Data Systems

Reversi/Othello - Group Assignment 2

Sudipta Halder (2021202011)
Vishal Pandey (2021201070)
Group Number - 13

Similarity between 2 board orientations of the game:

We are using Manhattan distance to find out the similarity between two boards. There can be three possibilities for a cell.

Case 1: the cell can be **empty**(let's denote it by **'0'**),

Case 2: the cell can have **white** piece in it(let's denote it by **'1'**),

Case 3: the cell can have **black** piece in it(let's denote it by **'2'**).

Now, we can basically convert the whole state of a board to a string. We can traverse each row of the board one by one from top to bottom and then stringify the entire board. So, the resultant string will be of length (n*n) if the board is of size (n*n)(Since each row is of size n and there are n such rows).

Refer to the example below for better understanding:

0	1	2
2	0	2
1	1	1

Board 1

1	1	0
2	2	1
2	1	2

Board 2

So, Board 1 will generate a string like this : “0 1 2 2 0 2 1 1 1”

And, Board 2 will generate a string like this: “1 1 0 2 2 1 2 1 2”

So, now the distance between two boards will be = sum of manhattan distance of each character present in those two strings.

So, diff between two boards = |1-0| + |1-1| + |2-0| + |2-2| + |2-0| + |2-1| + |2-1| + |1-1| + |2-1| = 8.

Note: For two exactly identical boards(same blank cell, same number of white and black pieces, same position of balck and white pieces), the difference should be = 0.

KNB(K Nearest Boards) Algorithm:

We have millions of played games. For each played game, we have (10^6 - 3) possible state of boards(Initial move + (10^6 - 4) moves thereafter). So, **for each state of the board of Game$_i$** we would store a **metadata** which would store the following things:

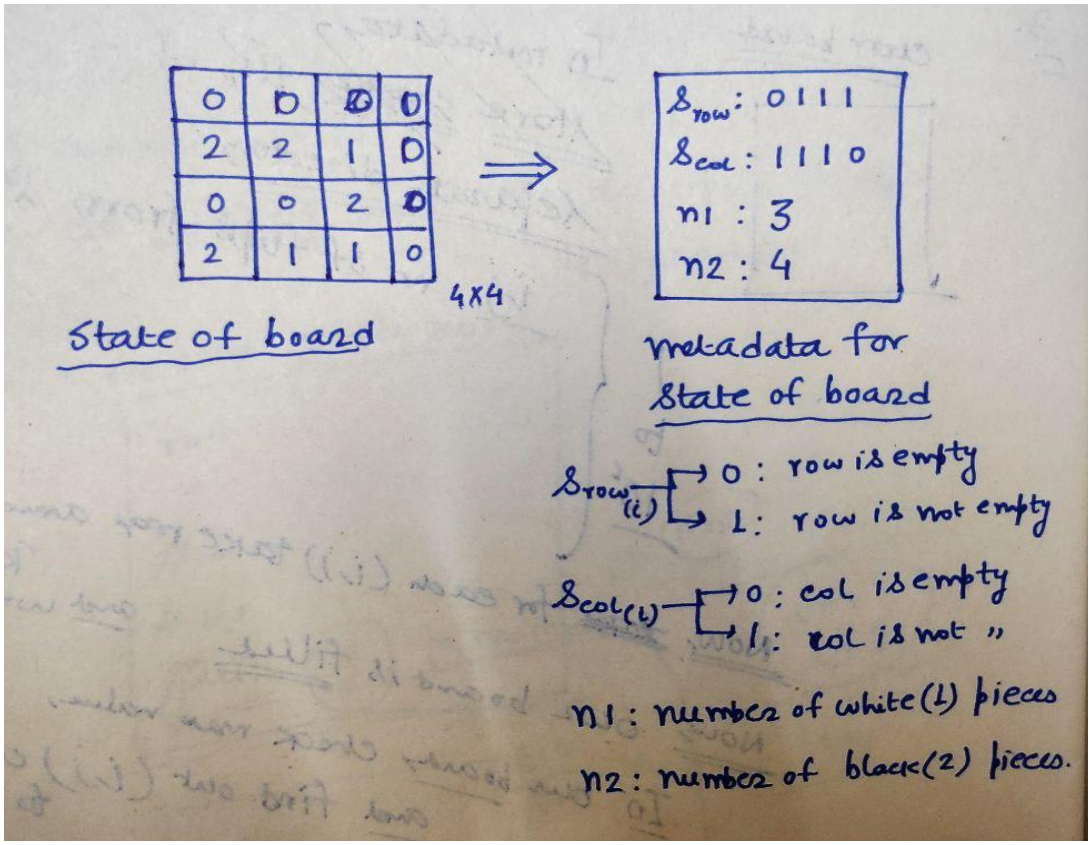
1. A string of length n (consisting of 0 or 1), the ith position of the string actually represents the ith row of the board. If it is 0, then the row is empty, if it is 1 then the row will be occupied by at least one piece(BLACK OR WHITE).

2. A string of length n (consisting of 0 or 1), the ith position of the string actually represents the ith column of the board. If it is 0, then the column is empty, if it is 1 then the column will be occupied by at least one piece(BLACK OR WHITE).

3. The number of white pieces present in the board at ith state(say n1).

4. The number of black pieces present in the board at ith state(say n2).

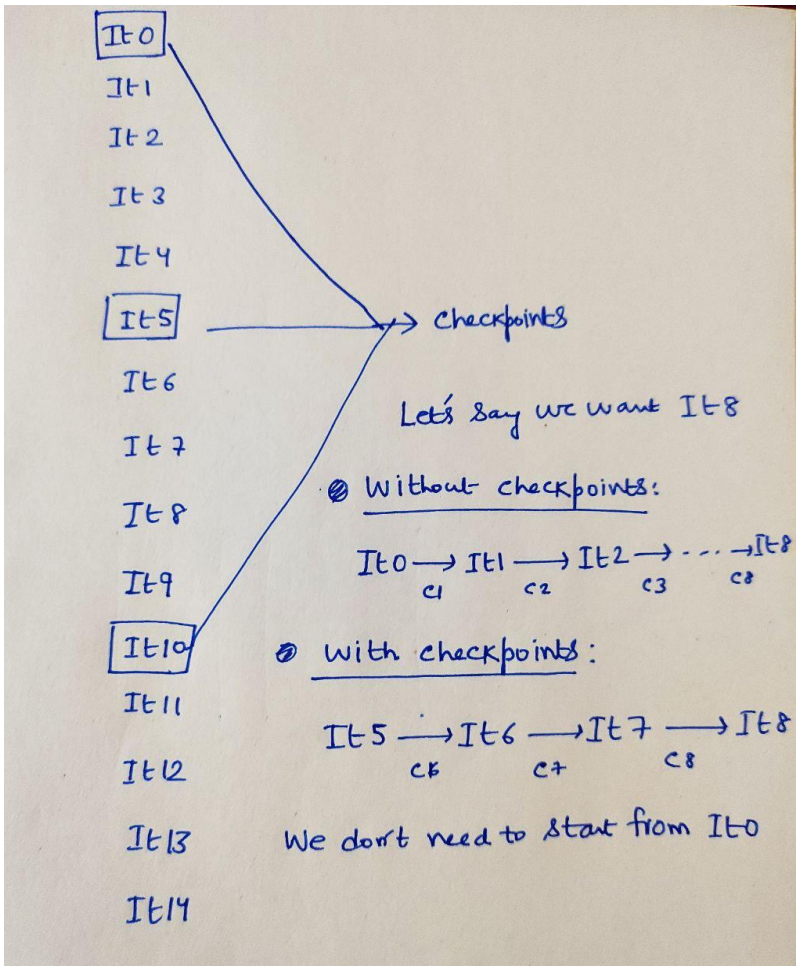
Refer to the below diagram to understand the metadata clearly:



Similarly we will have metadata for our current game also. If for our current game, we have played 'i' moves till now, then there will be (i+1) state of the boards. So, we will have (i+1) metadatas.

Step 1: So, the first step of KNB will be to pull out the metadata of all the boards (which have already been played) at ith state and match the metadata with our current board at ith state. The boards whose metadata would match completely, would be considered for further verification.

Step 2: Now, we have filtered out some boards based on metadata matching. Let's say, n boards have passed this test. Then, we would fetch the n boards at move i.(By applying checkpoint and changes.patch as discussed in Assignment 1, see below pic).



Then, we would apply the previously declared algorithm (Similarity between 2 board orientations of the game) (stringify two boards and then apply manhattan distance between two strings). So, for each pair of boards, we would get 1 manhattan distance.

So, we would get a list of Manhattan distances {d1, d2, d3, ..., dn}. So, among them we have to find out 'k' such boards who score minimum among them (We would try to find the boards which have manhattan distance = 0 with our current board).

Note: Since we have million such boards, we can be optimistic here that we would get **exactly 'k' identical boards** (who would have **Manhattan distance '0'** with our current board). If it is not possible to get 'k' identical boards, then we would take 'l' identical boards (l<k) and (k-l) similar boards.

This is how we would select 'k' nearest boards.

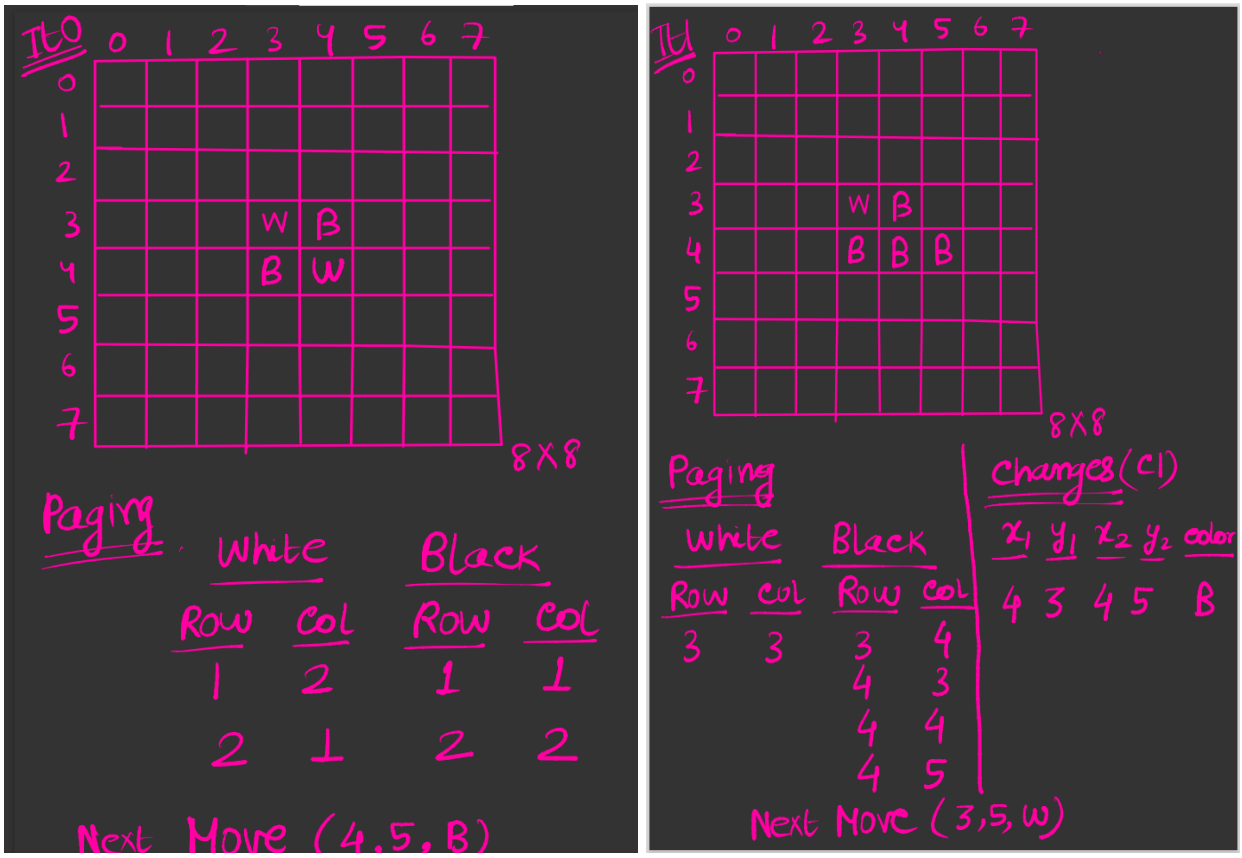
1) Query 1 [LONGEVITY]: You are given a position (i,j) and the move on which you are currently at.

a) **Whether the color at position (i,j) will be flipped in the next move or not?**

Ans: We are at xth move. So, we have our board ready for our current game. Now, we would find out the k nearest boards(k identical boards) who are also at xth move corresponding to our current board. Now, we would analyze those k boards, whether the color for (i,j) position has flipped in them or not. If between k boards, at least for 1 board the (i,j) position has flipped color then we can say that, for our board also, the (i,j) position will flip its color in the next move (because, we have got one possible move for which it's possible).

Now, we **don't need to fetch the complete state** of the k boards for this. We **can do this** by analyzing the **changes.patch(for each move separate change.patch file)** file which we mentioned in Assignment 1. Refer to the below diagram for clarification of change.patch.

In $change_i$, we store the end points of the line segments ((x1,y1) and (x2,y2)) in which the changes have occurred. So, if (i,j) satisfies the equation of any of those lines, then it is proved that for that coordinate the color has flipped.



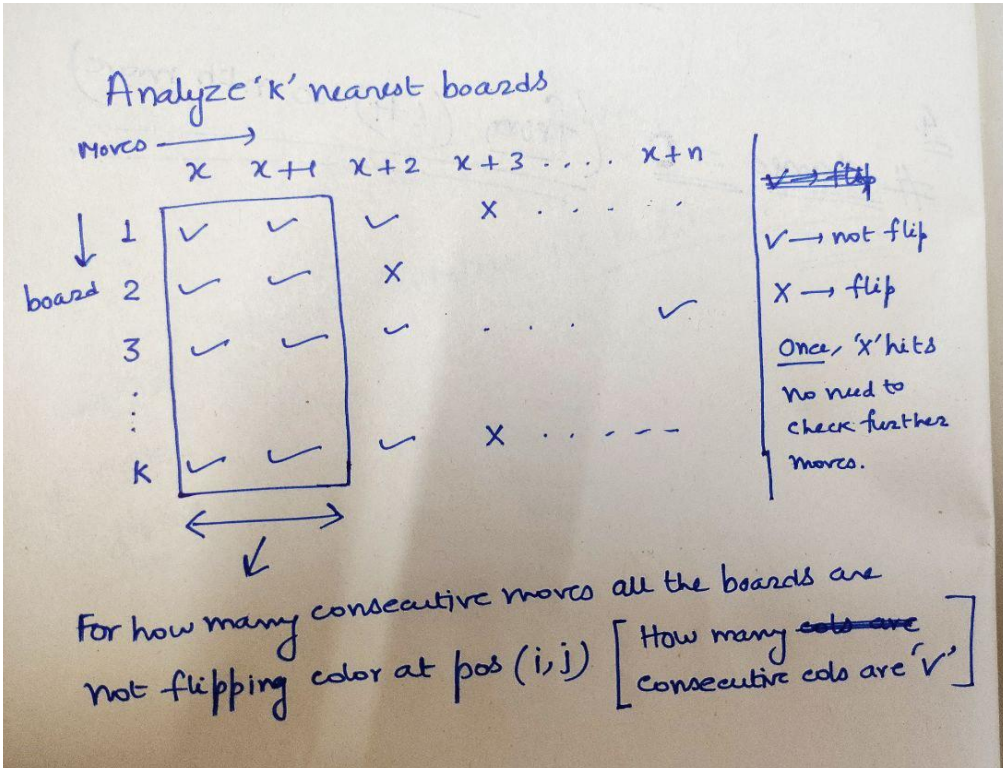
Changes in this diagram

b) For how many moves you can guarantee that this position will not be flipped?

Ans: Similarly, we would find out the k nearest boards(k identical boards) who are also at xth move corresponding to our current board. Now, for position (i,j) we have to check each board and find out whether it has flipped its color in next moves or not. Let's assume, we run this experiment n number of times. So, if we are at xth move, then we go till (x+n)th move for each of k boards.(x, x+1, x+2, x+3, ..., x+n).

We have to find out for how many consecutive moves, all the boards are not flipping color at pos (i, j).

Refer to the below diagram for clarification:



Again, for finding out whether position (i,j) has flipped or not for a board, we don't need to fetch the entire board. We can achieve this by only referring to the change.patch file for that particular move.

c) In the next move, all positions in the board that are going to become flippable

Ans: Similarly, we would find out the k nearest boards(k identical boards) who are also at xth move corresponding to our current board. Now, we would analyze those k boards.

So, for each (i,j) we would analyze k boards. If any one of the boards has changed color in that move then we would declare that pos (i,j) would also flip color in our board.

So, we would analyze 10^6 positions(i, j). Among them, the positions which would satisfy the criteria, we will return them.

Again, for finding out whether position (i,j) has flipped or not for a board, we don't need to fetch the entire board. We can achieve this by only referring to the change.patch file for that particular move.

2) Query 2 [DIFF IN FLIPS]: Suppose you are at the i th move and when player 1 plays (white) it turns "k" blacks to white and subsequently when player 2 plays (black) it turns "l" whites to black. Then we can define gain for player 1 as (k-l).

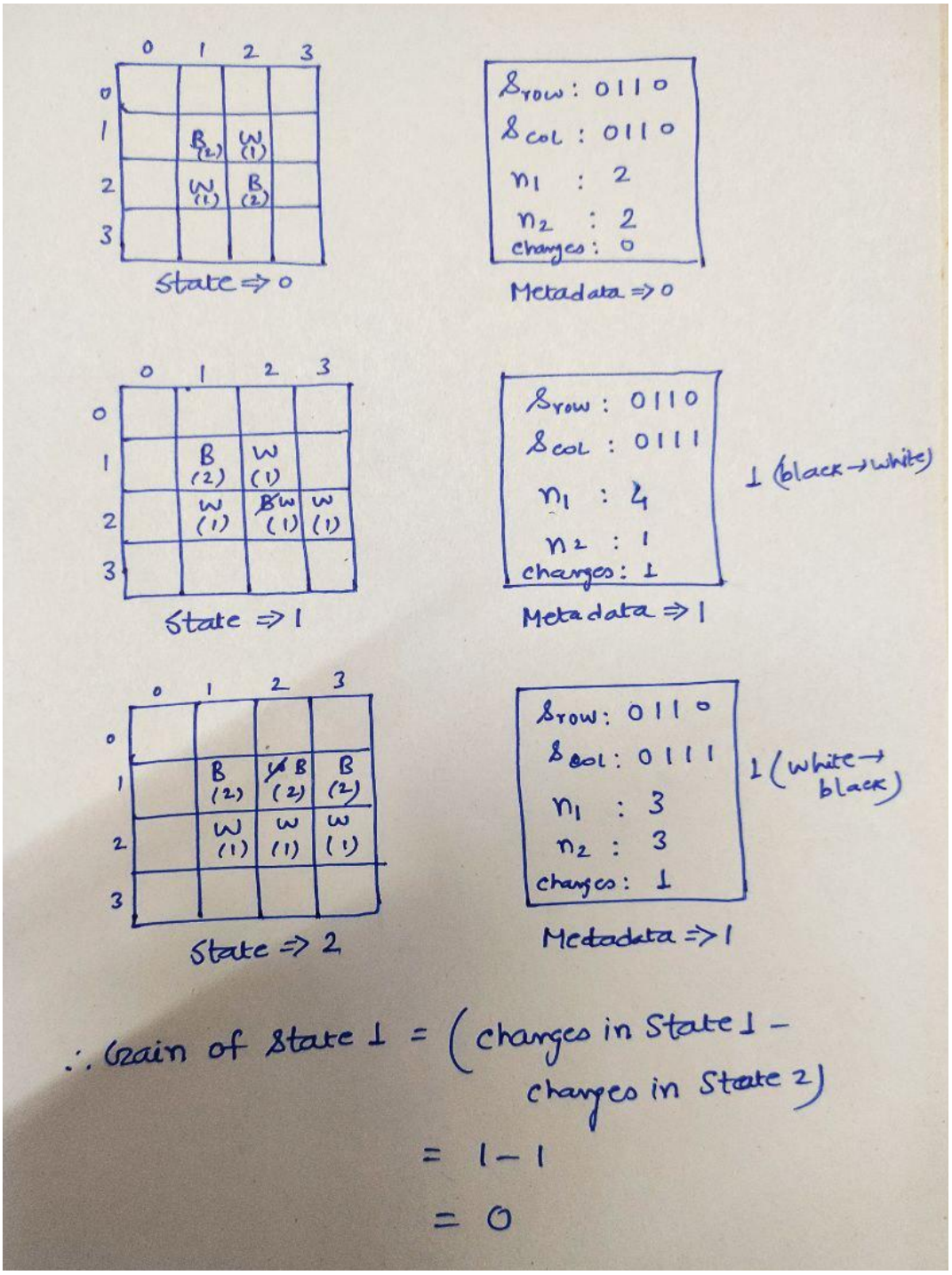
a) **Return all the i's such that player 1 is gaining at least "b"**

Ans: Similarly, we would find out the k nearest boards(k identical boards) who are also at ith move corresponding to our current board.

Now, we need to also update the metadata. We would also want to introduce a **number of changes**(**black to white** or **white to black**) in the metadata.

So, gain for ith move would be calculated by (number of changes in metadata of ith move - number of changes in metadata of (i+1) th move).

Refer to the picture below for clarification:



So, we would analyze all k boards. **For ith move, if for all the k boards, the gain is >=b then only** we would declare that **for that ith move our current board will also have gain >= b.**

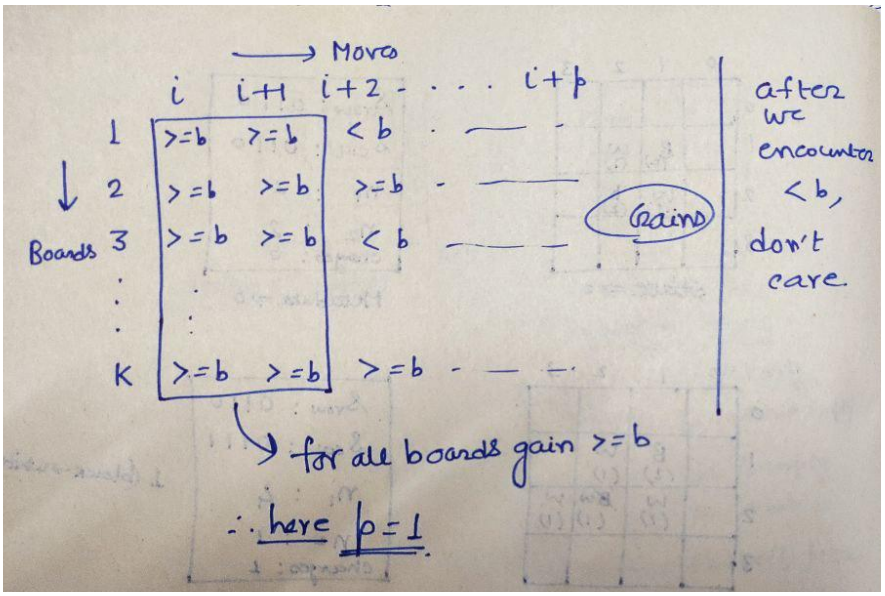
In this way, we will run from i from (i+1 to n(last move)), whichever i would satisfy the criteria, we would include that i in the result set.

b) **Return all the sequences of moves (i,i+1,i+2, ... i+p) such that player 1 is continuously gaining at least b in all these moves and p is the highest such number.**

Ans: This is similar to Q1b. We would analyze the K nearest boards.

We have to check for how long (i, i+1, i+2, ..., i+p), all the k boards are maintaining a streak of gain >= b. We have to return that p.

Refer to the following picture for clarification:



3) Query 3 [MAX FLIPS]: Return all locations (i,j) which suffered maximum number of flips which happened throughout the games played.

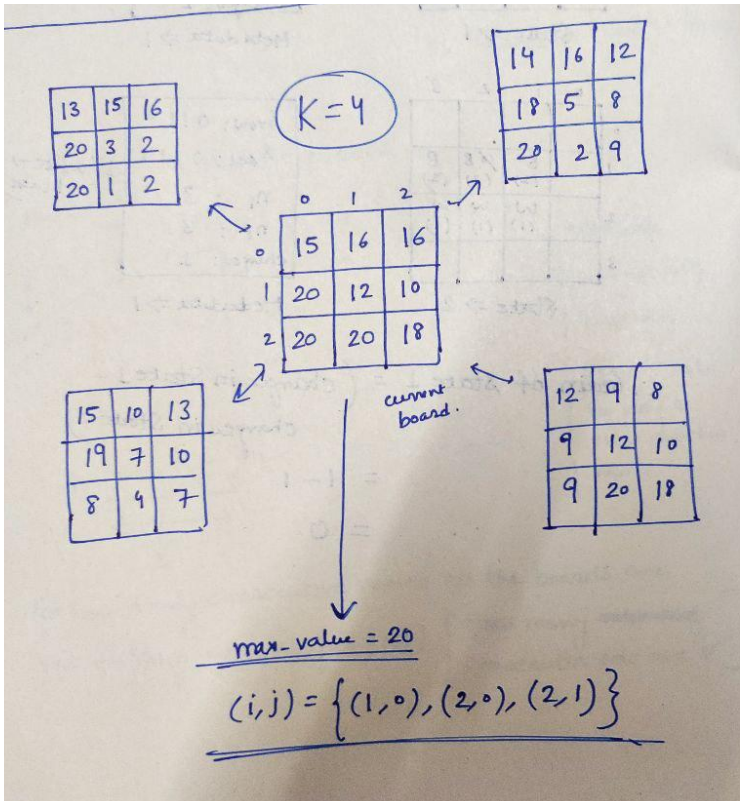
Ans: We would make a separate global directory for each Game. In that directory, there will be n*n cells similar like boards, in which how many times the cell (i,j) has flipped will be stored(from start to end).

Now, we would analyze the k nearest boards.

For each position (i,j) we would go through all k boards and whichever board has max value in its directory, we would take that.

Now, we have got max no of flips for each (i,j). Now, among all the numbers of flips, we need to find out the max flip. Then we would answer those (i,j) from our current board which satisfy those maximum number of flips.

Refer to the below picture for clarification:



4) Query 4: For the given board orientations, give me all the positions (i,j) which will not change forever (based on the KNB) . That is No matter what you or the other player plays the color at this position (i,j) will not change.

Ans: Now, we would analyze the k nearest boards.

We would go from (i+1) th move to the last move. In each move, we would consider k boards and its directory. For each move (i,j) if all the boards are not changing color at position (i,j) (basically value = 0 in directory) then we would include that (i,j) in the result set.

We would return the result set consisting of such (i,j)'s.

5) Open Ended: What all other queries (at least 2) might be there in such a given scenario and how will you execute them?

Ans: Following are the possible queries:

a) **Query 1:** What is the probability that player 1 will win?

Ans: The game starts with black pieces. Basically, between two black pieces, all the white pieces get converted into white. So, for winning of player 1, the number of black pieces must be greater than white pieces at the end of the game.

So, here also we would find out K nearest boards(K identical boards). Among these k boards, we would find that in how many boards the number of black pieces are greater than the number of white pieces. Let's say, among k boards, m boards are such that where (number of

black pieces > number of white pieces). Then the probability of winning of Player 1 is k/m.

b) **Query 2:** What is the maximum possible margin of victory by which any player can win after i^{th} move?

Ans: Fetch K nearest boards at move i. Then, analyze those k boards, in the last move, in how much margin they have won the game. So, if we get k margins (m1, m2, m3, ..., mk), then we need to just find the maximum among them.

6) KEY TAKEAWAY: Understanding the connection between Indexing, Page Design and Query Execution. (Comment on it)

Ans: There is a strong correlation between indexing, page design, and query execution. We use indexing to aid in the database's structural design. Different architectures will lead to the delivery of responses to different queries at various rates. Indexing on a parameter will make it easier to discover the solution to a query that depends on that parameter but may make other queries take longer time.
Page design is also highly correlated with query execution. It is easily visible when we need to make changes in the page design according to the type of query. For example, here we had to add metadata to make queries faster(Refer to KNB). If we had to compare all the boards, then the time complexity could have been much higher. But since we applied metadata for each state of the board, it was pretty easy filtering out only the useful boards and then eventually applying the similarity algorithm to those specific boards only. Also, we took the help of a global directory for each board, which helped us to find the maximum number of flips throughout the game.
Also, secondary indexing is really helpful while calculating gains of a particular move, which reflects the relation between page designing and indexing in a good way.

Thus, all these 3 parameters are correlated and highly connected with each other