Advertising on the Web

1 Project description

1.1 Project Overview

On-line advertising has been one of the most booming markets since the first on-line ads showed up on the web and now it is a market of hundreds of billions of dollars globally and it still has not lost the steam. Google got $32.2 billion in advertising revenue which is 97% of Google's total revenue in Q3 2010 - Q2 2011. By far the most lucrative venue for on-line advertising has been search, and much of the effectiveness of search advertising comes from the "AdWords" model of matching search queries to advertisements. For this project, you need to use Java to access an Oracle database and accomplish an on-line ads auction system using various bidding algorithms.

1.1.1 Definition of the AdWords Problem

We are going to consider on-line algorithms for solving the AdWords problem, which is defined as follows. Given:

- A set of bids by advertisers for search queries;
- A click-through rate for each advertiser;
- A budget for each advertiser;
- A limit on the number of ads to be displayed with each search query.

Respond to each search query with a set of advertisers such that:

- The size of the set is no larger than the limit on the number of ads per query.
- Each advertiser has bid on the search query.
- Each advertiser has enough budget left to pay for the ad if it is clicked upon.

The data set consists of three database tables: (primary keys are underlined)

```
Queries(qid:INTEGER, query:VARCHAR(400))
Advertisers(advertiserId:INTEGER, budget:FLOAT, ctc:FLOAT)
Keywords(advertiserId:INTEGER, keyword:VARCHAR(100), bid:FLOAT)
```

A *query* is a sequence of tokens typed on the search engine, e.g., "the best restaurant in Gainesville". One token may appear multiple times in one query. The click-through rate (*ctc*) is the number of times a click is made on the advertisement divided by the number of total impressions (the times an advertisement was served). To make the output unique, we assume the *ctc* is constant for each advertiser. Also we assume the first x of the 100 impressions will be clicked and the rest $100 - x$ impressions will not be clicked if the *ctc* is x%(x will be an integer). The same simulation process repeats for the ads' second 100 impressions, third 100 impressions and so on so forth. The `Advertisers` table stores the *advertiserId* and the corresponding *budget*. The *budget* is in the unit of dollars. The advertisers can't bid if the balance is less than the advertiser's bid. Each advertiser must provide a set of keywords that best describe the ads to match the search query. One advertiser has one and only one ads. The *bid* in the `Keywords` table is also in the unit of dollars.

1.2 Matching Bids and Search Queries

Each ads has a set of case insensitive keywords to match the query. Each keyword in the keyword set is associated with the price the advertiser would pay if the ads is clicked. The advertiser will bid the query with the sum of the price for each matched keywords. You may need to tokenize the search query. For simplicity,the project assumes the delimiter to tokenize the query is the empty space ' '.
$T \leftarrow matched\_tokens\_set(query, keywords)$; T contains no duplicate tokens
$bid(keywords, query) = \sum T_i \times bid_i$

1.3 On-line Auction Algorithms to the AdWords Problem

One advertiser bid one query *iff* the $bid > 0$ else the advertiser will not go into the auction process. Assuming that only K ads can be displayed for each query. It is also possible that the K slots are not fulfilled if there are not enough advertiser to bid the query.

1.3.1 The Greedy Algorithm

For each query, the greedy algorithm picks advertisers who has the highest bid for it. The Top-K advertisers who bid the query and have the highest bids will be shown.
Ad Rank = Bid

1.3.2 The Balance Algorithm

There is a simple improvement to the greedy algorithm. This algorithm, called the Balance Algorithm, assigns a query to the Top-K advertisers who bid on the query and have the largest balance.
Ad Rank = Balance

1.3.3 The Generalized Balance Algorithm

Suppose that a query q arrives, advertiser A_i has bid x_i for this query. Also, suppose that fraction f_i of the budget of A_i is currently unspent. Let $\Psi_i = x_i(1 - e^{-f_i})$. Then assign q to the Top-K advertisers who bid the query and have the highest Ψ_i.
Ad Rank = Ψ

1.3.4 Add the Ads Quality into Ranking

The above auction algorithms don't take the quality score into consideration which is a metric to determine how relevant and useful your ad is to the user. The higher your quality score is, the better. In this project, we take the Ads quality into consideration. we define the quality score as the product of ctc and cosine similarity between query and advertiser keywords: The attribute vectors A and B are the term frequency vectors of the query and the advertiser keywords.

$QualityScore = ctc * similarity$ where similarity $= \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$

Click Here to show an example of how to calculate cosine distance of two texts. The section 1.3.1, 1.3.2, 1.3.3 is just a general description of these algorithms. In our project, we need to implement a revised version of these algorithms which take the quality score into ranking. Finally, the Ad rank is redefined as follows:

The Greedy Algorithm Ad Rank = Bid × Quality Score

The Balance Algorithm Ad Rank = Balance × Quality Score

The Generalized Balance Algorithm Ad Rank = $\Psi\times$ Quality Score

1.4 Charging Advertisers for Clicks:

The advertiser will be charged *iff* the ads was clicked.

first-price auction when a user clicks on an advertiser's ad, the advertiser is charged the amount they bid. This policy is known as a first-price auction.

second-price auction In reality, search engines use a more complicated system known as a second-price auction. An advertiser for a search will pay the bid of the next highest bid advertiser who bid the query. The next highest bid is the bid which is smaller than the current advertiser's bid but closest to the current advertiser's bid. If the advertiser's bid itself is the smallest, it will pay the price it bids in this case. It has been shown that second-price auctions are less susceptible to being gamed by advertisers than first-price auctions and lead to higher revenues for the search engine.

1.5 Tasks

There are 6 tasks you need to finish in this project:

- Task 1: the greedy algorithm and the first price auction.
- Task 2: the greedy algorithm and the second price auction.
- Task 3: the balance algorithm and the first price auction.
- Task 4: the balance algorithm and the second price auction.
- Task 5: the generalized balance algorithm and the first price auction.
- Task 6: the generalized balance algorithm and the second price auction.

2 Input and Output

You must exactly follow the input and output format described in the following subsections. If you fail to follow the format, your program may not be able to parse the input file and the TAs' grading script may not be able to interpret your output. Also the input files, output files and your code files should be in the same directory. Don't add any extra file path except the file name to the path argument when program tries to read/write some files. For the purpose of automated grading, the TAs intend to make the output unique. Let us know if any issue in the project that would to lead nondeterministic results.

2.1 Input

There are four input files, namely Queries.dat, Advertisers.dat, Keywords.dat and system.in. We also provide sample inputs and it may not be the same as the final dataset to test your code. The system.in contains one and only one test case for each task. The first line in the system.in contains the your credential to access the oracle database. I will replace your system.in with my system.in to test your code. That means I will use my cise oracle account to test your code and you can assume nothing is already in the database. The following lines contains the corresponding parameters. *num_ads* (K in above context) is the maximum number of ads can be shown per query.

```
username = myusername
password = mypassword
TASK1: num_ads = 4
TASK2: num_ads = 3
TASK3: num_ads = 2
TASK4: num_ads = 5
TASK5: num_ads = 2
TASK6: num_ads = 3
```

2.2 Output

There is one output file for each task. Your program should generate six output files, namely, system.out.1,system.out.2,system.out.3,system.out.4,system.out.5 and system.out.6. The rank i is the i_{th} slot in the ads section. The rank starts from 1. The balance is the amount of money left *after* this ads impression. The budget is same as what defined in table `Advertisers`.
Sample system.out.1 for TASK1:

```
qid, rank, advertiserId, balance, budget
```

Sample system.out.2 for TASK2:

```
qid, rank, advertiserId, balance, budget
```

Sample system.out.3 for TASK3:

```
qid, rank, advertiserId, balance, budget
```

Sample system.out.4 for TASK4:

```
qid, rank, advertiserId, balance, budget
```

Sample system.out.5 for TASK5:

```
qid, rank, advertiserId, balance, budget
```

Sample system.out.6 for TASK6:

```
qid, rank, advertiserId, balance, budget
```

2.3 How to Break Ties

To make the output unique, you *must* break the tie according to the *advertiserId* in ascending order whenever necessary, which means the smaller the *advertiserId*, the higher the *rank*(rank 1 is the highest).

2.4 Project Report

You are required to submit a 1-2 page TXT report. Name it report.txt. Note the small case 'r' and NOT capital 'R'. Your report must contain a brief description telling us what are the steps you took to develop the code, what difficulties you faced, how you solved them for this project and what you learned from this project. Remember this has to be a simple text file and not a MSWORD or PDF file. It is recommended that you use vi or gedit or pico/nano applications in UNIX/Linux to develop your report.

2.5 Additional Instructions

1. You should tar all you java code files including the txt report in a single tar file using this command on Linux/SunOS:
   tar cvf adwords.tar <file list to compress and add to archive>

2. Your project must be named 'adwords.tar' and also your java file which contains the main function must be named 'adwords.java'.

3 Automatic Testing Script

The TAs will run the following script to grade your program on the CISE Linux machines. **Please do your own implementation. We will be checking similarities between the solutions.** Sample input and output files will be provided. The grading test cases will not be disclosed until the grades are released.

```
#!/bin/bash
source /usr/local/etc/ora11.csh
tar xvf adwords.tar;
./plagiarism-detector
rm *.class system.out.*
```

```
javac -cp ojdbc5.jar adwords.java
cp /path/to/*.dat .\
cp /path/to/correct.system.out-* .\
java -cp ojdbc5.jar:. adwords
java autograder
cat report.txt
```

4 Grading Guidelines:

The core logic of this project *must* be implemented using SQL. Java can only be used to process the inputs/outputs and drive the compuation as a driver function. You can't use Java data structures. Array can be used to store the data in system.in. The project which violates this policy will be graded below 60. It is your responsibility to make sure you follow the input and output format. Your project will be graded 0 if either your program cannot read the input files provided by the TAs or the TAs' autograder cannot parse your output files.

Task	1	2	3	4	5	6	Report
Grade(%)	10	10	10	10	10	10	40

5 Late Submission Policy:

Late submissions are not encouraged but we rather have you submit a working version than submit a version that is broken or worse not submit at all. Every late submission will be penalized 10% for each day late for up to a maximum of 3 days from the due date.

6 Resources

6.1 Accessing the CISE Oracle database

You need to create a CISE Oracle account to access the CISE Oracle database. Please refer to the following CISE website:
`http://www.cise.ufl.edu/help/database/oracle.shtml`
However some information in the above website is not up to date. The location for the jdbc driver is no longer valid.
Note: To use Oracle's JDBC on Department Solaris machines, add the following to your CLASSPATH: /usr/local/libexec/oracle/app/oracle/product/11.2.0/client_1/ojdbc5.jar.
The solution for you is just downloading the attached 'ojdbc5.jar' and put it into your source code.

6.2 How to execute sql script in Java

You may need to write procedures to implement the core part of this project. The following two lines of code enable you to import sql script containing procedure code to the database.

```
Process p = Runtime.getRuntime().exec("sqlplus username@orcl/passwd
   @adwords.sql");
p.waitFor();
```

Please append `exit;` to the end of your procedure code file `adwords.sql`. This line of code allows you to exit the sqlplus environment.

6.3 Tutorials

You can have a look at the following tutorials that might be helpful in doing this project:

1. `http://i.stanford.edu/~ullman/mmds/ch8.pdf`
2. `http://www.wordstream.com/articles/what-is-google-adwords`
3. `http://infolab.stanford.edu/~ullman/fcdb/oracle/or-jdbc.html`
4. `http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_jdbc.htm`
5. JDBC is covered a bit in your textbook, there are tons of books on JDBC at UF libraries, and Google will get you more info on JDBC than you ever wanted to know.

6.4 Questions and Answers

This section will list the questions from you and the TA's clarifications. Please let the TAs know *ASAP* if you find any ambiguity or any issue which may lead to non-unique outputs. Bonus points are possbile!

- Coming soon...