5: 16	10: 12	7: 23	8: 8	20: 9	15: 9	30: 7

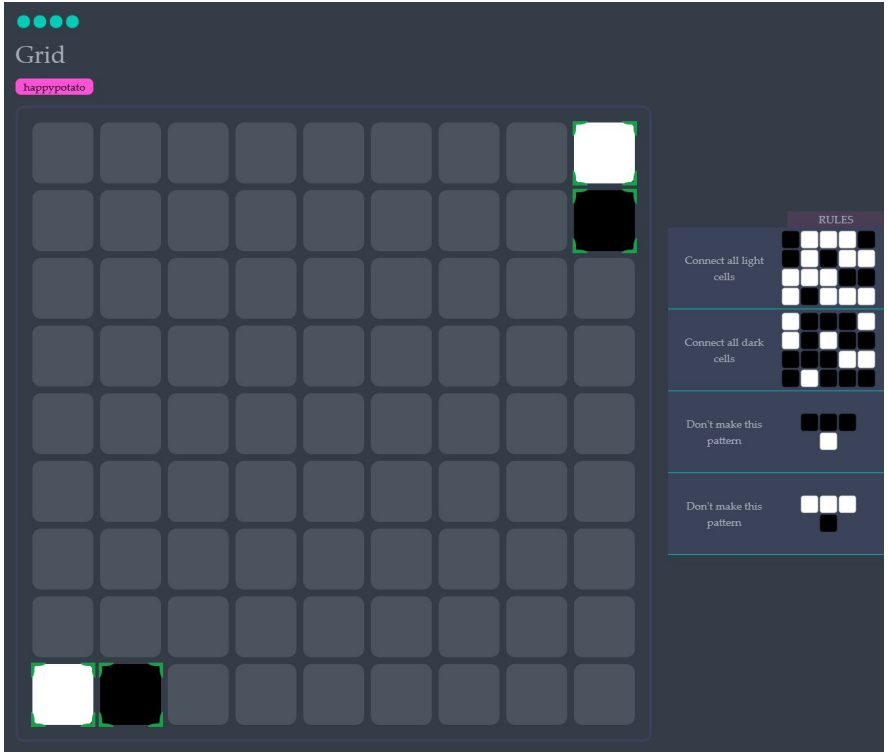
First AC by **yellowtoad** at **0:48**

References

The statement is a reference to the games Islands of Insight and Taiji.

They are both “logic grid”-based puzzle games. You can try an example here.

I highly recommend checking them out :)



Subtasks 1-3: Small cases

Subtask	Score	Constraints
1	**5**	**K = 2**
2	**5**	**N = 3**
3	**10**	**N ≤ 7, Q ≤ 1000**
4	7	$A_0 = A_1 = \ldots = A_{N-1}$
5	8	$A_i = i$ for all $0 \le i < N$
6	20	N, S, Q ≤ 100
7	15	N, S, Q ≤ 1000
8	30	No additional constraints

Subtasks 1, 2: K = 2

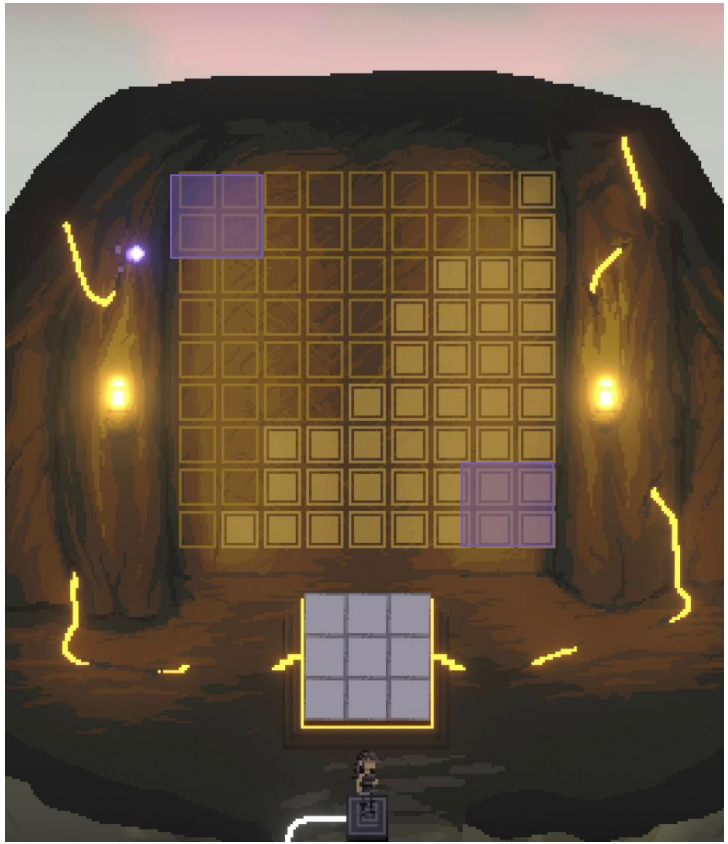
Actually, N = 3 implies K = 2 (since $2 \leq K < N$).

We can do casework on all possible inputs.
Here is one possible method to perform casework:

Subtasks 1, 2: K = 2

Case 1: `B = [0, 0] or [2, 2]`

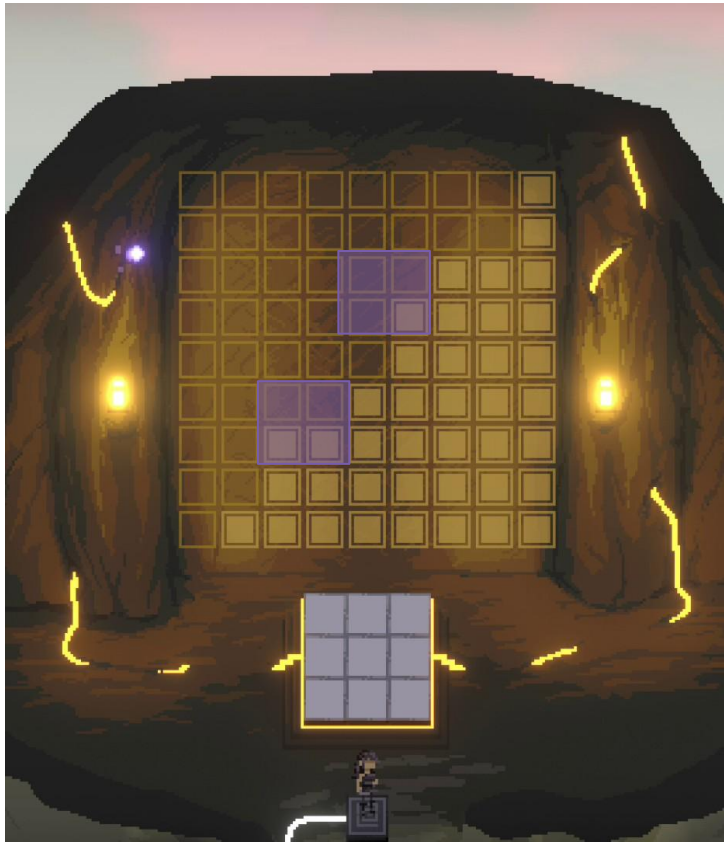
Then if a solution exists, it must be at the top left or bottom right respectively.



Subtasks 1, 2: K = 2

Case 2: `B = [0, 1] or [0, 2]`

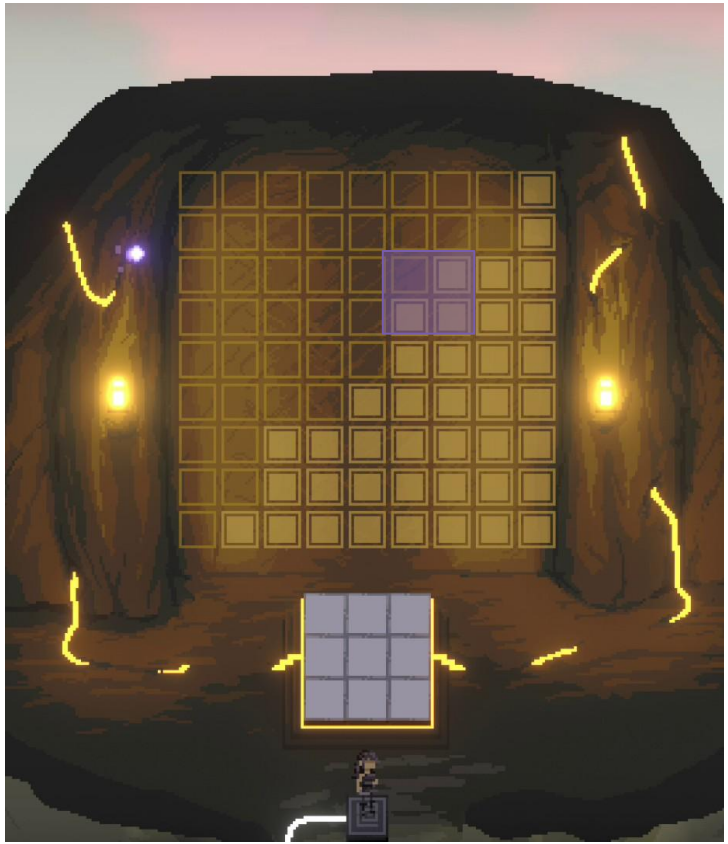
Then a solution exists if and only if there is some i such that $A_{i+1} - A_i >= 1$ or 2 respectively.



Subtasks 1, 2: K = 2

Case 3: `B = [1, 2]`

Then a solution exists if and only if there is some i such that $A_{i+1} - A_i >= 1$ AND $A_i >= 1$.

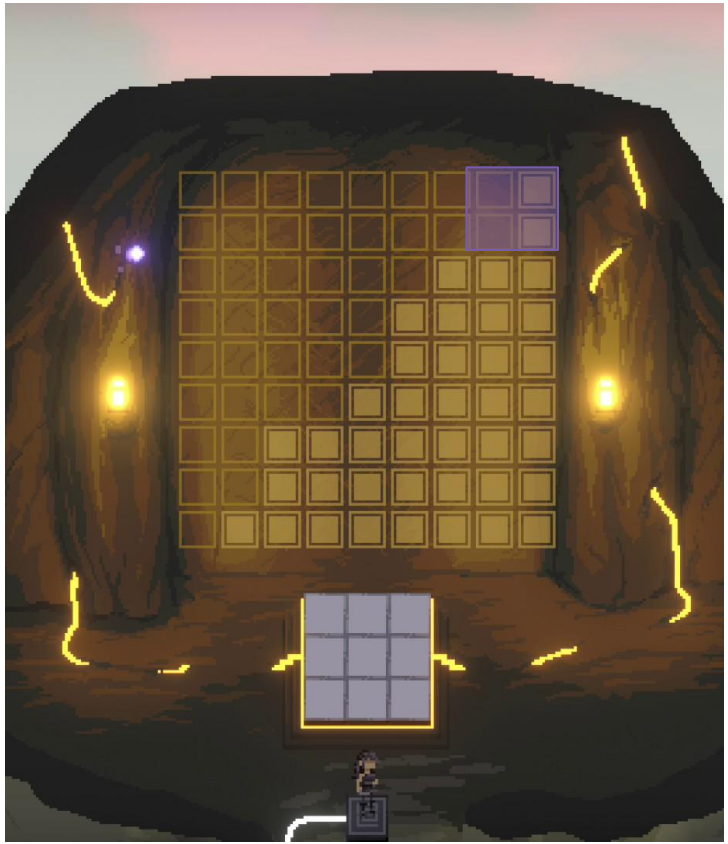


Subtasks 1, 2: K = 2

Case 4: `B = [1, 1]`

Then a solution exists if and only if there is some i such that $A_{i+1} = A_i$ AND $A_i >= 1$.

(Why can't this case be generalised with case 3?)



Subtasks 1, 2: K = 2

Handle all the above cases in O(N) time.

Time complexity: O(N + Q)

Expected score: 10

Subtask 3: N ≤ 7, Q ≤ 1000

N is small. We can brute force something.

For each query, brute over all the N choose K possibilities of both r and c, then check if it matches. Actually, you can even brute over all 2^N possibilities; it is fast enough.

Time complexity: O(2^2N * Q)

Expected score: 15 (Cumulative: 20)

Subtasks 4-5: Gathering observations

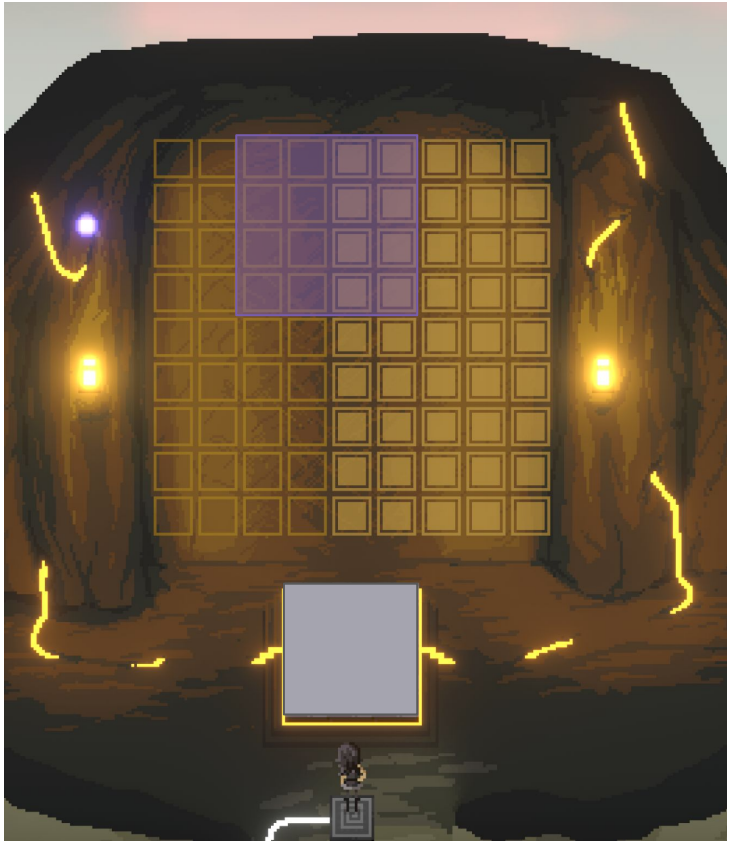
Subtask	Score	Constraints
1	5	$K = 2$
2	5	$N = 3$
3	10	$N \le 7, Q \le 1000$
4	**7**	$A_0 = A_1 = \ldots = A_{N-1}$
5	**8**	$A_i = i$ **for all** $0 \le i < N$
6	20	$N, S, Q \le 100$
7	15	$N, S, Q \le 1000$
8	30	No additional constraints

Subtask 4: $A_0 = A_1 = \ldots = A_{N-1}$

All valid submatrices must be in the form $B_0 = B_1 = \ldots = B_{K-1}$. After that, check if you have enough tiles for each row.

Time complexity: O(N + S)

Expected score: 7 (Cumulative: 37)



Subtask 5: $A_i = i$ for all $0 \le i < N$

Here, observe that the “difference” between each row is always 1.

Greedily pick the next row as early as possible.

Subtask 5: $A_i = i$ for all $0 \le i < N$

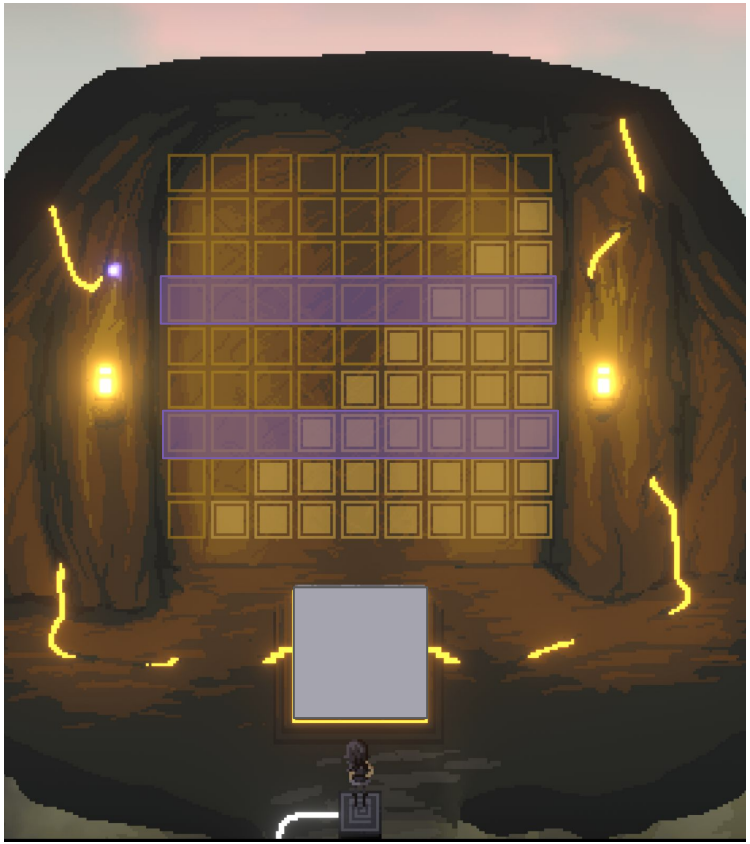
Since $A_i = i$, the next row you should take can be calculated beforehand.

Specifically, you should take the earliest row that has enough 0 to 1 transitions.

Expanding that condition, you get $r_{i+1} = r_i + \max(1, B_{i+1} - B_i)$.

Time complexity: $O(N + S)$

Expected score: 8 (Cumulative: 45)



More subtasks

- $B_0 = B_1 = \ldots = B_{K-1}$
  - Find the first row such that $A_i \geq B_0$.
  - Take the K consecutive rows starting from there.

- $B_i = i$ for all $0 \leq i < K$
  - Consider a similar argument for $A_i = i$.
  - The maximum value of K you can get is the number of $A_i \leq A_{i+i}$.

- This hints a more general observation...

Subtasks 6-8: General solution

Subtask	Score	Constraints
1	5	$K = 2$
2	5	$N = 3$
3	10	$N \leq 7, Q \leq 1000$
4	7	$A_0 = A_1 = \ldots = A_{N-1}$
5	8	$A_i = i$ for all $0 \leq i < N$
6	**20**	**$N, S, Q \leq 100$**
7	**15**	**$N, S, Q \leq 1000$**
8	**30**	**No additional constraints**

General observation

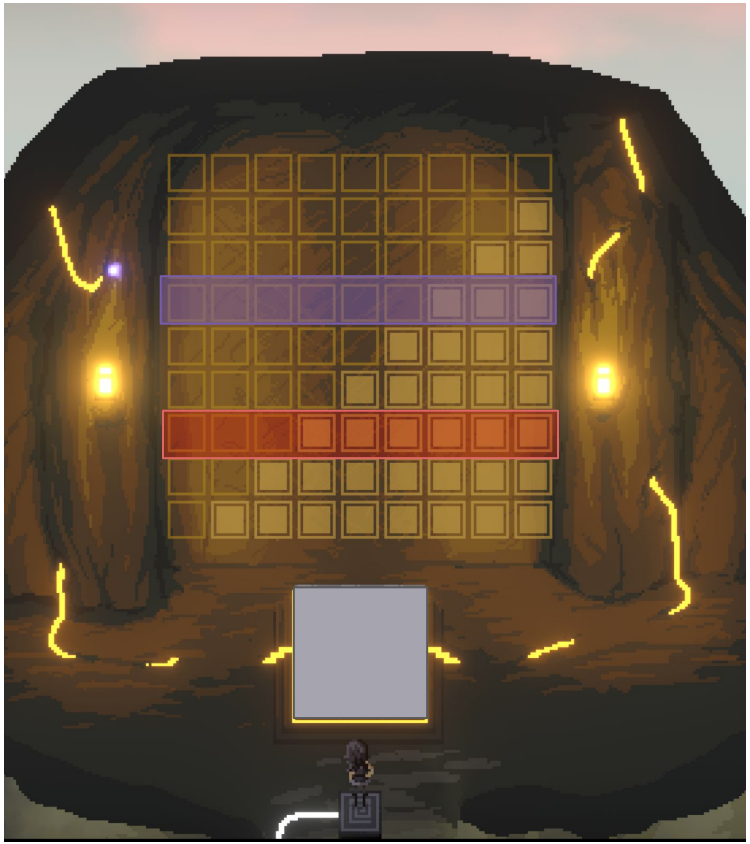
Observation 1: If a submatrix exists, we can find one by **greedily** matching its rows with the big matrix **as early as possible**.

I think this is extremely intuitive, to the point that you probably shouldn't bother proving it during contest. I will now show an outline of the proof.

Proof: Assume you took a later row and found a valid construction. We now try to show that using the greedy method also gives a valid construction.

General observation

Consider the following picture.



Observe that picking the “column set” for either row doesn’t change the value for other rows **below the selected row**. (The induction hypothesis assumes the rows above is valid already.)

Therefore, the construction for the lower row can be changed to the upper row.

General observation

Observation 1: If a submatrix exists, we can find one by **greedily** matching its rows with the big matrix **as early as possible**.

Note that you can apply the same argument on columns as well.

Actually, there is a stronger version of this:

General observation

Observation 2: This greedy algorithm works for every 2D prefix in B, i.e. For all "prefix solutions" ($r_{0..p}$, $c_{0..q}$), the greedy algorithm works best.

Proof: In the proof for observation 1, we actually showed that if there is some other "prefix solution" with ($r_{0..p}$, $c_{0..q}$), there exists another solution ($r'_{0..p}$, $c'_{0..q}$), generated by the greedy algorithm, such that $r'_p \leq r_p$ and $c'_q \geq c_q$.
Since both dimensions are better, this implies (r', c') is a **unique** optimal solution.

Solution 1: Sweep through rows

Briefly speaking, consider the difference array on A and B.

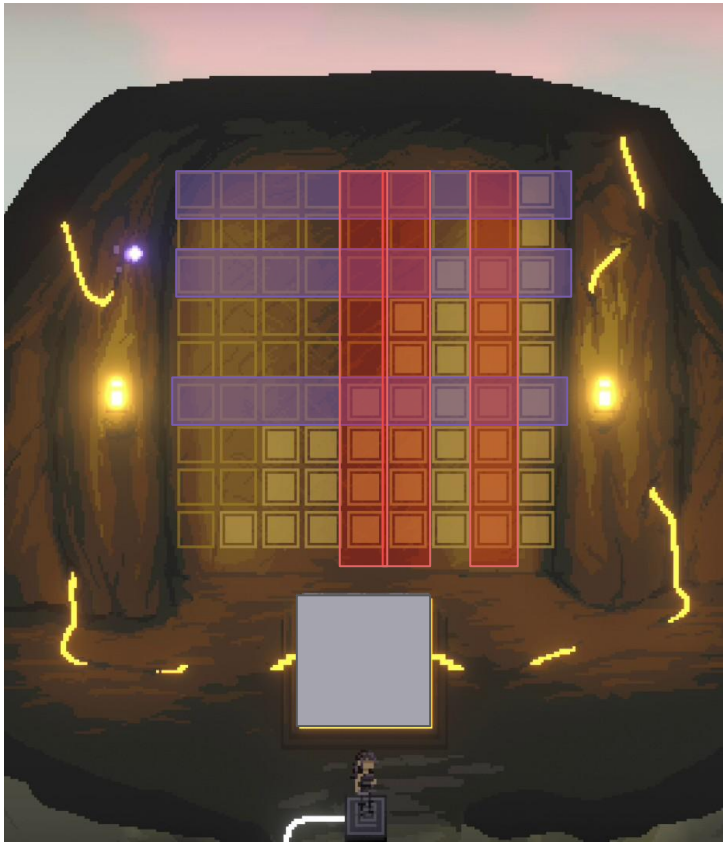
Consider calculating r_i given all its previous values. Then, we have the condition $A[r_i] - A[r_{i-1}] \geq B_i - B_{i-1}$ (and $r_i > r_{i-1}$). In words, we need at least $B_i - B_{i-1}$ places to switch from 0 to 1 when sweeping through rows.

For c_i, simply take the rightmost one you can take when B increments.

Solution 1: Sweep through rows

For example, let `B = [0, 1, 3]`, `A = [1, 1, 3, 4, 4, 5, 7, 7, 8]`.

- Previous = -1, $B_0 = 0$. Pick $r_0 = 0$.
- Previous = 0, $B_1 - B_0 = 1$.
  Need $A[r_1] - A[r_0] \geq 1$. Pick $r_1 = 2$, $c_2 = 7$.
- Previous = 1, $B_2 - B_1 = 2$.
  Need $A[r_2] - A[r_1] \geq 2$. Pick $r_2 = 5$, $c_1 = 5$, $c_0 = 4$.
- Final answer: r = (0, 1, 3), c = (4, 5, 7).



Solution 1: Sweep through rows

Use a binary search to find the smallest possible r_i, and just use a pointer to find the values of c_i.

Depending on implementation, you might have to handle some corner cases ($B_0 = B_1 = \ldots = B_{K-1} = 0/N$, $A_0 = A_1 = \ldots = A_{N-1} = 0/N$ etc.)

Time complexity: $O(N + S \log N)$

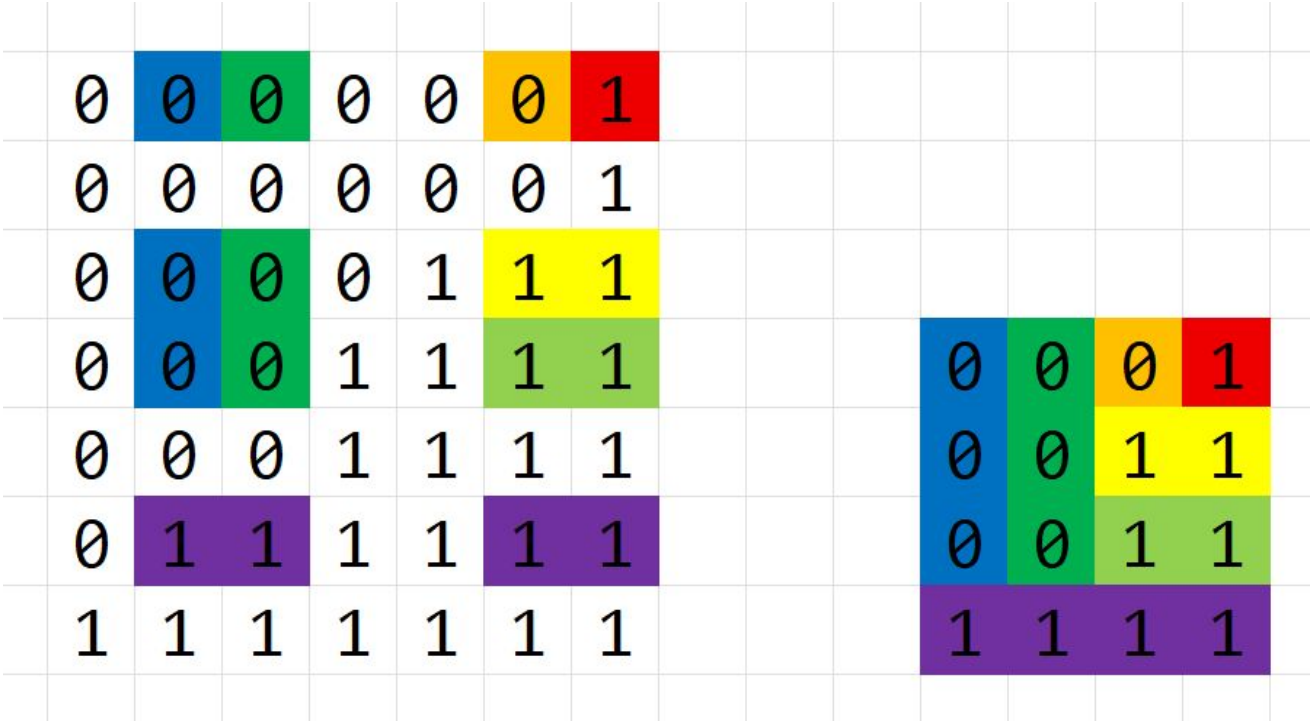
Expected score: 100

Solution 2: Walk along both directions

Alternatively, we can do the induction **simultaneously** on both directions!

This is better because the “jump” is easier to calculate. You also don’t need the second observation to prove its correctness.

Solution 2: Walk along both directions



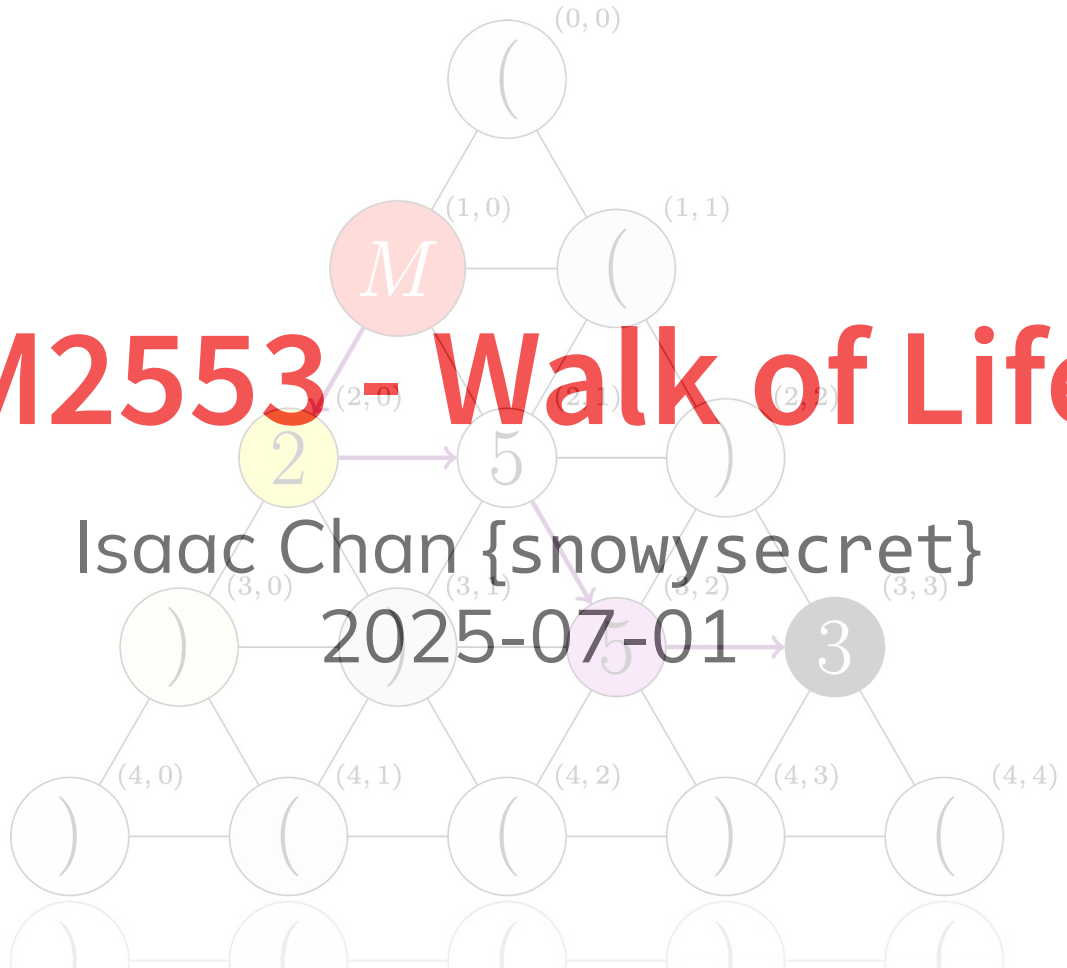
Solution 2: Walk along both directions

For each direction, simply maintain the position of the “cuts”. Then each “jump” can be computed in O(1) time.

Time complexity: O(N + S)

Expected score: 100

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



M2553 - Walk of Life

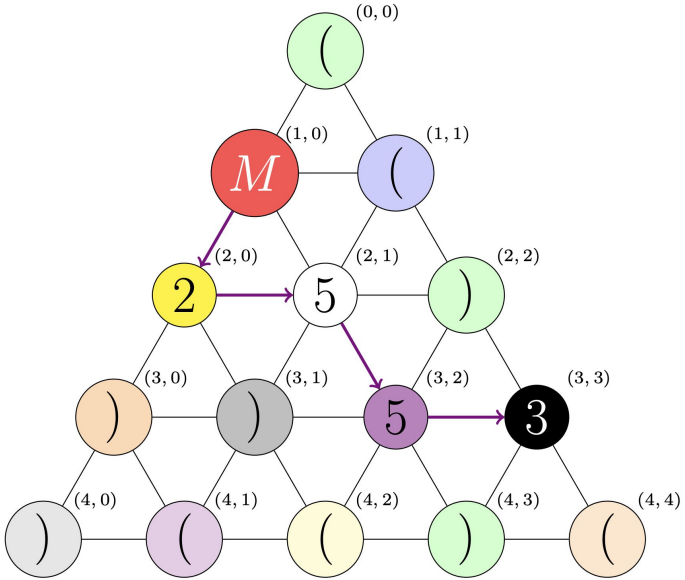
Isaac Chan {snowysecret}

2025-07-01

Background

Problem idea by snowysecret
Preparation by snowysecret
Presented by snowysecret

This problem was thought of in 2023 (back in my trainee days) 🧓



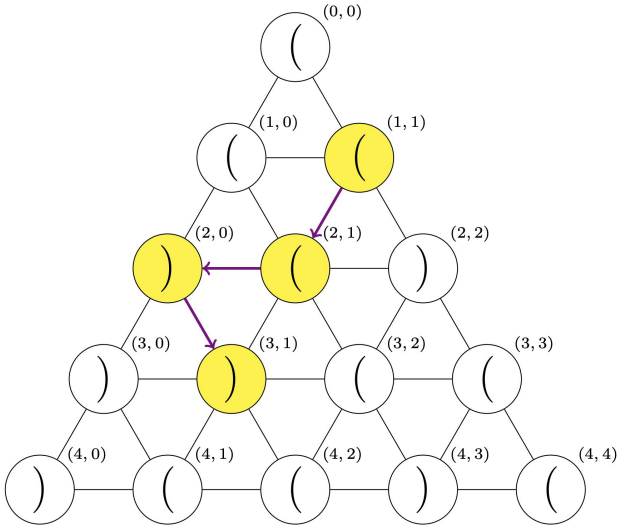
Story inspired by [PC2325 Walk of Love] by ethening

(Note: I am actually not familiar with philosophy. Please tolerate mistakes made in the first few lines.)

The Problem

Given a triangular grid of N layers, where each vertex comes with a single bracket.

Given a starting vertex (S_r, S_c) and an ending vertex (E_r, E_c), find a walk from (S_r, S_c) to (E_r, E_c) such that the bracket sequence formed by the characters of vertices on the walk is a **regular bracket sequence**. Or determine it is impossible to do so.



Subtasks

	Points	Constraints
1	4	N is even $(S_r, S_c) = (0, 0)$ $E_r = N - 1$ $B[r][c] =$ `(` if r is even, and $B[r][c] =$ `)` if r is odd
2	14	$S_r + 3 \leq E_r < N - 1$ $B[E_r][E_c] = B[E_r + 1][E_c] = B[E_r + 1][E_c + 1] =$ `)` All other vertices have $B[r][c] =$ `(`
3	15	$S_r + 2 \leq E_r < N - 1$ $B[E_r][E_c] = B[E_r + 1][E_c] = B[E_r + 1][E_c + 1] =$ `)` $B[S_r][S_c] = B[S_r + 1][S_c] = B[S_r + 1][S_c + 1] =$ `(`
4	24	It is always possible to construct a walk of life There exists a walk of life, such that if we write down the bracket on each vertex in the walk in that order, the resulting bracket sequence is `()()...()`, i.e. alternating between open and close brackets
5	17	$S_r + 2 \leq E_r < N - 1$ $B[E_r][E_c] = B[E_r + 1][E_c] = B[E_r + 1][E_c + 1] =$ `)`
6	26	No additional constraints

Partial Score Available: 45%, if you successfully distinguish between Possible and Impossible, **and** construct a walk such that its resulting bracket sequence can be turned into regular by adding zero or more closed brackets at the end.

(Non-Serious) Statistics

As we all know, breakfast is the most important meal of the day.

What is the effect of breakfast ordered on your score on this task?

1. Effect of Main on Score

Main Cat.	Q3 score ▼
Twisty Pasta	41.583333333333336
McGriddle	35.003846153846155
Hotcake item	33.375
Big Breakfast set	27.099999999999998
Burger / Muffin	27
Filet-O-Fish	18.5
Skip	11

2. Effect of Side on Score

Side Cat.	Q3 score
Hash Browns	41.375
Corn Cup	27.15
Yoghurt	11.1
Skip	7.659999999999999

3. Effect of Drink on Score

Drink Cat.	Q3 score ▼
Juice	61
Tea	57.4375
Hot Chocolate	33.45909090909091
Soft drink	29.060000000000002
Milk	18.5
Skip	16.5
Coffee (McCafé)	8.005555555555555

Actual Statistics

M2553 - Walk of Life	29	100	36.832	29.891	4: 27	14: 18 6.3: 3	15: 14 6.75: 3	24: 12	17: 7	26: 2

First solved by **sms27098** at 2h 35min

Revision: Regular Bracket Sequence

Definition (Regular Bracket Sequence). A string consisting of only brackets, such that this sequence, when inserted 1 and +, gives a valid mathematical expression.

Revision: Regular Bracket Sequence

Definition (Regular Bracket Sequence). A string consisting of only brackets, such that this sequence, when inserted 1 and +, gives a valid mathematical expression.

Properties of RBS:

1. The empty string is a regular bracket sequence.
2. If **s** is a regular bracket sequence, then so is **(s)**.
3. If **s** and **t** are regular bracket sequences, then so is **st**.
4. If S is a regular bracket sequence, then for every prefix of S, the number of open brackets, cnt_open, is **at least** the number of closed brackets, cnt_closed.
5. If S is a regular bracket sequence, the number of open brackets equals the number of closed brackets.

“Almost” Regular Bracket Sequence

The problem introduced the concept of “**almost** walk of life”, which basically means the resulting bracket sequence can be **turned into regular** by adding zero or more closed brackets at the end.

What are the properties of such a sequence?

"Almost" Regular Bracket Sequence

The problem introduced the concept of "**almost** walk of life", which basically means the resulting bracket sequence can be **turned into regular** by adding zero or more closed brackets at the end.

What are the properties of such a sequence?

Recall properties 4 and 5 of RBS:

Properties of "Almost RBS":

4. If S is a regular bracket sequence, then for every prefix of S, the number of open brackets, cnt_open, is **at least** the number of closed brackets, cnt_closed.
5. ~~If S is a regular bracket sequence, the number of open brackets equals the number of closed brackets.~~

Surplus

Throughout this solution explanation we will use the word “surplus” quite a lot.

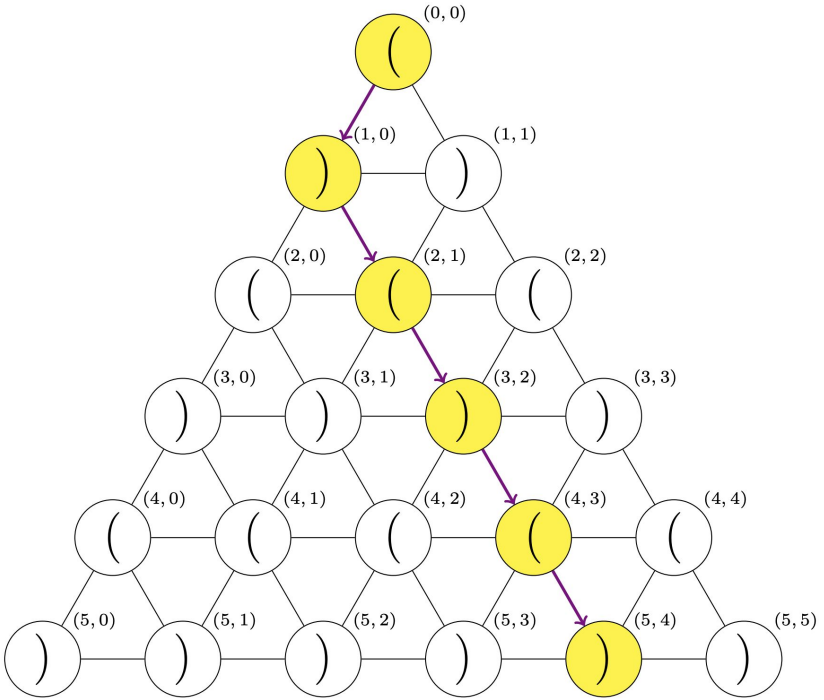
Surplus:
Given a bracket sequence S that can be possibly turned into an RBS by inserting closed brackets at the end, its **surplus** is equal to the number of ‘(‘s it contains **minus** the number of ‘)’s it contains.

E.g. the surplus of **(()()** is 1, the surplus of **(((** is 3.

Subtask 1 (4%): N even, (Sr, Sc) = (0, 0), Er = N – 1, B[r][c] = '(' iff r even

The triangular grid looks like this:

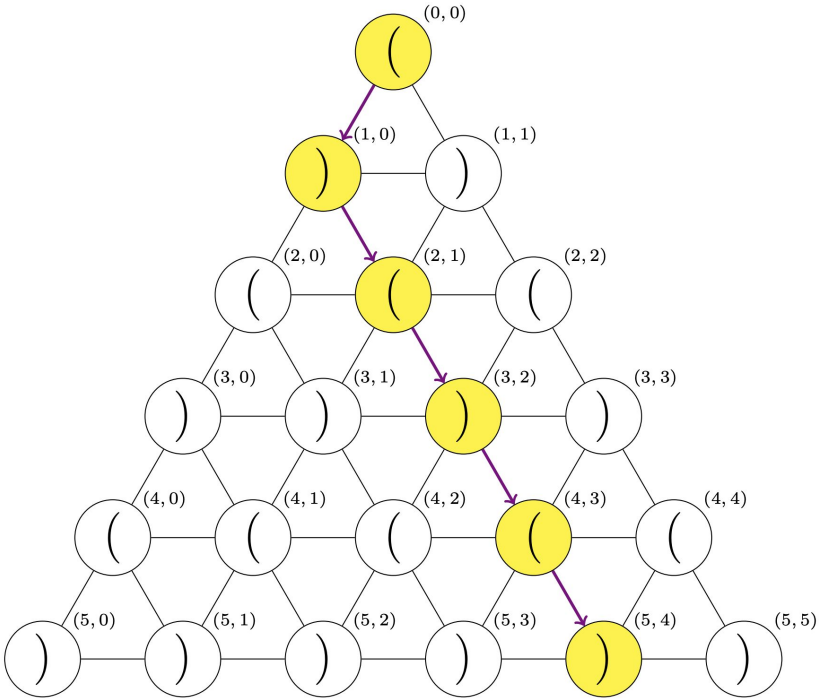
Under these constraints, a simple construction is to go from (0, 0) to your desired cell by moving one row down every time. Then we will have a RBS in the form ()()()...().



Subtask 1 (4%): N even, (Sr, Sc) = (0, 0), Er = N – 1, B[r][c] = '(' iff r even

How do we construct a path from (0, 0) to (N – 1, x)?

Notice that moving diagonally right preserves the value of **row – column**; meanwhile, moving diagonally left preserves the value of **column**.

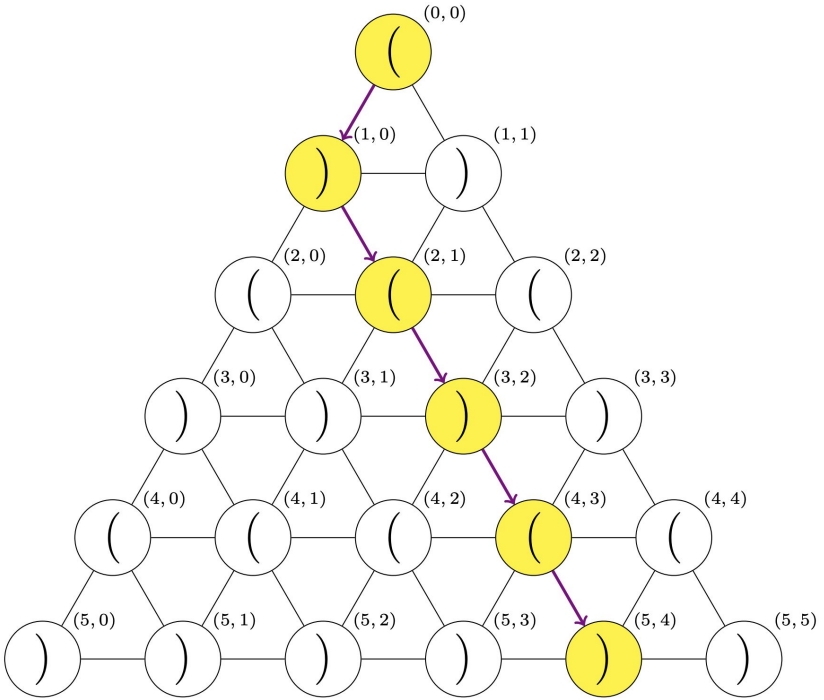


Subtask 1 (4%): N even, (Sr, Sc) = (0, 0), Er = N – 1, B[r][c] = '(' iff r even

To walk from (0, 0) to (N – 1, x), we can:

- Let d = N – 1 – x
- Walk from (0, 0) to (d, 0), each step is diagonally left
- Then walk from (d, 0) to (N – 1, x), each step is diagonally right

Expected score: 4



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

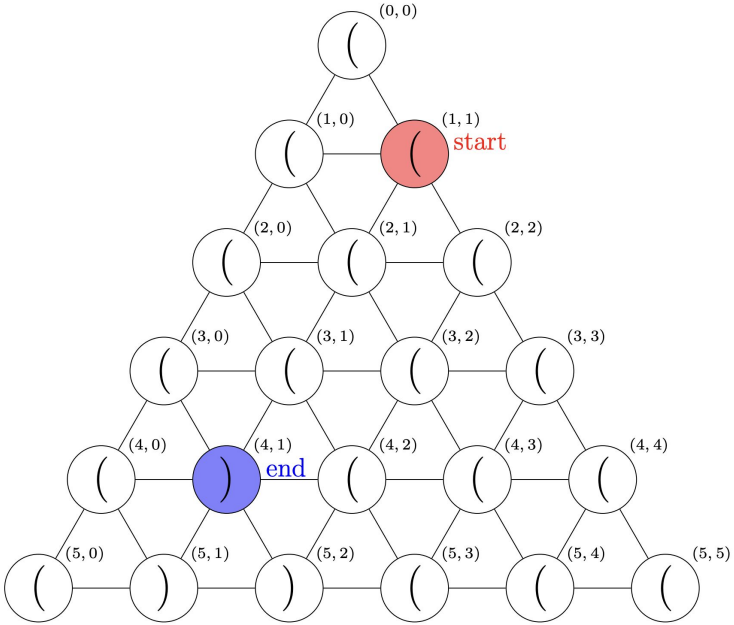
Example:

Starting vertex: (1, 1)

Ending vertex: (4, 1)

The only ‘)’s are at (4, 1), (5, 1) and (5, 2).

Do we ever have to visit a ‘)’ before reaching (4, 1)?



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

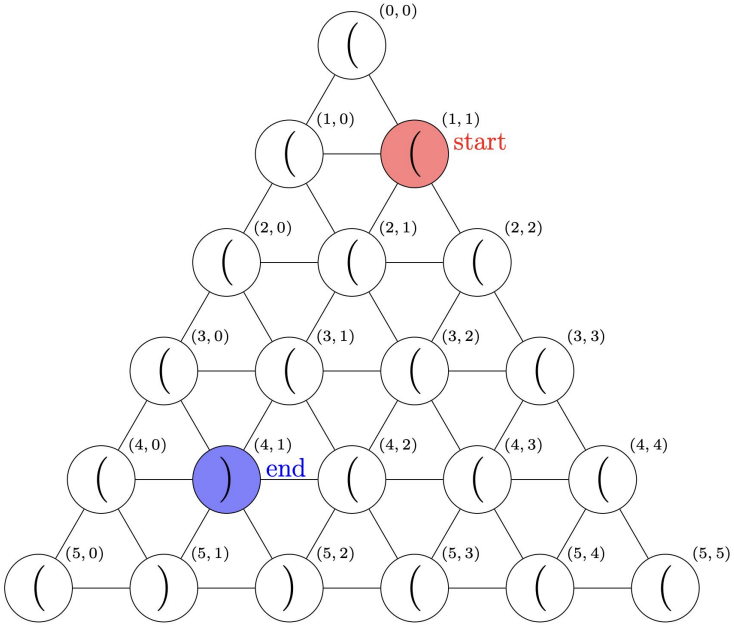
Example:

Starting vertex: (1, 1)

Ending vertex: (4, 1)

The only ‘)’s are at (4, 1), (5, 1) and (5, 2).

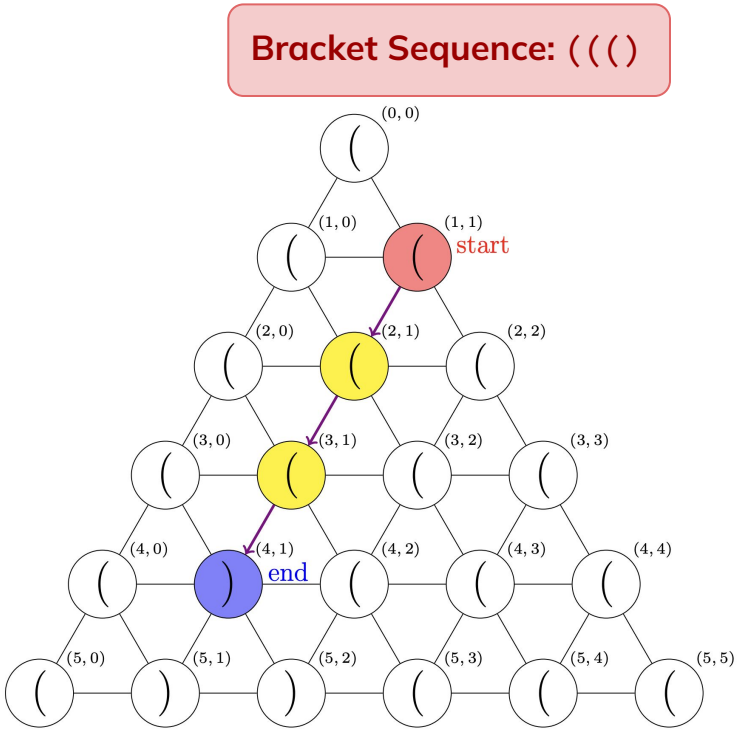
Do we ever have to visit a ‘)’ before reaching (4, 1)? **No; if we walk down row by row and never reach row 5.**



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Idea: we basically won’t get any closed brackets on the way from (Sr, Sc) to (Er, Ec).
- So we can safely walk from (Sr, Sc) to (Er, Ec) collecting open brackets. Now we have an almost walk of life.

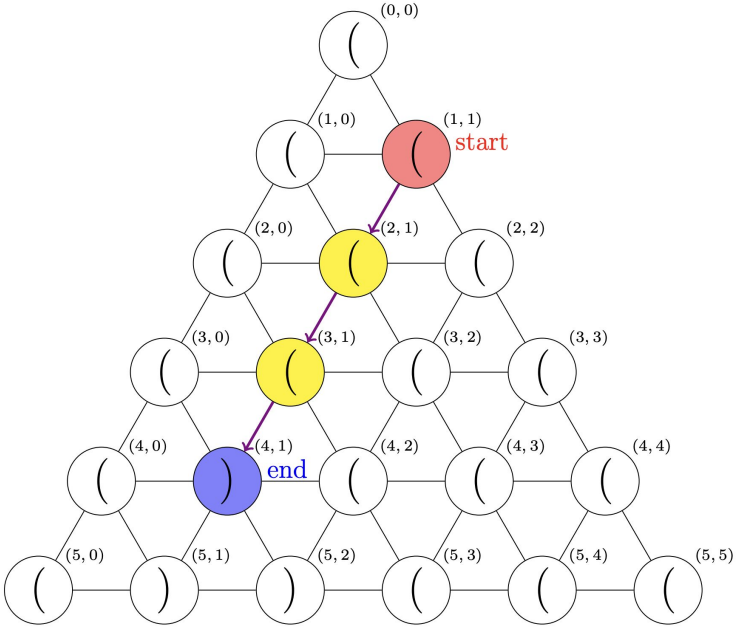
Expected score: 6.3



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Idea: we basically won’t get any closed brackets on the way from (Sr, Sc) to (Er, Ec).
- So we can safely walk from (Sr, Sc) to (Er, Ec) collecting open brackets.
- Aiming for full credit, we will need to handle the closed brackets after we reach (Er, Ec).

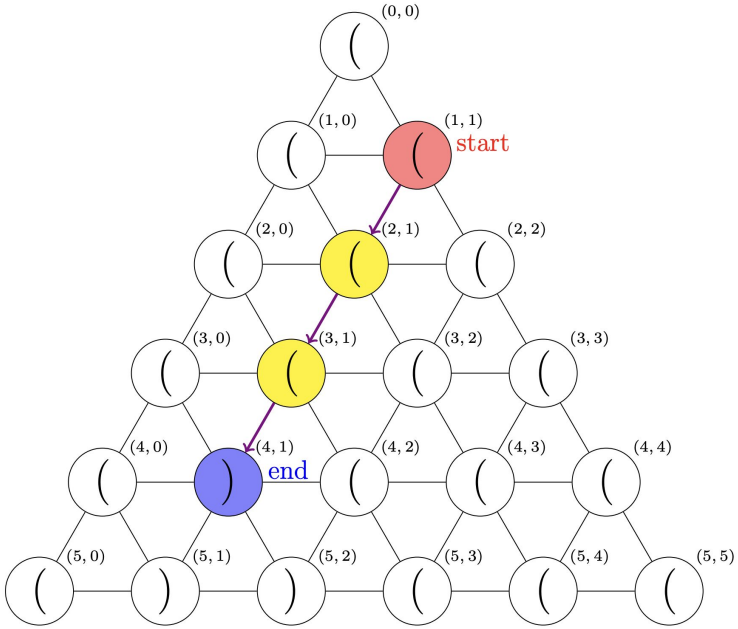
Bracket Sequence: ((( )



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Idea: we basically won’t get any closed brackets on the way from (Sr, Sc) to (Er, Ec).
- So we can safely walk from (Sr, Sc) to (Er, Ec) collecting open brackets.
- Aiming for full credit, we will need to handle the closed brackets after we reach (Er, Ec).
- When we do reach (Er, Ec), what is the minimum surplus?

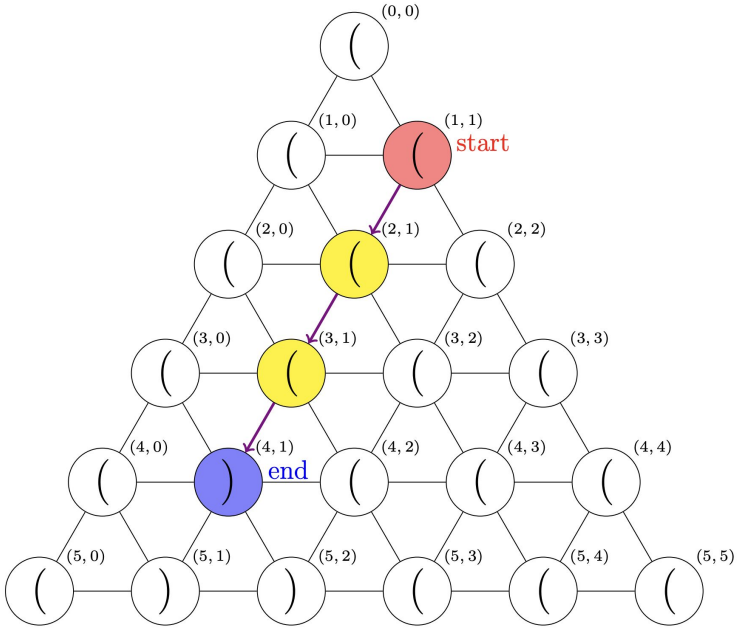
Bracket Sequence: ((( )



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Idea: we basically won’t get any closed brackets on the way from (Sr, Sc) to (Er, Ec).
- So we can safely walk from (Sr, Sc) to (Er, Ec) collecting open brackets.
- Aiming for full credit, we will need to handle the closed brackets after we reach (Er, Ec).
- When we do reach (Er, Ec), what is the minimum surplus? **3 – 1 = 2.**

Bracket Sequence: ((( )

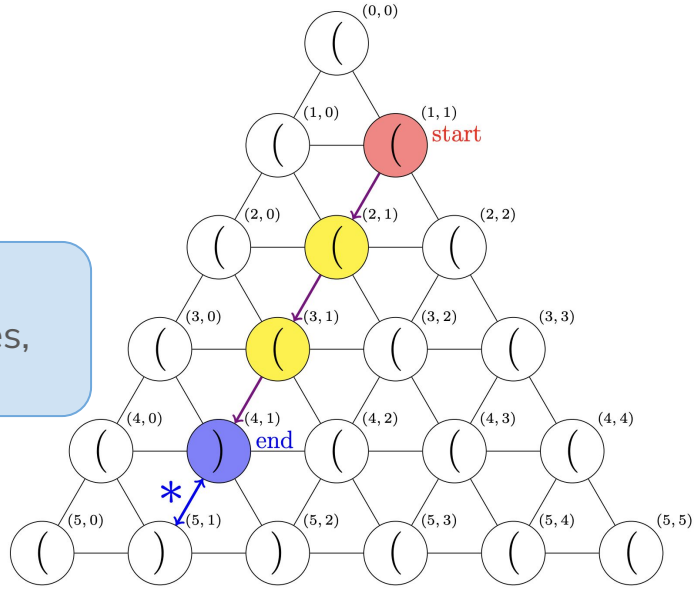


Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- It remains to reduce the surplus to zero. Here, it is guaranteed that surplus ≥ 2.
- Conveniently, (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1) are all closed brackets.
- What ways are there to reduce the surplus using these few vertices?

Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

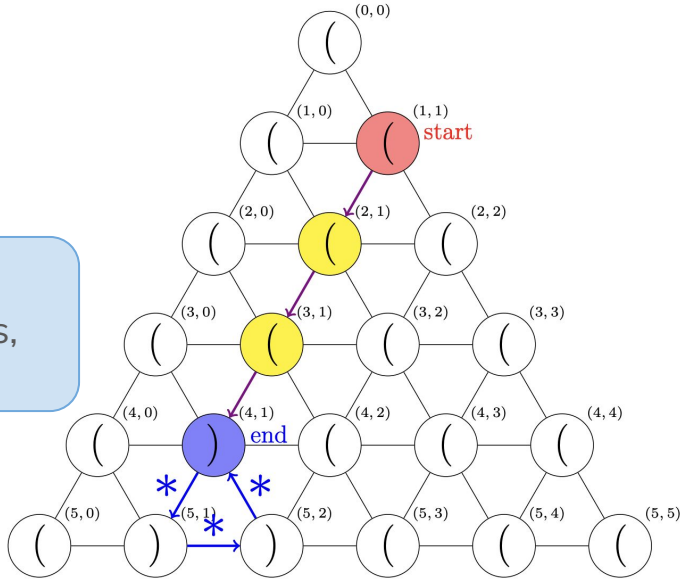
- Method 1: Walk from (Er, Ec), to (Er + 1, Ec), then back to (Er, Ec).



This reduces the surplus by 2.
Recall we can do this many times, denoted by the asterisk **(*)**.

Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Method 2: Walk the cycle (Er, Ec) → (Er + 1, Ec) → (Er + 1, Ec + 1) → (Er, Ec).

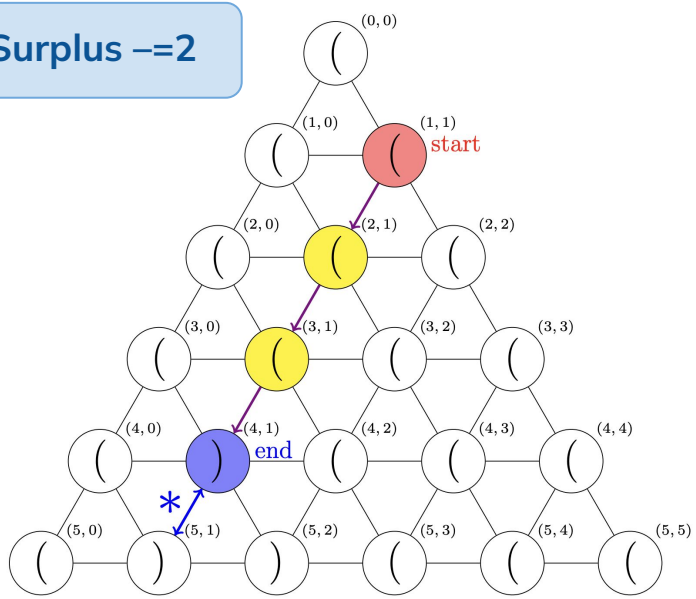


This reduces the surplus by 3.
Recall we can do this many times, denoted by the asterisk **(*)**.

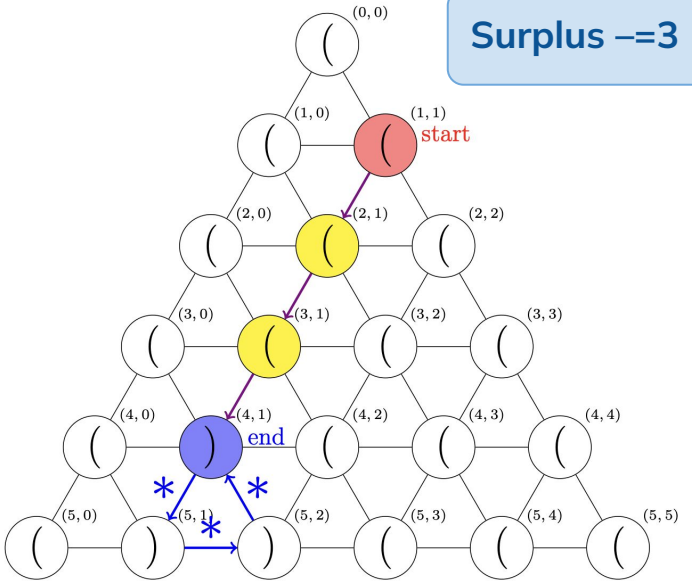
Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

- Method 2: Walk the cycle (Er, Ec) → (Er + 1, Ec) → (Er + 1, Ec + 1) → (Er, Ec).

Surplus –=2



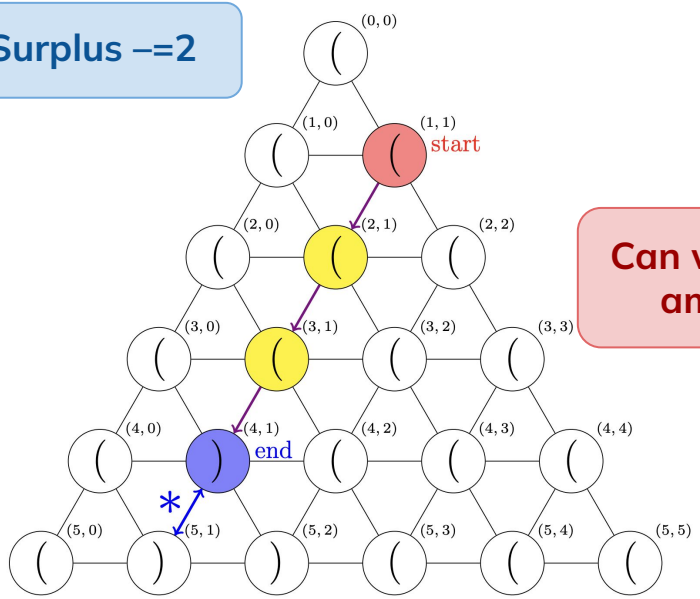
Surplus –=3



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

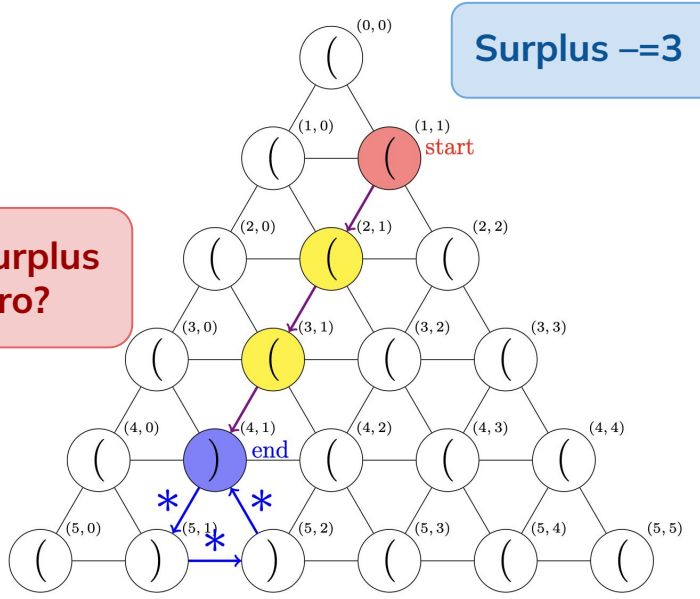
- Method 2: Walk the cycle (Er, Ec) → (Er + 1, Ec) → (Er + 1, Ec + 1) → (Er, Ec).

Surplus –=2



Can we reduce any surplus amount (≥ 2) to zero?

Surplus –=3



Subtask 2 (14%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); everything else is ‘(’; Sr + 3 ≤ Er < N – 1

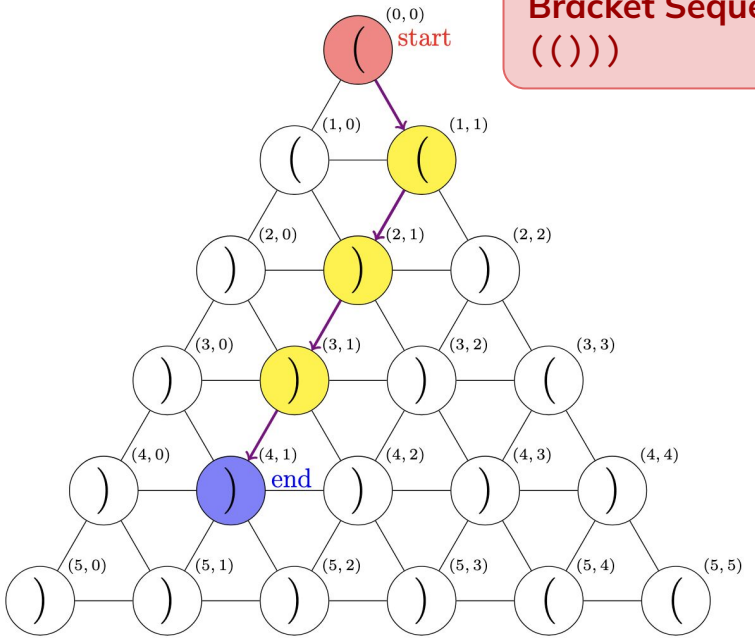
- Hopefully this is obvious by now, the answer is **YES**!
- Our strategy would be as follows:
- Walk from (Sr, Sc) to (Er, Ec), obtaining a surplus of ≥ 2.
- Then reduce the surplus to zero:
  - If surplus is odd, firstly reduce it by 3 using Method 2.
  - Now the surplus must be even. Keep reducing by 2 every time using Method 1.

Expected score: 14

Subtask 3 (15%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); 3 ‘(’s at (Sr, Sc), (Sr + 1, Sc) and (Sr + 1, Sc + 1); Sr + 2 ≤ Er < N – 1

- If we walk arbitrarily from (Sr, Sc) to (Er, Ec), we are no longer guaranteed to have more ‘(’s than ‘)’s anymore...
- There can be many ‘)’s everywhere
- We are only given that the vertices around (Sr, Sc) are ‘(’s

Bracket Sequence:
(())))

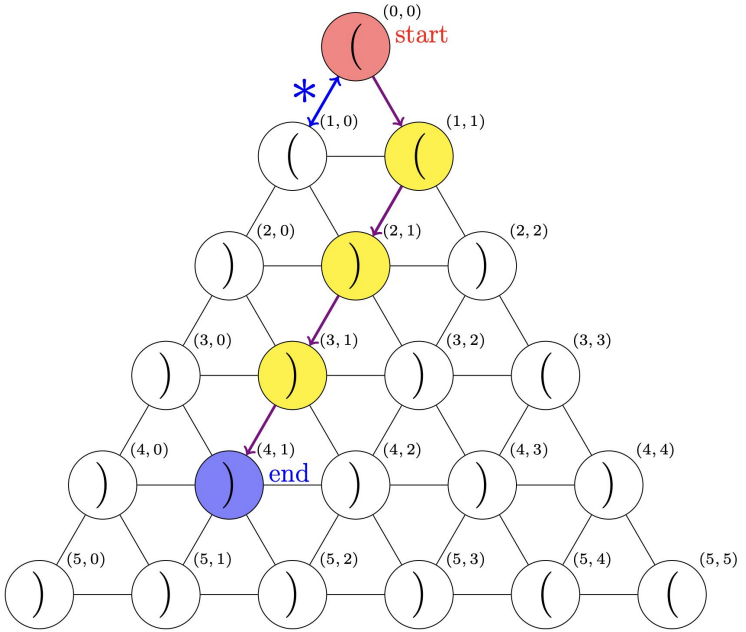


Subtask 3 (15%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); 3 ‘(’s at (Sr, Sc), (Sr + 1, Sc) and (Sr + 1, Sc + 1); Sr + 2 ≤ Er < N – 1

- However, note that before we move from (Sr, Sc) to (Er, Ec) we can first increase the surplus by a lot* by moving back and forth between (Sr, Sc) and (Sr + 1, Sc)
- Then we can simply run Subtask 2’s solution: Move to (Er, Ec), then reduce the surplus

Expected score: 29

* Something like 2000 will do.

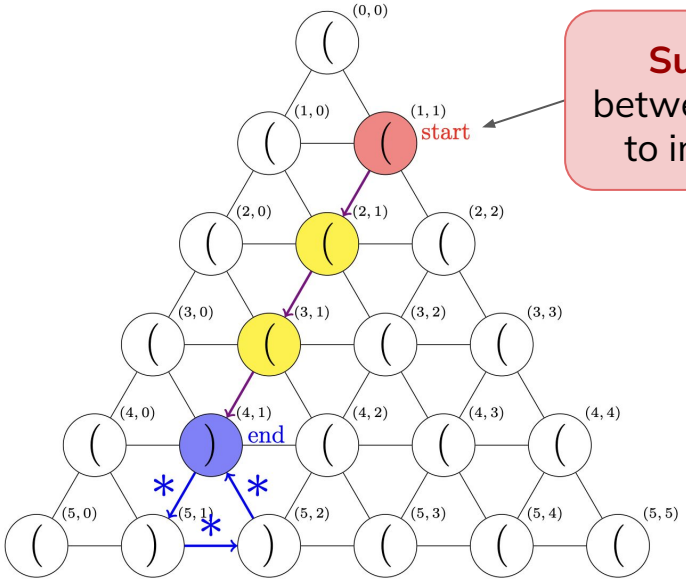


Subtask 4 (24%): Exists walk of life of the form ( ) ( ) . . . ( )

- We are guaranteed that there exists a walk of life such that the resulting bracket sequence is alternating.
- We can find such a walk of life easily using breadth-first search on the graph, where we **erase** all the edges connecting two vertices that have the same bracket (i.e. each edge connects one open bracket with one closed bracket).
- If the end vertex is reachable from the start vertex, backtrack to find the appropriate walk.
- Expected score: 28 (incl. Subtask 1)

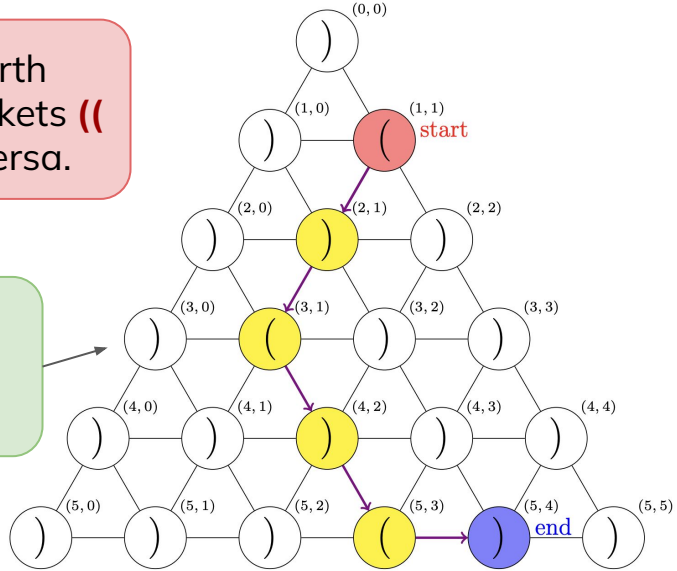
Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

- Can we use the ideas from previous subtasks?



Subtask 3: Walk back and forth between vertices that have brackets ((to increase surplus, and vice versa.

Subtask 4: Maybe, ()()()...() is a walk of life. Find it using a BFS.



Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

- Take a closer look at all regular bracket sequences.

Observation 1: For any RBS, either `((` is a substring of the RBS, or the RBS is of the form `()()()...()`.

Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

- Take a closer look at all regular bracket sequences.

Observation 1: For any RBS, either `((` is a substring of the RBS, or the RBS is of the form `()()()...()`.

Observation 2: If an RBS contains the `((` substring, it should also contain the `))` substring; and if `((` and `))` are the **first** and **last** occurrences of `((` and `))` respectively (note the colour), then the RBS can be written in the form `()()...()((...))()()...()`.

Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

- So, we only need to consider these cases:

(1) If there is a solution of the form `()()()...()`, then use that solution.

⇒ **Subtask 4 Solution**

(2) If there isn’t, then check for a solution with substring `((`; if that exists, then use that solution.

(3) If there isn’t, then there is just no solution.

Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

- So, we only need to consider these cases:

(1) If there is a solution of the form `()()()...()`, then use that solution.
⇒ **Subtask 4 Solution**

(2) If there isn’t, then check for a solution with substring `((`; if that exists, then use that solution. ⇒ **How do we find such a general solution?**

(3) If there isn’t, then there is just no solution.

Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Finding a solution with substring ((

Observation 2: An RBS with `((` substring is of the form `()()...()((...))()()...()`.

- Split into three parts: (Sr, Sc) → `()()...()` → `(( ... ))` → `()()...()` → (Er, Ec)
- So we want to find some ((that is reachable from (Sr, Sc) using only a ()()...() path. We can achieve this via BFS (similar to Subtask 4).
- For Subtask 5, the `))` is simply (Er, Ec) and its neighbours.

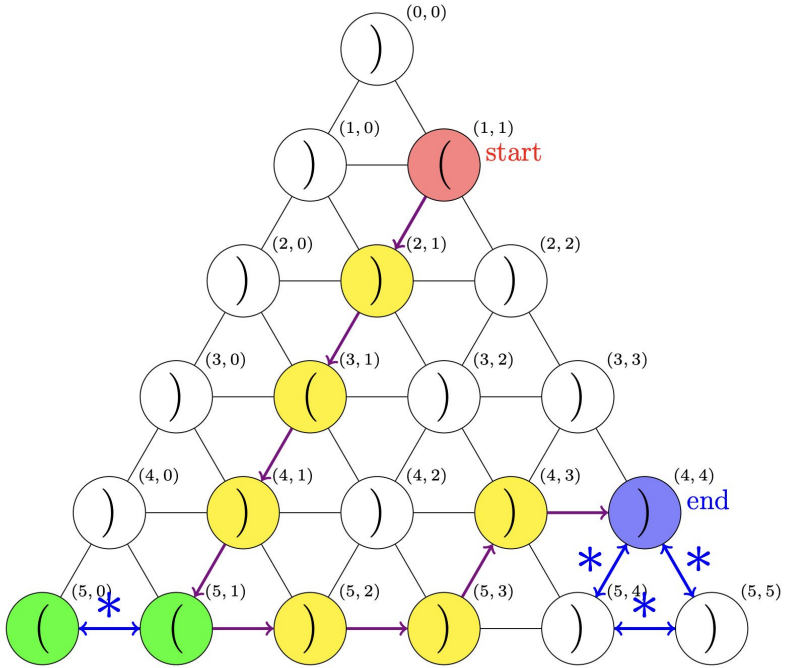
Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Finding a solution with substring ((

Observation 2: An RBS with `((` substring is of the form `()()...()((...))()()...()`.

- Recall: What do we do when we encounter `((`?
- Walk back and forth many times to increase the surplus.
- Then the middle part (`...`) doesn’t really matter, just pick an arbitrary path.
- Then walk back and forth many times / in a cycle, to reduce the surplus to zero when we are at (Er, Ec) (for subtask 5 only).

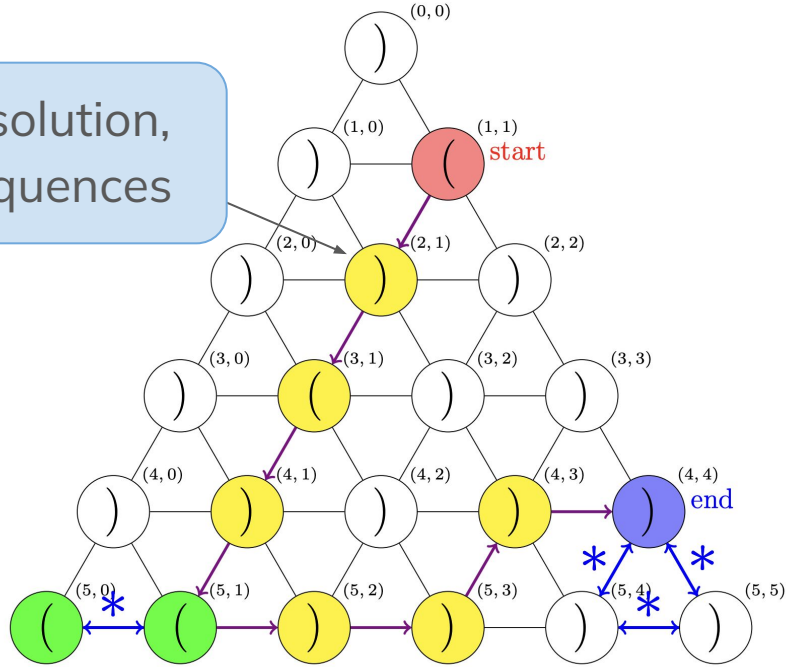
Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1



Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: ( ) ( )

Step 1: Use subtask 4 solution, BFS using only ()()... sequences

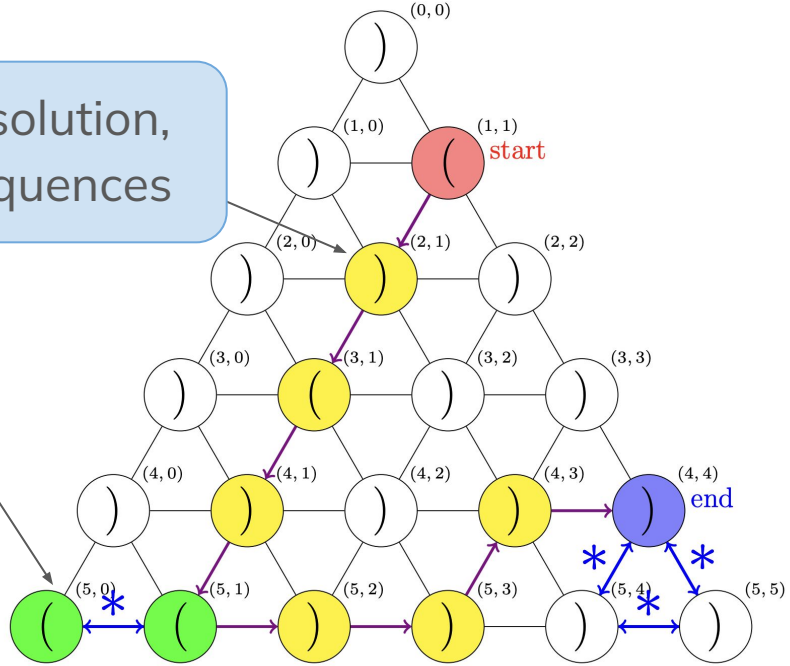


Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: ()()(((((((

Step 1: Use subtask 4 solution, BFS using only ()()... sequences

Step 2: Build up the surplus by repeating the green vertices



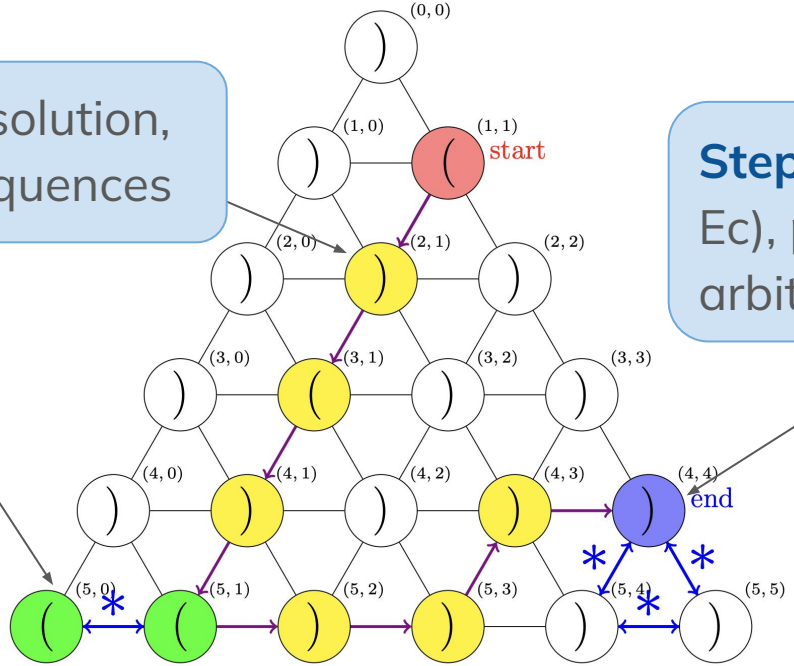
Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((((()))))`

Step 1: Use subtask 4 solution, BFS using only ()()... sequences

Step 2: Build up the surplus by repeating the green vertices

Step 3: Walk to (Er, Ec), picking an arbitrary path



Subtask 5 (17%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((((()))))))`

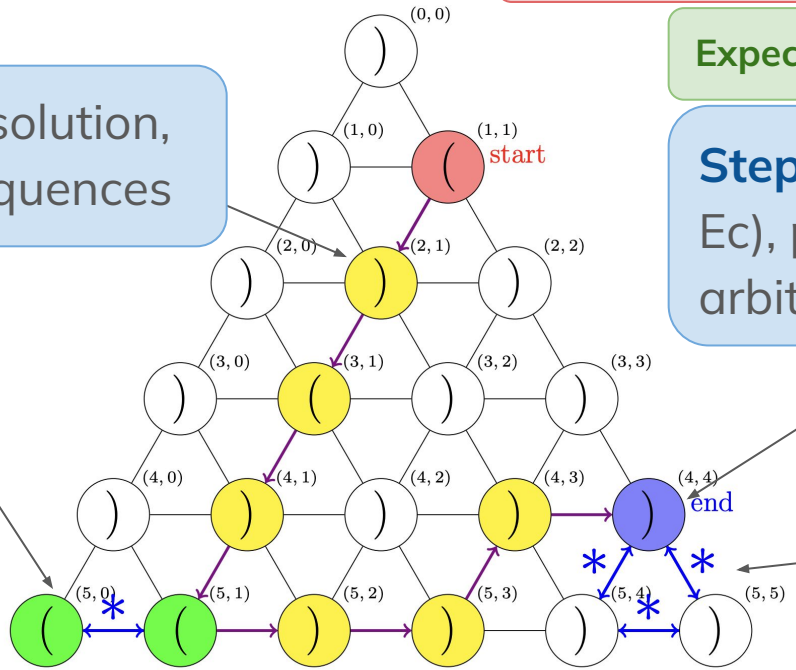
Expected score: 74 (Cumulative)

Step 1: Use subtask 4 solution, BFS using only ()()... sequences

Step 2: Build up the surplus by repeating the green vertices

Step 3: Walk to (Er, Ec), picking an arbitrary path

Step 4: Reduce the surplus to zero, e.g. walk in a 3-cycle.



Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Ok, now we no longer have the guarantee that there are ‘)’s nearby the end.

How do we modify the previous solution?

Observation 2: An RBS with `((` substring is of the form `()()...()((...))()()...()`.

Approach 1: Just find the `((`, increase the surplus arbitrarily, then walk straight to the end.

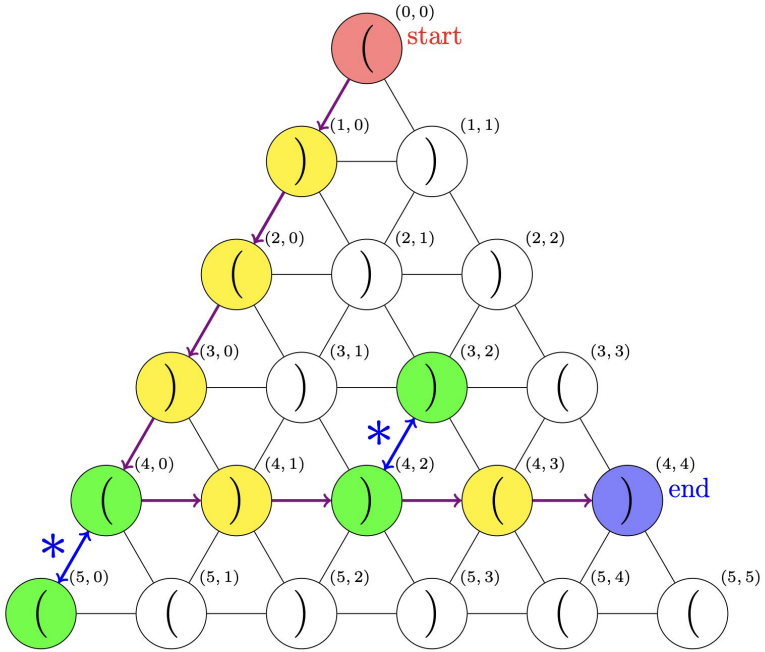
Expected score: 74 + 26 * 45% = 85.7

Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Approach 2: It turns out that we need to do the “analogous” BFS from the ending vertex as well. Specifically, we:

- BFS from the starting vertex, find a `((` that is reachable from it via a sequence of ()()...()
- BFS from the ending vertex, find a `))` that is reachable from it via a sequence of ()()...()
- The desired walk is
  [start to `((`] + [repeat `((` many times] + [arbitrary path from `((` to `))`]
  + [repeat `))` many times] + [`))` to end]

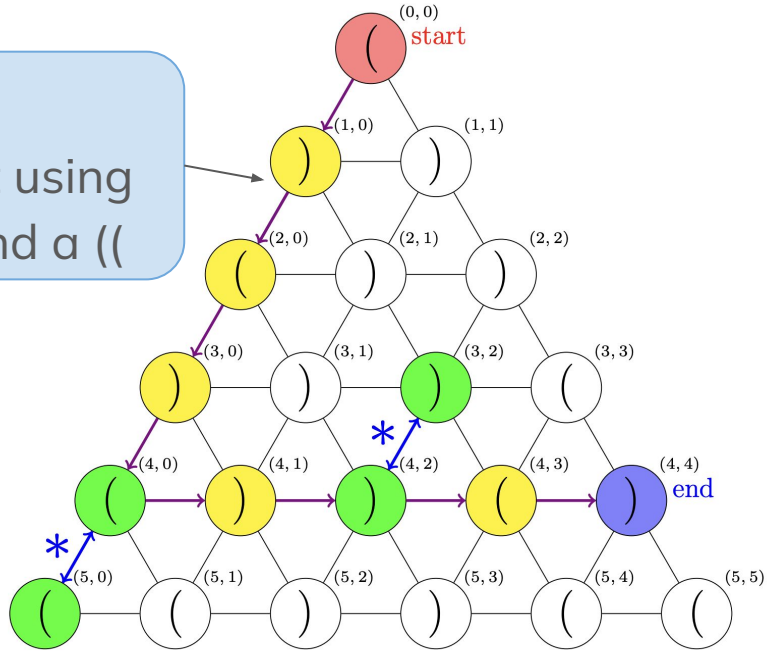
Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1



Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: ( ) ( )

Step 1: Use subtask 4 solution, BFS from start using only ()()... sequences, find a ((

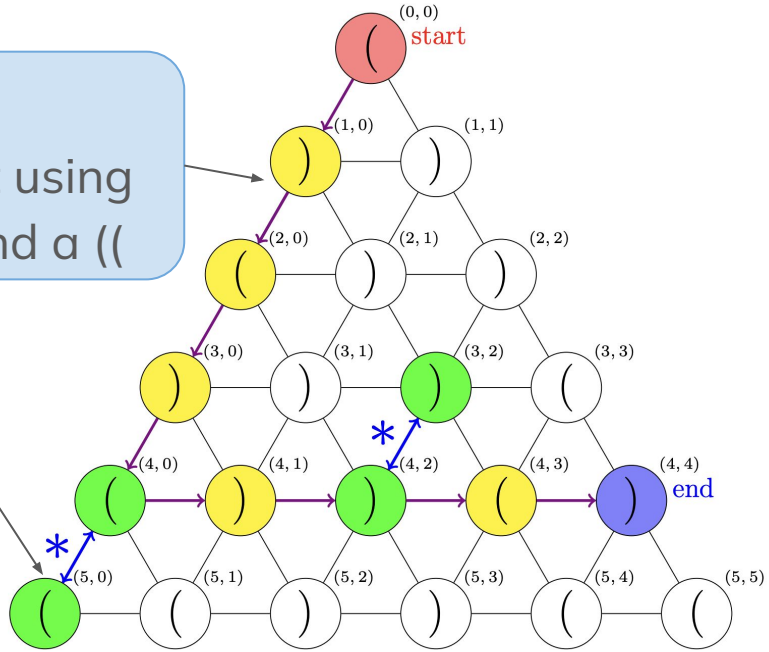


Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: ()()(((((

Step 1: Use subtask 4 solution, BFS from start using only ()()... sequences, find a ((

Step 2: Build up the surplus by repeating the green vertices



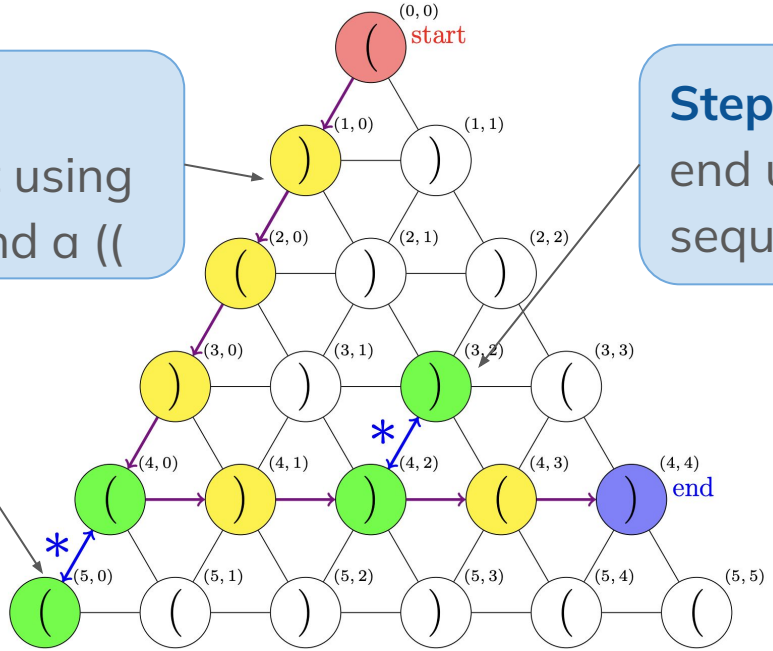
Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()(((((` + `?` + `()`

Step 1: Use subtask 4 solution, BFS from start using only ()()... sequences, find a ((

Step 2: Build up the surplus by repeating the green vertices

Step 3: BFS from end using only ()()... sequences, find a))

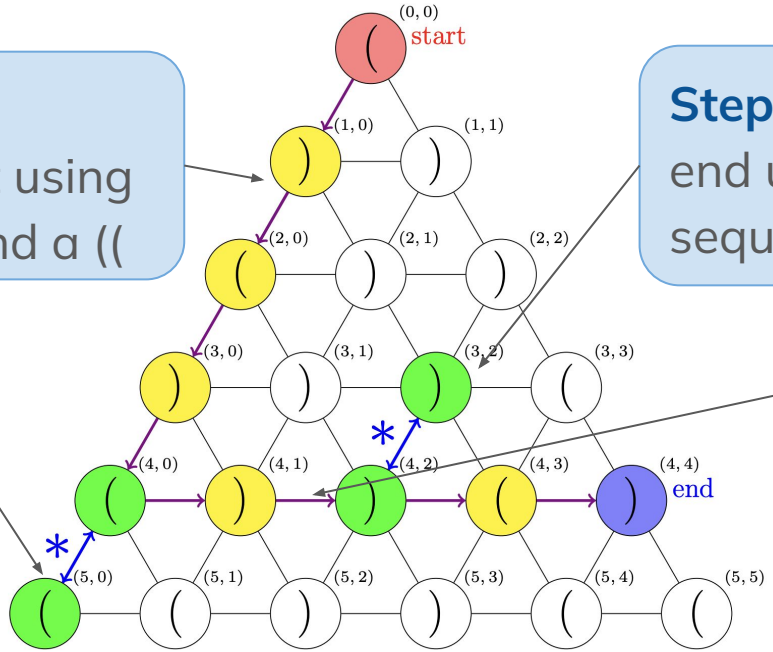


Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((( + )) + ()`

Step 1: Use subtask 4 solution, BFS from start using only ()()... sequences, find a ((

Step 2: Build up the surplus by repeating the green vertices



Step 3: BFS from end using only ()()... sequences, find a))

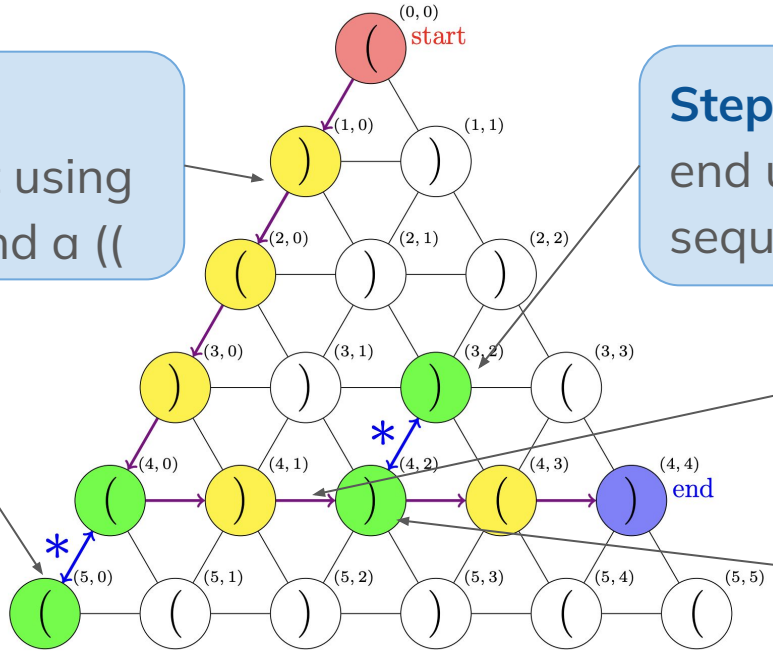
Step 4: Connect ((to))

Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((()))))()`

Step 1: Use subtask 4 solution, BFS from start using only ()()... sequences, find a ((

Step 2: Build up the surplus by repeating the green vertices



Step 3: BFS from end using only ()()... sequences, find a))

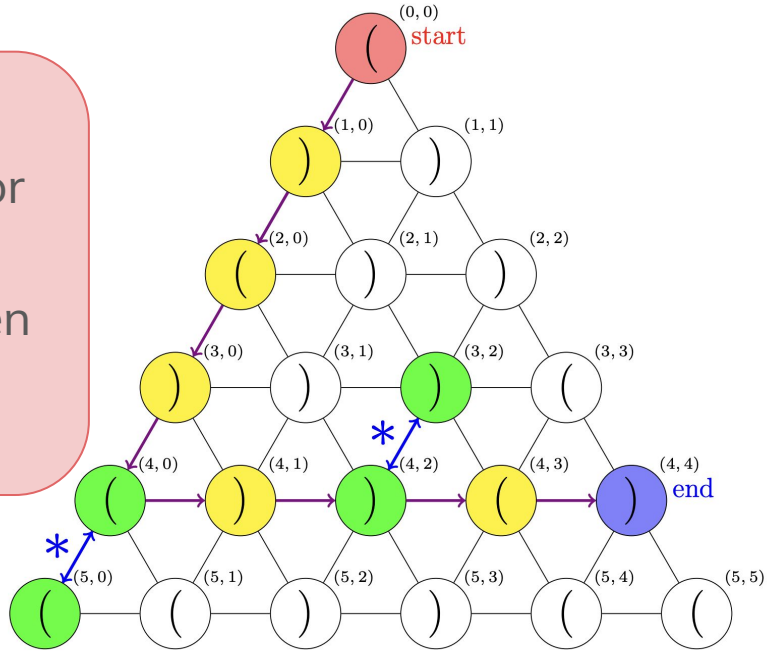
Step 4: Connect ((to))

Step 5: Reduce surplus by repeating green vertices

Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((()))))()`

Parity Issue: Step 5 is not always possible. For example, in this case if we just repeat the green vertices, we obtain a surplus of 1 at the end.

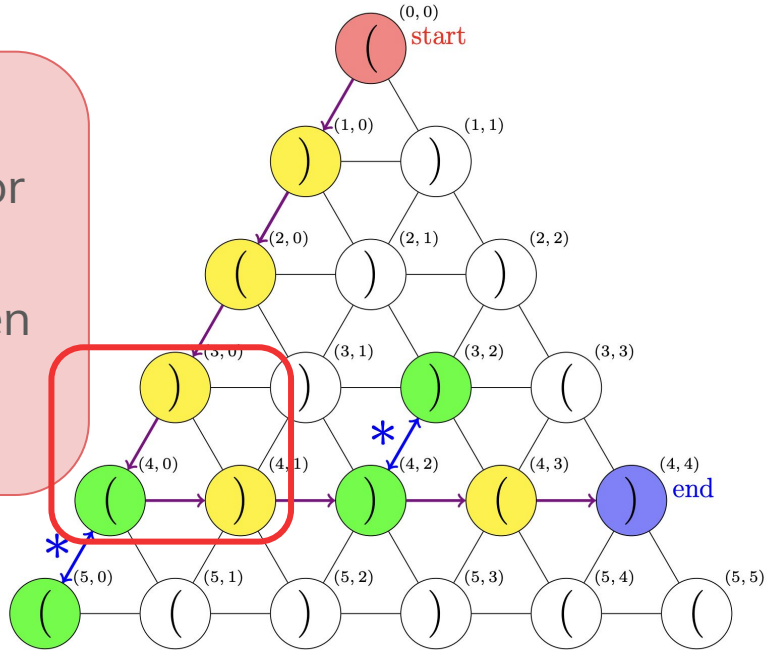


Subtask 6 (26%): 3 ‘)’s at (Er, Ec), (Er + 1, Ec) and (Er + 1, Ec + 1); Sr + 2 ≤ Er < N – 1

Bracket Sequence: `()()((((())()))))()`

Expected score: 100

Parity Issue: Step 5 is not always possible. For example, in this case if we just repeat the green vertices, we obtain a surplus of 1 at the end.



How to fix: just find an arbitrary 3-cycle when we are at the ((stage; each vertex visit toggles the parity, so overall the parity must be fixed with 3 more vertex visits.

Full Solution: Summary

- BFS from start to some vertex V_1, only using alternating-bracket paths, such that both itself and at least one neighbour is an open bracket.
- BFS from end to some vertex V_2, only using alternating-bracket paths, such that both itself and at least one neighbour is a closed bracket.
- Building the walk itself: First walk from start to V_1, increase the surplus by repeatedly visiting V_1 and its neighbour.
- Then walk to V_2 arbitrarily.
- Then decrease the surplus by repeatedly visiting V_2 and its neighbour.
- Finally, walk from V_2 to end.
- Fix the parity by adding an arbitrary 3-cycle when the surplus is large.

Multi-layered Graph Solution

- For small N (say N ≤ 100) there exists a multi-layered graph solution.
- But this makes the task uninteresting, so we don't allow such solutions.
- Nevertheless, a 2-layered graph would earn you 28 points (Subtasks 1, 4)

Takeaways

- Subtasks are (usually) here to guide your thinking. In fact, in this task I would say that even **reading** the subtasks helps a lot in thinking of the solution (e.g. subtask 4 is a special case that you must handle).
  (*I am aware that some contestants prefer shooting for the full solution directly. While this might save you some time, you must be sure when to do so and when not to do so.)

- Sometimes there are implementation-heavy tasks in IOI (my solution for this task is 200 lines), be sure to have skills in implementation

- Constructive tasks are fun!

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

M2554 Playing Cards

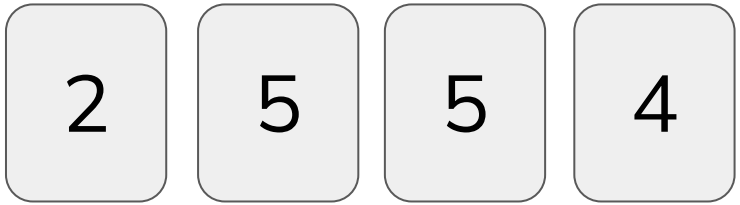
Hsieh Chong Ho {QwertyPi}

2025-07-01

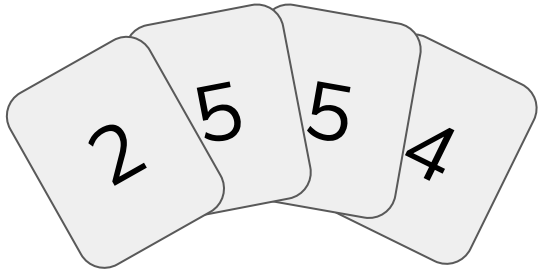
Problem Background

Problem Idea by `QwertyPi`
Preparation by `QwertyPi`
Presented by `QwertyPi`

Another card problem!



The Problem



Given N cards of integer values A[1] A[2] ... A[N].

The **score** of a card sequence B[1] B[2] ... B[M] is defined to be

$$\#\{B[K + 1] > B[K]\} - \#\{B[K + 1] < B[K]\}$$

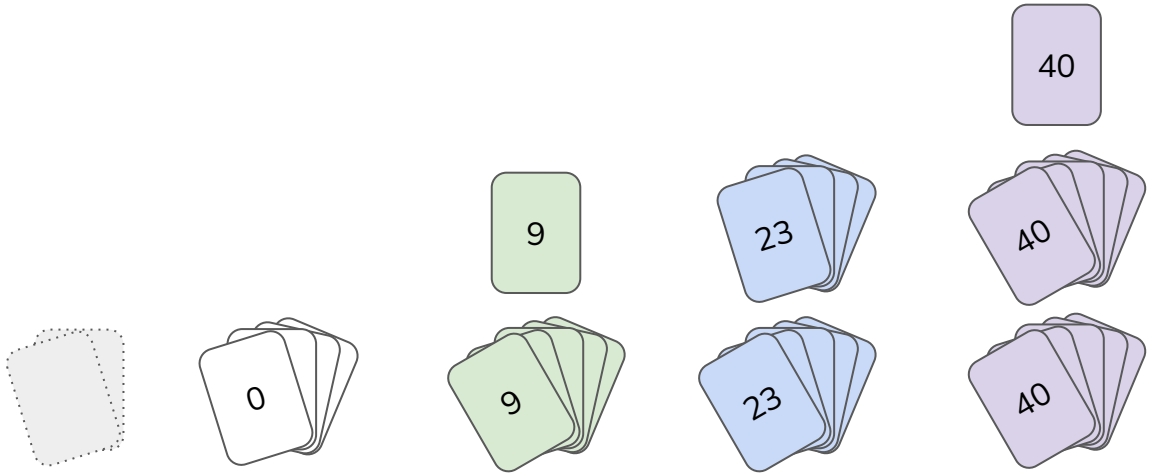
Given Q subarrays A[L] ... A[R], maximise the **score** over all **card subsequence**.

Subtask Overview

For all cases: $N, Q \leq 2 \times 10^5$

Subtask	Score	Main Constraints
1	9	$N \leq 10, Q \leq 10, A[i] \leq 3$
2	14	$N \leq 1000, Q \leq 10, A[i] \leq 1000$
3	17	$Q \leq 10, A[i] \leq 2 \times 10^5$
4	15	$A[2k] < A[2k + 1] > A[2k + 2]$ for all k
5	24	$A[i] < A[i + 1]$ for at most 10 i
6	21	/

Statistics

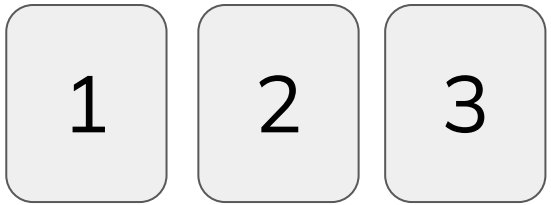


Subtask 1 (9%, $N \le 10$, $Q \le 10$, $A[i] \le 3$)

Since N and Q are small, you may enumerate through all the possible card subsequences in $O(Q2^N)$.

Alternatively, you use $A[i] \le 3$, and observe that the picked subsequence is either multiple copies of 1 2 3 or ≤ two numbers.

Expected Score: 9



Subtask 2 (14%, N ≤ 1000, Q ≤ 10, A[i] ≤ 1000)

Let's define the following function:

```
function score(a, b) // a: old score, b: new score
    if (a < b) return +1
    if (a > b) return -1
    return 0
```

With this, you can in fact solve this with dynamic programming in two ways. You can do it based on either index or value, both are fine for this subtask.

Subtask 2 (14%, N ≤ 1000, Q ≤ 10, A[i] ≤ 1000)

Let's say the subarray is `B` of length `M`. If we solve this based on index:

```
dp[i] := answer for which last picked card is card i
for r from 1 to M:
    dp[r] = max(0, max_{1≤i≤r-1}(dp[i] + score(B[i], B[r])))
```

Time Complexity: $O(QN^2)$

Expected Score: 23

Subtask 2 (14%, N ≤ 1000, Q ≤ 10, A[i] ≤ 1000)

Alternatively, we can also solve this based on values.

$V := \max_{1 \le i \le M}\{B[i]\}$

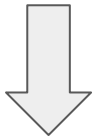
```
dp[v] := answer for last picked value = v
for i from 1 to M:
    for j from 1 to V:
        dp'[B[i]] := max(dp[j] + score(j, B[i]))
```

Time Complexity: O(QNV)
Expected Score: 23

Subtask 3 (17%, $Q \le 10$, $A[i] \le 2 \times 10^5$)

In fact, you can optimise the previous solution with segment tree:

```
for j from 1 to V:
    dp'[B[i]] := max(dp[j] + score(j, B[i]))
```



$$dp'[B[i]] := \max(\max_{1 \le j < B[i]}(dp[j]) + 1,\ dp[B[i]],\ \max_{B[i] < j \le V}(dp[j]) - 1)$$

Time Complexity: O(QN log V)

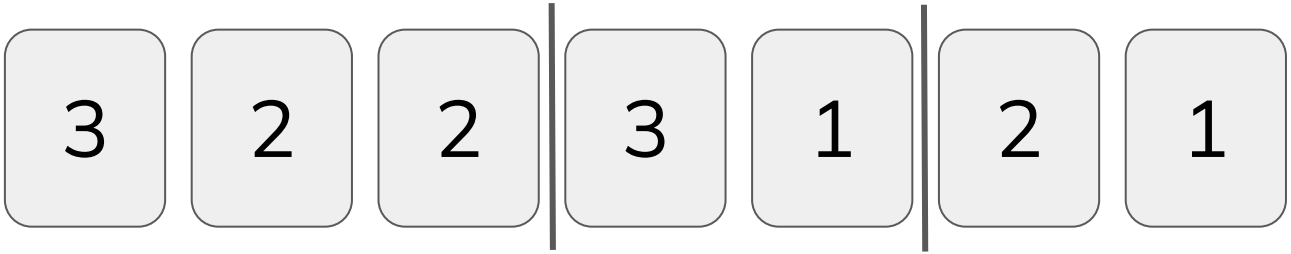
Expected Score: 40

Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

Card values are “oscillating frequently”. How does this help?

Let’s go through some general definitions:

Fact 1. We can **partition** the cards into **decreasing segments**.



Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

Fact 2. There exists a way of **getting maximum score**, such that **for each decreasing segment, we pick at least one card**.

Let's say the decreasing segments of B are L_1 to R_1, L_2 to R_2, ..., L_K to R_K.

Then

$$B[L_1] \geq \ldots \geq B[R_1] < B[L_2] \geq \ldots \geq B[R_2] < \ldots < B[L_K] \geq \ldots \geq B[R_K].$$

Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

$$B[L_1] \geq \ldots \geq B[R_1] < B[L_2] \geq \ldots \geq B[R_2] < \ldots < B[L_K] \geq \ldots \geq B[R_K]$$

Step 1. If no cards is picked, pick any card. Answer remains as 0.

Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

$$B[L_1] \geq \ldots \geq B[R_1] < B[L_2] \geq \ldots \geq B[R_2] < \ldots < B[L_K] \geq \ldots \geq B[R_K]$$

Step 2. Suppose card X_1 is the leftmost card picked where $L_1 \leq X_1 \leq R_1$. Then adding cards $R_1, L_2, \ldots, R_{1-1}$ and L_1 (if $X_1 > L_1$) won't negatively affect the result. Similarly for the rightmost card picked.

We can pick **at least one card from the first and last decreasing segments**.

Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

$$B[L_1] \geq \ldots \geq B[R_1] < B[L_2] \geq \ldots \geq B[R_2] < \ldots < B[L_K] \geq \ldots \geq B[R_K]$$

Step 3. Suppose card X_l and X_r are picked where $L_l \leq X_l \leq R_l$ and $L_r \leq X_r \leq R_r$ where $r - l > 1$, and there is no card between X_l and X_r is picked.

- Case $B[X_l] > B[X_r]$: $X_l \to R_l \to L_{l+1} \to X_r$.
- Case $B[X_l] = B[X_r]$: $X_l \to L_{l+1} \to X_r$.
- Case $B[X_l] < B[X_r]$: $X_l \to R_{l+1} \to L_{l+2} \to X_r$.

Each case, we can add cards in between to achieve same or higher score.

Therefore, we can recursively show that we can pick at least one card from each decreasing segment.

Subtask 4 (15%, A[2k] < A[2k + 1] > A[2k + 2] for all k)

Therefore, when card values are “oscillating frequently”, the gap between adjacent selected cards is at most 3.

Therefore, you can simply maintain matrices at segtree nodes.

For each segtree node for range `L ... R`, maintain `a[i][j]` (`0 ≤ i, j ≤ 3`), which is the answer when the position of picked cards being `L + i` till `R - j`.

Time Complexity: O((N + Q) log N)

Expected Score: 15 (Cumulative: 55)

Subtask 5 (24%, A[i] < A[i + 1] for at most 10 i)

Fact 3. There exists a way of getting maximum score, such that **for each decreasing segment, we pick at most two card**.

- This is true since we can always remove the card in-between.

Fact 4. We can first consider all the decreasing segments of A. Then, for each query, we can **"extend" the query** to include the **whole segments**.

- In the first segment, the only make-sense choice is to pick the smallest value. Similarly, in the last segment, the only make-sense choice is to pick the largest value.

Subtask 5 (24%, A[i] < A[i + 1] for at most 10 i)

Fact 5. At each index, only two values that it make sense to jumps to:

- Reward = 1: Jump to the value which is greater than the current value in the next segment
- Reward = 0: Jump to the lowest value of the current segment, then jump to the value which is greater than the current value in the next segment

Fact 6. The answer is bounded by the number of A[i] < A[i + 1].

Subtask 5 (24%, A[i] < A[i + 1] for at most 10 i)

There are only **at most 11 decreasing segments** in this subtask.

Note that for each answer in each segment, we only need to store the current minimum value that attains that.

Therefore, with some precomputation on jumping, you can simply do some dynamic programming.

Time Complexity: $O(11^2Q)$

Expected Score: 64 (Cumulative: 79)

Full Solution

Fact 7. Actually, for each segment, it only make sense to consider the values for max answer and (max - 1) answer. Therefore, you can do some binary lifting on that.

The implementation is left as exercise for readers.

Time Complexity: O((N + Q) log N)
Expected Score: 100