Data Management Assignment 5

Cuihuan Zhang

May 14, 2025

---

1. (a) i. Scenarios when insertion from a heap would take 1 operation when the new record is directly insert in front (such like the head of a linked list) or null value of the block, without access the other data.

   ii. Scenarios when insertion from a heap would take N operations when the heap is compaction or cannot easily access the back of the heap. For example, insert the value in between, then we need to move the rest of the data.

   (b) Scenarios when insertion from a sorted sequence would take log N + N operations: a sorted sequence requires log N using binary search to find the position, then move the rest of the data cost N when the sequence is compaction.

   (c) Scenarios when insertion from a 2-3 tree would take log N operations when insertion a integer without insert a new node at the parent nodes. Just find the position (node) and insert the value when the node has null value of the block or at the bottom level and bubble up.

2. (a) i. A dense index contains a dedicated pointer (index entry) for every unique key value in the data file (number of indexes = number of records). A sparse index contains pointers only for selected key values (e.g., the first key of each data block) (number of indexes smaller than number of records).
   For example:
   Block1: [ID=1, Alice] → [ID=2, Bob]
   Block2: [ID=3, Mary] → [ID=4, David]
   Dense index file:
   1 → Block1 (Alice)
   2 → Block1 (Bob)
   3 → Block2 (Mary)
   4 → Block2 (David)
   Sparse index file:
   1 → Block1
   3 → Block2

ii. Sparse index requires less storage of the index file, which also leads to less index maintenance operations (no need for index maintenance operations when every time we need to insert or delete a value).

(b) A clustered index ensures that the physical order of records in the data file matches the logical order of the index keys. An unclustered index is independent of the physical order of data records.
For example:

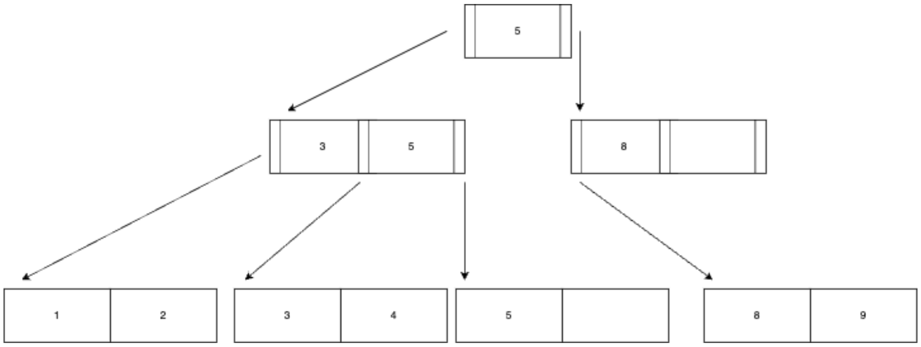

Figure 1: clustered index

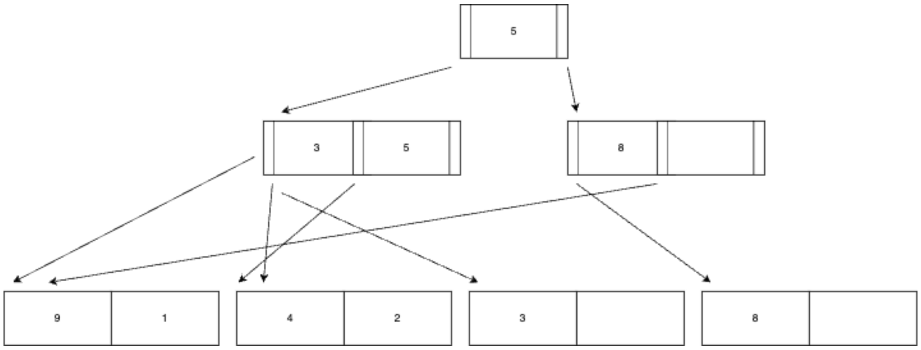

Figure 2: unclustered index

(c) Best: sparse index and clustered file. High storage efficiency since only record the head of blocks. Less index maintenance operation since when inserting/deleting data, the update frequency of sparse indexes is lower than the dense indexes. High query speed since after locates the block by sparse index, the target record must be in the block.
Worest: sparse Index and unclustered file. Low query efficiency since after locates the centain block by sparse index, the target record may not include in the block due to the unclustered file. Require more storage since the localization of sparse indexes fails under the unclustered file, and additional redundant index entries may required to

cover the scattered data. For example, require an additional dense and clustered index.

3. (a)
- **Disk size**: 5 GiB = 5×2^{30} bytes
- **Block size**: 256 bytes
- **Key size**: 8 bytes
- **Pointer size**: ?

Step-by-Step Calculation

1. Total Number of Blocks:

$$\text{Number of blocks} = \frac{\text{Disk size}}{\text{Block size}} = \frac{5 \times 2^{30}}{256} = 5 \times 2^{22} = 20{,}971{,}520 \text{ blocks}$$

2. Block Address Size:

$$\log_2(20{,}971{,}520) \approx 24.32 \text{ bits}$$

- Round up to 25 bits (requires **4 bytes** ($4 \times 8 = 32$ bits, 1 byte = 8 bits) for alignment)

3. The largest possible number of children: Each node contains m pointers and $m - 1$ keys:

$$m \times (\text{Pointer size}) + (m - 1) \times (\text{Key size}) \leq \text{Block size}$$

Substitute values (4 bytes pointers, 8 bytes keys):

$$4m + 8(m - 1) \leq 256$$

$$12m \leq 264 \implies m \leq 22$$

$$\boxed{m = 22}$$

The smallest number of children are 11, and the largest number of children are 22.

(b)

Level	Nodes in a narrow tree ($m = 11$)	Nodes in a wide tree ($m = 22$)
1	1	1
2	2	22
3	22	484
4	242	10,648
5	2,662	234,256
6	29,282	–
7	322,102	–
8	3,543,122	–

For the narrow tree, 8 levels is too much. If we had 8 levels there would be at least 3,543,122 × 11 = 38,974,342 pointers, but there are only 5,000,000 records. So it must be 7 levels.
For the wide tree, 4 levels is too little. If we had 4 levels there would be at most 10,648 × 22 = 234,256 pointers, but there are 5,000,000 records. So it must be 5 levels.

(c) I will use B+ trees because the quantity involves range queries. Quickly locate all records within the interval through the traversal of the B+ tree structure.

4. Computing A Join Of Two Tables

- Table R: 2,000 blocks.
- Table S: 4,000 blocks.
- Available RAM: 6 blocks (2 for R, 4 for S).

(a) Method 1

i. Read 4 blocks of S into memory:

$$\frac{4,000}{4} = 1,000$$

ii. Repeat for all subsets of 4 size of S, need to do it total of 1,000 times:

$$1,000 \times 2000 = 2,000,000$$

iii. Total reads: 1 read of S + 1,000 read of R

$$4,000(S) + 2,000,000(R) = \boxed{2,004,000}$$

(b) Method 2

i. Read 2 blocks of R into memory:

$$\frac{2,000}{2} = 1,000$$

ii. Repeat for all subsets of 2 size of R, need to do it total of 1,000 times:

$$1,000 \times 4000 = 4,000,000$$

iii. Total reads: 1,000 read of S + 1 read of R

$$2,000(R) + 4,000,000(S) = \boxed{4,002,000}$$

Therefore, without sorting the data, the total number of block reads required to execute the given query in the most efficient way is 2,004,000.