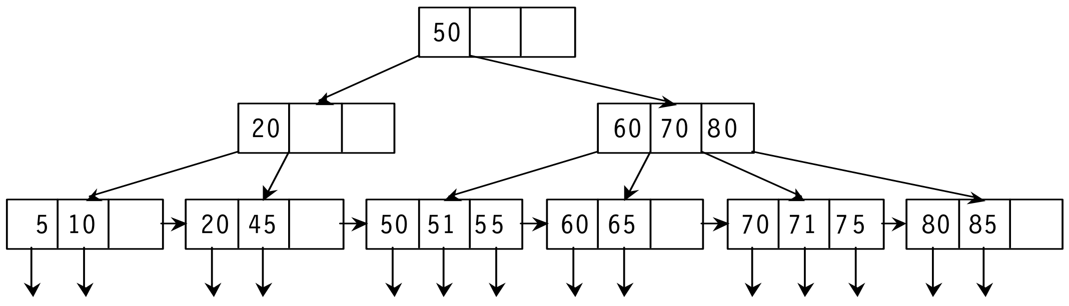
CPS 216 Spring 2005
Homework #2
Assigned: Tuesday, February 15
Due: Thursday, March 3

Problem 1.

Consider the following B$^+$-tree with a maximum fan-out of 4.



For all questions below, assume that you always start with the tree shown above—*not* the result tree you get for the previous question.

(a) Show the result tree after inserting 49.
(b) Show the result tree after deleting 50.
(c) Show the result tree after inserting 72.
(d) Show the result tree after deleting 5.
(e) What is the minimum number of keys you must delete for this tree to shrink down to two levels? Show the sequence of deletions.

Problem 2.

The following question is based on the paper "Generalized Search Trees for Database Systems," by Hellerstein et al. Suppose we want to use GiST to index ranges of the form $[x, y)$, where x and y are integers. These ranges may overlap with each other. The queries are of the form "find all ranges that overlaps with $[a, b)$." Discuss briefly how you would implement the six basic methods required by GiST, and whether you should set *IsOrdered* to true and define *Compare*. Highlight the differences between your implementation and the B$^+$-tree implementation described in the paper.

Problem 3.

Suppose keys are hashed to 4-bit sequences, and each block can hold three records. We start with a hash table with two empty blocks (corresponding to 0 and 1), and insert 16 records with keys 0000, 0001, 0010, …, 1111, in order. Show the final state of the index:

(a) If the index is based on extensible hashing.
(b) If the index is based on linear hashing, with a capacity threshold of 100%. Here, we define capacity to be (actual number of records indexed) / (maximum number of records that can be held by primary buckets).

Problem 4.

To build a hash index for a multi-attribute search key, we can use an approach called partitioned hashing. The partitioned hash function is really a list of hash functions, one for each attribute in the search key. Suppose that we wish to build a partitioned hash index on $R(A, B)$ with 2^n buckets numbered 0 to $2^n - 1$. In this case, the partitioned hash function consists of two hash functions h_A and h_B. Hash function h_A takes a value of A as input and produces a result with n_A bits, and h_B takes a value of B as input and produces a result with $(n - n_A)$ bits. The two results are concatenated together to produce the result of the partitioned hash function, which is then used to index the buckets. To locate records with $A = a$ and $B = b$, we simply go directly to the bucket numbered $h_A(a)\ h_B(b)$ (in binary).

(a) Which buckets do we have to examine in order to locate all records with $A = a$?
(b) Suppose we are given a query mix. Each query in this mix will either ask for records with a given value of A, or it will ask for records with a given value of B (but never both). With probability p, the value of A will be specified. Give a formula in terms of n, n_A, and p for the expected number of buckets that must be examined to answer a random query from the mix.
(c) Find the value of n_A, as a function of n and p, that minimizes the expected number of buckets examined per query.

Problem 5.

Consider the join $R \bowtie_{R.A = S.B} S$, given the following information about the tables to be joined. The cost metric is the number of disk I/O's and the cost of writing out the result should be uniformly ignored.

- R contains 10,000 rows and has 10 rows per page.
- S contains 2,000 rows and also has 10 rows per page.
- $S.B$ is a key of S.
- Both tables are stored compactly on disk in no particular order.
- No indexes are available.
- 52 memory blocks are available for query processing.

(a) What is the expected cost of joining R and S using a block-based nested-loop join, with R as the outer table?
(b) What is the expected cost of joining R and S using a block-based nested-loop join, with S as the outer table?
(c) What is the expected cost of joining R and S using a sort-merge join? What is the minimum number of memory blocks required for this cost to remain unchanged?
(d) What is the expected cost of joining R and S using a hash join? What is the minimum number of memory blocks required for this cost to remain unchanged?

Problem 6.

The *mode* of a list of values is the most common (frequent) value. There can be more than one mode for a list. For example, the list 1, 2, 2, 3, 4, 4, 2, 1, 1 has two modes 1 and 2.

(a) Assume that no index is available. Design the best algorithm you can come up with to compute one mode (any one) of a column in a table. Analyze the worst-case performance of your algorithm in terms of number of I/O's.

(b) Consider the following query that compute one mode for a subset of a table:
`SELECT mode(r.A) FROM R r WHERE r.B > 0;`
Suppose that there is a B$^+$-tree index on *R*(*A*) and a bit-sliced index on *R*(*B*); both are secondary indexes. What would be a good algorithm for processing this query?