SOEN 363 - Database Systems for Software Engineers
Assignment 3

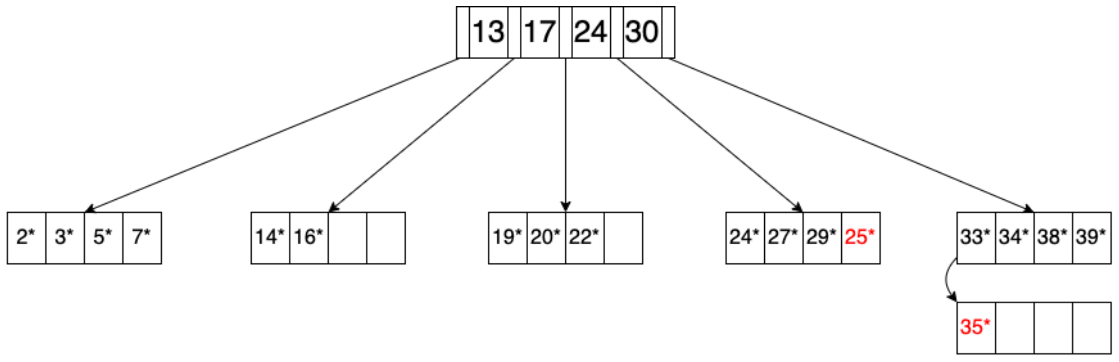
Ali Hanni - 40157164

Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada

Winter 2022

Question 1

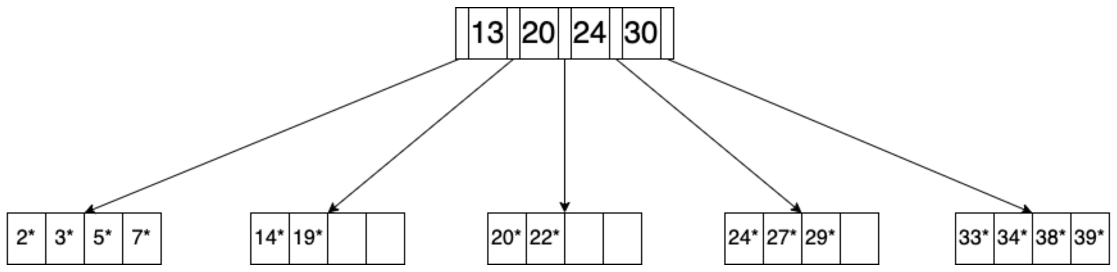
(a) We assume that we have an ISAM structure with 4 entries per page. While 25 is directly inserted in a page, 35 will be an overflow node. New ISAM structure is shown bellow.



(b) The correct tree is Tree B

(c) The correct tree is Tree A

(d) Since a the borrowing should occur from the right sibling, none of the trees are correct. Bellow is shown the correct B^+-tree resulting from deleting of 16* and then borrowing a node from the right sibling;



Question 2

(a) First, we try to find the order of the B^+ tree d. We know that the maximum number of keys is $2d$ and the maximum number of pointers to children is $2d+1$. Knowing that key size is 50-byte, that pointer size is 8-byte, and that page size is 2000 bytes, we have to maximize the value of d such that:

$$\begin{aligned}(2d) * 50 + (2d+1) * 8 &\leq 2000\\ 100d + 16d + 8 &\leq 2000\\ 116d &\leq 1992\\ d &\leq 17.17\\ d &= 17\end{aligned}$$

Hence, when maximizing the value of d we obtain $d = 17$. We know that we can have 34 keys and 35 pointers. Then, we know that a record on any given page holds a key field (50-byte) and a pointer (8-byte). Since the size of one page on the disk is 2000 bytes, we can find the maximum number of entries in a page by doing:

$$\lfloor 2000/(50+8) \rfloor \lfloor 2000/58 \rfloor \lfloor 34.48 \rfloor = 34$$

Hence, there is a maximum of 34 entries per page. We can proceed and compute the number of levels in the tree by doing:

$$\begin{aligned}&\lceil \log_{35}(20000/34) + 1 \rceil\\ &\lceil \log_{35}(588.2353) + 1 \rceil\\ &\lceil 1.79367 + 1 \rceil\\ &\lceil 2.79367 \rceil = 3\end{aligned}$$

There are 3 levels in the B^+ tree described.

(b) Since we deal with a dense B^+ tree, we know that the nodes are filled at each level we know that there are $\lceil 20000/34 \rceil = 589$ leafs on level 3. Since each full node has $2d+1 = 2*17+1 = 35$ children, there are $\lceil 589/35 \rceil = 17$ nodes on level 2 and a single node on level 1 which is the root of the tree.

(c) We try to find the order of B^+ but this time with a key size of 10-bytes and a page size of $2000 * 0.70 = 1400$:

$$\begin{aligned}(2d) * 10 + (2d+1) * 8 &\leq 1400\\ 20d + 16d + 8 &\leq 1400\\ 36d &\leq 1392\\ d &\leq 38.66\\ d &= 38\end{aligned}$$

We can have 76 keys and 77 pointers. Since the size of one page on the disk is 2000 bytes, but pages are only 70% full, we can find the maximum number of entries in a page by doing:

$$\lfloor 1400/(10+8) \rfloor \lfloor 1400/18 \rfloor \lfloor 77.7778 \rfloor = 77$$

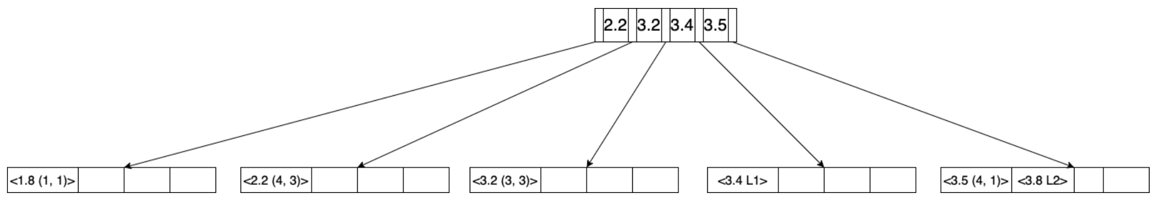
Hence, there is a maximum of 34 entries per page. We can proceed and compute the number of levels in the tree by doing:

$$\begin{aligned} \lceil \log_{77}(20000/77) + 1 \rceil \\ \lceil \log_{77}(259.74026) + 1 \rceil \\ \lceil 1.2799 + 1 \rceil \\ \lceil 2.2799 \rceil = 3 \end{aligned}$$

There are 3 levels in the B^+ tree described.

Question 3

(a) Here is the B^+-tree index :



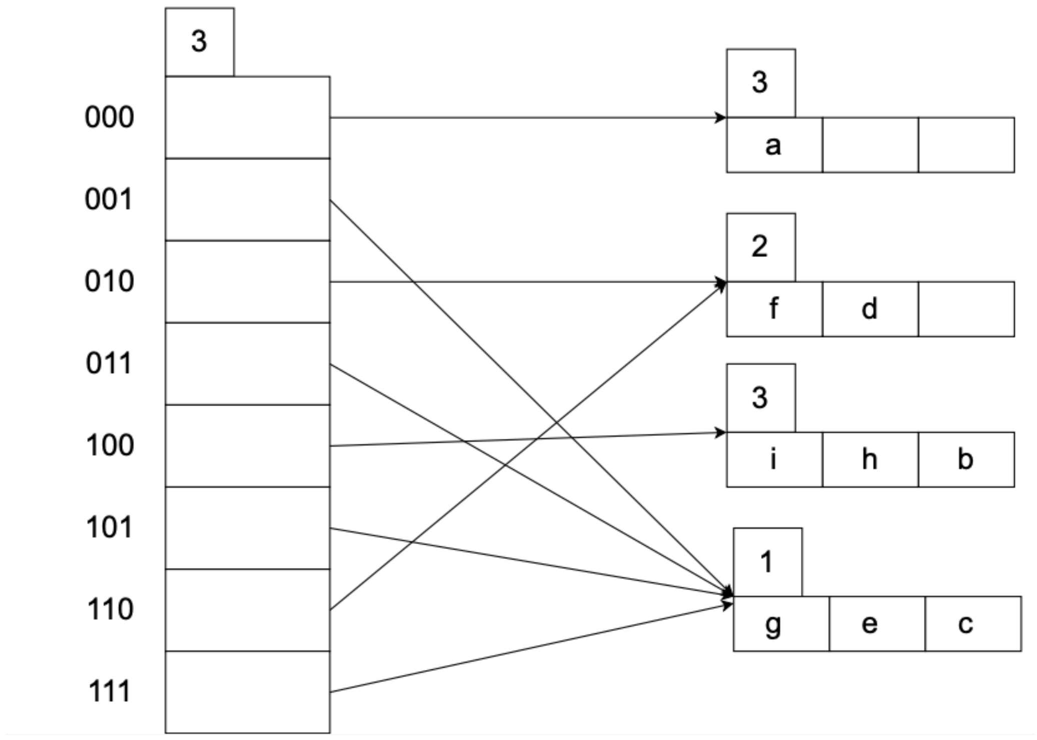
The position of the tuples in the file (page #, slot #) are used to identify the different tuples. Note that for practical reasons, when a certain key (GPA) corresponds to a number of distinct entries that is too large, the list is replaced with a variable. List variables definitions are bellow:
L1 = [(1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1)]
L2 = [(1, 2), (3, 2), (3, 4), (4, 2)]

(b)
1. If the tuples in f are sorted, they will appear in the following order:
   <1.8 (1, 1)>
   <2.2, (1, 2)>
   <3.2, (1, 3)>
   <3.4 (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2)>
   <3.5 (3, 3)>
   <3.8, (3, 4), (4, 1), (4, 2), (4, 3)>
   First, the system would need to access the first leaf node with an index value corresponding to the search's criteria (i.e. gpa between 3.0 and 3.5 inclusive). The first value is 3.2 for an entry at page 1 slot 3. Since data is sorted, the system can take advantage of the fact that each node has a bidirectional pointer to the next leaf block. It can keep reading leaf nodes as long as the index value respects the criteria and return the 9 corresponding entries. In total, the system would need to read 3 pages (pages 1, 2, and 3) costing only 1 I/O per page of records thus, 3 I/O.
2. If the tuples in f are not sorted, they will appear in the following order:
   <1.8 (1, 1)>
   <2.2, (4, 3)>
   <3.2, (3, 3)>
   <3.4 (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1)>
   <3.5 (4, 1)>
   <3.8, (1, 2), (3, 2), (3, 4), (4, 2)>
   Here, since entries are not sorted in the pages, the system would need to return and read the next index from the index tree before proceeding. This time, we would therefore need at least 1 I/O per record (note that for clustered indexes, it is only 1 I/O per page of records). Since there are 9 entries that respect the criteria, we can expect at least 9 I/O.

3. We realize that the number of pages accessed tripples when tuples are unsorted (unclustered indexes) in comparison to when they are sorted (clustered indexes). This factor can quickly increase as the distribution of GPA widen and the number of tuples and tuples per page increases. Also note that clustered indexes are particularly efficient for range searches since it takes advantage of the fact that entries are sorted and can quickly go through pages without having to references the B^+-tree each time. However, clustered indexes are more expensive to maintain as files requires periodical sorting as data is added.

Question 4

(a) The table resulting form the insertion of above records is shown bellow:



(b) The global depth is 3.

(c) There will be 4 buckets.

(d) i, h, and b. The bucket has a local depth of 3.

(e) g, e, and c. The bucket has a local depth of 1.

(f) **a)**. The number of bits to be used in the hash function are kept in the global depth.

(g) No.

(h) If the local depth of a bucket is equal to the global depth of the directory, then the bucket is pointed to by exactly one directory entry.