Data Systems: Assignment-2

P Shiridi Kumar (2021121005) Shikhar Saxena (2021121010)

Page Design Summary

We have used the same page layout as in our first assignment. The only change made is that we have also added `game_id` field in our records. For more detail refer to our first assignment.

Each step in a game consists of bunch of updates on the positions on the board. Each of these updates can be stored as a fixed length record.

The record attributes are as follows:

`update_tuple(game_id, step_number, row_position, column_position, updated_value, pre-viously_empty)`

Attribute	Description	Domain of Values
`game_id`	The associated Game ID for this record	Positive Integer values (starting from 1 onwards)
`step_number`	The associated step/move in the game (for this update)	Positive Integer Values
`row_postion`	The row position in the board where this update applies	Integer with `min value = 1` and `max value = Number of rows`
`column_postion`	The column position in the board where this update applies	Integer with `min value = 1` and `max value = Number of columns`
`updated_value`	The final value at the corresponding update position. Semantically, takes `BLACK` or `WHITE` value	Storing `WHITE` as 1 and `BLACK` as 2
`previously_empty`	Boolean value representing whether the corresponding position was empty before the update or not	Storing 1 for `NOT EMPTY` and 0 for `EMPTY`

Similarity between two board games

We compute similarities between two board games based on **Hamming Distance**. Essentially we compare each cell of the two boards. If both cells have the same configuration (both black, both white or both empty) then `distance=0` otherwise `distance=1`. We add the Hamming distances for all the cells, to get the Hamming distance between the two boards respectively.

Similarity is computed as the inverse of hamming distance *i.e.*, more the hamming distance, lesser the similarity and vice versa.

Example

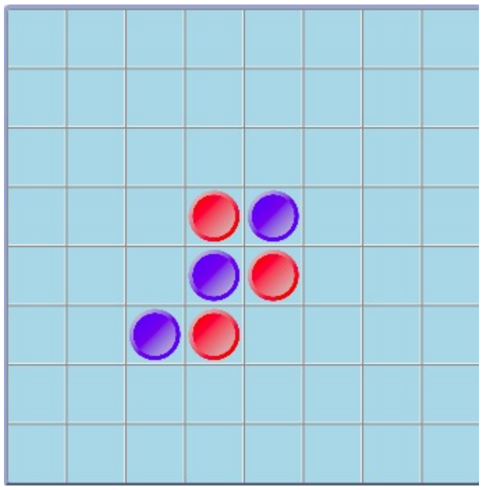
Figure 1: First Board Configuration

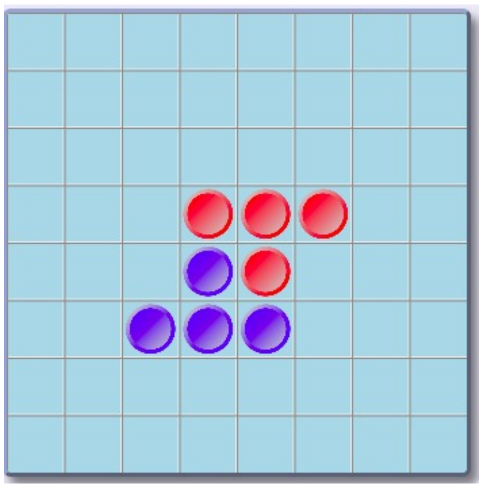
Figure 2: Second Board Configuration

The Hamming distance between these two boards is 4. The different cells are (6,4), (6,5), (4,5) and (4,6).

KNB Algorithm

For finding the K nearest board positions , we initially find the hamming distnace for the inital board configuration and our query board position ,for that specific game and let the board configuration at each move be stored in temp.For the rest of the configurations in that specific game we follow the below steps :

- Lets suppose the hamming distance is x for the previous state . Since our page design actually stores only the updated positions at each move in the game we will actually have the board positions which are actually updated in the next move for that game.
- Now for checking the hamming distance between the query board position and the next board position , we just compare the updated moves for the next board configuration with the current board configuration and our query board configuration. i.e, lets suppose our current board differs the query board by just 4 board positions (let them be :(1,4),(1,0),(2,1),(3,1)). and next move has updated values stored which are (4,2),(1,4),(2,3) . for each updated move we can have the following changes :
  - If the updated cell value actually matches the query board value, at that position, we decrease the hamming distance by 1 (i.,e -1)
  - If the current board value (before the update) and the query board value, at the updated position, is same then we actually increase the hamming distance by 1(+1)

  - For other cases, there is no change in the hamming distance.

- After checking the above conditions, and changing the hamming distance accordingly, we just make the updates (at the updated positions) in the current board(i.e, TEMP). We go to the next move and the entire procedure repeats again till the end of the game.
- At each step we will keep a track of the board position(we have ids for both game and the move in our page design) which have the least hamming distance.
- We repeat this procedure for all the games in the database.
- By the end of this, we will have a map of `game_id`, `move_id` (which together form the `board_id`) with their corresponding `HammingDistance`.
- Now, we get the first k boards from this map based on the least `HammingDistance`. This will be our kNBs.

Query 1

A) We will find the k nearest board orientations for the given board orientation (we can get the board orientation since we know the move) and then since we know the game id and move id of the fetched k nearest boards , we just check if the given (i,j) is present in the next moves updates(where previously empty field in the table would be 1) in all the k nerest boards orientations of their respective game if they are flipping then the position (i,j) will be present in updates (where previosuly empty field in the table would be 1) else it the coin didnt flip in that game in the next move. we check this condition for all k nearest boards and return the result as maximum vote (i.e, if for majority of the k nearest boards the coin at (i,j) flips in the next move the it will be outputed as flip and vice versa)

B) To fnd the guaranteed number of moves for which the position (i,j) doesnt flip , we can fetch the K nearest board orientations(we will even get game id and move id of that game) and check the number of moves till which the given position (i,j) will not be present in the move updates entries and if present in the updates entries then its previously empty attribute should be 0 (which indicates its previously empty and doesnt come under as flipped move) in the database for that game(which indicates that the position is never flipping) and each k nearest board will return the number of moves for which its following the above rule. After getting the values from all the K boards we can guarantee that the number of moves for which the position (i,j) doesnt flip is minimum of all the values returned by the k nearest boards . and the expected would just be the average of all the values returned by k nearest boards.

C) We fetch the K nearest boards and and check all the positions that are going to be flipped in the next move we just return all the (i,j) in the update entries for which previously_empty attribute is 1.And after the positions from all the k nearest boards we return the union of all these positions as moves that have chance to be flipped in the next turn.

Query 2

A) for each game for the given board we need to traverse through all the moves till the of the game . since each entry in our page block contains the update value for that row we can sequentially calculate all the i's where the player 1 (assuming player 1 plays white) is atleast gaining b. starting from the first player move we find the number of coins that turn from black to white by traversing the rows in the page for that move udpates and find the count of rows where (updated_value=white and previously empty=1) and then subtract the count of the rows where (updated_value=black and previously_empty =1) for the next immedeate move . And then do the same for all other alternate moves(i.e, take difference of that blacks turned to white in that move and numer of whites turned to black in the next immedeate move). Now after finding the gain for all these moves ,return the moves for which the gain is atleast b.

B) We can follow the same procedure as explained in the above part except for a change that we maintain a reocord of the sequences of all the moves where theere is continous gain in b. i.e, we maintain two sets, temp set and final set. After calculating gain for a move i we add the i and i+1 to the set temp if the gain is atleast b else we add the temp set to final if it not empty and make it empty after adding to final set .and at the end after checking for all the moves we once again check if the temp set is empty or not, if it is not then we add it to final. we return final set which represents the set of sequences of moves where the player 1 is continously gaining b.

Query 3

since we have the record of all the games , we can traverse all the game records and find (i,j)'s which suffered maximumnumber of flips .i.e, since our entries in the page contain only updated position values of the board , we traverse through all the entries in page and increment the flip for the specific (i,j)'th position present in each row if its previously_empty attribute is 1(i.e, there is a coin at that position before that flip). and after continuing the procedure for all the games we just return the positions of all position (i,j) which suffered maximum number of flips.

Query 4

For the given board orientation after fetching the k nearest boards we will have the game id and move id of the k boards. we will iterate through all the updates in the game of the each k nearest board from that particular move and return all the (i,j) which were never flipped ,i.e, it will be present in move update entries in the table if and only if the previously_empty is 0 or else it will never be in the update entries.

Open ended

- **Query1** : In this query we need to find the position (i,j) which best stops the other player to gain more score in their next turn(i.e, by turning more whites to blacks , assuming white is for the current user and black is for the other user).This helps the player to get to know about the positions where the opponent could actually score more and he can thereby place his coin in that position before the opponent places his coin there.

  Solution: After fetching the k nearest boards we just go till the next player move and check his moves and calculate the gain he is getting for that particular move (i.e, the number of whites which are flipped to black in that move) by traversing through the update entries in the page ,i.e, we count the number of rows where there is an update from white color to black (i.e, value = white , and previously empty =1) ,and return the gain and the position for all the k boards. Now after getting the results from all the k boards just return the move with maximum gain (for player 2). And now since the player 1 has the idea of where the opponent (player 2) could score more he can stop him by placing his coin in that particular position.

- **Query 2**: In this query we need to find the best immedeate move for the player.

  Solution: After fetching the k nearest we compute the gain of each board for the next 2 moves in their respective games by analyzing the next 2 moves (i.e, gain= number of black flipped to white in the current move - number of whites to black in the next move , assuming black is for player 1 and white is for player 2) .this could be done by iterating theough the updates for the two moves and take the difference of the counts of the rows which have (value =white and previously_empty =1) and (value =black and previosuly empty=1).. We then return all the moves and their gains for all the k nearest boards. After which we choose the move with more gain and return that as the best immedeate move for the player .Note that this is different from query 2 where we are supposed to find for all move i's with gain as b but where as here we are just finding the immedeate best move possible for the player.

Key Takeaway:

The whole idea of how to effieciently organize data in pages and blocks and retrieve data for different queries operated on the database is best explained in the given assignments(1 and 2). Example : we have clustered index for storing the move updates(our page layout uses clustered index mentioned in assignment 1) so that we could retrieve or go to the i'th move in an efficient manner. We have learnt how effective the role of the page design plays for various query executions(eg : since our page design stores only updated position values in each move ,calculating hamming distance for finding similarity is done very efficiently(since we need to now just look for the updated values to update the hamming distance) but where as if we store the whole board , it would be inefficient to compute hamming distance for each move , since we need to again compare all the positions in both the boards)