Assignment-8 Accessing and Indexing Data

Solutions CS411 Summer ‘18

---

1 - B+ Tree Min Nodes Used in Search

Consider a B+ tree index with $n=50$, where n is the maximum number of keys fitting in a block. The B+ tree indexes 100,000 records. What is the minimum number of nodes in the tree that we need to examine when searching for a record?

- noFigure

✖ 1

✖ 2

✔ 3

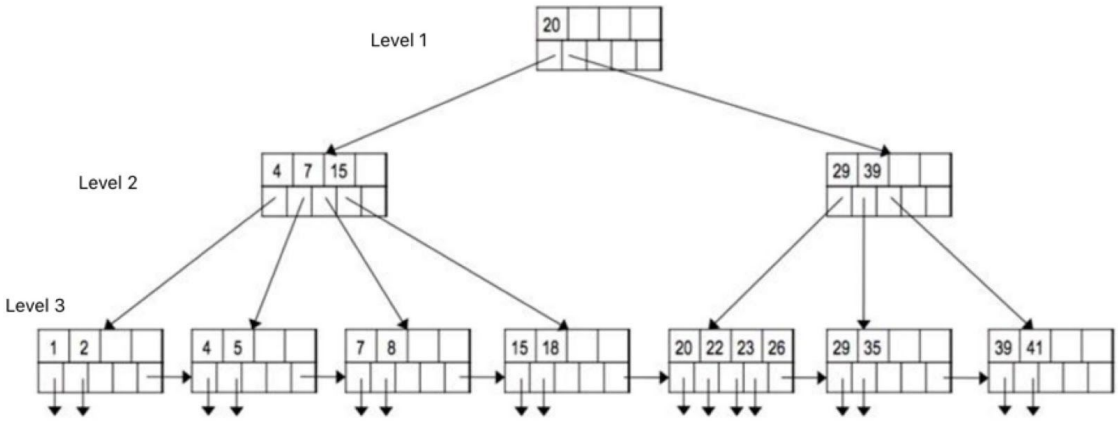
- 1) Need to read in root node (Level 1) 2) A root node can have maximum $n+1=51$ children. (Level 2) 3) Each of the 51 interior node can have maximum 51 children. (Level 3) 4) At this point, we have 51^2 leaf nodes. Each of these leaf nodes can have 50 pointers to records (the 51st pointer points to the block on its right), thus allowing it to index $51^2 \times 50=130050$ records. If you reduce the level by one, you can only index $51 \times 50=2550$ records, thus we need to have at least 3 levels in the B+-tree. With 3 levels, we will need to read in 3 nodes.

✖ 4

---

2 - B+ Tree Insert

Consider the following B+ tree in the figure. Each index node can hold at most 4 keys. A student from our CS411 class listed two solutions for inserting key 24 into the original tree: Solution 1: Split the 5th leaf into [20, 22, 23] and [24, 26]. Add a key and a pointer to the rightmost node at level 2, which then has 3 keys [24, 29, 39] and 4 pointers. Solution 2: Split the 5th leaf into [20, 22] and [23, 24, 26]. Add a key and a pointer to the rightmost node at level 2, which then has 3 keys [23, 29, 39] and 4 pointers. Which of the above two solutions is correct?



✖ Both answers are incorrect.
✖ Solution 1.
✖ Solution 2.
✔ Both answers are correct.

---

3 - Dense Indexing

Suppose we have a relation of 3,000 tuples with no duplicate keys, and each block can hold 5 tuples. How many key-pointer pairs do we need for a dense index of this relation?

- noFigure

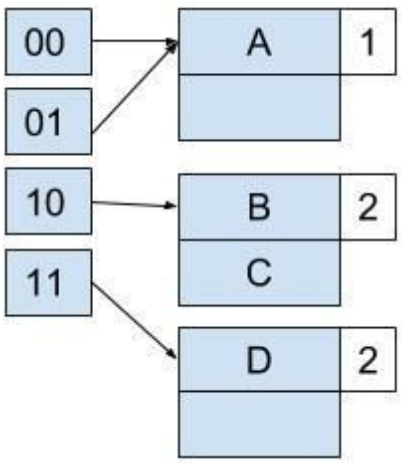
✖ 5
✖ 600
✔ 3,000
- Since this is a dense index, we need a pair for each tuple in the relation. Therefore, we need 3,000 pairs.

✖ 15,000

---

4 - Extensible Hash Table

We consider using the hash function h(n)=n mod 16h(n)=n mod 16 for inserting the following integer keys: 29, 59, 40, 23. According to the resulting extensible hash table in the figure, which of the following option is possible? On top of the result we have in the figure, if we continue to insert 16 and 20, how many buckets will we have in the end?



-

✖ A = 59, B = 29, C = 40, D = 23. 4 buckets in the end.
✖ A = 23, B = 40, C = 59, D = 29. 3 buckets in the end.
✔ A = 23, B = 40, C = 59, D = 29. 4 buckets in the end.

- Because A = 23 (0111), B = 40 (1000), C = 59 (1011), D = 29 (1101), they fall into the corresponding buckets in the correct order as in the picture. After inserting 16(0000) and 20(0100), we will split the first buckets into 2 buckets, so we will have 4 buckets in the end.

---

5 - Extensible Hash Tables vs. Linear Hash Table

Select the true statement.

- noFigure

✖ Extensible Hash Tables do not have overflow chains like Linear Hash Tables, so they are always faster.

- Though they do not have overflow chains, extensions on the table can be costly and take a long time.

✔ The search cost for a Linear Hash Table can vary significantly.

- Depending on the number of overflow chains, a Linear Hash Table can have very low search costs or very high search costs. Since a Linear Hash Table has no control over the length of its overflow chains, the search cost can suddenly become very high.

---

6 - Possible size of a node

Consider a B+ tree: The key size is x byte, the pointer size is x bytes, and the degree d is x−2. What is a possible size of the actual data (size of keys + size of pointers) stored in a node in this tree? Assume that 4<x.
Hint: Considering the lower bound will be enough for this question.

- noFigure

✔ 22

- The node size should be larger than (x−2)(x)+(x−2+1)(x)==$2x^2$ −3x, which is larger than 20 since 4<x.

✖ 16

✖ 12

---

7 - Overflow blocks

Both linear hash tables and extensible hash tables can have overflow blocks when the data block is full.

- noFigure

✖ True

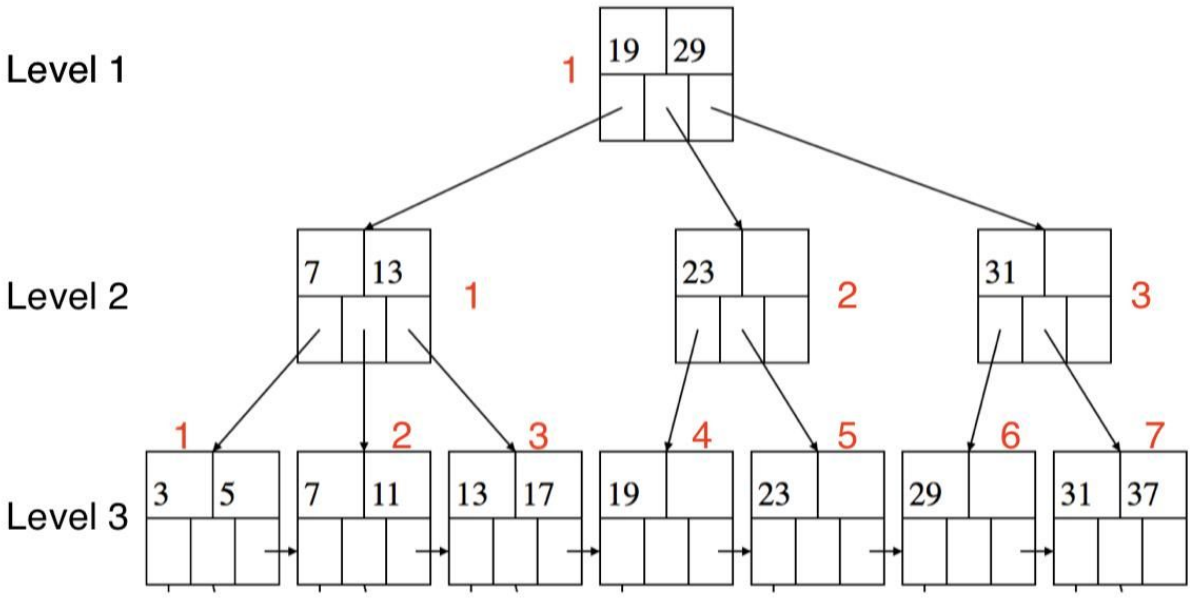
- Only linear hash tables can have overflow blocks. Extensible hash tables cannot.

✔ False

---

8 - Valid solution for insertion/deletion

Consider the following B+ tree in the figure. Each index node can hold at most 2 keys. A student from our CS411 class listed two solutions for inserting key 15 into the original tree: Solution 1: Split the 3th leaf (level 3) into [13, 15] and [17]. Add a key and a pointer to node 1 at level 2, which then has 3 keys [7, 13, 15] and 4 pointers. Then we move its child [17] to its sibling (node 2 at level 2). Now node 1 at level 2 has 2 keys [7, 13] and 3 pointers while node 2 at level 2 has 2 keys [19, 23] and 3 pointers. Solution 2: Split the 3th leaf (level 3) into [13] and [15, 17]. Add a key and a pointer to node 1 at level 2, which then has 3 keys [7, 13, 15] and 4 pointers. Then we move its child [13] to its sibling (node 2 at level 2). Now node 1 at level 2 has 2 keys [7, 15] and 3 pointers while node 2 at level 2 has 2 keys [19, 23] and 3 pointers. Which of the above two solutions is correct?



-

✖ Both solutions are correct

- In solution 2, after adding a key and a point to node 1 at level 2, the rightmost child ([15, 17]) of the node 1 at level 2 should be moved to the sibling on the right instead of [13], otherwise the ordering will not be correct.

✔ Solution 1

- The solution satisfies the constraints in B+ tree.

✖ Solution 2

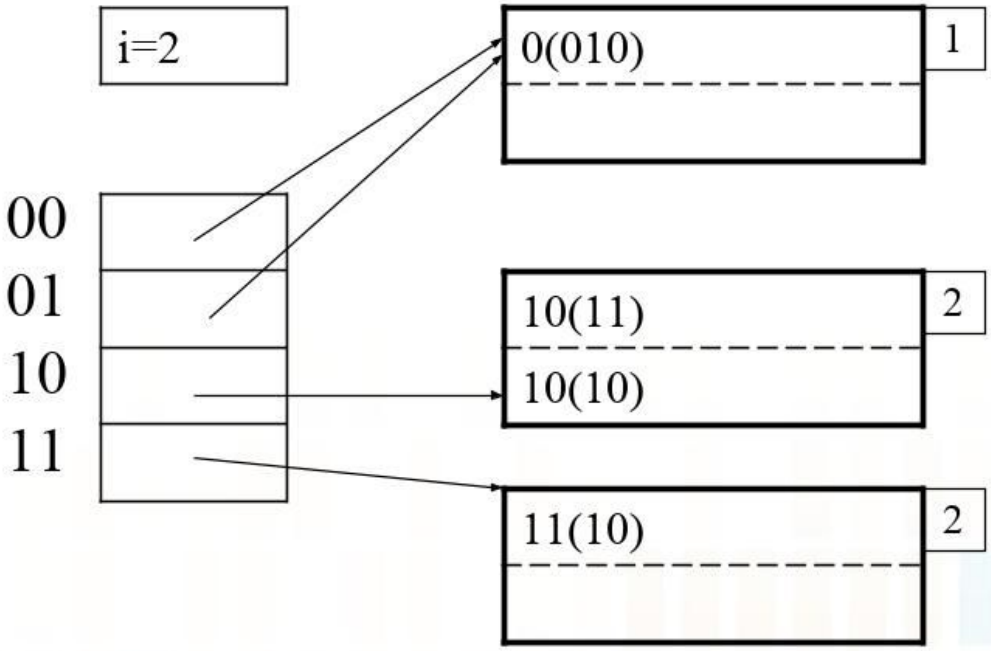
- In solution 2, after adding a key and a point to node 1 at level 2, the rightmost child ([15, 17]) of the node 1 at level 2 should be moved to the sibling on the right instead of [13], otherwise the ordering will not be correct.

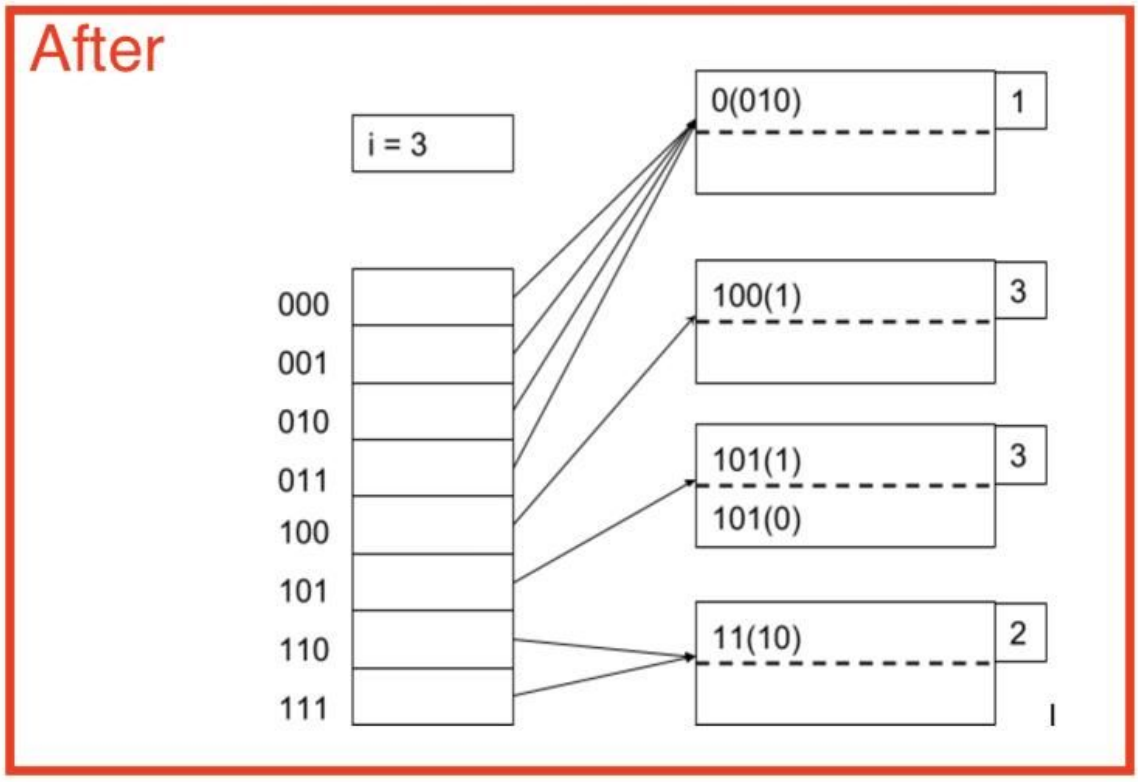
✖ Both solutions are incorrect

---

9 - Extensible hash table after insertion/deletion

Consider the following extensible hash table, where h(n) = n. After inserting 1001, is the resulting hash table shown below correct?





-

✔ Correct

- 1001 will be put into the second bucket, and since the second bucket is full, thus the second bucket is splitted to two buckets with nub value 2+1. Because 3 > i at this point,

the whole hash table need to double the size and make i = 3. The result is the table after expansion and rearrangement.

✖ Incorrect

- 1001 will be put into the second bucket, and since the second bucket is full, thus the second bucket is splitted to two buckets with nub value 2+1. Because 3 > i at this point, the whole hash table need to double the size and make i = 3. The result is the table after expansion and rearrangement.

---

10 - Clustered vs. Unclustered

Select the true statement.

- noFigure

✔ Inserts into clustered indexes take longer than unclustered indexes.

- Inserts and updates take longer because we need to put the data back into a sorted order in clustered indexes.

✖ Clustered indexes are always dense

- They can be either dense or sparse, depending on the attribute we index on. If the attribute is unique, it will be dense since we will need to give every value an entry in the index table. If it is not unique, then it can be a sparse index.
