Computer Science 425
Fall 2006
Second Take-home Exam
Out: 2:50PM Wednesday Dec. 6, 2006
Due: 5:00PM SHARP Friday Dec. 8, 2006

Instructions:
This exam must be entirely your own work. Do not consult with anyone else regarding this exam. If you have questions about what is being asked, contact Prof. LaPaugh. You are allowed to use the following reference materials: You may use your personal notes and problem set submissions, *Database Management Systems* by Ramakrishnan and Gehrke, the solutions to odd-numbered exercises provided by the authors at http://www.cs.wisc.edu/~dbbook/openAccess/thirdEdition/supporting_material.htm and copies of any of the material on the Fall06 COS 425 Web site for the course (staying within the site http://www.cs.princeton.edu/courses/archive/fall06/cos425/...). Furthermore, for this exam you may use the strongly suggested sections of [Abraham Silberschatz, Henry Korth, and S. Sudarshan, *Database System Concepts, Fifth Edition*, McGraw-Hill, 2006] on XML: 10.1, 10.2, 10.3, 10.4.1, 10.4.2, and 10.7.
No other materials are allowed. Also you are **NOT allowed to use any online database or search interfaces — in particular, no use of SQL or XML tools.**

There are 6 problems, with point values indicated, totaling 100 points.
Be sure to give explanations for your answers.

On the cover page of your exam submission, write and sign the acknowledgement of original work:

"This exam represents my own work in accordance with University regulations."

Turn in your exam by 5:00pm Friday December 8, 2006 to Professor LaPaugh at her office in Room 304 of the Computer Science building. If you wish to turn in your exam when Professor LaPaugh is not in her office, you may give it to the course secretary, Mitra Kelly, in Room 323 of the Computer Science Building or you may put it in Professor LaPaugh's mailbox on the second floor of the Computer Science Building. If you put it in her mailbox, please also email her that it is there.

Problem 1 (15 points) This problem uses the XML Schema of an example done in class on 10/18/06 and reproduced here:

```
< xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="books" type="ListBooksType"/>
        <xs:element name="book" type="BookType"/>
        <xs:element name="author" type="AuthorType"/>
        <xs:complexType  name="BookType">
                <xs:attribute name="in_print"/>
                <xs:sequence>
                        <xs:element name="title" type="xs:string"/>
                        <xs:element name="isbn" type="xs:string"/>
                        <xs:element name="edition" type="xs:string"/>
                        <xs:element name="date" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType  name="AuthorType">
                <xs:sequence>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="dob" type="xs:string"/>
                        <xs:element name="place_ob" type="xs:string"/>
                        <xs:element name="do_death" type="xs:string"/>
                        <xs:element name= "isbn" type="xs:string" minOccurs= "0"
                                                        maxOccurs=  "unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType  name="ListBooksType">
                <xs:sequence>
                        <xs:element ref="book" minOccurs="1"
                                                maxOccurs="unbounded"/>
                        <xs:element ref="author" minOccurs="1"
                                                 maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:key name= "bookKey" >
                <xs:selector  xpath= "/books/book"/>
                <xs:field  xpath= "isbn"/>
        </xs:key>
        <xs:keyref name= "authorBookFkey" refer= "bookKey" >
                <xs:selector  xpath= "/books/author"/>
                <xs:field  xpath= "isbn"/>
        </xs:keyref>
</xs:schema>
```

Consider this XQuery query:

```
for $au in /books/author
return
  <result>
       {$au/name}
       {let $au_bks := $au//isbn
                       return
                       <a_lists>{$au_bks}</a_list>}
  </result>
```

Part a: Give an English description of the query.
Part b: Give an XML Schema specification of the schema of the query result.

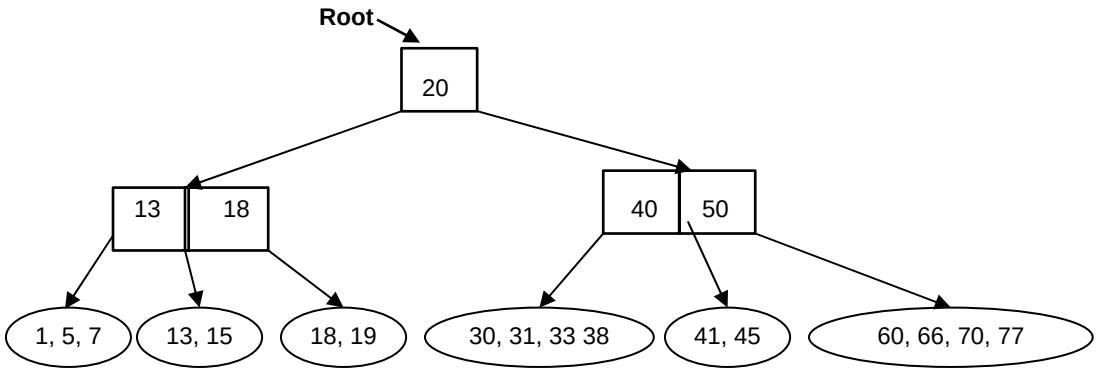
Problem 2 (15 points)
Pagerank is computed on the link graph for HTML documents (or more generally Web objects). It does not take into account the content of documents, only the links between them. Suppose we have a similarity score for each pair of HTML documents in a collection; this similarity score measures how similar the documents are in content. Scores range from 0 to 1 inclusive. (One example of a similarity score for a pair of documents is the normalized dot-product of their vectors in the vector model of terms and documents.) Modify the link graph of documents so that each edge, say from node i to node j, is labeled with a weight that is the similarity score of the documents represented by nodes i and j.

Part a: Suggest a modified version of the pagerank algorithm that uses the weights on the edges. Your modified algorithm should have the following properties:

1. A node propagates some pagerank value to a node it is linked to even if the similarity score on the edge is 0.
2. If all edges in the graph have identical similarity score, the modified pagerank is the same as the original pagerank.

Part b: Do you think this modification would improve the scoring of documents by pagerank? Justify your answer.

Problem 3 (10 points)
Consider this B+ tree of order 2:



Links forming a doubly-linked list of leaves are not shown

Part a: Delete the record with B+ tree key 15 according to the deletion algorithm. Show the resulting tree. How many disk pages were read? How many written?

Part b: Delete the record with B+ tree key 13 according to the deletion algorithm. Show the resulting tree. How many disk pages were read? How many written?

Problem 4 (20 points)
Your are going to build indexes on a set of data records with a search key that is 8 bytes long. You may assume that pointers are also 8 bytes long. A disk page is 1 kilobyte (1024 bytes). There are 14 million data records.

Part a: How many search key values and pointers can fit in a disk page? Given this, what is the order of a B+ tree that will have interior nodes that come as close as possible to filling a disk page when those nodes contain the maximum number of keys for a B+ tree of that order? Give your reasoning.

Part b: Suppose we want a B+ tree whose leaves hold key values and pointers to data records (Alternative 2 in Ramakrishnan and Gehrke terminology.) Assume the search key is a candidate key. If the B+ tree interior nodes and leaves are full or nearly full, what is the height of the tree?

Part c: Suppose we have enough buffer pages in main memory to keep the root and children of the root of the B+ tree in buffer. How many pages of disk are used for the tree, including the leaves? How many disk page reads will it take to access the data record for a given key value?

Part d: Now suppose we want to create an extendible hash index instead of the B+ tree index but using ***no more buffer space*** than the B+ tree. The hash index will again be

Alternative 2; each bucket will occupy one disk page. Describe the hash index: how many buckets are there? How big is the directory? How full are the buckets? Are there any overflow pages? What can be kept in the main memory buffers?

Part e: How many disk page reads will it take to access a data record for a given key value using the hash index you designed in Part d.

Problem 5 (20 points)**:**
Let R be a relation with fields (i.e. attributes) named *a_1, a_2, …, a_p, b_1, b_2, … b_q* and S be a relation with fields named *b_1, b_2, … b_q*. Consider the relational algebra query $\Pi_{a\_1, a\_2, \ldots, a\_p}$ **(**R ◊◊ S) where ◊◊ is the natural join on shared fields *b_1, b_2, … b_q*. Suppose that S is stored sorted on (*b_1, b_2, … b_q*). That is, *q*-tuples of S appear in order of *b_1* value and for equal values of *b_1* in order of *b_2*, etc. Furthermore, suppose R is accessed by a B+-tree on (*a_1, a_2, …, a_p*) whose data entry for a particular value of the p-tuple points to the first tuple of R with that value for (*a_1, a_2, …, a_p*) . Remaining tuples of R with a given value for (*a_1, a_2, …, a_p*) are stored contiguously on disk in sorted order of (*b_1, b_2, … b_q*). Let M be the number of tuples in R, N be the number of tuples in S and k be the number of unique values of (*a_1, a_2, …, a_p)* appearing in R.

Consider $\Pi_{a\_1, a\_2, \ldots, a\_p}$ **(**R ◊◊ S) as one operation on R and S when R and S are of the form described above. Describe an algorithm for computing this operation that uses the B+ tree index. What is the approximate disk I/O cost to compute the operation, including removal of duplicates, in terms of M, N, and k? (If you need additional parameters, state what they are.) Under what circumstances is this an improvement over re-sorting R on (*b_1, b_2, … b_q*), doing a merge-join and projecting during the join? Your algorithm will be judged on correctness and efficiency.

Problem 6 (20 points)**:**
Consider the calculation of query (R $\lozenge\lozenge_{R.a=S.b}$ S $\lozenge\lozenge_{S.b=T.c}$ T) for relations R, S, and T. R contains 50,000 tuples and has 5 tuples per page; S contains 100,000 tuples and has 10 tuples per page; T contains 1000 tuples and has 25 tuples per page. R has a clustered, Alternative 1 hash index on R.a. S has a clustered, Alternative 1 B+ tree index on S.b, which is the primary key for S. T is stored as a heap and has an unclustered (Alternative 2) index on T.c. 45 buffer pages are available in main memory for the evaluation of the query .

Part a: Consider using hash-join for both joins.

i. What is the best order to do the joins in terms of minimizing disk I/O?
ii. Can any of the indexes be used productively during either hash-join? Explain.
iii. Can pipelining be used? Explain
iv. Calculate the disk I/O cost of the two joins for the order you have chosen.

Justify all your answers!!

Part b: Do you think your solution to Part a is the best way to compute the pair of joins? If so, justify your answer *without* doing exhaustive calculations of the costs of other methods. If not, what method do you think is better? Justify your answer by sketching its cost.