Team Members

Sudipta Halder (2021202011)

Vishal Pandey (2021201070)

Group Number - 13

Problem

Othello Game. Design the page to store the board position of the game. It should be possible to replay the complete game, or check what the board position is at some move.

Assumptions

- The board size is greater than data block size.
- Data Block size = 4 Bytes
- The maximum number of moves possible is 10^6 - 4.
- We are storing 200 million completely played games - each with a unique identifier.

Criteria and Main Idea for Page design

- To store the state of the board, we have thought of using sparse matrix representation. Also, we would be storing the details of black and white entries separately.
- Now, let's say we have a 4x4 board. Then, if the state of the board is like the following then

	B	W	
	W	B	

We can store the state of the board like this:

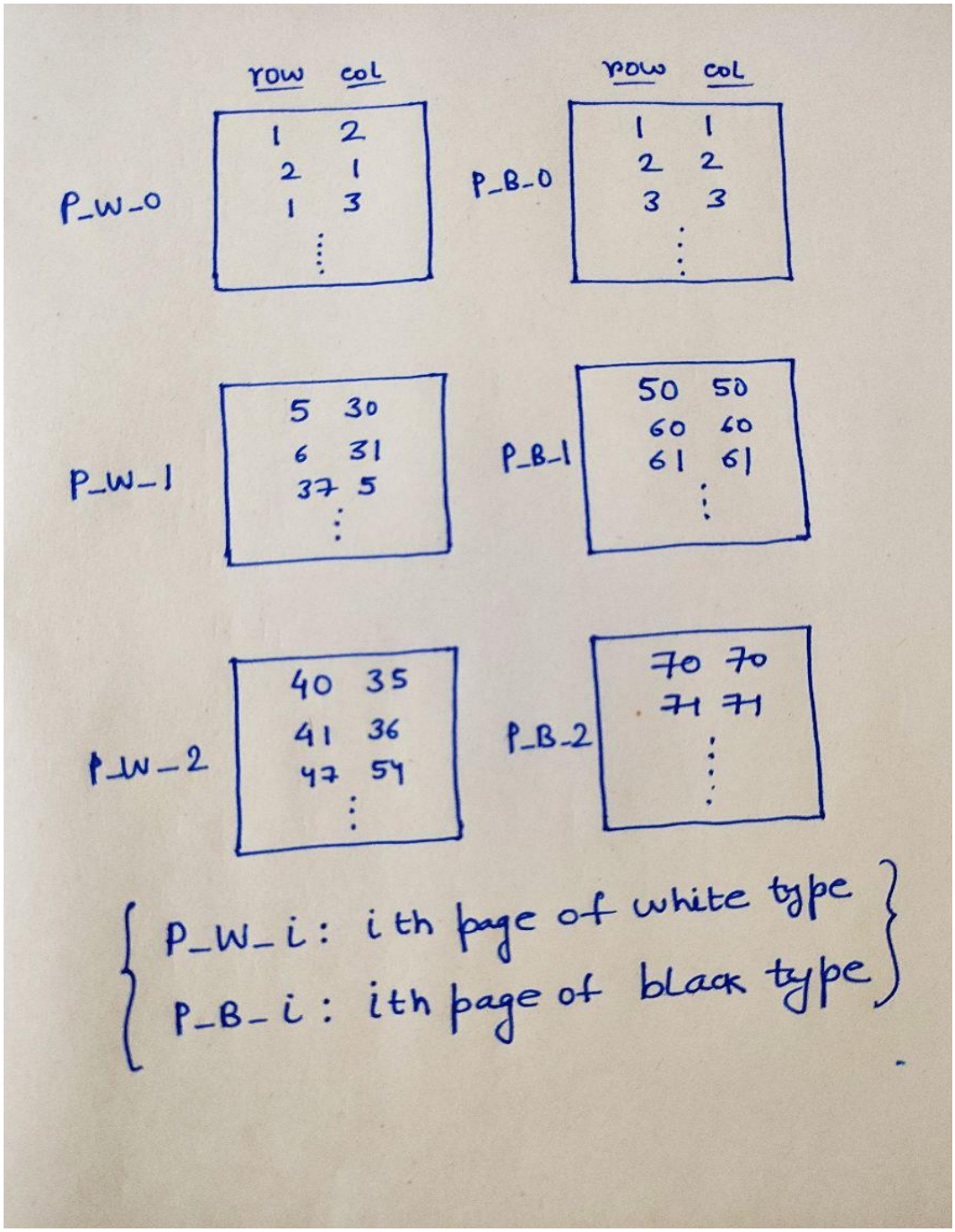
White		Black	
Row	Col	Row	Col
1	2	1	1
2	1	2	2

Page Design in details

- Now, as the game progresses, the number of entries would increase in both white and black. So, we have to divide them into pages.

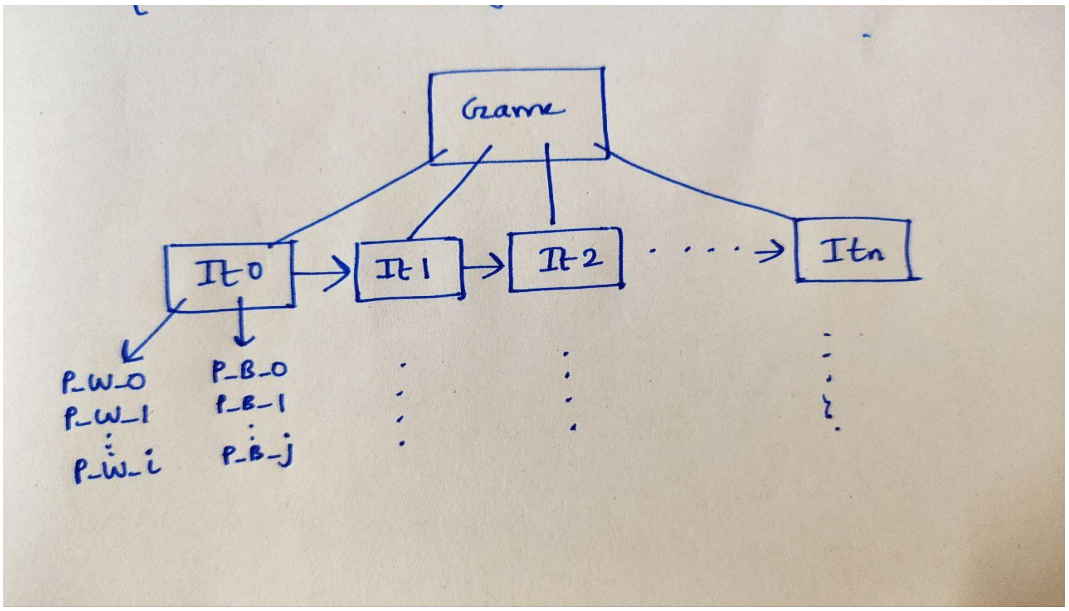
- Now, let's assume, in a page, only 100 entries can fit and we have 300 entries for both White and Black in iteration It_i

  So, the paging scheme will look like this:



- Now, as we keep on playing the game, we can store iteration It_i (store the pages corresponding to that iteration). So, let's say if we have 100 iterations of the game, we need to store all the pages for both black and white for every iteration.

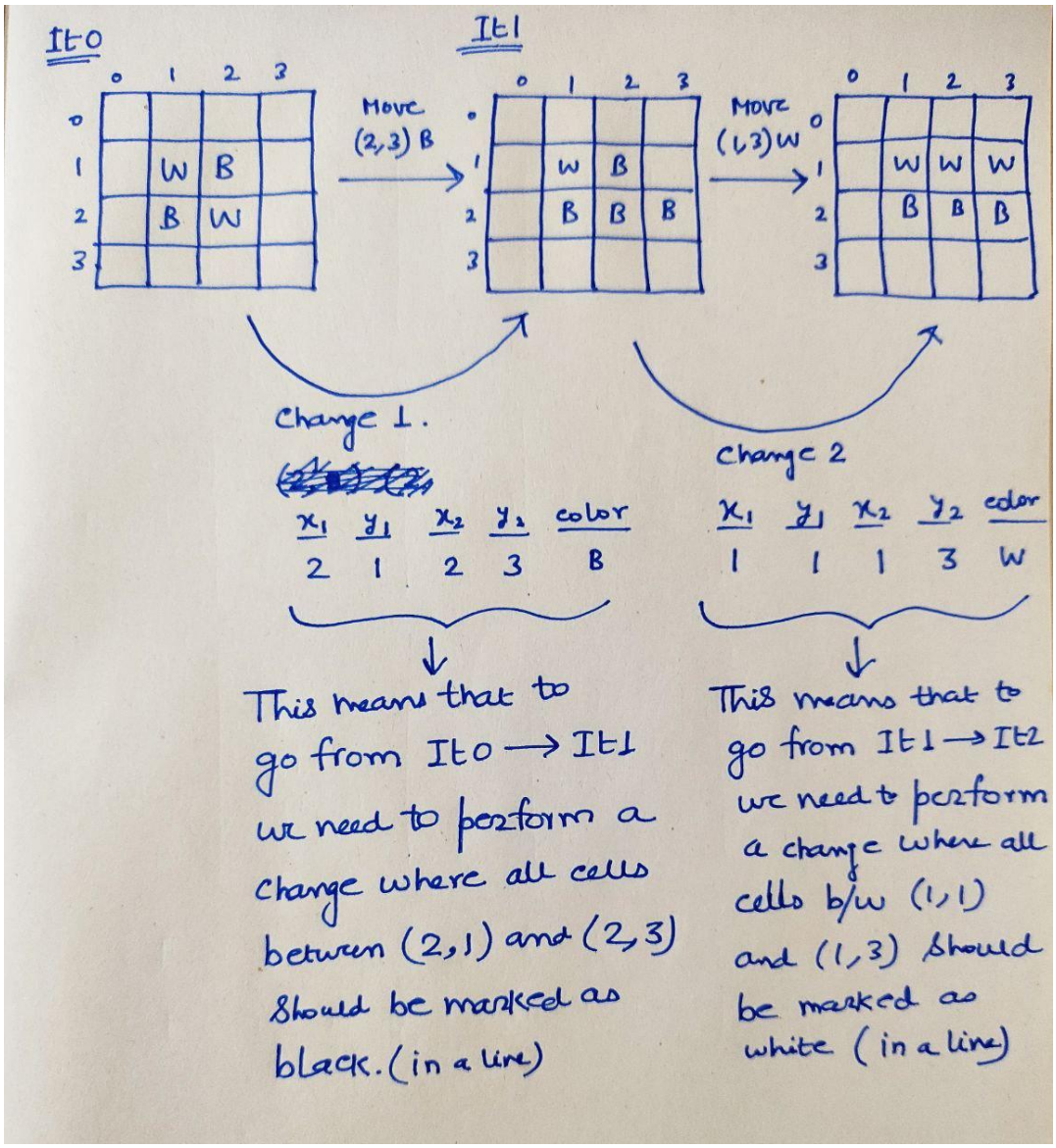
  The situation looks like this:



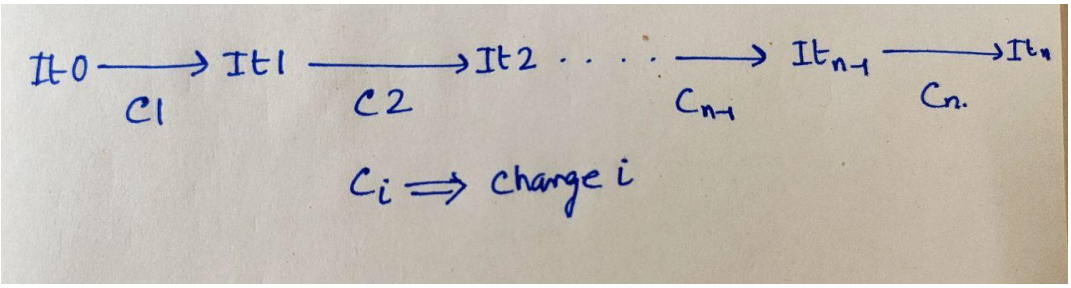
- Now, we need to think, whether we need to store all the iterations(It_i) of the game. Definitely storing all the iterations would be very calculation efficient, since if we are asked to return the ith iteration of the game, then we can return it in O(1) time. But, then it would be every memory heavy since, for each iteration we are storing the entire state.
- **Solution:** What if we change only the changes that happened in each iteration??

  We can store the coordinates of the points whose color have been changed after each iteration **OR** we can just store the end points of the line ((x1, y1) & (x2,y2)) in which the points have changed. Since, storing the details of all the points would take a lot of space, we can use lines to denote the changes.

- The following picture is an example:



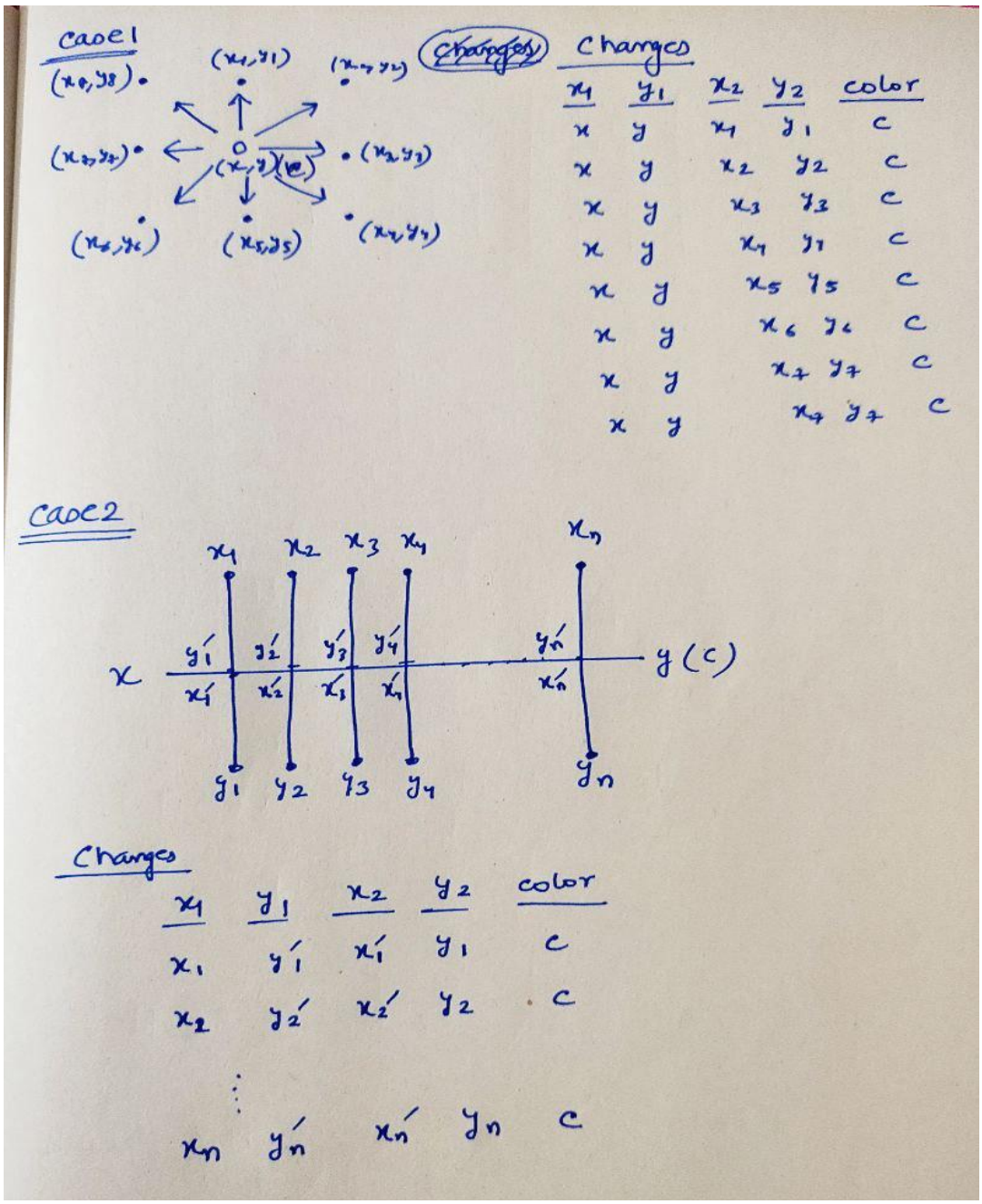
- Now, we can only store the initial state of the board(It_0) and keep on storing the changes(c1, c2, ..., cn). So, if we need state of the board at nth Iteration(It_n), then we can get it like this:



We don't need to store all the iterations in this case, only storing It_0 and storing all the changes would work.

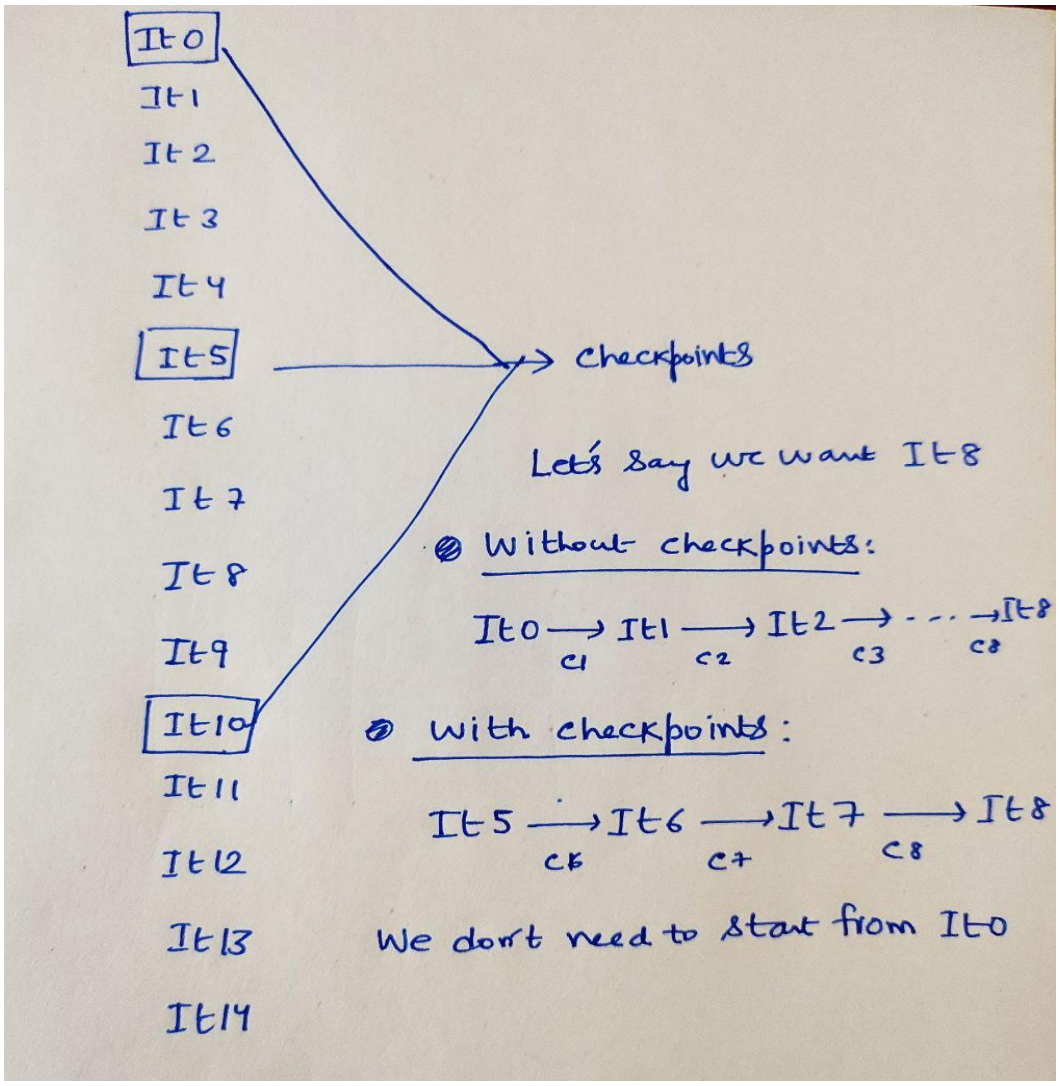
- Now, how big can each change be?

  In average cases, there will be 1-2 entries per change. But in cases like following, the changes can be O(n), where n = 10^3



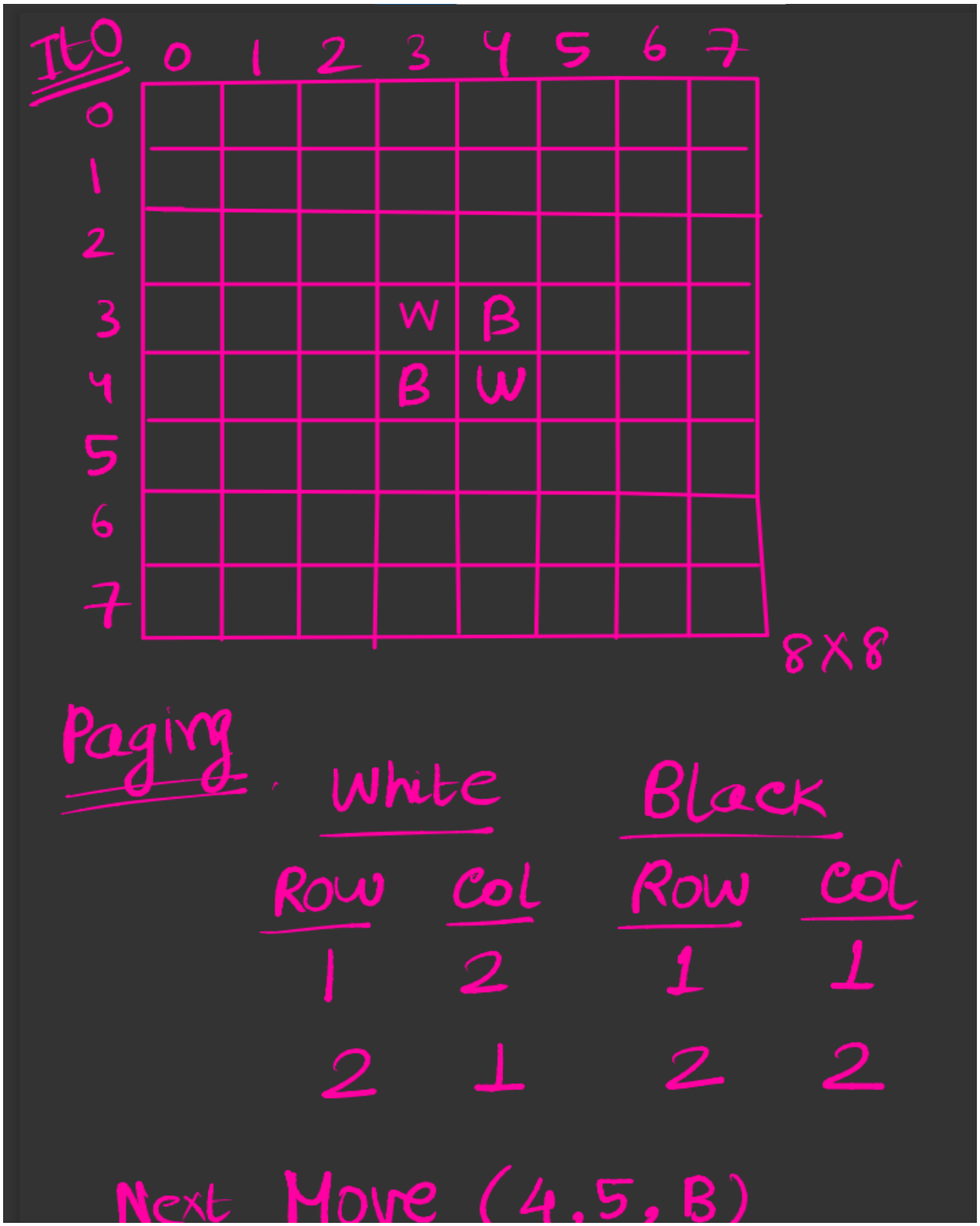
So, in case of changes also, we need to do paging if the number of entries grows very big.

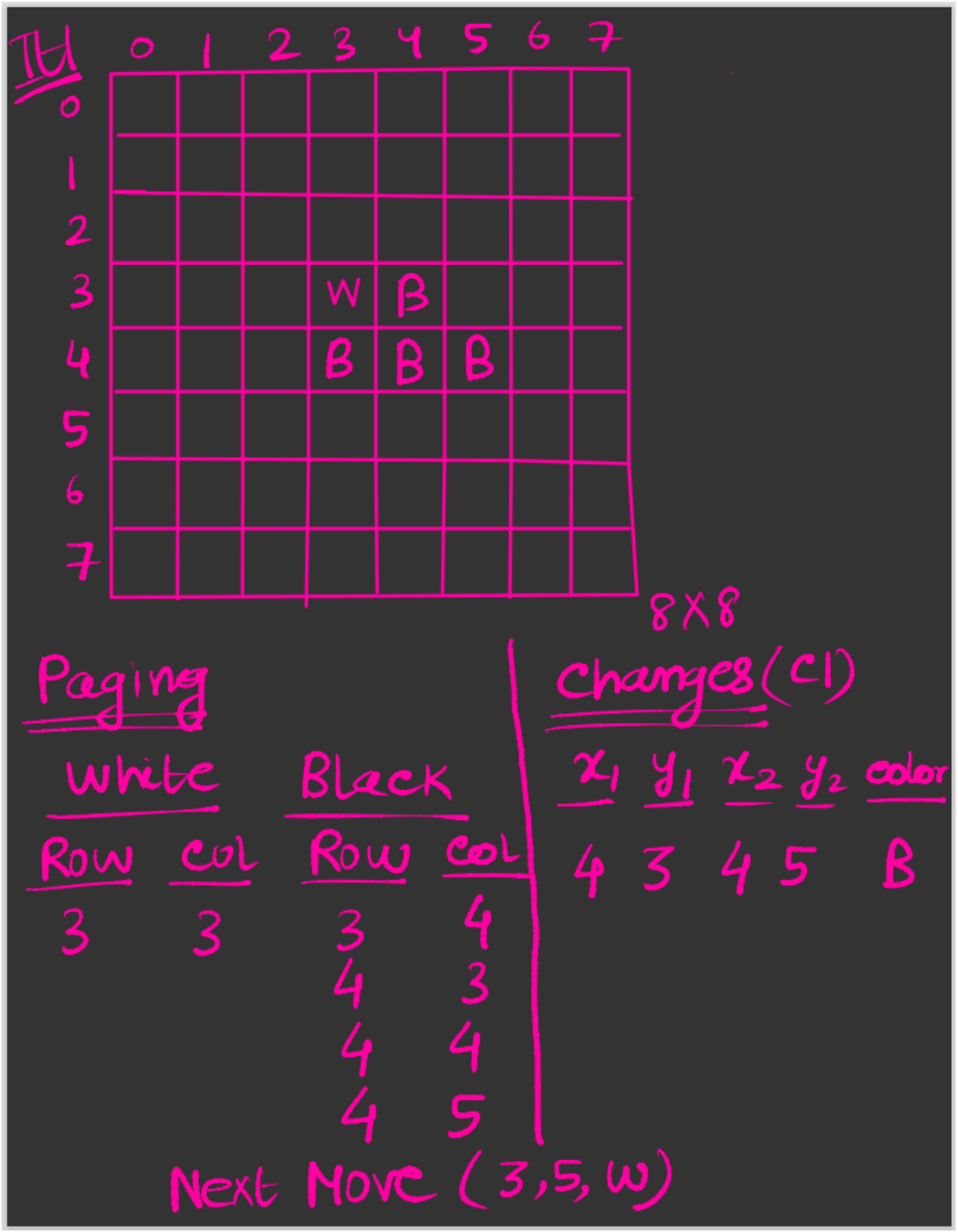
- Now, storing only the first iteration and only the changes can be memory efficient. But, we need to understand that if we need iteration n where n = 10^6, then we have to apply so many changes to get to that state from state It_0.
- **Solution:** What about storing in between states as checkpoints??

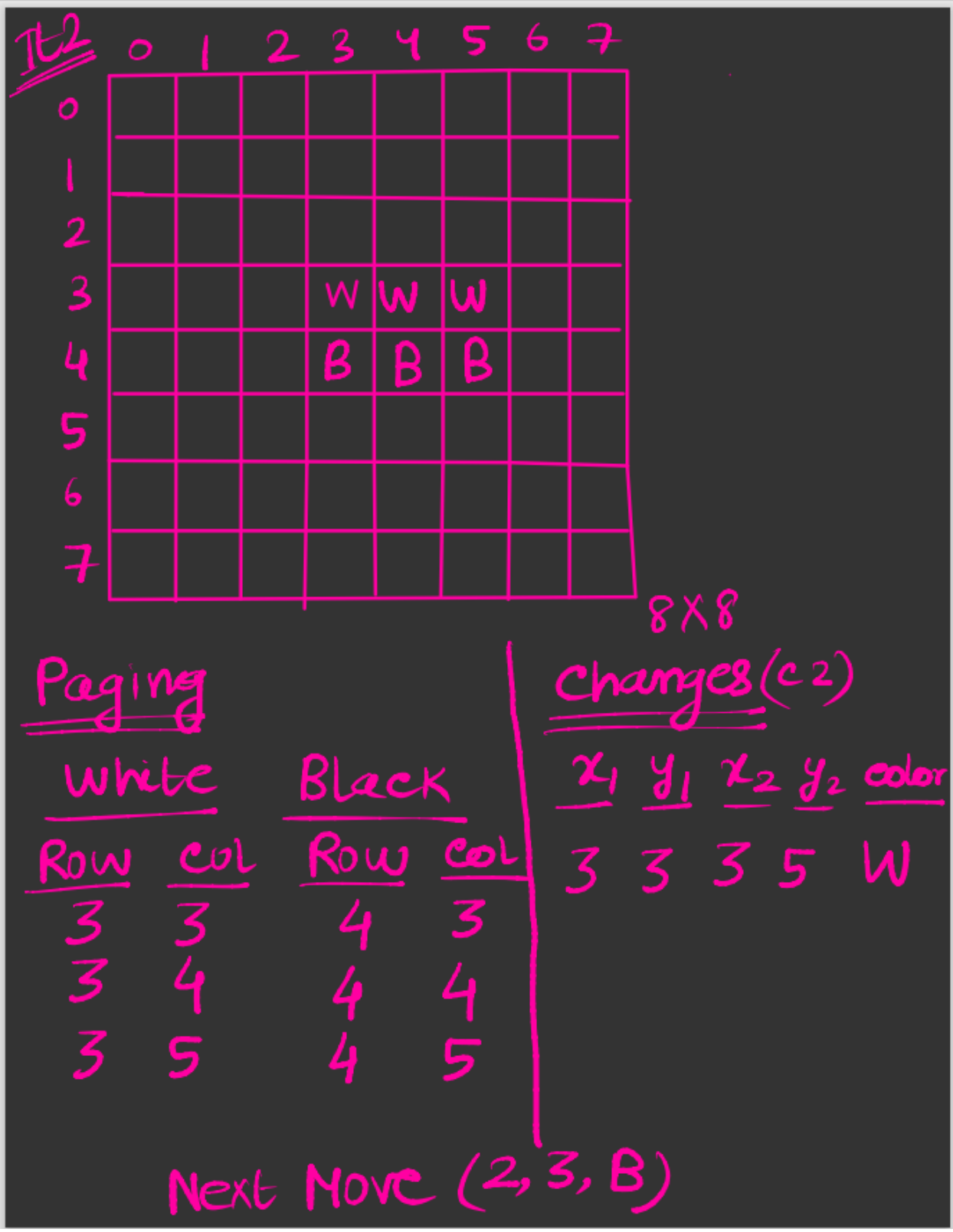


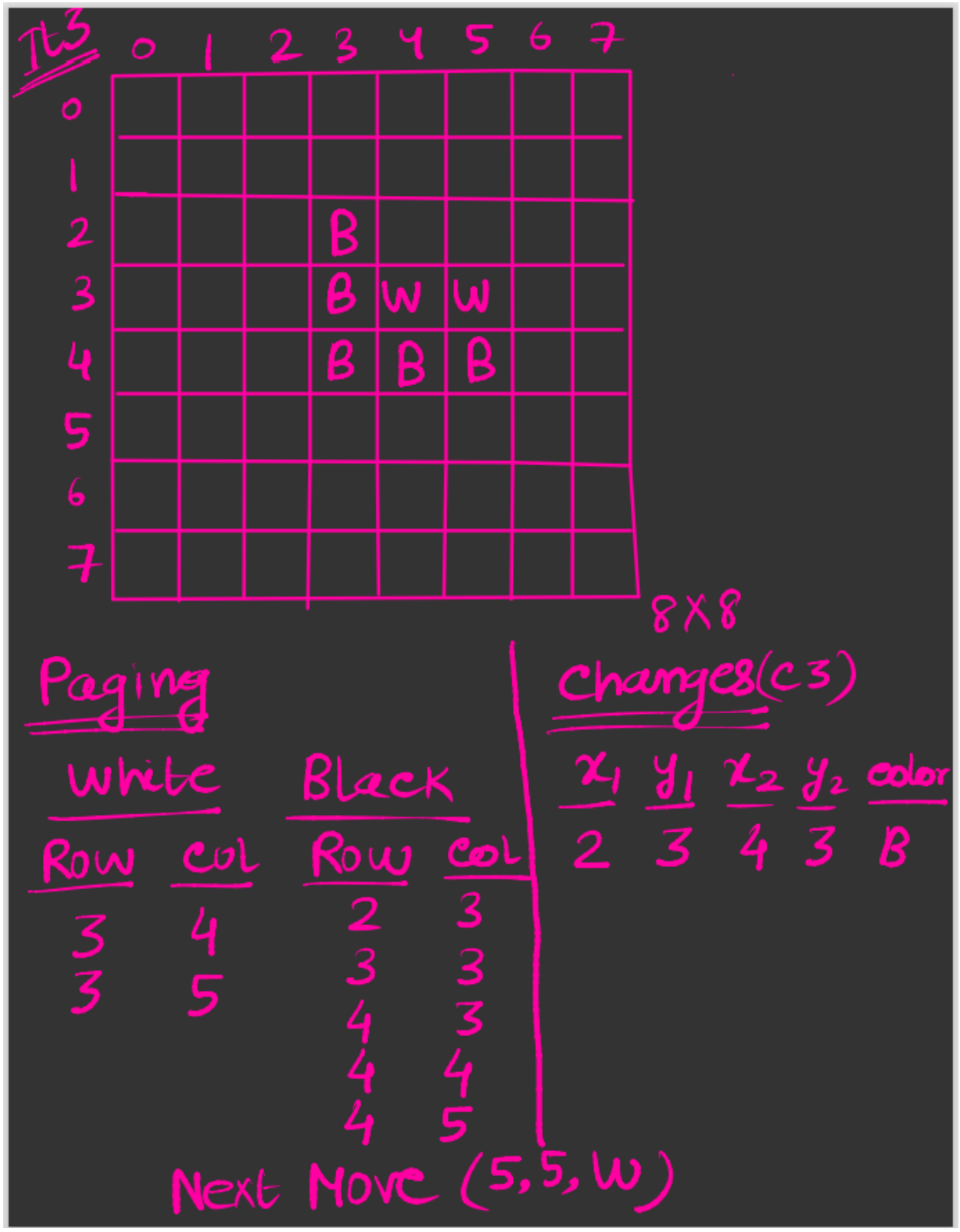
This will be calculation efficient, since we don't have to start from the beginning to get the nth iteration. We can start from the nearest checkpoint to reach the nth iteration.

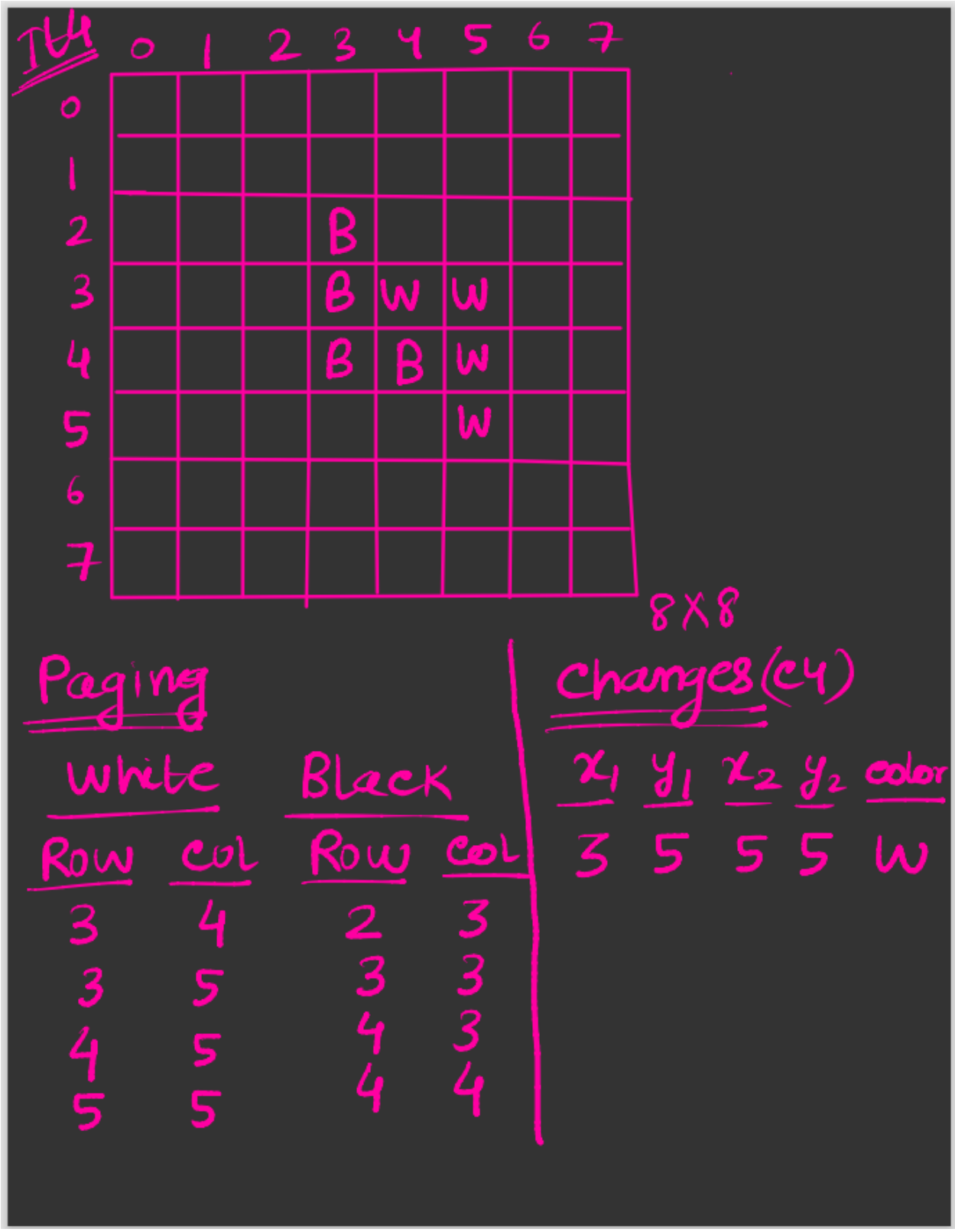
Example in 8x8 board




IH
0 1 2 3 4 5 6 7
0
1
2
3 W B
4 B B B
5
6
7
8X8
Paging
White
Row col
3 3
Black
Row col
3 4
4 3
4 4
4 5
changes (C1)
x1 y1 x2 y2 color
4 3 4 5 B
Next Move (3,5, W)

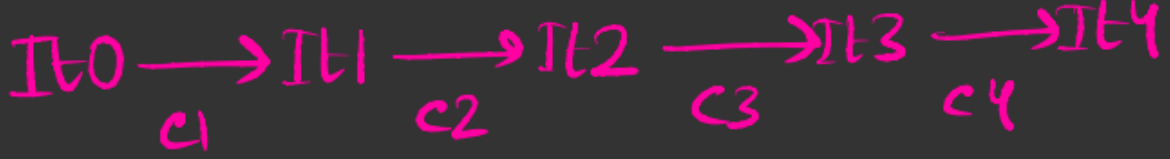

It2
0 1 2 3 4 5 6 7
0
1
2
3 W W W
4 B B B
5
6
7
8X8
Paging
White
Row col
3 3
3 4
3 5
Black
Row col
4 3
4 4
4 5
Changes (c2)
x1 y1 x2 y2 color
3 3 3 5 W
Next Move (2, 3, B)


It3
0 1 2 3 4 5 6 7
0
1
2 B
3 B W W
4 B B B
5
6
7
8X8
Paging
White
Row col
3 4
3 5
Black
Row col
2 3
3 3
4 3
4 4
4 5
Changes(c3)
x1 y1 x2 y2 color
2 3 4 3 B
Next Move (5,5,W)

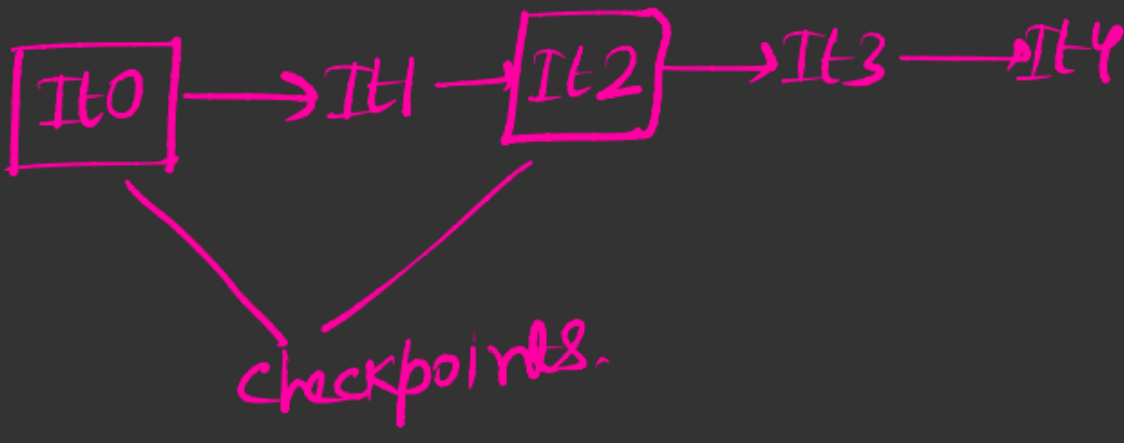

IH
0 1 2 3 4 5 6 7
0
1
2 B
3 B W W
4 B B W
5 W
6
7
8X8
Paging
White
Row col
3 4
3 5
4 5
5 5
Black
Row col
2 3
3 3
4 3
4 4
Changes (C4)
x1 y1 x2 y2 color
3 5 5 5 W

So, we have 4 iterations (It 0, It 1, It 2, It 3, It 4).

Now,



We can store It 0, It 2 as checkpoint. So, to access It 4 we don't need to start from It 0, we can start from It 2



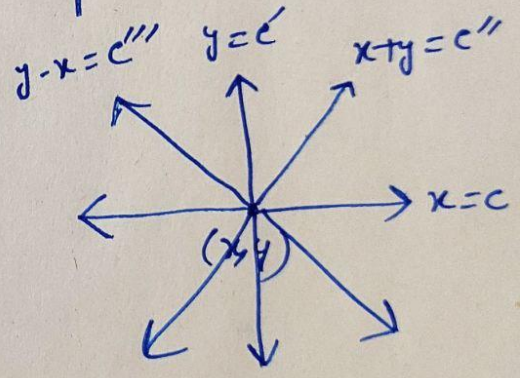
Gameplay at a particular move

Given a new move, (x, y, W/B) we have to find nearest points of same colour in all possible directions.

Then, we need to check whether these two points have points of opposite colour in between them or not. If yes, then perform operation otherwise discard the points.

Since we have the points in sorted manner, we can use binary search to find points.

~~We have to~~



Storage needed to complete one game

Let's assume, block size = 4 KB

1) Storage required to store each iteration/checkpoint

black		white	
row	col	row	col
4B	4B	4B	4B

row, col both of (int) type

∴ Size of each row = $2 \times 4B = 8B$

∴ No of rows per block = $\frac{4KB}{8B}$

$= 512$

NOW, table of size = $10^3 \times 10^3$

∴ In the worst case, we may need to store the entire matrix.

∴ Combining both black and white, total no. of entries in worst case $= 10^6$

∴ no of blocks required to accomodate that $= \left\lceil \frac{10^6}{512} \right\rceil$

$= 1954$

∴ Total storage required to store 1 checkpoint in worst case $= 1954 \times 4\ KB = 7\ MB$

∴ If we keep 50 checkpoints, then total storage required for checkpoints $= (7 \times 50)\ MB$

$= 350\ MB$

2) Storage required to store each change

In worst case, changes can happen in 8 directions.

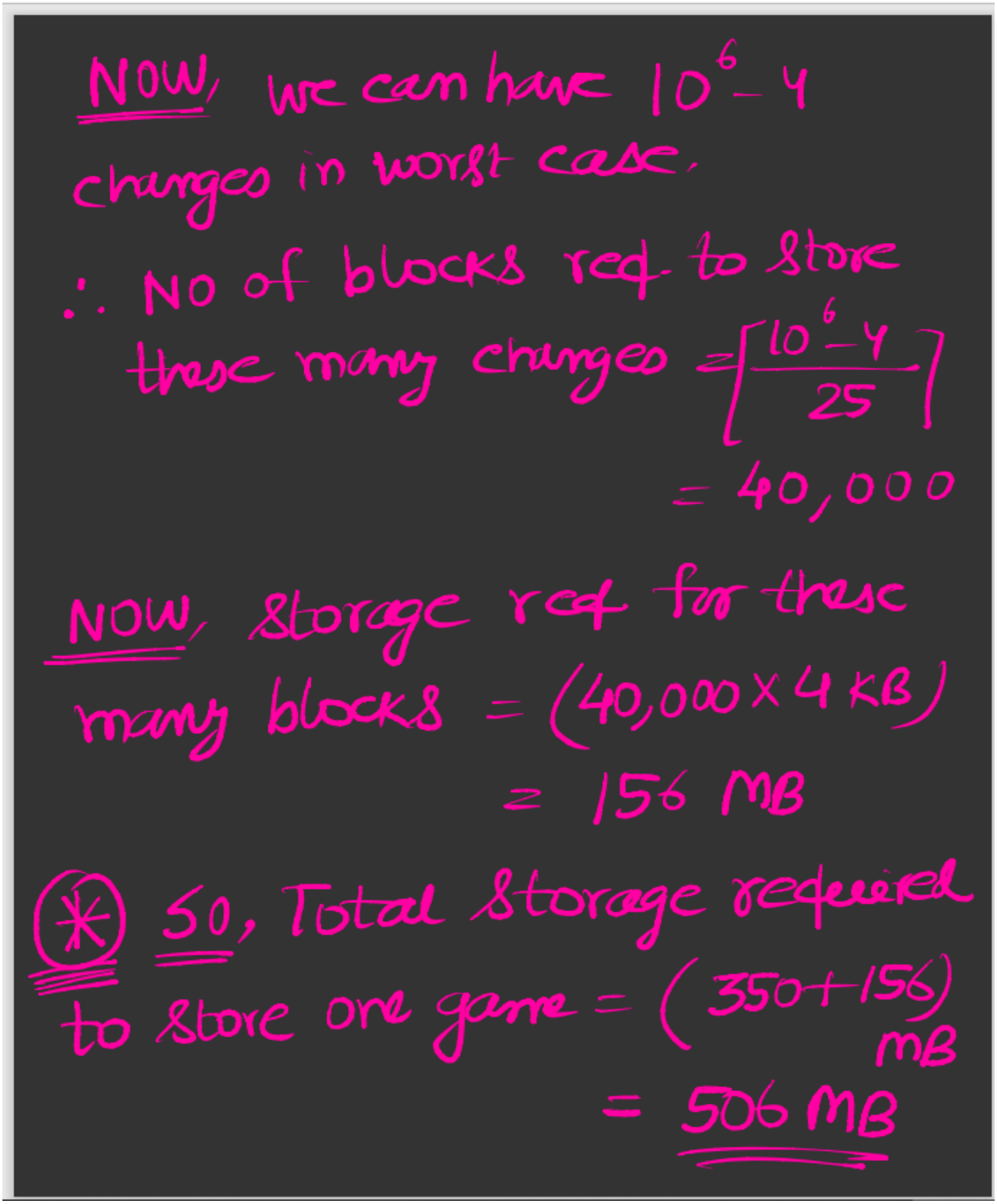
. $\frac{x_1}{4B}$ $\frac{y_1}{4B}$ $\frac{x_2}{4B}$ $\frac{y_2}{4B}$ $\frac{color}{4B}$

∴ Each row = $(4B \times 5) = 20B$

∴ no. of entries possible per block = $\left\lfloor \frac{4KB}{20B} \right\rfloor = 204$

NOW, maximum no. of entries possible in 1 change = 8

∴ In 1 block, we can store $\left\lfloor \frac{204}{8} \right\rfloor = 25$ changes



Again, it depends on the number of checkpoints. If we increase the number of checkpoints, memory consumption will be higher. If we decrease the checkpoints, memory consumption will be lower.

Why is it the best design??

- We have adopted sparse matrix representation here. It is memory efficient since it does not store unnecessary information like storing unoccupied cells.
- We are not storing the state of the board for each iteration, we are only storing a limited number of iterations as checkpoints.
- The advantages of using checkpointing are:
    - we don't need to apply all the changes one after another on top of the first iteration(It_0) to get the nth iteration(It_n). We can go to the nearest checkpoint It_p and start applying changes on top of that to get the desired iteration.
- We are storing only the changes, not the state of the board for each iteration. Also, for the changes, we are not storing each and every point that is being modified or added. We are only storing the endpoints ((x1,y1), (x2,y2)) of the lines whose points are being changed. So, there also we are optimizing memory space.
- At the end of the day, it's a tradeoff between space (secondary storage and memory) and time (calculation efficiency and access efficiency).
    - If we want efficiency in calculation and access(Get state of the ith iteration in O(1) time), then we can store each iteration separately.
    - But if we want memory optimization, we can store only the initial state of the board(It_0) and the changes after each game play.
    - If we want both memory and calculation optimization, we can store some intermediate states of the board as checkpoints along with the changes and initial state. So, if we want the ith iteration, we don't need to start from the 0th iteration, starting from the nearest state and then applying changes accordingly would work fine.

Directory Structure