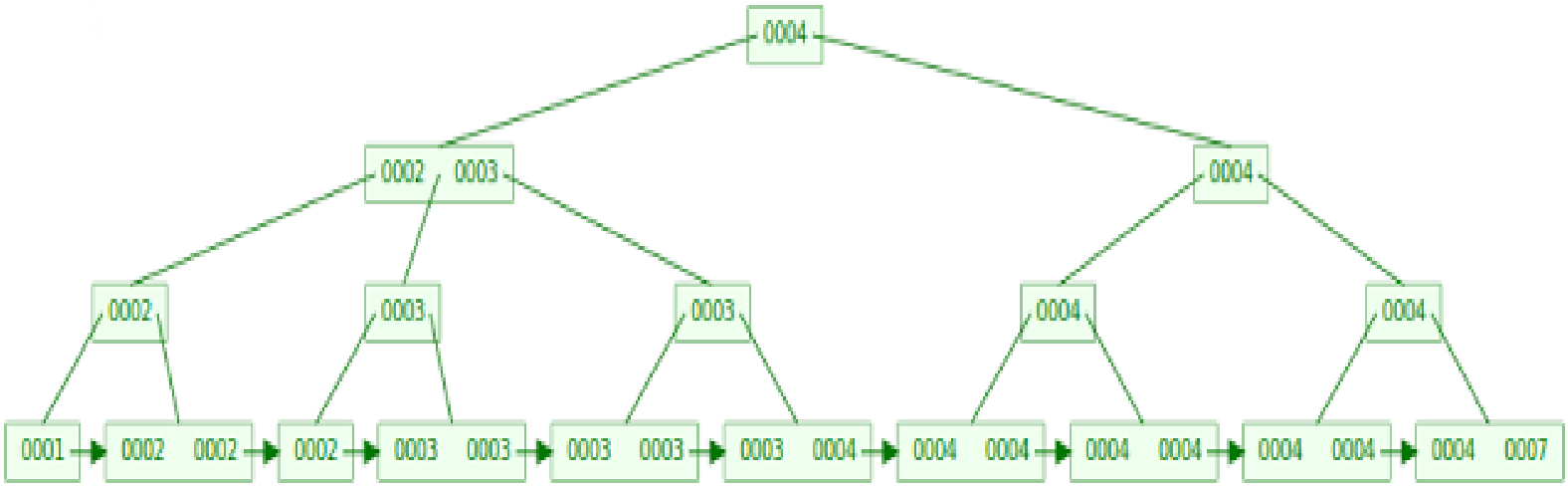
Design Documentation

In this project, we plan to implement a secondary index for repeating attributes(i.e non-unique) so that search and other queries operate faster. Underneath, AM layer implements the index using a B+ tree. We process the B+ tree for the index corresponding to the non-unique attribute and create our own B+ tree (the secondary index) which is used for all queries.

For ease of explanation, We will describe our design using an example.

A	B
1	1
3	2
5	2
7	2
15	3
2	3
4	3
6	3
8	3
10	4
12	4
17	4
18	4
16	4
14	4
13	4
11	4
9	7

In the above example, B is the non-unique attribute that we are improving upon and A is the primary key for the above relation. The underneath B+ tree that is created to index attribute B by the AM layer looks like,



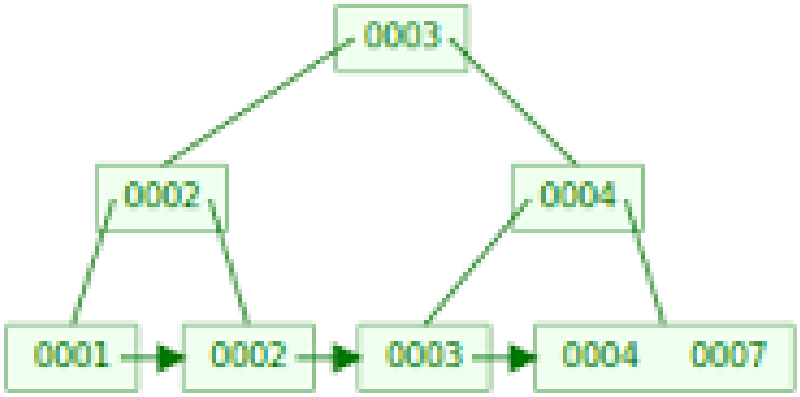
The leaf nodes contain the actual values for the attribute B and the internal nodes contain the keys by which the B+ tree is created.

We have two approaches in creating the secondary index:

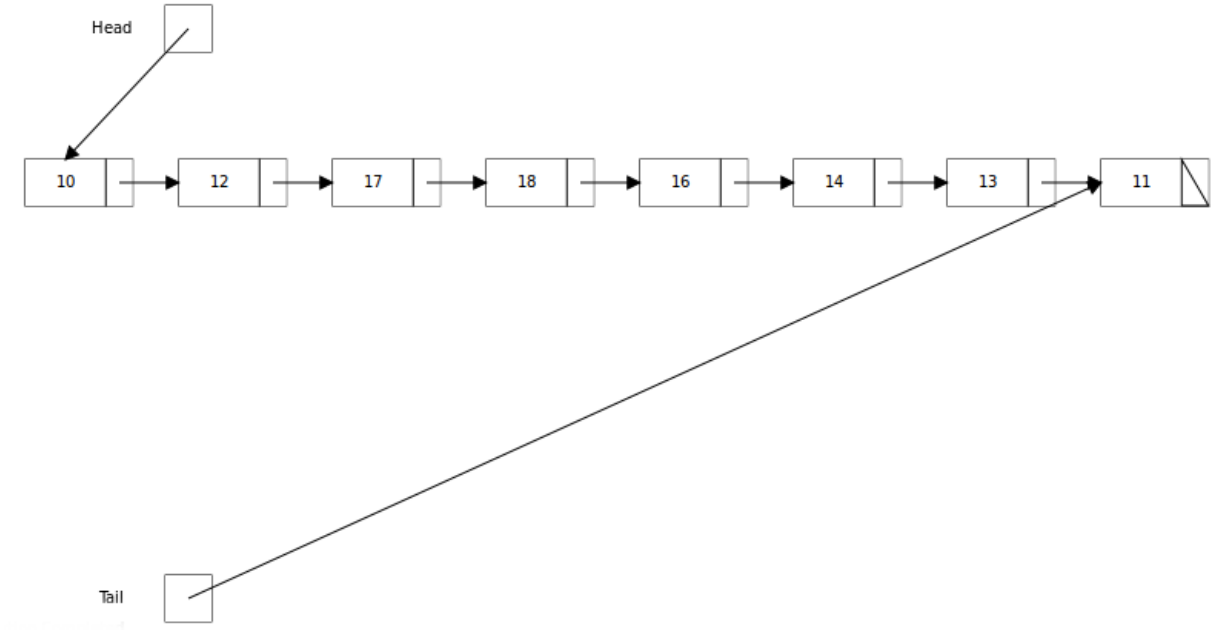
1. **Using a bucket of pointers:**

In this approach, instead of having a separate node for each value of the attribute B, we have a separate node for each unique value of the attribute B. Each node of the B+ tree also contains a pointer to a linked list containing all the records(identified by record IDs/primary key) in the relation which have the same B value as that present in the node.

The B+ tree in this approach will be quite small and saves a lot of space although the time it takes to search will be almost same.



Here, each node has a payload that points to a linked list containing all the instances of relation having the same B value identified by the primary key. The linked list corresponding to the B value of 4 looks like:



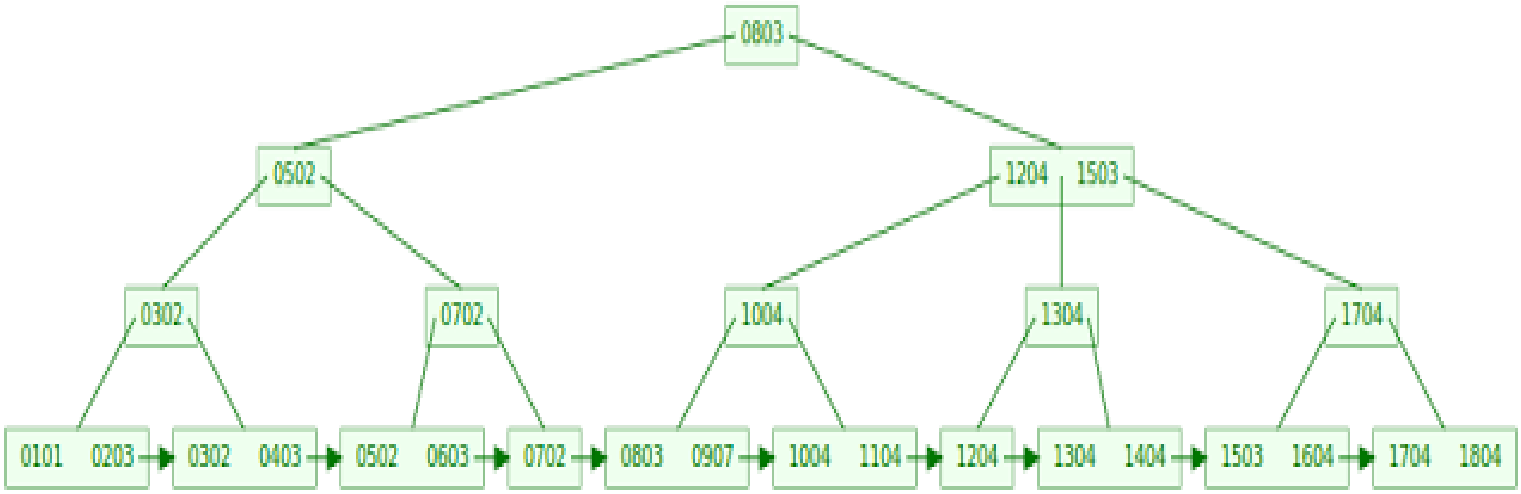
The payload associated with the B+ tree node containing 4 can be the head pointer to the above list.

2. **Appending the primary key value:**

In this approach, we append the value of the corresponding primary key(i.e. A in this case) with the value of B so as to make it unique. Thus the secondary index that we create corresponds to an attribute that is unique for all the tuples in the relation.

The B+ tree created in this case will be equal in size with the original one created by the AM layer but it will be more faster and efficient in search.

For our example, we have appended the values as newB = B + A*100. The new B+ tree looks like



We plan to design in the above way so that different records having the same B value have unique values in the B+ tree.

For the first approach, we need to design our own B+ tree algorithms to implement the linked list-B+tree combination. However, for the second approach we can directly use the AM layer to create the required B+ tree.

For the above description, we assume that the B+ tree uses two keys for ease of depiction although this may not be the case.

Citations: All the visualizations in this report are made using the visualization engines from http://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Yeddu Vamsidhar 110050051
M Rajeev Kumar 110050052
Anirudh Vemula 110050055
K Kartik Pranav 110050062