Data Systems

Group Assignment - 2
Querying For Patterns

Team 48

Name	**Roll No**	**Email**
Anurag Gupta	2020101019	anurag.g@students.iiit.ac.in
Madhusree Bera	2022202007	madhusree.bera@students.iiit.ac.in
Sanya Gandhi	2022201066	sanya.gandhi@students.iiit.ac.in

Assumptions:

1. The software identifies which patterns yield how many number of boxes in how many number of turns.
2. One box can be part of one Pattern only
3. The game has been played almost halfway through so that almost all the boxes are a part of some pattern.

Design Idea:

Pattern Definition:

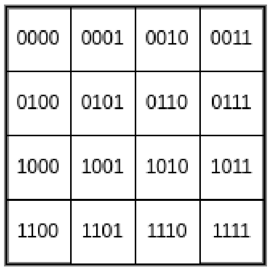
❖ Every Pattern is a collection of boxes, the edges that are active on them and the number of cells taken in that turn.

❖ The data records are shown in the figure below

 ➢ **Box ID Array**: Box IDs (starting from 0 and stored as binary) sorted in ascending order

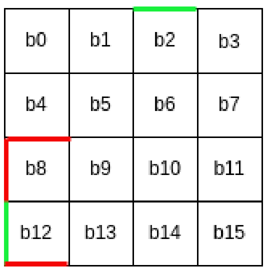
- ➢ **Active edges array** : Bitmap index (Up, Right, Down, Left) of each box according to the order of Box IDs
- ➢ **No. of cells taken** : Maximum No. of cells that can be taken in a particular turn and must be taken

[Box ID array]	URDL (Active Edges Array in Binary)	No. of cells taken	[Box ID array]	URDL (Active Edges Array in Binary)	No. of cells taken	...

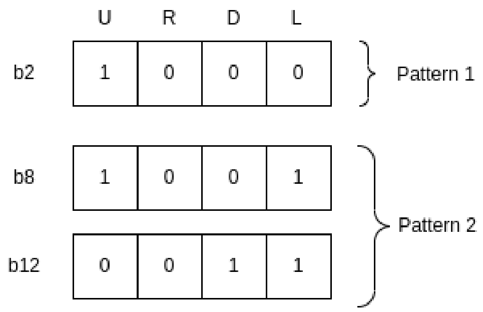


0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111

Storing of box indices in binary
(for 10k x 10k, 2B required for box index)



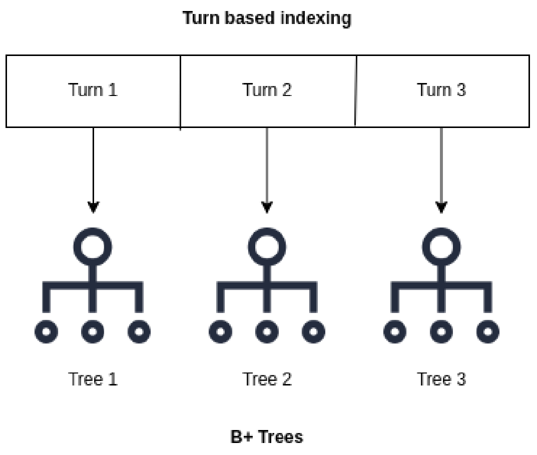
State of Board



Representing boxes

Indexing:

- ❖ **Level 1 indexing** : We propose index based on Turn 1, Turn 2, Turn 3 patterns
  - ➢ **Turn 1 patterns**: Patterns that will give atleast 1 box in turn 1
  - ➢ **Turn 2 patterns**: Patterns that can give at least one box in turn 2, but zero boxes in turn 1.
  - ➢ **Turn 3 patterns**: Patterns that can give at least one box in turn 3, but zero boxes in turn 1 and turn 2.



- **Level 2 indexing**: B+ Tree will be used to index the patterns based on the search key
    - **Search Key**: the box ID of the first box (lowest index) in the pattern. This search key will be unique since one box will belong to one pattern only
    - **Search, Insertion and Deletion in B+ Tree**: Algorithm 17.2, 17.3 and algorithm followed in figure 17.13 from Elmasri & Navthe book
    - **Time complexity:** logarithmic time complexity as it is a B+ Tree

Querying:

- The query asks for the patterns that might yield a box after **K** turns which is returned by the cluster pointer given in the indexing page.
- First level 1 indexing returns the B+ Tree containing the Pattern search keys.
- Traversing to reach the least number in the B+ tree [O($\log_q$ (total search keys))] and then accessing the record pointers sequentially. [O (total records in the turn)] (since the leaf nodes in B+ Tree links to the next leaf node for the special purpose of querying sequentially)

Update:

- When an action is performed, it can be seen as an update in the edges corresponding to only **two boxes.** The patterns that involve those boxes are updated.
- The updated patterns are checked by the software for the possibility of cluster shifting or if any new patterns have come up due to the action.
- There can be 3 cases:
    - **Case 1: Only the edges of the pattern are modified**
        - Step 1: Search for the pattern in the B+ tree i.e find the leaf node (worst case read h internal nodes where h is height of tree) and then perform binary search on the keys in the leaf node. [O($\log_q$ (total search keys) * $\log_2$ (total keys in the leaf))]
        - Step 2: Go to the record pointer, update the edges and number of cells taken by over-writing.
        - Step 3: If the pattern search key is modified, then delete and insert the search key accordingly. (logarithmic time complexity)
    - **Case 2: Pattern gets moved from turn i to turn j**
        - Step 1: Search for the pattern in the B+ tree of turn i and store the record pointer Pr.
        - Step 2: Go to the record and update the values if needed,
        - Step 3: Delete the pattern from the turn i B+ tree. [O ($\log_q$ (total search keys))]
        - Step 4: Insert the pattern and its record pointer in the turn j B+ tree. [O ($\log_q$ (total search keys))]
    - **Case 3: Pattern is consumed**
        - Step 1: Delete the pattern key and record pointer from the corresponding B+ tree. [O ($\log_q$ (total search keys))]
        - Step 2: Mark the record pointer as free space so that it can be used for storing other records by overwriting
        - Now, the record of the pattern will not be pointed by any key in any B+ Tree.

Space Complexity:

Pattern storage:

Total boxes: 1000 * 1000 = 10^6 approximately

Box ID: ceil ($\log_2$ (10^6)) = 20 bits = 3 B approx for one single box

=> total space for box IDs = 3MB approx.

Active edges per box = 4 bits = 0.5 B

Active edges for all boxes = 0.5 * 10^6 B = 0.5MB approx.

No. of cells taken = Integers (maximum value as same as number of boxes) = 3B, total = 3MB

Total space for records of all patterns = 3MB + 0.5MB + 3MB = 6.5MB

Assuming one block = 4KB, approximate number of blocks required : ceil(6.5 * 10^3/ 4) = 1625 blocks approximately

One record : Variable length depending on the number of boxes in the pattern: say ‘b’

One record length = b * (3B + 0.5B) + 3B = (3.5b + 3) Bytes

Level 1 Indexing:

Turn 0 Pointer (B+ Tree root)	Turn 1 Pointer (B+ Tree root)	Turn 2 Pointer (B+ Tree root)

Pointer: 4B or 8B depending on architecture (32-bit or 64-bit).

Level 2 Indexing:

Internal Node: <P1, K1, P2, K2, ... K_{q-1}, P_q >

Let p be the maximum number of tree pointers of internal node

p * (size of tree pointer) + (p-1) * (size of key) <= Block size

= p * 4B + (p-1) * 3B <= 4KB

= 4p + 3p - 3 <= 4000

So p = 571 approximately

Internal node can have maximum 571 tree pointers, maximum 570 keys, minimum [ceil (571/2)] 286 keys

Leaf Node: < <K1, Pr1>, <K2, Pr2>, ... <K_{pleaf} , Pr_{pleaf}>, P >

Let pleaf be the maximum number of keys in the leaf node

pleaf * (size of key + size of record pointer) + size of tree pointer <= block size

= pleaf * (3B + 8B) + 4B <= 4000 B

= 11pleaf + 4 <= 4000

So pleaf = 363 approx

Leaf nodes can have maximum 363 keys and record pointers and minimum 182 key and record pointers.

Assuming tree to be 69% full,

0.69 * p = 394 approx

0.69 * pleaf = 250 approx

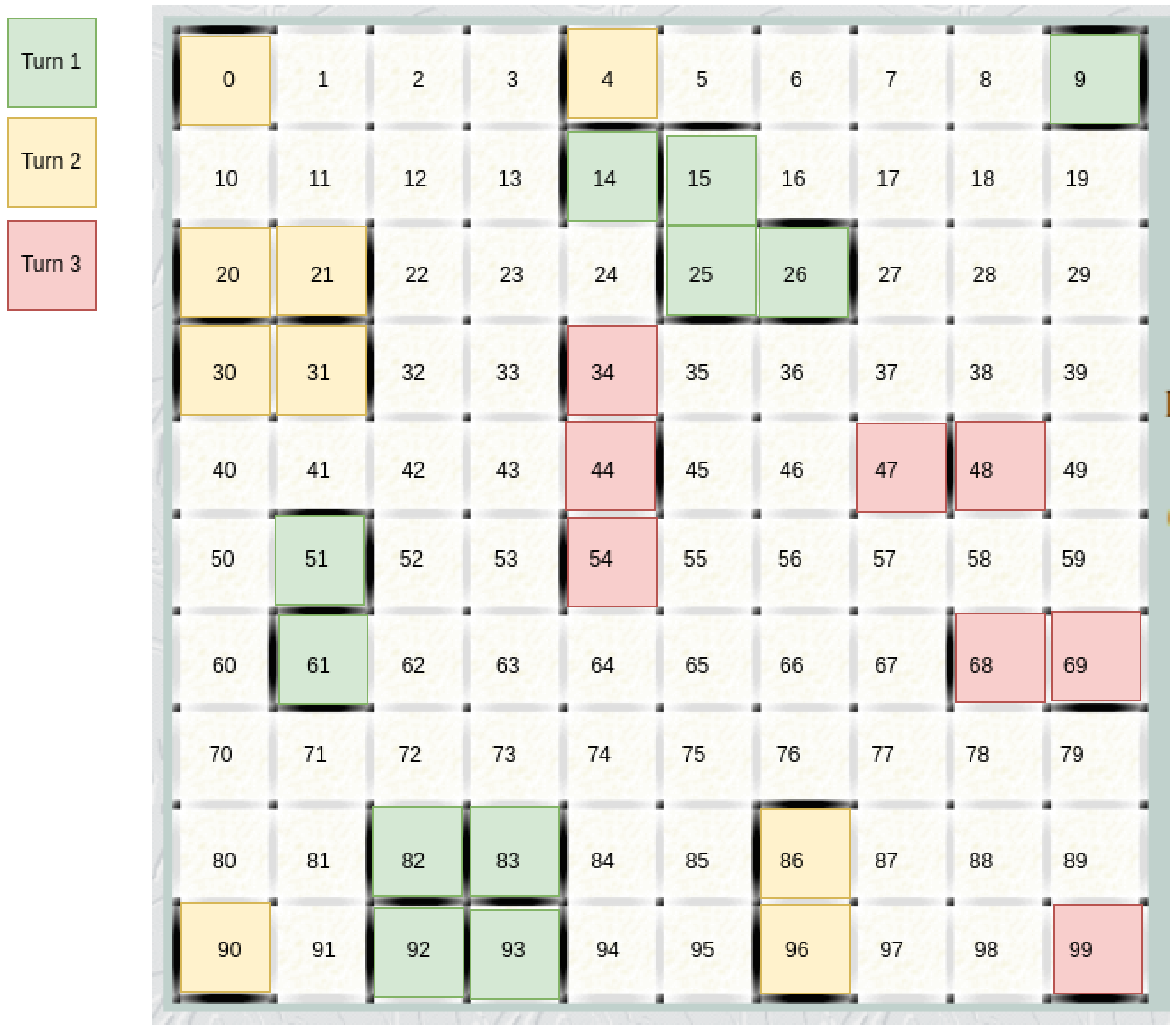
Root: 1 node, 393 keys, 394 pointers

Level 1: 394 nodes, 1,54,842 keys, 155236 pointers

Leaf level: 155236 nodes, 3,88,09,000 data records

We can see that 2 level B+ tree can store a huge amount of data records

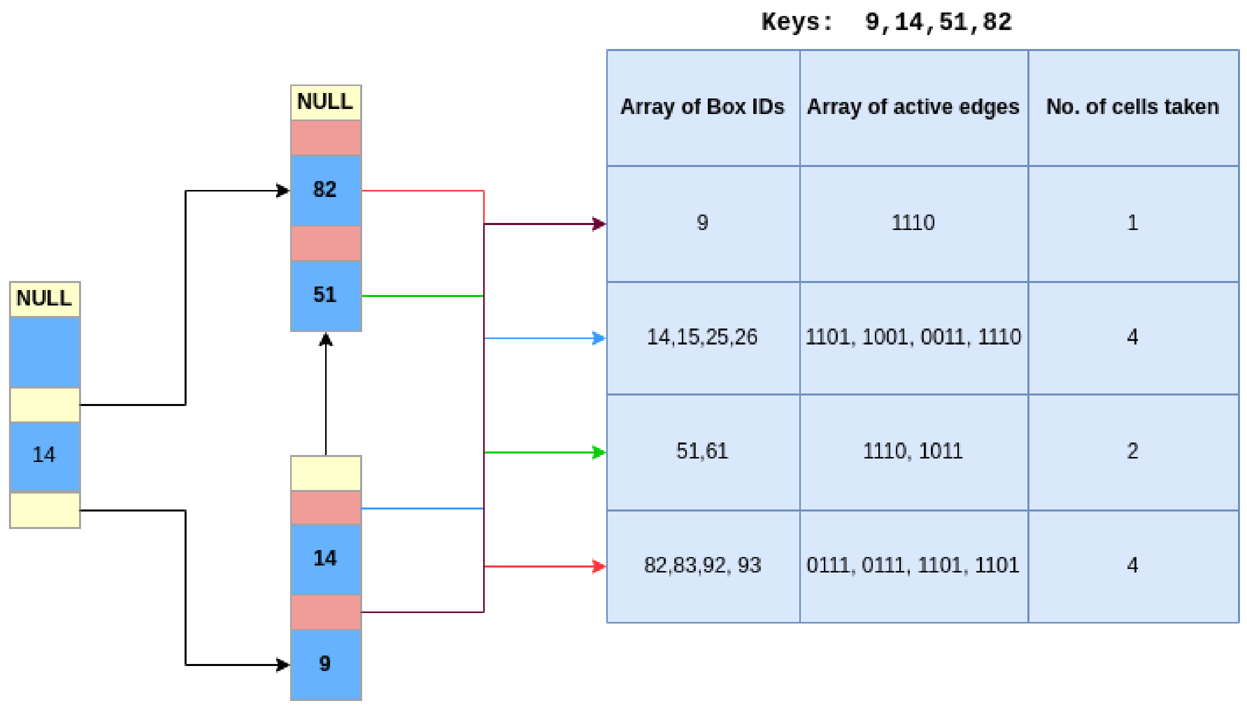
Example Game State



In this example, we will be assuming that order of B+ tree p=3 and pleaf=2

Insertion:

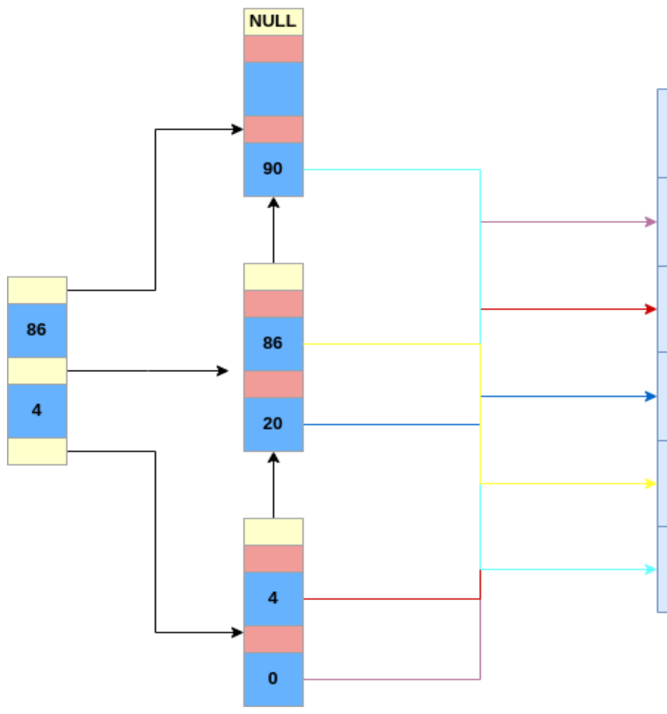
Inserting turn 1, turn 2 and turn 3 patterns into the B+ tree.



Array of Box IDs	Array of active edges	No. of cells taken
9	1110	1
14,15,25,26	1101, 1001, 0011, 1110	4
51,61	1110, 1011	2
82,83,92, 93	0111, 0111, 1101, 1101	4

Turn 1 B+ Tree

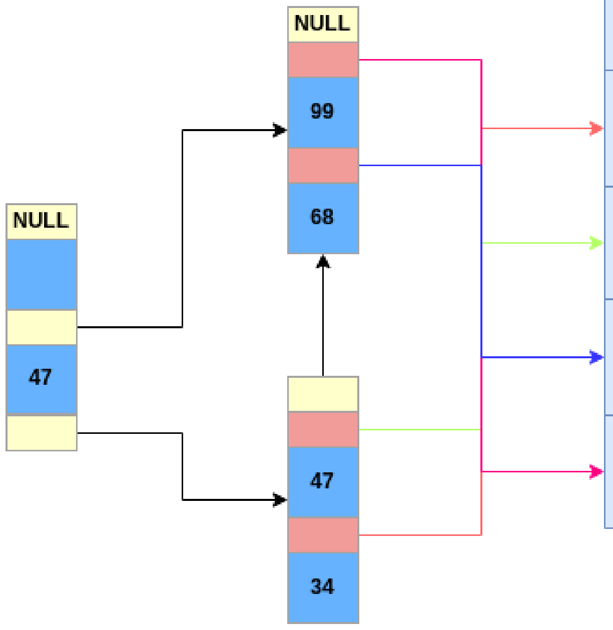
Keys: 0,4,20,90,86



Array of Box IDs	Array of active edges	No. of cells taken
0	1001	1
4	0011	1
20,21,30,31	0011,0110,1001,1100	2
86,96	1001,0011	2
90	0011	1

Turn 2 B+ Tree

Keys: 34,47,68,99



Array of Box IDs	Array of active edges	No. of cells taken
34,44,54	0001, 0100, 0001	1
47,48	0100, 0001	1
68,69	0001, 0010	1
99	0010	1

Turn 3 B+ Tree

Updates from first submission:

Case 1 Update:

Eg: Pattern [20,21,30,31] => this is a turn 2 pattern. Let us assume that after 2 moves, boxes 20 and 21 are taken. So the pattern will be updated to => { [30, 31], [1001, 1100] , 2} which is also turn 2 pattern. So the new pattern search key is 30 instead of 20. We need to first update the pattern record, delete the key 20, and insert the key 30 and its record pointer in the same turn 2 B+ tree.

Case 2 Update:

Eg: Considering pattern [34,44,54], let us assume after 3 moves, box 44 is taken. Now we can update the records into 2 patterns of turn 2: {[34], [0011], 1} and {[54], [1001], 1}.

So we must delete the pattern key 34 from turn 3 B+ tree and insert pattern keys 34 and 54 and their corresponding records in turn 2 B+ tree

Case 3 Update:

Eg: Considering pattern [82,83,92,93] , all the 4 boxes will be taken in turn 1 and hence this pattern is consumed. So, it must be removed from the turn 1 B+ tree. The record ptr is marked as free space so that space can be now utilised for some other storage.

Limitations:

❖ For queries such as fetch the pattern that will give the maximum number of boxes in turn i, then all the patterns in that turn have to be sequentially scanned.
❖ Extra space required for B+ tree indexing