Carnegie Mellon University
15-826 – Multimedia Databases and Data Mining
Fall 2012, C. Faloutsos
Homework 1, Due Date: Oct 2nd, at noon

Reminders

- Deposit your work in Blackboard - see '**What to turn in**' instructions.
- All homeworks are to be done **INDIVIDUALLY**.
- Expected effort for this homework (approximate times):
  - Q1: 2-5 hours
    - 1-3 hours to write and debug all sql queries
    - 1-2 hours to run them queries and record the answers.
  - Q2: 4-7 hours
    - 30' to download and `make` the package
    - 3-6 hours to write and debug your algorithm
    - 30' - 1 hour to run your algorithm and answer questions
  - Q3: 4-9 hours
    - 1 hour to download and `make` the package
    - 3-7 hours to write and debug your algorithm
    - 1 hours to run your algorithm and answer questions

Q1 – DBMS and SQL [30pts]

Problem Description: For each question in this part, provide both the SQL statement(s) and the resulting answer(s), unless specified otherwise. You'll be using the *Marvel Universe* dataset from HW0, which lists the appearances of super-hero characters from the Marvel universe in comic books.
Hint: Please use SQLite3; version 3.6.20 is already on the andrew cluster machines. You may use your own machine and your own sqlite3 installation, as long as your submitted code runs correctly on the andrew cluster machines.

Implementation Details: Write sql code for the following queries.

1. **[1 pt]** *Loading:* Create and load the following three tables, using the text files provided. (Run `wc -l FILENAME` on each of them, to double-check the number of lines).
   - `characters(char_id, name)` (marvel_characters.txt [1], lines: 6,486)

[1] `www.cs.cmu.edu/~christos/courses/826.F12/HOMEWORKS/HW1/data/marvel_characters.txt`

- `comics(comic_id, name)` (marvel_comic_books.txt [2], lines: 12,942)
- `appearances(char_id, comic_id)` (marvel.txt [3], lines: 96,662)

2. **[2 pt]** *Cast size:* How many distinct Marvel Universe characters does this dataset contain?
3. **[2 pt]** *Library size:* How many distinct comic books does this dataset contain?
4. **[2 pt]** *Popularity:* Which comic book character is the most popular (appears in the most comic books)? Give his/her *name*.
5. **[2 pt]** *Large Cast:* Which comic book has the most characters appearing in it? Again, give its name.

   Definition: two distinct characters are defined as *co-actors* if they appear together in at least one comic book.
6. **[4 pt]** *Co-actors:* Create a view called `co-actors` that contains one row for each pair of co-actors in the dataset. The view should have the format `(id1 int, id2 int)`. Exclude self-pairs (`id1 != id2`), but include mirror pairs: *e.g.* if (1,3) is in the set, (3,1) should also be in the set.
7. **[8 pt]** *Most "social" characters:* List the names and co-actor counts, for the top 3 most "social" characters (that is, the characters that have many co-actors). Sort your answer most-social-first.
   Hint: You will need to join with the `characters` table, and use the SQL statements `GROUP BY`, `ORDER BY`, and `LIMIT`).
   [1 pt] *Report* the wall-clock running time of your query, using, e.g., the `time` Linux/Unix command.
8. **[2 pt]** Create indices on the columns `appearances.char_id`, `appearances.comic_id`, and `characters.char_id` (follow the syntax shown here).
   [2 pt] Run the query from the last question again ("most social characters"), and *report* the wall-clock running time.
9. **[2 pt]** *Show* the output of `explain select` for the previous query in the last two questions, i.e., with, and without indices.
   [2 pt] *Justify* the speed-up with the indices. *Hint:* See `http://www.sqlite.org/draft/eqp.html` for a (rough) description of the output of `explain`.

What to turn in:

- **Code:** A text file with name `hw1.q1.sql.txt`, with all the SQL queries and commands you need to answer the queries above.. Make sure it runs: we will grade it using
  `sqlite3 marvel.db < hw1.q1.sql.txt`
- **Answers:** A text file with your results and responses. Call it `hw1.q1.output.txt`; create it with
  `sqlite3 marvel.db < hw1.q1.sql.txt > hw1.q1.output.txt`.
  and append the wall-clock times and your justification to Q9 to this text file.

[2] `www.cs.cmu.edu/~christos/courses/826.F12/HOMEWORKS/HW1/data/marvel_comic_books.txt`
[3] `www.cs.cmu.edu/~christos/courses/826.F12/HOMEWORKS/HW1/data/marvel.txt`

Q2 – KD-Trees [30pts]

Problem Description: Your task is to add new functionality to an existing KD-Tree package, and specifically, to find the minimum bounding box of all its elements. That is, find the smallest n-dimensional bounding box that contains all the points in a KD-tree.

Set up

- Build the KD-Tree Package[4] (`tar -xvf kdtree_base.tar.gz; make`).
- Test it:
  `make hw1`
  should load the appropriate dataset, and call a place-holder function, which prints "*Algorithm not implemented*",

Your task is to replace this place-holder function with a working one.

Implementation Details: Implement a new command, 'b' (for 'b'ox), that prints out the minimum bounding box for the KD-tree. It should print out the low-end n-D vector and the high-end n-D vector of the bounding box. For example, for a 3-d case, the answer may be *(1,1,3) (5,8,19)*.
Run your code on 3 provided input files and report the minimum bounding box in each case. For your convenience, we have provided a suitable `makefile`:

- (a) Run your code on dataset 1 (using `make hw1_1`),
- (b) ..... on dataset 2 (using `make hw1_2`), and
- (c) ..... on dataset 3 (using `make hw1_3`).

What to turn in:

- **Code:** **[18 pt]** A tar ball file named `hw1.q2.tar.gz` with the code you implemented for the KD-tree. Make sure it compiles and runs because we will use it for grading (using: `make hw1_1`, etc)
- **Answers:** **[12 pt]** A text file named `hw1.q2.output.txt` with your the answers to the above three KD-trees.

Q3 – R-Trees [40pts]

Problem Description: The goal is to become familiar with the R-tree algorithms. Your task is to add new functionality to an existing R-Tree package.

[4] http://www.cs.cmu.edu/~christos/courses/826.F12/HOMEWORKS/HW1/kdtree.tar.gz

Set up Please build the R-Tree Package[5] (`tar -xvf drtree.tar.gz; make demo`). This creates the `bin/DRmain` program and runs it on some small datasets.
It has been tested on the Unix platform in the andrew machines - it most probably runs under mac-osx, and Cygwin on Windows.
Running

```
make hw1
```

should load the appropriate dataset, and print "*Algorithm not implemented*" message

Currently the R-tree package supports 's' for range search, 'i' for insertion etc.

Implementation Details: Using the input dataset given in `hw1.input` (which is called by default with `make hw1`), implement the following commands:

- '`l`' for 'print at (l)evel' : your program should print the coordinates ($x_{1,low}$, $x_{1,high}$, $x_{2,low}$, $x_{2,high}$, ...) of the MBRs of the pages at the specified level.
  The input format to the command is

  ```
  l LEVEL
  ```

  with the leaf nodes at `LEVEL` =0.
  - *Clarification 1*: The command "`l 0`" should print out all data rectangles.
  - *Clarification 2*: If the root is e.g. at level 3, the command "`l 3`" should print out the MBRs of the contents of the root page (about 5-100 rectangles).
  - *Clarification 3: In the same case (root is at level 3), the command "`l 4`" should print out an error.*
- '`c`' for '(c)ount' : your program should print the count of data rectangles in the R-tree.

What to turn in:

- **Code:** **[20 pt]** A tar ball file named `hw1.q3.tar.gz` with your code. We will grade it using the commands:

  ```
  tar -xvf hw1.q3.tar.gz; cd DRTree; make hw1;
  ```

  When typing in "`make hw1`", we should see the answers.
- **Answers:** **[20 pt]** The output of running "`make hw1`", in a file named `hw1.q3.output.txt`. They should be the count of the data rectangles and the printout of the MBRs at level 2.

[5] http://www.cs.cmu.edu/~christos/courses/826.F12/HOMEWORKS/HW1/drtree.tar.gz