15-415: Database Applications
Problem Solving Assignment 3

School of Computer Science
Carnegie Mellon University, Qatar
Spring 2016

Assigned Date: March 8, 2016
Due Date: March 24, 2016 by 11:59 PM

I. B^+ Trees Basics [20 points]:

For the following sub-questions, consider the B+ tree structure with order $\boldsymbol{d = 2}$ (i.e. there are at most 4 keys per node, and at most 5 pointers to children) as shown in Figure 1.

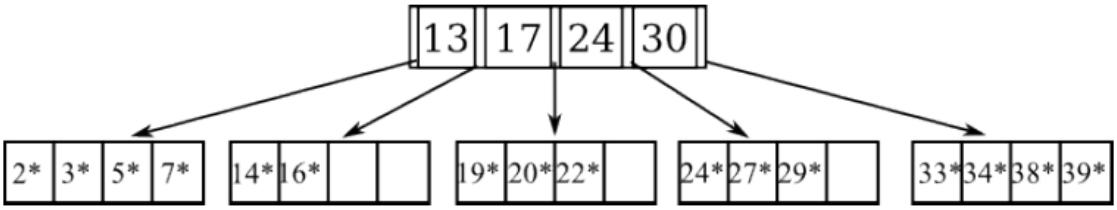
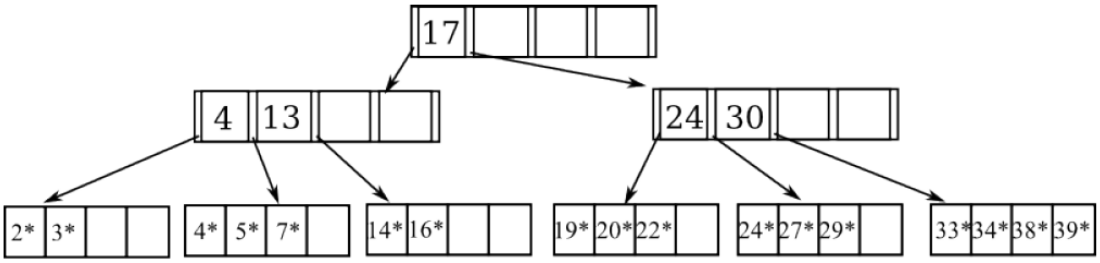


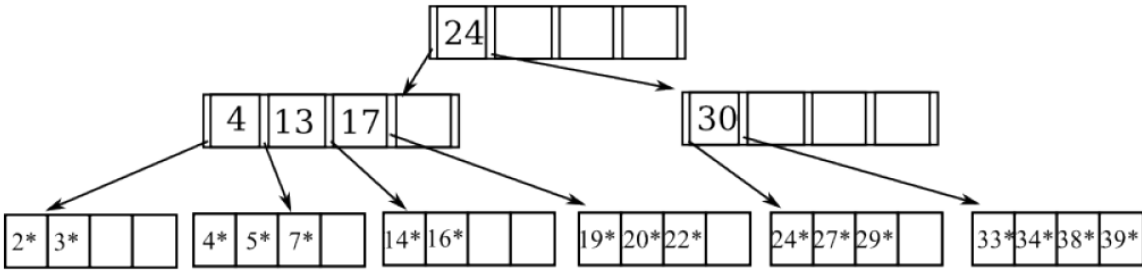
Figure 1: A B^+ tree with order d = 2

Q1.1 Assume the structure in Figure 1 is not a B+ tree, but an ISAM structure. Show the new structure after inserting keys 35 and 37. **[5 Points]**

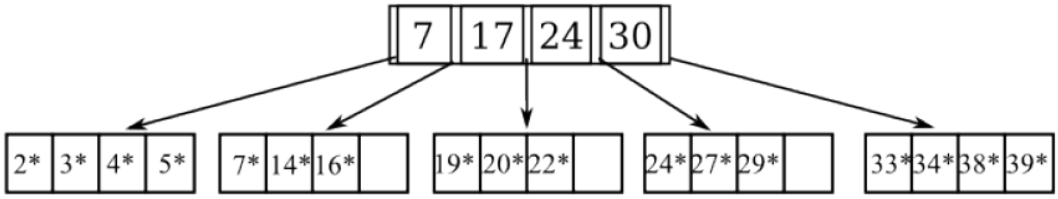
Q1.2 Starting from the original B+ tree in Figure 1, we insert the record 4* with a key 4. From the choices in Figure 2, which is the resulting tree if we use the default "no redistribution" strategy: A, B, C, or neither? If neither, draw the correct tree. **[5 Points]**



Tree A



Tree B

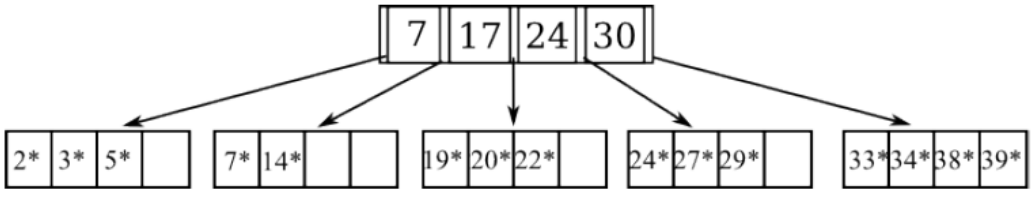


Tree C

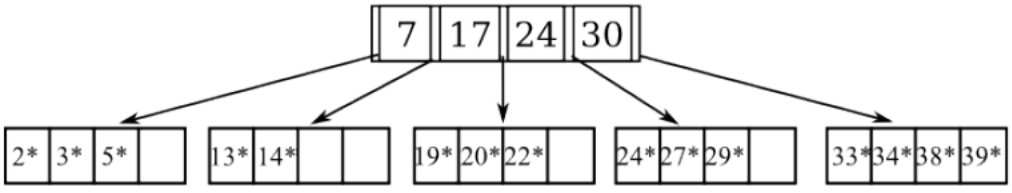
Figure 2: Alternatives for questions 1.2 and 1.3

Q1.3 Starting from the original B+ tree in Figure 1, we insert the record 4* with a key 4. From the choices in Figure 2, which is the resulting tree if we use the default "redistribution with a sibling" strategy: A, B, C, or neither? If neither, draw the correct tree. **[5 Points]**

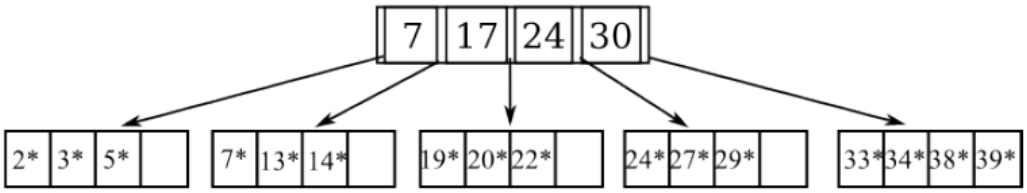
Q1.4 Starting from the original B+ tree in Figure 1, we delete the record 16*. From the choices in Figure 3, which is the resulting tree if we borrow a node from the left sibling: A, B, C, or neither? If neither, draw the correct tree. **[5 Points]**



Tree A



Tree B



Tree C

Figure 3: Alternatives for question 1.4

II. B^+ Trees in Numbers [20 points]:

Assume that you have just built a dense B+ tree index using Alternative (2) on a heap file containing 20,000 records. The key field for this B+ tree index is a 40-byte string, and it is a candidate key. Pointers (i.e., record ids and page ids) are (at most) 10-byte values. The size of one disk page is 1000 bytes. The nodes at each level were filled up as much as possible.

1. How many levels does the resulting tree have? **[5 Points]**

2. For each level of the tree, how many nodes are at that level? **[5 Points]**

3. How many levels would the resulting tree have if key compression is used and it reduces the average size of each key in an entry to 10 bytes? **[5 Points]**

4. How many levels would the resulting tree have without key compression but with all pages 70 percent full? **[5 Points]**

III. B^+ Trees (Clustered vs. Un-clustered) [35 points]:

Consider the instance of the Students relation (shown in Table 1) stored in file *f*.

1. Construct a B+ tree index of order 2 on the *gpa* field using Alternative (3). The tuples in *f* are stored as: the first tuple is in page 1, slot1; the second is in page 1, slot 2; and so on. Each page can store up to four tuples. You may use (page #, slot #) to identify a tuple. Clearly indicate what the data entries are (i.e., do not use the *k** convention). **[15 Points]**

2. Consider the following query:

```
SELECT sid, name FROM Students WHERE gpa >= 3.0 AND gpa <= 3.5
```

   Calculate the IO cost of finding all the tuples that fit the criteria when **(a)** tuples in *f* are sorted, and **(b)** tuples in *f* are unsorted (i.e., they appear in the order shown in table 1). What do you conclude from this? **[20 Points]**

sid	*name*	*login*	*age*	*gpa*
53831	Maclayall	maclayan@music	11	1.8
53832	Guldu	guldu@music	12	3.8
53666	Jones	jones@cs	18	3.4
53901	Jones	jones@toy	18	3.4
53902	Jones	jones@physics	18	3.4
53903	Jones	jones@english	18	3.4
53904	Jones	jones(ggenetics	18	3.4
53905	Jones	jones@astro	18	3.4
53906	Jones	jones@chem	18	3.4
53902	Jones	jones@sanitation	18	3.8
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8
54001	Smith	smith@ee	19	3.5
54005	Smith	smith@cs	19	3.8
54009	Smith	smith@astro	19	2.2

Table 1: Instance of the students relation stored in file *f*

IV. Extendible Hashing [25 points]:

Assume we have the following records where we indicate the hashed key in parenthesis (in binary):

i [001100]
h [001100]
g [101101]
f [010010]
e [111111]
d [010010]
c [100001]
b [001100]
a [000000]

Consider an Extendible Hashing structure where buckets can hold up to three records. Initially the structure is empty (only one empty bucket). Consider the result after the records above have been inserted in the order shown, using the lower-bits for the hash function. As mentioned in the textbook, assume that the directory doubles in size at each overflow.

1. What will be the global depth of the resulting directory? **[2 Points]**

2. How many buckets will we have? **[3 Points]**

3. List all the elements in the bucket which contains the element “i.” What is the local depth of this bucket? **[4 Points]**

4. List all the elements in the bucket which contains the element “c.” What is the local depth of this bucket? **[4 Points]**

5. Do we store the number of bits to use in the hash function in the (a) global or (b) local depth? **[4 Points]**

6. Consider the case that the directory just doubled. Is it true that every bucket will be split in two? (Yes/No) **[4 Points]**

7. If the local depth of a bucket is equal to the global depth of the directory, is this bucket pointed to by (a) exactly one, or (b) multiple directory entry(s): **[4 Points]**