Database Systems, CSCI 4380-01
Homework # 6
Due Monday April 18, 2016 at 11:59:59 PM

Introduction.

In this homework, you are allowed to work in groups of at most 2. If you do, please write the name of your group members in a README file included in your submission. Be careful: if you do not include group member names with your submission, we will not allow addition of names after you submit.

This homework is on indexing and secondary access methods. You will write a simple program (really a very simple one) to search through index and data pages to answer simple queries. We will simulate disk access by representing disk pages as individual files.

You are given the pages for the following database objects in separate directories:

Directory	Attributes	Notes
`actors_table`	`atype, id, name, surname`	tuples from `actors` table
`movieroles_table`	`actorid, info_1, info_2, movieid, role`	tuples from `movieroles` table
`actors_id_idx`	`id, pageid`	index on `actors(id)`, 400 key values max
`movieroles_ma_idx`	`movieid, actorid, pageid`	index on `movieroles(movieid,actorid)`, 360 key values max

Basically, each table is stored on a page with one tuple per line and each tuple as a comma separated list of values. The list of attributes is given in the table above.

Note that the index is stored in a slightly different format than we have covered in class, in particular how the index and key values are paired in nodes. Please use the course notes for answering questions in the exam. For now, read carefully the format for indices used in this homework which should make the logic for scanning the index simpler.

Index leaf nodes. All index nodes have a first line that tells whether the node is an internal node or a leaf node.

Each page of the index at the leaf level stores one line for indexed tuples. At each line, we have the key value for the indexed tuple and the link to the page id for the tuple containing given key value.

For example, the entry `43880,2` in a leaf node for `actors_id_idx` means that actor with id `43880` is stored in data page `2` (which is in file `actors_table/page2.txt`).

The second index `movieroles_ma_idx` is sorted by two atributes, so leaf level nodes will contain two attributes and a link to the page containing the given pair of values. Here are a few example entries from this index:

```
3300239,145986,2653
3300239,149008,2653
3300330,4532,2653
3300330,4934,2653
```

Each leaf node (except for the last node) has a pointer to the next index page (name of the file containing that node). This is the last line in the page for that node.

Index internal nodes and root. Both of the indices given are B-tree indices. Hence, upper level nodes are sparse indices that point to the level below.

The `actors_id_idx` index has only one level: a single `root.txt` node and the leaf nodes.

The second index has two levels: a single `root.txt` node, two internal nodes `int1.txt` and `int2.txt` and the leaf nodes.

Non-root nodes have a pointer pointing to the next sibling page. For example `int1.txt` points to `int2.txt`, while `int2.txt` does not point to any other node as the last node at that level.

All internal and root nodes point to the nodes in the level below with ranges of values. For example, for actors, the internal nodes contain tuples of the form:

```
100357,leaf1.txt
100718,leaf2.txt
```

which means that `leaf1.txt` page for this index contains tuples with `actorid<=100357`, and `left2.txt` page contains tuples with `100357<actorid<=100718`.

Problem Specification

Write a program using Python 2.7 or C++ (same requirements as the Data Structures class) that is assumed to run in the same directory as the `files` folder.

Your program can hard code a few things: (1) the directory location for the database objects, (2) the files for root nodes of indices and that which table each index points to and (3) the schema (list of attributes) for each object into your program. You can use whatever method you wish to hard code these. You will not be graded on the flexibility of your program, just correctness of the results and the accounting of query costs.

To scan the table, you can simply scan all the files in the directory for that table. To scan the index, you need to start from the root and follow pointers (resolving the location of the data pages).

Make sure your program runs as follows:

```
python hw6.py input_file_name
```

or

```
./hw6 input_file_name
```

and outputs the results on standard output. Each line of the input will be as follows:

```
movieid_low,movieid_high,actorid_low,actorid_high
```

indicating a search for a range of `movieid` and `actorid` values. The special symbol `*` stands for any value. Here are some examples:

movieid_low	movieid_high	actorid_low	actorid_high	notes
10	20	30	40	movies with id between 10-20 (inclusive) and actors with ids between 30-40 in those movies
10	10	*	*	all actors in movie id 10
10	10	30	*	all actors in movie id 10 with actor id 30 or higher.
*	10	*	*	all actors in a movie with id 10 or lower.

Given this search criteria, you are expected to find all the matching actor names and return them in the order you find them in the index for movieroles.

You must implement three possible methods to answer this query.

In the first method, you will scan first the `movieroles_ma_idx` index to find all actors, then `actors_id_idx` to find the names of these actors and print.

In the second method, you will scan first the `movieroles_ma_idx` index to find all actors, then scan the whole `actors_table` to find the names of these actors (if their id is in the set of actor ids found).

In the third method, you will scan both tables to answer the question, no indices. Scan `movieroles_table` first and then the `actors_table`.

After printing the output once, you must print the total number of disk pages (index and data) scanned for for both methods.

Then print a line of stars to distinguish between different inputs.

Example output for `2834638,2834638,147039,*` is given below (correctness is not yet guaranteed):

```
Query: 2834638,2834638,147039,*
Results:

    Sean (I) Cronin (actor)
    Josh Herdman (actor)
    Gemma Padley (actress)

Method 1 total cost: 13 pages

    3 pages movieroles_ma_idx
    1 page movieroles_table
    6 pages actors_id_idx
    3 pages actors_table

Method 2 total cost: 2108 pages

    3 pages movieroles_ma_idx
    1 page movieroles_table
    2104 pages actors_table
```

Deliverables

We will supply an example set of queries later this week.

Turn in a ZIP folder containing (at least) three files:

```
README.txt
  should contain the names of your group members
  and documentation of how to run your program, include
  all potential problems that you know exist with your program

filesforyourcode
  you are welcome to divide your code into multiple
  files if you wish, but document your code well so it is
  readable

example output
  the output for the example input file we give you to show
  what to expect from your program. please match our input format
  as closely as possible (but you do not have to match all the spacing
  etc.)
```