Indexing moves of Dots and Boxes game

TeamAG:
2019201011
2019201049

Background:

- Problem statement: For a dots and boxes game, indexing of scoring moves based on some rank.

- These moves are then used to identify ‘opportunities’ to make next move or ‘pitfalls’ to not make a certain move.

- Defiinition of game structure:

➢ A board with 10^4 x 10^4 sized board.
➢ A move by a player from a dot (i,j) is defined by either a horizontal line to (i,j+1) or vertical line from (i+1,j). Hence a move is defined by following tuple,
(i,j, horizontal/vertical, player_id).

1. Overview of what your game board storage structure is.

Data to store:

Based on the operations to be performed, following data need to be stored,

1. **Moves made**: A move by a player from a dot (i,j) is defined by either a horizontal line to (i,j+1) or vertical line from (i+1,j). Hence a move is defined by following tuple,
(i,j, horizontal/vertical, player_id).

2. **Player status**: (Player_id,Number of boxes)

3. **Opportunity**:

   Concept**:**

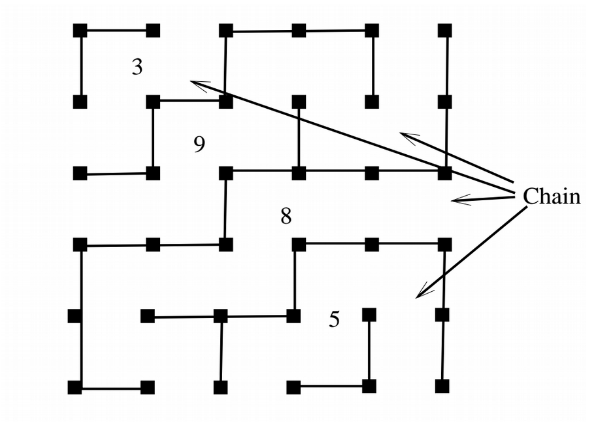
   - We Rank the opportunity and pitfall moves by the length of chain that they yield when making a that move. Their rank is also defined by this length.

   - Chain:A chain is a sequence of boxes with valence 2, where every empty line is part of two adjacent boxes, except for the two outer-most empty lines, which exist on the side of the grid. A chain is of length 1 or more.

   - Length of chain: Number of edges in above defined chain.It might be close chain also.

   - O/P: Every chain is initial status is pitfall (P) for Bot until another user makes single move in it. If user makes a move, then all flags corresponding to that move in chain are set to opportunity(O)

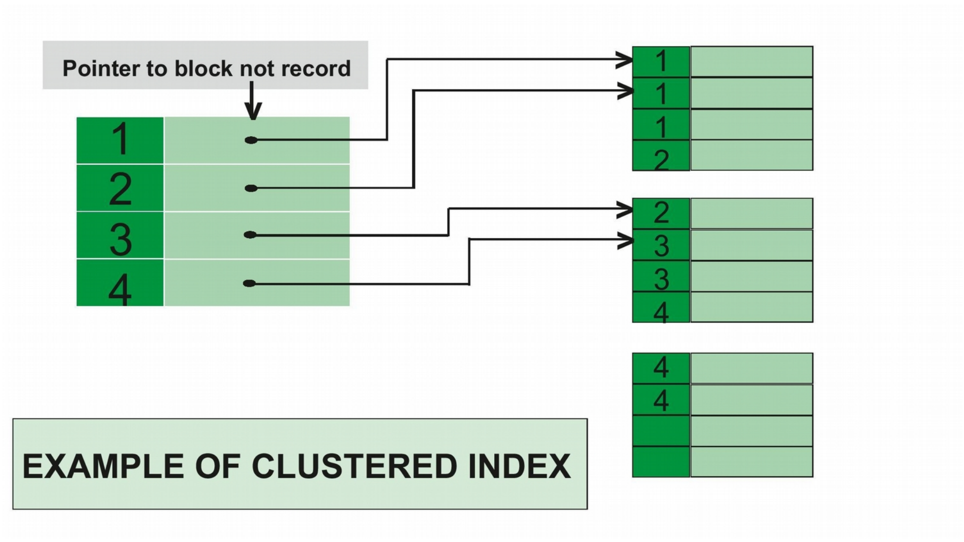
   - i , j , horizontal/vertical: when chain/cycle with valency 2 is formed then there are moves between this two chain from starting to ending we store that moves



<u>Schema of table opportunity:</u>

(length_of_chain, i , j , horizontal/vertical ,o/p)

2. One or more indices for the game board, with clear semantics for LHS and description of pointers on the RHS, with pictorial representation.



Clustered Indexing:

- As from the schema, we create clustered indexing on length of the chain as key.
- As the multiple opportunities/pitfalls of same chain length can occur, this is a suitable indexing technique.
- Each key stores block pointer to the block in which the oportunities/pitfalls of chain length equal to the key value are stored.
- **Semantics of indexing:** The indexing semantics will be as follows
  LHS: It is the key which corresponds to length of chain
  RHS: This stores the pointer to the block having records of possible moves of length equal to key.

- We implement this clustered indexing using B+ tree which is a type of multilevel indexing where each node, depending on internal or leaf, stores key and the block pointers to other nodes or to records.
- Hence we use this ‘length’ as the key for indexing the opportunistic and pitfall moves.
- We propose the following indexing structure to store and rank the opportunistic and pitfall moves

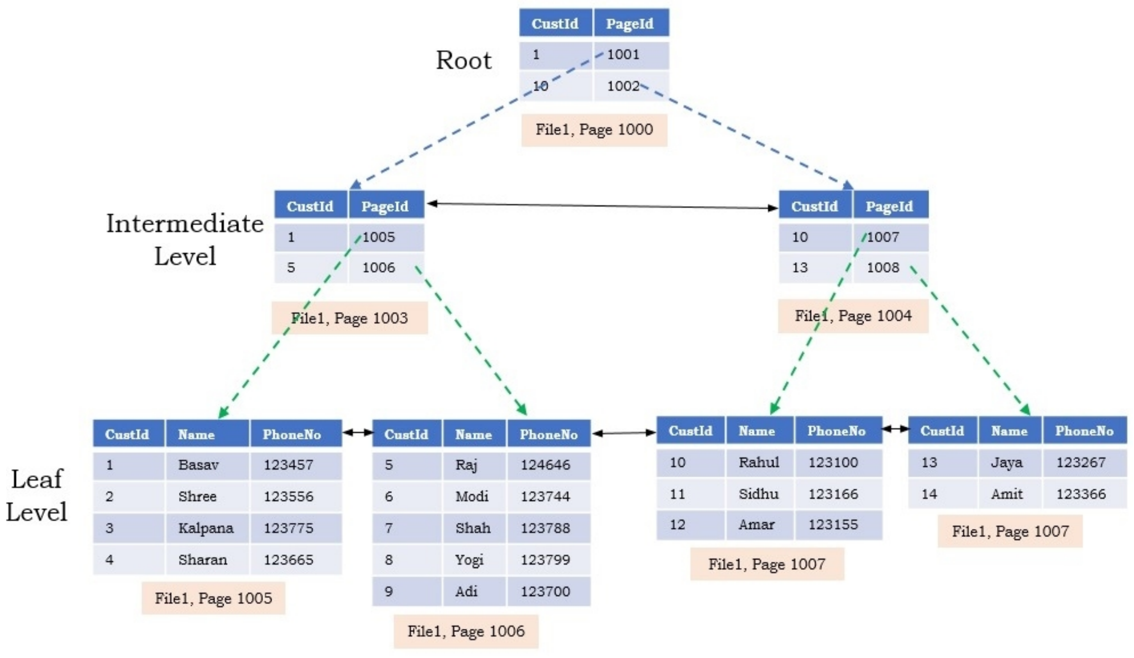
➢ <u>A B+tree with the internal and leaf nodes structure</u>:

i. **Internal node:** An internal **node stores key(Chain Length) and block pointers** to the nodes of next level.

ii. **Leaf node:** The leaf nodes store keys and block pointers to the actual moves of length equal to key.

iii. Since the keys in B+-tree are sorted in increasing order, the opportunity moves with larger lengths are at the right end of the tree and the pitfall moves which will let the least score to opponent with least key value are stored at the left end of the tree.

B+ Tree Structure of a Clustered Index



Root (File1, Page 1000):

CustId	PageId
1	1001
10	1002

Intermediate Level (File1, Page 1003):

CustId	PageId
1	1005
5	1006

Intermediate Level (File1, Page 1004):

CustId	PageId
10	1007
13	1008

Leaf Level (File1, Page 1005):

CustId	Name	PhoneNo
1	Basav	123457
2	Shree	123556
3	Kalpana	123775
4	Sharan	123665

Leaf Level (File1, Page 1006):

CustId	Name	PhoneNo
5	Raj	124646
6	Modi	123744
7	Shah	123788
8	Yogi	123799
9	Adi	123700

Leaf Level (File1, Page 1007):

CustId	Name	PhoneNo
10	Rahul	123100
11	Sidhu	123166
12	Amar	123155

Leaf Level (File1, Page 1007):

CustId	Name	PhoneNo
13	Jaya	123267
14	Amit	123366

Storage capacity of B+tree node:

- Suppose a block is of size B KB, the record is of R KB, block pointer is of P KB, and key is of size K KB. Then,

  Number of index entries per leaf page = floor (B / (K + P))

Move based update of index:

- After each move is made, a find_chain() utility function scans for possible chain formation at the place where the move was made. It returns the start position of the chain i.e. (i,j,verticle/horizontal) and the possible length that it would yield.

- This function also handles the case where an existing chain was broken or updated into two by some inefficient move.

- Thus, the function is assumed to return multiple results of possible chain with its start position and chain length.

- Once these chains are obtained, they are inserted in B+ tree depending on the chain's length. A new key is inserted or the record is updated if the key is already present.

- Also the already present chain is deleted from the tree
- Two type of indexing is implemented one is on all enries where o/p is "O" (opportunity) and another on entries where all are entries are set as "P"(Pitfall)

3. A list of queries where the indices are useful. Give explanation on how the index access is useful, and how efficient it is.

- Print all the highest opportunistic moves so that we can select any moves from table

```
SELECT i,j, horizontal/vertical, MAX(length_of_chain) AS op
FROM opportunity WHERE op < (SELECT MAX(length_of_chain)
FROM opportunity) and O/p="O"
```

Explanation: In this query with the help of B+ tree it traverse through right most node of tree where we can get all moves with Max chain length it reduces number of block acess

- Print all pitfalls with minimum loss

```
SELECT i,j, horizontal/vertical, MIN(length_of_chain) AS op  FROM opportunity
WHERE op < (SELECT MIN(length_of_chain)
FROM opportunity) and o/p ="P"
```

Explanation: Again As explained above it will goes to directly leftmost node and reurns the solution with least chain length but it uses another indexing we use on o/p "P"

- Print number of length entries which are used as opportunity

1. `SELECT length_of_chain, Count(*)FROM Student Where O/p="O"GROUP BY length_of_chain;`

Explanation: Here also as we defined seperate index for " O" flag then average block acess reducess to logN+1

4. After each move an update index function that updates or maintains the current index for immediate use. A notion of real-time index.

- After each move, once the B+ tree is updated, the current index for immediate use would be the one with maximum chain length and hence the maximum key in the tree.
- The B+ tree has its node with maximum value as its right-most node. So we read this node and the corresponding record from the block and store this record in the current index for immediate use.

Pseudo-Code for maintaining current index:

- After each move if new move added with new chain length we insert unique key (length of chain) in tree.
- There is probability that previous old chain length increases hence when we got move, we check that if duplicate move is present or not if yes then we update old chain length.
- Then find the right-most node in the tree. (We Store the value in B+ Binary tree such that rightmost node contains longest chain length)
- As each block in B+ tree is also in sorted order we retrieve last pointer from block.

```
Update_current_index (Node *tree, current_index):
        chain_move = find_chain(current_chain)
        if chain_move is not empty:
            node = tree->find(chain_move.key)
            if node is empty:
                    tree->insert(chain_move)
            else:
                    tree->update(chain_move)

            while(tree->right)
                    tree = tree->right

         blk_pointer = tree->block
         page = read_block(blk_pointer)
         current_index = get_record(page)
```

- When we return record pointer, we delete that moves from our table and check if any other moves is present with same chain lenth or not if no then we delete same key from B+ tree

Monitor function()

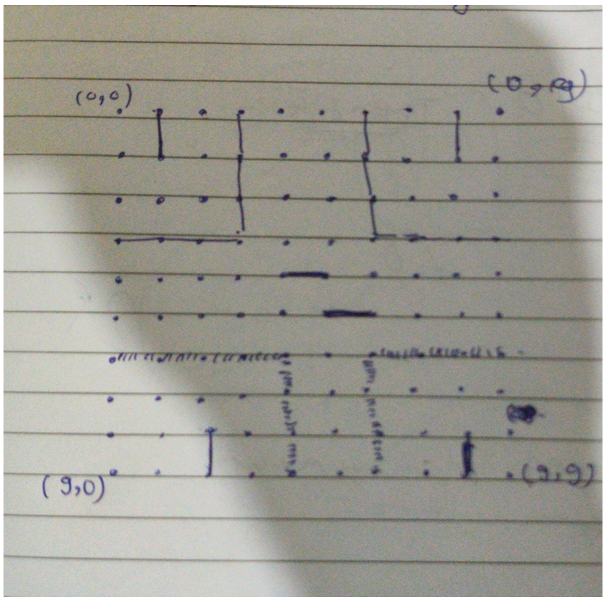
We need to constantly monitor each moves of both player after each move, we have to check that if

1. Old chain length has increased, then update each record
2. If boxes are scored, then reduce the chain length for each move. If it reaches to zero, then delete than move
3. Also, initially any first move between chain is set as pitfall (P) so another player makes any move with-in chain then set all moves within chain as O(opportunity).

4. When searching for opportunity (O) at the right-most node of the tree to update current_index, we should check for the record with highest key and its flag set as O, since a move with highest key can also be set as a pitfall (P).

5. An example of 1,2,3,4 on 10x10 game board.

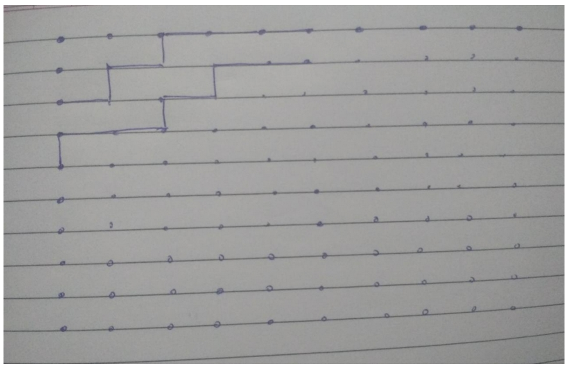
- **No Indexing is used**



- In starting phase when there is no chain of valency 2 as shown in image there is no indexing
- Each move has equal opportunity.
- Hence gradually as chains are created they are indexed in the tree as mentioned above.

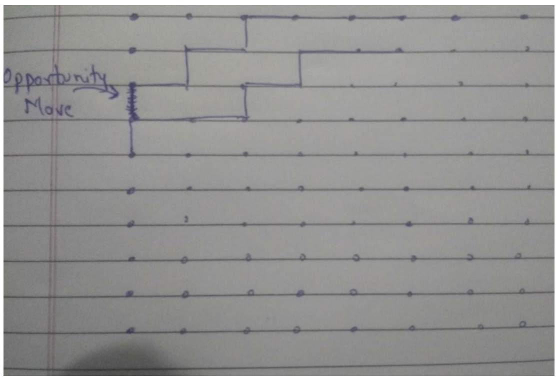
- **We started using Indexing**

As shown in above image this is chain of length 7 with valency 2 this chain length is stored in table as pitfall and entry is made in pitfall b+ tree   with key value 7
Now we store coordinate of each possible move for example
(7,0,2,vertical,p)
(7,1,2,verticle,p)
(7,1,2,horizontal,p)...

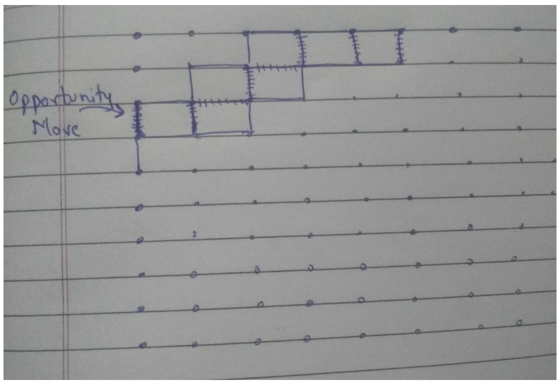


When the first move made by opponent in chain then all entries are set as O and we delete this entries from this tree and add to Opportunistic B+ tree indexing and set the flag as “O”

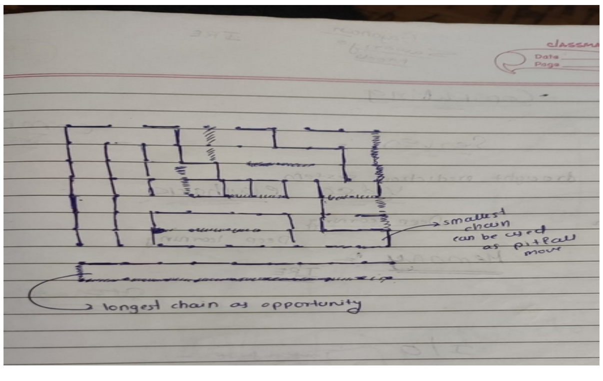
Hence after this whenever BOT asked for opportunistic move it will return a move with chain length 7
(7,0,2,vertical,o)
(7,1,2,verticle,o)
(7,1,2,horizontal,0)...



In this image we can see that opponent completed the chain length. After the completion his move we delete the entry from our table. Similarly we delete the key from B+ tree.

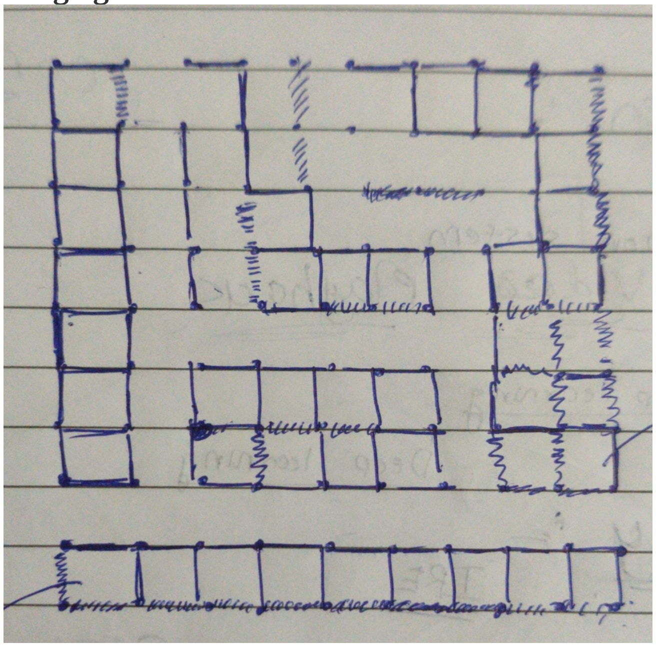


Indexing At peak:


smallest chain can be used as pitfall move
longest chain as opportunity

- In such situation indexing is at peak
- All open chain stored as pitfall
- In such stiuation it is necessary to give opportunity to opponent, so we use smallest pitfall from our B+ tree
- But if opponent makes a move, we use Max opportunity (Longest chain)
- During this situation monitor function continuously update chain length and check if any Opponent move makes pitfall to opportunity and if yes we add new index in B+ tree
- From as more and more blocks form indexing start reducing

Indexing again reduces:



- Here as we can see that longest chain changes
- Now there are very less entry in both B+ tree (opportunity and pitfall B+ tree)
- As the chains are completed each time update_current_next points to next opportunity
- Monitor function continuously update length of chains as moves of players completed