Variants of B+ Trees with Multiway Splitting

To accommodate insertion of a new entry, sometimes we have to go through splitting of nodes in the B+ tree. Adjustment of tree structure is a costly operation. We want to reduce the frequency of such adjustments. Whenever we have to adjust the tree structure, our goal is to create the extra capacity for new keys so that possibility of next split is delayed.

Proposed mechanism: Introduce multiway splits only in the data level nodes

We need to insert a new data entry "Y" into a data node "D0". The data node is full. We have to split the node. The capacity of data node is "2m" keys. Node the node is supposed to accommodate "2m+1" keys. In your existing code, you are splitting this data node into two parts "m+1" and "m". Instead, we will split the data node into three parts, and adjust the indexing structure as follows.

Step 1: Split the D0 into three parts: D0, D1, and D2. Also, D0 D1 D2 represents the ascending order of entries. In other words: D0 retains the smallest values, D2 gets the largest values, and intermediate values go to D1. Depending on whether "2m+1" is divisible by 3, there will be three cases:

Remainder 0: All three nodes have equal number of values

Remainder 1: D0 has one extra entry as compared D1 and D2

Remainder 2: D1 and D2 each have one extra entry as compared to D2

Step 2: Adjust the indexing structure to reflect this addition of new data nodes D1 and D2. Separator for D0 and D1 will be (LargestEntry(D0) + 1). Separator for D1 and D2 will be (LargestEntry(D1) + 1). There are three possible cases

Case 1: Parent of D0 has sufficient space to accommodate two new separators. In this case, simply add two new entries to the parent.

Case 2: Parent of D0 has sufficient space to accommodate only one new separator. In this case, first insert the smaller of the two separators. Then insert the other new separator and adjust the index structure.

Case 3: Parent of D0 is already full. In this case, first insert smaller of the two entries and adjust the index structure. Then insert the other new separator.

Note that there is no change in splitting criteria for the index nodes. We have introduced changes only in the splitting method for data nodes.

As a follow up improvement can you also introduce multiway splits in the index nodes as well? This improvement is not required for this lab submission.

How to submit your code:

Submit single file. <roll number>_bplustreevar.cpp

How your code should compile:

g++ <roll number>_bplustreevar.cpp

How your code should run:

./a.out <input.txt >output.txt

Consider the given input file.

Lines 1 to 3 ask you to create a new empty B+ tree with capacity of 3 keys per index node and capacity of 4 keys per data node.

Lines 4 to 11 ask you to insert four keys: 1000, 2000, 3000, and 4000. All these keys should be accommodated in a single data node.

Line 12 asks you to display the structure of the B+ tree. It should show following structure

Data Node 0: Parent: NULL Contents: 1000 2000 3000 4000

Lines 13 and 14 ask you to insert entry 5000 in the tree. Current data node is full. We will create two new data nodes and arrange the data as follows:

Data Node 0: 1000 2000

Data Node 1: 3000 4000

Data Node 2: 5000

As the D0 did not have any parent, this is case 3 of step 2. We have insert separators 2001 and 4001. First we will insert 2001. This is achieved by creating a new index node and inserting 2001. Now, we can also accommodate 4001 in the same node. The children pointers should be adjusted accordingly.

Line 15 asks you to display the structure of the B+ tree. It should show following structure

Index node 0: Parent: NULL Sequence of keys and children: Data Node 0, 2001, Data Node 1, 4001, Data Node 2

Data Node 0: Parent: Index node 0 Contents: 1000 2000

Data Node 1: Parent: Index node 0 Contents: 3000 4000

Data Node 2: Parent: Index node 0 Contents: 5000

Lines 16 through 25 ask you to accommodate entries 6000, 7000, 8000, 3500, and 3900. There is enough space in corresponding data nodes to accommodate these entries.

Lines 26 and 27 ask you to accommodate entry 3600. It should be accommodated in data node D1. It is full. We go for three way split as follows

D1: 3000 3500

D3: 3600 3900

D4: 4000

We have insert two new separators 3501 and 3901. The parent node has space only for one. This is case 2 of step 2. We accommodate 3501. To insert 3901, we go through usual insertion method of B tree with splitting of index nodes.

Line 28 asks you to display the structure of the B+ tree. It should show following structure

Index node 2: Parent: NULL Sequence of keys and children: Index Node 0, 3901, Index Node 1

Index node 0: Parent: Index Node 2 Sequence of keys and children: Data Node 0, 2001, Data Node 1, 3501, Data Node 3

Index node 1: Parent: Index Node 2 Sequence of keys and children: Data Node 4, 4001, Data Node 2

Data Node 0: Parent: Index node 0 Contents: 1000 2000

Data Node 1: Parent: Index node 0 Contents: 3000 3500

Data Node 3: Parent: Index node 0 Contents: 3600 3900

Data Node 4: Parent: Index Node 1 Contents: 4000

Data Node 2: Parent: Index Node 1 Contents: 5000 6000 7000 8000

Lines 29 and 30 ask you to insert key 9000. Corresponding Data Node 2, is full. We split it in three parts as follows

D2: 5000 6000

D5: 7000 8000

D6: 9000

Two separators to send to parent node are 6001 and 8001. The parent Index Node 1 has available space for both of them. This is case 1 of step 2.

Line 31 asks you to display the structure of the B+ tree. It should show following structure

Index node 2: Parent: NULL Sequence of keys and children: Index Node 0, 3901, Index Node 1

Index node 0: Parent: Index Node 2 Sequence of keys and children: Data Node 0, 2001, Data Node 1, 3501, Data Node 3

Index node 1: Parent: Index Node 2 Sequence of keys and children: Data Node 4, 4001, Data Node 2, 6001, Data Node 5, 8001, Data Node 6

Data Node 0: Parent: Index node 0 Contents: 1000 2000

Data Node 1: Parent: Index node 0 Contents: 3000 3500

Data Node 3: Parent: Index node 0 Contents: 3600 3900

Data Node 4: Parent: Index Node 1 Contents: 4000

Data Node 2: Parent: Index Node 1 Contents: 5000 6000

Data Node 5: Parent: Index Node 1 Contents: 7000 8000

Data Node 6: Parent: Index Node 1 Contents: 9000

Line 32 asks you to exit.

Note that while printing the structure of the tree, order of the nodes in the print order is not important. You can have your own scheme of assigning ids to the nodes.