CPS216 Advanced Database Systems - Fall 2007
Assignment 2

- Due date: Friday, Oct. 12, 2007, noon (12.00 PM). Late submissions will not be accepted.
- Submission: In class, or email solution in pdf or plain text to shivnath@cs.duke.edu.
- Do not forget to indicate your name on your submission.
- State all assumptions. For questions where descriptive solutions are required, you will be graded both on the correctness and clarity of your reasoning.
- Email questions to shivnath@cs.duke.edu.
- Total points = 200.

Question 1 **Points 20 = 4 x 5**

For each subquestion, state the answer only. Explanations are not required.

A. Consider the portion of B-Tree shown in Figure 1. What is the permissible range of values for X and Y?

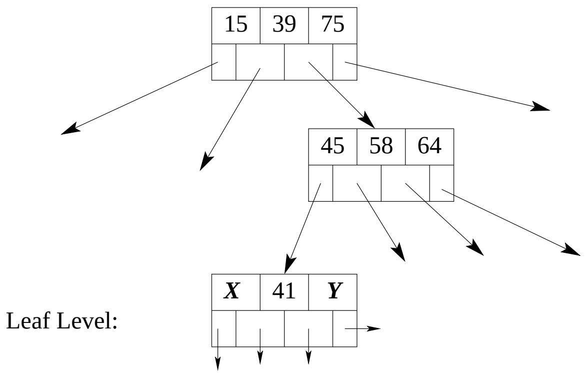

Figure 1: Part of a B-Tree with $n = 3$

B. Consider the B-Tree shown in Figure 2. What is the maximum number of keys that can be inserted into the B-Tree without necessitating the addition of a new level?

C. What is the minimum number of key insertions that causes a new level to be introduced in the B-Tree of Figure 2? Give an example insertion sequence having the minimum number of keys that causes a new level to be added.

D. Consider the portion of B-Tree shown in Figure 3. Delete key 62 and update the B-Tree so that only the three nodes shown in Figure 3 are modified. Show the state of the three nodes after the deletion.

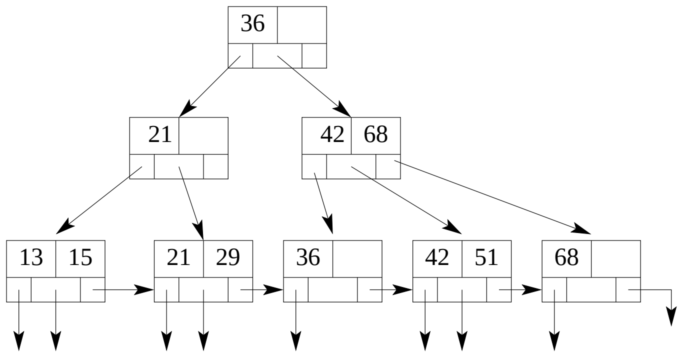

Figure 2: A B-Tree with $n = 2$

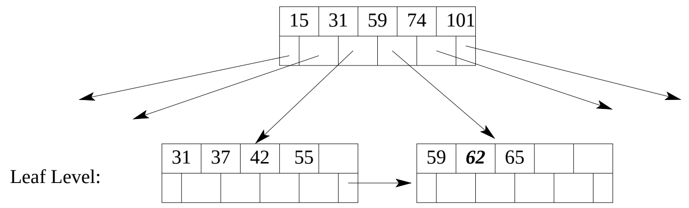

Figure 3: Part of a B-Tree with $n = 5$

E. We start with an empty B-Tree (with a reasonably large n, around 100 say) and insert N ($N \gg n$) keys in sorted order. What can you say about the space utilization of the resulting B-Tree?

Question 2 **Points 20 = 10 + 10**

We are given a large relation $R(A, B, \ldots)$ with 100 million tuples. Attribute A is a primary key for the relation. The relation is *static*, meaning there are no updates, insertions, or deletions to the relation. The only operation over the relation is to search for tuples of R having a specified value of attribute A. Since A is a key, there can be 0 or 1 tuples in R having the specified value. Our task is to perform this operation as fast as possible.

We do not have control over how tuples of R are stored on the disk. In other words, our solution cannot make any assumptions about the storage of R. But each tuple of R is stored in a contiguous region of a single disk block. Therefore, we can maintain pointers to tuples of R, if we chose to.

Values of attribute A are drawn from an ordered domain. We are only permitted to perform the basic comparison operations over values of attribute A, i.e., if a_1 and a_2 are values of attribute A, then we can perform the boolean operations $(a_1 \, Op \; a_2)$, where $Op \in \{<, >, =, \leq, \geq, \neq\}$. In particular, we are not permitted to compute hashes over attribute A values.

We have access to a limited main memory (specified later), but nearly unlimited disk space (around a few tens of GB). The disk characteristics are as follows:

- A random seek takes 10 ms (milli-seconds) on average.
- Size of a disk block is 4096 bytes.
- The sustained transfer rate is 50 MB/sec. Therefore, once the head is over a block, we can read the block in about 0.08 ms. For simplicity, we ignore gaps: so reading two contiguous blocks takes $2 * 0.08$, reading three takes $3 * 0.08$, and so on.

Finally, we need 8 bytes to store a value of attribute A, and 8 bytes to store a disk-based pointer (for simplicity, we assume both record pointers and block pointers require the same 8 bytes).

Clearly, we need to build an auxiliary index-like structure to support the search operation. One obvious approach to the problem is to build one of the indexes that we have studied (e.g., B-Tree). But most of these are general purpose indexes, which may not be best for our specialized task.

A. Design an index structure that optimizes the search operation over R. Characterize its performance in terms of the time required to perform each search operation. For this part of the problem, design a solution that is completely disk-based, i.e., you are not allowed to use main memory to hold parts of the index.

B. Suppose you have access to 512 MB of main memory. You can now store a portion of your index in main memory to speed up search. Design an index for this case, and characterize its performance.

For all the cases above, assume you have sufficient memory for performing I/O operations. In your solutions, clearly state all your assumptions. Some portion of credit is for clear and concise descriptions.

Question 3 **Points 20**

What is the minimum number of insertions required to increase the number of levels of a B-Tree from 2 to 4. State your answer in terms of a general n, but assume that n is odd to avoid floors and ceilings in your expressions. Justify your answer briefly. A portion of credit is allocated for clear, concise justifications.

Question 4 **Points 45 = 10 + 7 * 5**

Suppose you have two clustered relations R(A,X,Y) and S(B,C,Z). You have the following indexes on S.

- A non-clustering B-tree index on attribute B for S.
- A clustering B-tree index on attribute C for S.

Assume that both indexes are kept entirely in memory always (i.e., you do not need to read them from disk). Also, assume that all of the tuples of S that have the same value of attribute C are stored in sequentially adjacent (i.e., contiguous) blocks on disk. That is, if more than one block is needed to store all of the tuples with some value of C, then these blocks will be located sequentially on the disk.

You have the following information about R and S:

- 100 tuples of R are stored per block on disk. Assume that blocks of R are laid out contiguously on disk.
- T(R) = 360,000 (number of tuples of R). The values of attribute A in R range from 1 to 360,000. Assume that A is a key of R, so each tuple in R has a unique value of A in [1,...,360,000].
- 5 tuples of S are stored per block on disk.
- T(S) = 1,200,000 (number of tuples of S).
- V(S,B)= 1200, i.e., there are 1200 distinct values of attribute B in S. Assume that these values are distributed uniformly in S, so each value of B occurs T(S)/V(S,B) = 1000 times in S. Furthermore, assume that these values range from 1 to 1200. That is, for each value v in [1,...,1200], there are 1000 tuples in S with S.B = v.

- V(S,C)= 120,000, i.e., there are 120,000 distinct values of attribute C in S. Assume that these values are distributed uniformly in S, so each value of C occurs T(S)/V(S,C) = 10 times in S. Furthermore, assume that these values range from 1 to 120,000. That is, for each value v in [1,...,120,000], there are 10 tuples in S with S.C = v.

You want to execute the following query:

```
SELECT *
FROM R, S
WHERE R.A = S.B AND R.A = S.C
```

We present you with two indexed-nested-loop-join plans:

Plan 1:

```
For every block BLK of R, retrieved using a scan of R
    For every tuple r of BLK
        Use the index on B for S to retrieve all of the tuples s of S such that s.B=r.A
            For each of these tuples s, if s.C=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z
```

Plan 2:

```
For every block BLK of R, retrieved using a scan of R
    For every tuple r of BLK
        Use the index on C for S to retrieve all of the tuples s of S such that s.C=r.A
            For each of these tuples s, if s.B=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z
```

Note that both plans read R one block at a time, and retrieve all S tuples that join with tuples in the current block of R (using one of the indexes on S) before reading the next block of R.

a. Analyze each of these plans in terms of their behavior regarding accesses to disk. For each plan compute the number of sequential accesses and the number of random accesses to blocks on disk. Given that random accesses are at least an order of magnitude costlier than sequential accesses, which of the plans performs better?

b. Assume all statistics remain the same except for the number of tuples of S stored per block on disk, which now reduces to 2 (from 5). How does this change your answer to (a)?

c. Let the variable X represent the number of tuples of S stored per block on disk. Assuming all other statistics remain the same as before, what values of X in [1,...,10,000] will make the worse plan of (a) perform better than the other?

d. Which plan is better if both indexes are non-clustering, and everything else remains as specified originally in the question? Note that now tuples of S that have the same value of attribute C are not stored in contiguous blocks on disk.

e. Which plan is better if both indexes are non-clustering, and V(S,B) = 180,000? There are 180,000 distinct values of attribute B in S. Assume that these values range from 1 to 180,000 and are distributed uniformly in S. V(S,C)= 120,000 as before.

f. Suppose everything remains as specified originally in the question except that values of attribute B come from the domain 1-3,600,000. (That is, the domain is positive integers 1,2,3 and so on up to 3.6 million.) Assume that the values of attribute B in S are distributed uniformly in this domain, and V(S,B) = 1,200,000. Which plan is better in this scenario?

Question 5 **Points 20 = 10 + 10**

A set of indexes is called a *covering index set* for a query if the query can be evaluated using these indexes only (i.e, without fetching any data records). For queries Q1 and Q2 below:

(a) Give a minimal covering index set

(b) Give an efficient technique (need not be a query plan; an explanation will suffice) to evaluate the query using your minimal covering index set from (a)

(c) Compute the number of disk blocks read by your technique from (b)

Queries Q1 and Q2 are as follows:

```
Q1: SELECT R.a
    FROM R, S
    WHERE R.a = S.a

Q2: SELECT DISTINCT R.a
    FROM R, S, T
    WHERE R.a > S.a AND S.a >= T.b
```

Note that SQL's DISTINCT operator used in Q2 will eliminate duplicates from Q2's result. DISTINCT is the duplicate-eliminating project that we considered in Homework 1. DISTINCT is also discussed in Section 6.4.1 of the textbook.

Make the following assumptions about relations R(a,b), S(a,b), and T(a,b) (Note: you may not need all this information to compute the number of disk blocks accessed):

- R.a is the primary key of R, S.a is the primary key of S, and T.a is the primary key of T.
- All relations are clustered.
- B(R) = 1000, B(S) = 10,000, and B(T) = 100,000
- T(R) = 10,000, T(S) = 50,000, and T(T) = 300,000. (T(T) denotes the number of tuples in relation T.)
- There are clustering B-tree indexes on R.a, S.a, and T.a. There are non-clustering B-tree indexes on R.b, S.b, and T.b.
- For simplicity of computation, assume that all indexes contain two levels, with the root node in the first level and some number of leaf nodes in the second level. The indexes on R.a and R.b contain 25 leaf nodes each; the indexes on S.a and S.b contain 250 leaf nodes each; and the indexes on T.a and T.b contain 2500 leaf nodes each.
- Assume that root nodes of all indexes are always in memory so that access to a root node never incurs an I/O.

Question 6 **Points 35**

The following information is available about relations R and S:

- Relation R is clustered and the blocks of R are laid out contiguously on disk. B(R) = 1000 and T(R) = 10,000.
- Relation S is clustered and the blocks of S are laid out contiguously on disk. B(S) = 500 and T(S) = 5000.
- M = 101 blocks.
- For simplicity, we will assume that a random access can be done on average in time $t_r = 20$ ms, and a sequential access can be done on average in time $t_s = 1$ ms. For example, scanning five contiguous blocks on disk, assuming the first access is random, incurs a cost $t_r + 4t_s$.

1. [**10 Points**] How will you extend the "Efficient" Sort-Merge Join algorithm (that we learned in class) to minimize cost when our cost model distinguishes between random accesses and sequential accesses? (Note that in class we did not distinguish between random and sequential accesses.) Compute the cost of your algorithm.

2. [**10 Points**] Design an algorithm for the block nested-loop join of relations R and S which has minimum cost when we distinguish between random and sequential disk accesses. Compute this minimum cost using the parameter values specified above.

3. [**15 Points**] How does your answer to (2) change if blocks of R are not laid out contiguously on disk? All other assumptions and parameters remain the same as specified above. Compute the minimum cost possible for block nested-loop join in this case.

Question 7 **Points 20**

Consider the join of four relations $R1 \bowtie R2 \bowtie R3 \bowtie R4$. We have not shown the join predicates since they are not relevant to this problem. Consider two plans for joining these relations: one using a left-deep join tree (Figure 4) and one using a right-deep join tree (Figure 5). $X1, X2, X3, X4$ represent various intermediate relations produced in the plans. All the join operators are tuple-based, nested loop joins. The plans are fully pipelined. Only 4 blocks of memory are available. We have $B(R1) = B(R2) = B(R3) = B(R4) = 1000$ blocks, and $T(R1) = T(R2) = T(R3) = T(R4) = T(X1) = T(X2) = T(X3) = T(X4) = 10000$ tuples. What is the number of disk I/Os for the left-deep plan and the right-deep plan?

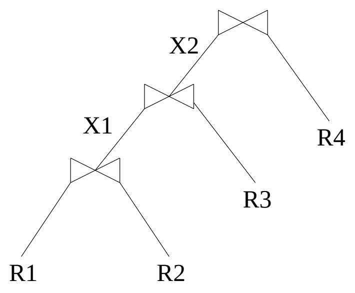

Figure 4: Left-deep plan

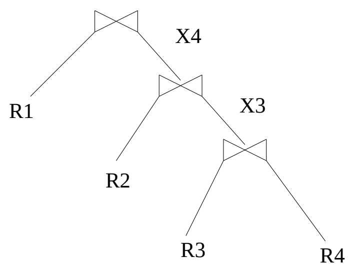

Figure 5: Right-deep plan

Question 8 **Points 20**

Consider the following query over relations R_1–R_4:

$R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$

Suppose there are three possible access methods for each R_i and two possible join methods for each join. Assume that all combinations of access and join methods are feasible, and that both join methods are asymmetric (e.g., the two join methods could be Nested-Loop join and Hash join, both of which are asymmetric).

1. [**5 Points**] How many different left-deep plans are there for this query?
2. [**7 Points**] How many different bushy plans are there for this query? Note that a plan that is not left-deep or right-deep is bushy.
3. [**8 Points**] How would your answer to (1) change if there is only one join method, but this join method is symmetric (e.g., the join method could be Sort-Merge join, which is symmetric)? Compute the number of different left-deep plans in this case.