Mohammad Ali Zahir

ID: 40077619

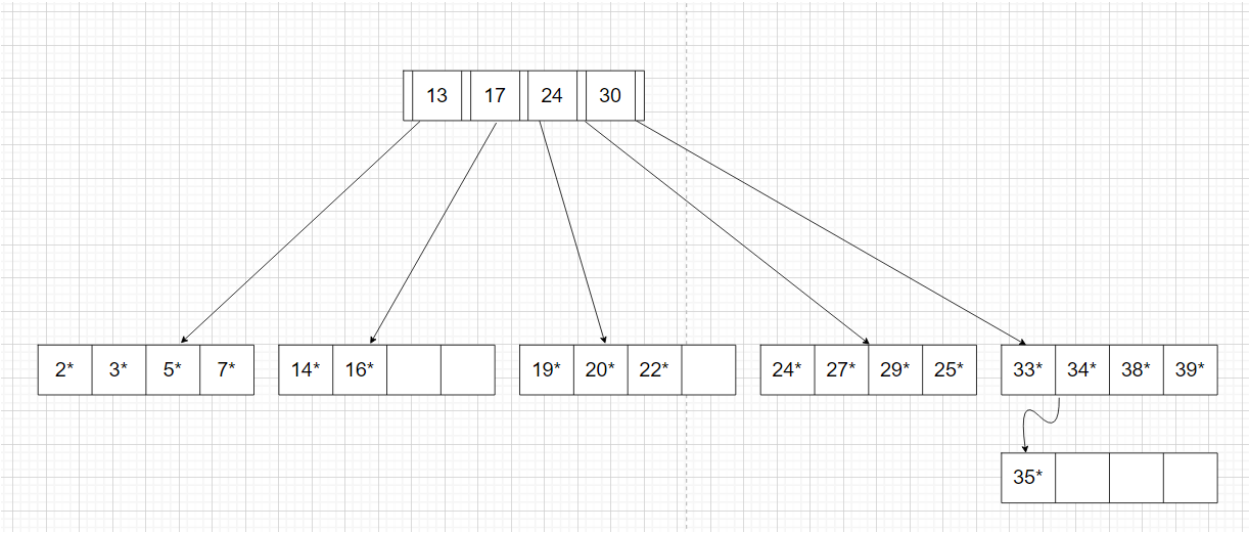
Date: 2022/03/18

SOEN 363: Data Systems for Software Engineers – Section S

SOEN 363 – Assignment 3

1.

a.



b. The resulting tree would be B

c. The resulting tree would be A

d. The resulting tree would be A

2.

a. We know that a B+ tree has at most 2d keys, and 2d +1 pointers. Hence, since we have the total size of the page which is 2000 bytes. We can derive the following equation.

$$(2d * string\ size) + ((2d + 1) * pointer\ size)) \leq 2000$$

$$(2d * 50) + ((2d + 1) * 8) \leq 2000$$

$$100d + 16d + 8 \leq 2000$$

$$116d \leq 1992$$

$$d = 17.17, but\ we\ need\ an\ integer\ so\ we\ will\ round\ down\ to\ d = 17$$

So since we found d = 17, we can then say that we have 2(17) = 34 keys, and 2(17) + 1 = 35 pointers. We also know that each record on a leaf page will take 58 bytes (50 + 8 bytes). Hence then we can do the following calculation to find the number of records.

$$\frac{2000}{58} = 34.48 \rightarrow which\ we\ will\ round\ to\ 34\ records$$

Hence to find the number of levels, we can now do

$log_{35}(\frac{20\ 000}{34}) + 1$ = 2.64, so this means we would need 3 levels.

b. For level 3, we would have, 20 000/34 = 589 nodes

For level 2, we would have, 589/35 = 17 nodes

For level 1, we would have the one node which is at the root of the tree.

c. Since we know that the 70% of the pages are full, we know that the new size of the disk page would 0.7 * 2000, which is 1400 bytes. If we assume no key compression, we can find the value for d, the same way we found it for question a.

$$(2d * string\ size) + ((2d + 1) * pointer\ size)) \leq 1400$$

$$(2d * 10) + ((2d + 1) * 8) \leq 1400$$

$$20d + 16d + 8 \leq 1400$$

$$36d \leq 1392$$

$$d = 38.66, but\ we\ need\ an\ integer\ so\ we\ will\ round\ down\ to\ d = 38$$

Hence, this would mean that we have 2(38) = 76 keys, and 77 pointers. Each record on the leaf page will be 18 bytes (10 +8 bytes). Hence, we could do the following calculation to obtain the number of records.

$$\frac{1400}{18} = 77.77 \rightarrow which\ we\ will\ round\ to\ 77\ records$$

Hence to find the number of levels, we can now do

$log_{77}(\frac{20\,000}{77}) + 1$ = 2.28, so this means we would need 3 levels.

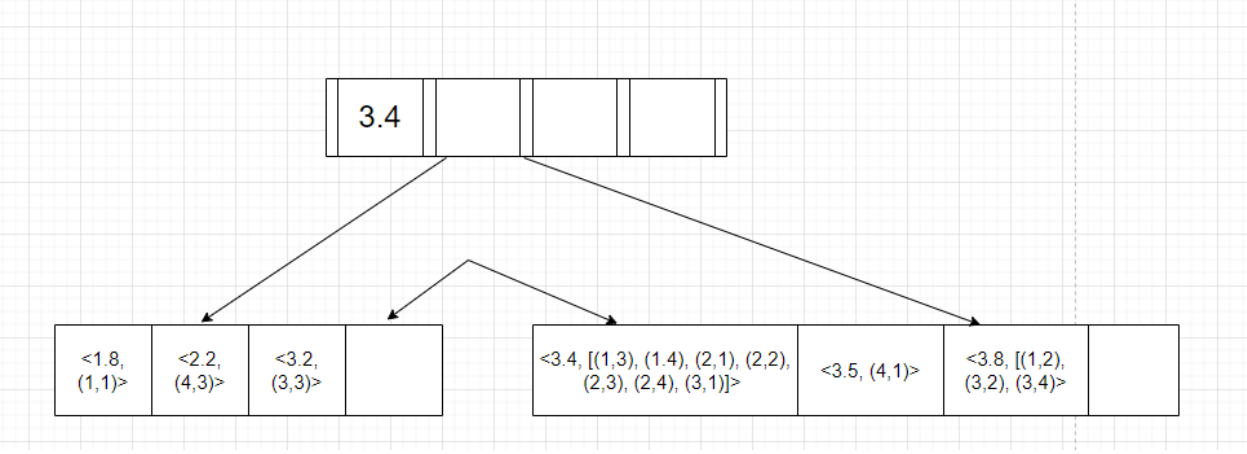
3.

a.

This displays the table for the unsorted tuples

SID	GPA	Tuples
53831	1.8	(1,1)
53832	3.8	(1,2)
53666	3.4	(1,3)
53901	3.4	(1,4)
53902	3.4	(2,1)
53903	3.4	(2,2)
53904	3.4	(2,3)
53905	3.4	(2,4)
53906	3.4	(3,1)
53902	3.8	(3,2)
53688	3.2	(3,3)
53650	3.8	(3,4)
54001	3.5	(4,1)
54005	3.2	(4,2)
54009	2.2	(4,3)

The B+ tree for the following tuple would be



b.

The sorted table for the data would be

SID	GPA	Tuples
53831	1.8	(1,1)
54009	2.2	(1,2)
53688	3.2	(1,3)
54005	3.2	(1,4)
53666	3.4	(2,1)
53901	3.4	(2,2)
53902	3.4	(2,3)
53903	3.4	(2,4)
53904	3.4	(3,1)
53905	3.4	(3,2)
53906	3.4	(3,3)
54001	3.5	(3,4)
53832	3.8	(4,1)
53902	3.8	(4,2)
53650	3.8	(4,3)

i. If the tuples are sorted, we would have 1 IO cost for the tuple with a GPA of 2.2, and 1 cost for the tuple with the GPA of 3.8. Then, we would have another one respectively for a GPA of 3.2, 3.4 and 3.5, which would bring the total for the sorted IO cost to 5.

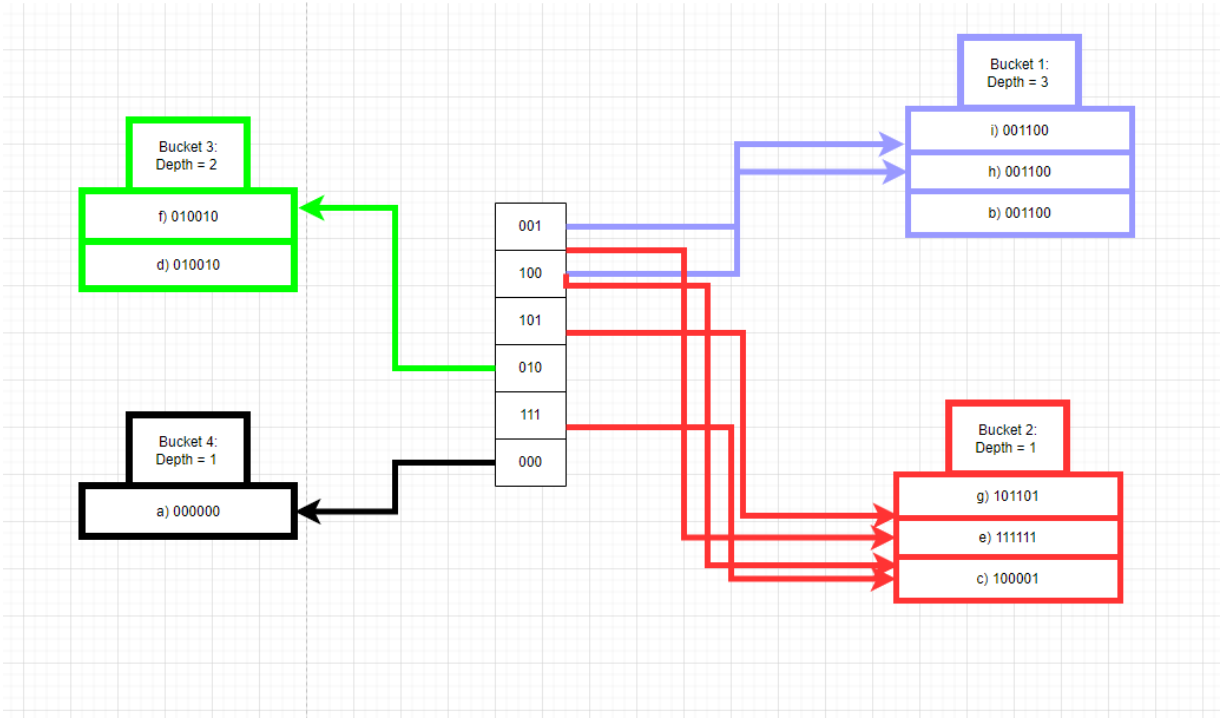
ii. For the unsorted IO cost (table in a), we would again have a total of 2 IO cost for the 2.2 and 3.8 GPA respectively, however, for the 3.4 GPA, we would have an IO cost of 3 since it appears in pages 1, 2, and 3. For the 3.2 GPA, since it only appears on page, the IO cost for it would be 0. Finally, for the 3.8 GPA, we would have an IO cost of 1, which brings our total IO cost for the unsorted tuples to 6.

iii. In terms of IO cost, we can determine that there was no real difference in this case because the value for the sorted IO cost was 5, while the value of the unsorted one was 6.

4.

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

a. If we divide it into binary with 3 values (since the above is a combination of 6 binary values), we will get.



The lines and the buckets have been color coded for simplicity.

b. The global depth of the directory would be 3

c. We will have a total of 4 buckets

d. The bucket which contains the element “i”, has a local depth of 3. The bucket contains the elements “i”, “h”, and “b”.

e. The bucket which contains the element “c”, has a local depth of 1. The bucket contains the elements “g”, “e”, and “c”.

f. We need to store the number of bits to use in the hash function **in the global depth.**

g. Even if the directory was doubled, the bucket would not be split in 2, **hence the answer is NO.**

h. The values as shown above will still be pointed to exactly one directory entry.