Advanced Databases 2017 **Dr. Pierpaolo Dondio**

Lab. 3: B-TREE and Bitmap Indexes

Tutorial

Review the B-Tree Index. Look at the tutorial on webcourses (folder Lab 2017) and this video on b-tree insertion: http://www.youtube.com/watch?v=bhKixY-cZHE

Exercise 1. B-Tree

Insert the following numbers into a 2-3 B-tree (2 places for data for each node and 3 pointers, as seen in class):

2, 6, 5, 7, 11, 3, 12, 15, 13, 16, 4, 1, 8, 14, 17, 18, 20

Exercise 2. Simple Tree

Insert the same list of numbers into a simple ordered tree with the same structure as a 2-3 B-tree (2 data + 3 pointers for each node) but with no rules to rebalance the tree.

Compare the resulting tree with the b-tree of exercise 1. What is the main difference? What is the main consequence?

Exercise 3. Bitmap Index

For this exercise you need to perform a short research. Using printed or web sources, you need to research about Oracle Bitmap Index.

You need to quickly describe

- What is a bitmap index?
- How does it work?
- What are the advantages and disadvantages?

And answer the following question:

- How big (in Mbytes) is a Bitmap index for a field "Month" (12 values) of a table of 5 million records?

Exercise 4.

Let us consider again the two trees of exercises 1 and 2. We call the tree of exercise 1 *"b-tree*" and the tree of exercise 2 "*simple tree*".

The database is receiving queries. Each query is a question of the kind:

"Is number x in the database?"

Where x is a number from 1 to 20. The tree is used to check if the number specified is in the index. Each query requires to visit part of the tree in order to be answered. For instance, if the query is looking for a number that is in the root node, only the root node will be visited.

If the number is in a node that is a child of a child of the root node, 3 nodes will be visited (=we have to go down three levels to find the number) and so on. A query for a number that is not in the index will require to visit all the levels (one node each level) until a terminal node is reached. The highest is the number of nodes visited, the less efficient is the query.

If we assume that each number requested has the same likelihood (=each number from 1 to 20 has the same 5% chance to be requested), can you estimate the gain in performance of the *B-tree* compared with the *simple tree*?

You need to find:

- the average number of nodes that needs to be visited for a b-tree to check if a number is in the index or not.
- the average number for the simple tree
- the gain in performance (for instance “the b-tree is 10% faster because it needs to visit on average 10% less number of nodes”).