CS144: Web Applications

Project Part 2: Data Conversion and Loading
Due Friday, January 29, 2016 11:00PM

- **Submission deadline:** Programming work is submitted electronically and must be submitted by Friday at 11:00PM. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your sumission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.

- **Late Policy:** Programming work submitted after the deadline but less than 24 hours late (i.e., by Saturday 11:00 PM) will be accepted but penalized 25%, and programming work submitted more than 24 hours but less than 48 hours late (i.e., by Sunday 11:00 PM) will be penalized 50%. No programming work will be accepted more than 48 hours late. Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but no single assignment may be more than two days late.

- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under Honor Code. In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.

Overview

In this project, you will have to (1) parse XML data and store them into a relational database system and (2) create an two-column Web page with a header and footer using CSS.

In Parts A through E, you will be given a large volume of data downloaded from the eBay web site and stored in XML files. You will examine the data and design a good relational schema for it. You will then write a Java program to transform the data from its XML form into MySQL's load file format, conforming to your relational schema. You will create your schema in your own MySQL database, load your transformed data, and test it by running some SQL queries. **Note that you won't be able to continue to Project 3 unless you complete this project. It is important that you finish this project on time to have enough time to do Project 3.**

By completing this project you will be able to learn:

1. How to parse an XML file in Java
2. How to place a Java class into a package
3. How to use Ant to help Java code compilation and execution
4. How to write a Unix shell script

We will be providing relevant tutorials and references to help you learn the above topics.

Part A: Download and examine the XML data files

First, download the XML-encoded ebay auction data set ebay-data.zip for this project and unzip the file in the `$EBAY_DATA` directory (`/home/cs144/ebay-data`) in our VM like the following:

```
$ wget http://www.cs.ucla.edu/classes/winter15/cs144/projects/project2/ebay-data.zip
$ unzip -d $EBAY_DATA ebay-data.zip
$ rm ebay-data.zip
```

1. As a *small data set* for initial experimentation and debugging we suggest you use just one file: `items-0.xml`. It contains 500 auctions, comprising about 900kb of plain-text data.

2. Your system also must work on the *large data set*, which consists of all 40 files: `items-*.xml`, for * = 0..39. There are a total of about 20,000 auctions, comprising about 38 megabytes of plain-text data.

Each XML data file is valid with respect to the XML DTD specified in the file `items.dtd`.

Your first task is to examine the DTD and the XML files to completely understand the data you will be starting with. You will be translating this data into relations and loading it into MySQL server on our VM. Please read the auction data help file in `items.txt`. The provided data was captured at a single point in time. (Very specifically, it represents the point in time December 20th, 2001, one second after midnight.) It contains items that have been auctioned off in the past and items that are "currently" up for auction.

Note that UserIDs of sellers and bidders uniquely identify a user. Whenever one user appears in the data, his/her Rating, Location, Country information are the same. For more, read items.txt in the dataset.

Part B: Design your relational schema

Now that you understand the data you'll be working with, design a good relational schema for it. Iterating through the following three steps should help you in the design process.

1. List your relations. Please specify all keys that hold on each relation. You need not specify attribute types at this stage.

2. List all completely nontrivial functional dependencies that hold on each relation, excluding those that effectively specify keys.

3. Are all of your relations in Boyce-Codd Normal Form (BCNF)? If not, either redesign them and start over, or explain why you feel it is advantageous to use non-BCNF relations.

4. Are all of your relations in Fourth Normal Form (4NF)? If not, either redesign them and start over, or explain why you feel it is advantageous to use non-4NF relations.

Document these steps in the `README.txt` file for submission; see **What to submit** below.

Part C: Write a data transformation program

Next you will write a program that transforms the XML data into MySQL load files that are consistent with the relational schema you settled on in Part B. Remember that our MySQL tutorial linked in Project 1 had a brief explanation on the MySQL LOAD command and the load file format. If necessary, read MySQL LOAD command reference to learn more details on MySQL load file.

To help you get started, we are providing a parser skeleton sample codes, `MyParser.java` and `MyParserPrint.java`, in the `src` directory of the project2.zip file. Note that the two Java classes are defined as part of the `edu.ucla.cs.cs144` package, which is why `MyParser.java` and `MyParserPrint.java` files are located in the `src/edu/ucla/cs/cs144` directory (`src` is the base directory of all Java code and the rest corresponds to the package name hierarchy). If you are not clear about what is a package and how a package name is related to the source directory structure, read Section 1.8 of A Crash Course from C++ to Java.

The provided two Java files are almost identical, except that `MyParserPrint.java` has additional code to print out the parsed XML data to screen. In the sample codes, all of the parsing is done for you by invoking the JDK's XML parser (JAXP). You need to fill in code that processes the internal representation of the XML tree and produces MySQL load files according to the relational schema you designed in Part B. Detailed information is provided as comments in the skeleton code file `MyParser.java` in the zip file.

The zip file also contains a sample Ant build script, `build.xml`. Ant is a "build tool" similar to `make` but it is specifically designed for cross-platform Java development. You must read the following Ant tutorial to learn how to use Ant and write and modify an Ant script file (we will use this tool throuout the rest of the quarter):

- A Short Tutorial on Ant

Since Ant is preinstalled in our VM, you will be able to compile your Java code simply by typing `ant` in your command line (inside the directory where `build.xml` is).

Note that the provided Ant script have targets "run" and "run-all". The target "run" executes the parser class `edu.ucla.cs.cs144.MyParser` on the single data file `$EBAY_DATA/items-0.xml` and the target "run-all" executes the class on all files at `$EBAY_DATA/items-*.xml`. You can use one of these two targets to test your code on the provided ebay data. *We strongly suggest that you fully debug your program on the small data set before unleashing it on the large one.*

Notes on MySQL Datatypes

We note that some of the `Description` elements in the XML data are quite long. If any text is longer than 4000 characters, you must **truncate it at 4000**. In particular, we suggest that you use the MySQL data type `VARCHAR(4000)` to store descriptions and during data transformation simply truncate any descriptions that will be too long to fit. For attributes representing dates/times, we suggest that you use MySQL's built-in `TIMESTAMP` type. For attributes representing money, we suggest you use `DECIMAL(8,2)` to specify a numeric type with 2 digits after the decimal point.

Note that in creating MySQL load files, you may have to reformat the input data to the one that MySQL expects. Information on the format of SQL `TIMESTAMP` type is available on the MySQL Dates and Times page. You can use the Java class `java.text.SimpleDateFormat` to parse a date/time string in a particular format and to output it in a different format. See the example code on this page to learn how to use `SimpleDateFormat.parse()` and `SimpleDateFormat.format()` methods to parse and format date/time strings in Java.

Duplicate elimination

When transforming the XML data to relational tuples you may discover that you generate certain tuples multiple times but only want to retain one copy. There are different ways to eliminate duplicates, including:

- Coding duplicate-elimination as part of your transformation program.
- Using Unix text utilities (e.g., sort and uniq) directly on the generated load files to eliminate duplicate lines. These tools are preinstalled in our VM.

You may *NOT* rely on the MySQL bulk loader to eliminate duplicates. MySQL will generate an error but continue loading when a tuple with a duplicate value in a key attribute is encountered. However, using this "feature" to eliminate duplicates is an unaesthetic hack that also complicates our grading procedure, and it should be avoided.

Notes on CR/LF issue: If your host OS is Windows, you need to pay particular attention to how each line of a text file ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux, Mac OS X, and tools in cygwin) use only a LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a Unix tool (such as `sort`, `uniq` and `mysql`). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected behavior from these tools. If you encounter this problem, you may want to run the `dos2unix` command in VM on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Part D: Load your data into MySQL

Now create and populate your table(s) in MySQL inside the `CS144` database. We suggest that you first debug the schema creation and loading process interactively using the MySQL command-line interface. When you are satisfied that everything is working, follow the instructions to set up for batch loading (see **Section D.3** below), which allows a database to be conveniently recreated from scratch with one command.

D.1 Creating your databases interactively

Using the command line interface to MySQL, issue a set of "`CREATE TABLE`" commands for all of the relations in your schema. Some suggestions:

- For efficiency we suggest that you specify a `PRIMARY KEY` for each table that has at least one key.

Once the tables are created, load your data in MySQL.

D.2 Maintaining your databases

You should consider two factors in the maintenance of your databases throughout the CS144 project:

1. Your MySQL databases are probably not backed up, so anything you need long-term should be saved on the file system as well.

2. As you start modifying data in a database, you will undoubtedly want the ability to get a "fresh start" easily from your original data.

We recommend that you establish a routine of saving all data in MySQL load files, and perhaps reloading the database each time you start working with it. Remember that you need to delete the contents of each table (or destroy and recreate the tables) before reloading. Otherwise, MySQL will happily append the new data to your old table, causing your table size to double, triple, quadruple, etc. To get rid of a table called T, you can use `DROP TABLE` command as follows:

```
DROP TABLE T;
```

If you want to get rid of all tuples in `T` without deleting the table itself, you can issue the command:

```
DELETE FROM T;
```

D.3 Automating the process

As we explained in our MySQL tutorial, MySQL provides a facility for reading a set of commands from a file and executing them in batch mode. You should use this facility for creating/(re)building databases and running sample queries, and you must use it extensively in your submitted work. (See **What to submit** below.) For starters, create a command file called `create.sql` that includes the SQL commands for database and table creation that you debugged interactively (recall **Section D.1**). This file will look something like:

```
CREATE TABLE Item ( .... );
CREATE TABLE AuctionUser ( ... );
...
```

Similarly, create a second command file called `drop.sql` that destroys all your tables. This file will look something like:

```
DROP TABLE Item;
DROP TABLE AuctionUser;
...
```

Lastly, create a command file called `load.sql` that contains the appropriate MySQL commands to load data from a file into a table. The file will look something like:

```
LOAD DATA LOCAL INFILE 'Item.dat' INTO TABLE Item ...
LOAD DATA LOCAL INFILE 'AuctionUser.dat' INTO TABLE AuctionUser ...
...
```

To automate the entire process, create a *bash shell script* called `runLoad.sh`. If you do not know what a shell script is or how to create one, first read our Tutorial on Shell Script. Your script should do the following things.

1. Drop any existing relevant tables in `CS144` database **IF EXISTS**.

2. Create all the relevant tables in `CS144` database.

3. Build and run your parser to generate fresh load files.

4. Load the data into MySQL.

5. Delete all temporary files (including your load file) that your parser generated

Here is a skeleton of what this script can look like (as a bash script) when it includes all four steps:

```
#!/bin/bash

# Run the drop.sql batch file to drop existing tables
# Inside the drop.sql, you sould check whether the table exists. Drop them ONLY if they exists.
mysql CS144 < drop.sql

# Run the create.sql batch file to create the database and tables
mysql CS144 < create.sql

# Compile and run the parser to generate the appropriate load files
ant
ant run-all
...

# If the Java code does not handle duplicate removal, do this now
sort ...
...

# Run the load.sql batch file to load the data
mysql CS144 < load.sql

# Remove all temporary files
rm ...
...
```

In your final submission, please **make sure that the ant target "run-all" takes all data at `$EBAY_DATA/items-*.xml` and produces the output. The value of `$EBAY_DATA` should not be hard-coded in your ant script.** It should be obtained from the environment variable. Our provided `build.xml` follows this requirement.

To run this script, make sure permissions are set properly by executing "`chmod u+x runLoad.sh`", then simply type "`./runLoad.sh`" at the shell prompt to run it. Again, **pay attention to the CR/LF issue if you modify your scripts using a Windows program. Run `dos2unix` in VM on the scripts if necessary.** Otherwise, you may get unexpected errors when you run your shell script.

Part E: Test your MySQL database

The final step is to take your newly loaded database for a "test drive" by running a few SQL queries over it. As with your initial database creation, use MySQL to test your queries interactively, then set up a MySQL script file, named `queries.sql`, to be able to run them in batch mode later. First try some simple queries over one relation, then more complex queries involving joins and aggregation. Make sure the results look correct. When you are confident that everything is correct, write SQL queries for the following specific tasks:

1. Find the number of users in the database.

2. Find the number of items in "New York", (i.e., items whose location is *exactly* the string "New York"). *Pay special attention to case sensitivity. You should match the items in "New York" but not in "new york".*

3. Find the number of auctions belonging to exactly four categories.

4. Find the ID(s) of current (unsold) auction(s) with the highest bid. *Remember that the data was captured at the point in time December 20th, 2001, one second after midnight, so you can use this time point to decide which auction(s) are current. Pay special attention to the current auctions without any bid.*

5. Find the number of sellers whose rating is higher than 1000.

6. Find the number of users who are both sellers and bidders.

7. Find the number of categories that include at least one item with a bid of more than $100.

Correctness check: Your answers to the above seven queries over the large data set should be (in order): 13422, 103, 8365, 1046740686, 3130, 6717, 150

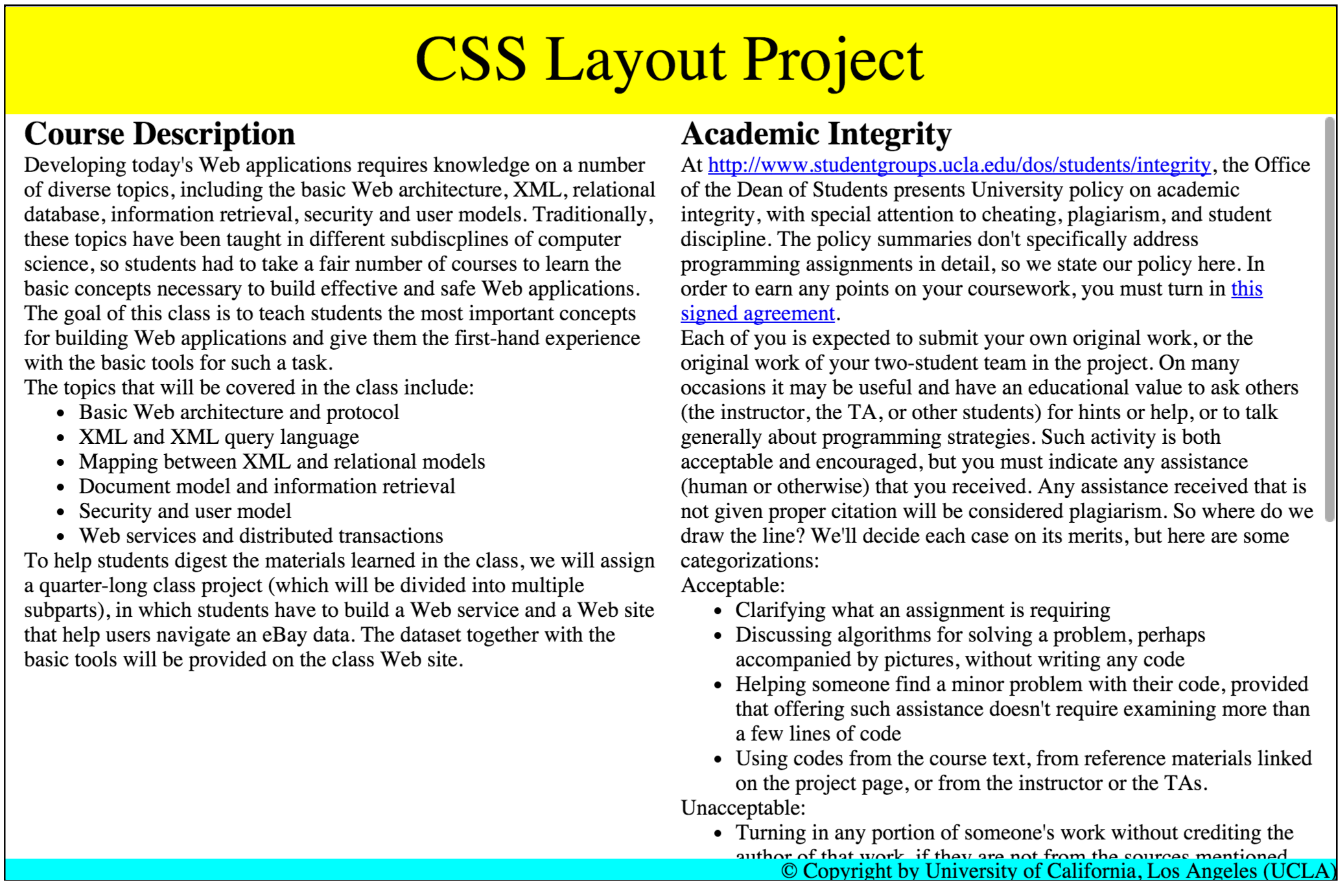
Put all seven queries in a MySQL batch script file called `queries.sql`. It should look like:

```
SELECT ... FROM ... ;
SELECT ... FROM ... ;
...
```

Make sure that the order of your queries in the file is exactly the same as the order specified above.

Part F: CSS Layout

As the last part of Project 2, you will practice on your CSS skill by creating a 2-column web page with a header and footer. After everything is one, your page should roughly look like the following screenshot:



CSS Layout Project

Course Description

Developing today's Web applications requires knowledge on a number of diverse topics, including the basic Web architecture, XML, relational database, information retrieval, security and user models. Traditionally, these topics have been taught in different subdiscplines of computer science, so students had to take a fair number of courses to learn the basic concepts necessary to build effective and safe Web applications. The goal of this class is to teach students the most important concepts for building Web applications and give them the first-hand experience with the basic tools for such a task.
The topics that will be covered in the class include:

- Basic Web architecture and protocol
- XML and XML query language
- Mapping between XML and relational models
- Document model and information retrieval
- Security and user model
- Web services and distributed transactions

To help students digest the materials learned in the class, we will assign a quarter-long class project (which will be divided into multiple subparts), in which students have to build a Web service and a Web site that help users navigate an eBay data. The dataset together with the basic tools will be provided on the class Web site.

Academic Integrity

At http://www.studentgroups.ucla.edu/dos/students/integrity, the Office of the Dean of Students presents University policy on academic integrity, with special attention to cheating, plagiarism, and student discipline. The policy summaries don't specifically address programming assignments in detail, so we state our policy here. In order to earn any points on your coursework, you must turn in this signed agreement.
Each of you is expected to submit your own original work, or the original work of your two-student team in the project. On many occasions it may be useful and have an educational value to ask others (the instructor, the TA, or other students) for hints or help, or to talk generally about programming strategies. Such activity is both acceptable and encouraged, but you must indicate any assistance (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered plagiarism. So where do we draw the line? We'll decide each case on its merits, but here are some categorizations:
Acceptable:

- Clarifying what an assignment is requiring
- Discussing algorithms for solving a problem, perhaps accompanied by pictures, without writing any code
- Helping someone find a minor problem with their code, provided that offering such assistance doesn't require examining more than a few lines of code
- Using codes from the course text, from reference materials linked on the project page, or from the instructor or the TAs.

Unacceptable:

- Turning in any portion of someone's work without crediting the author of that work, if they are not from the sources mentioned

© Copyright by University of California, Los Angeles (UCLA)

More precisely, the page should have the following parts:

1. It has a header of height 80 pixels, yellow background, and font size 35 points. The header has the text "CSS Layout Project" located at the center, both horizontally and vertically.
2. It has a footer of height 1em and cyan background with the text "© Copyright by University of California, Los Angeles (UCLA)" aligned to the right.
3. The content of the page is displayed in two columns. The first column has this content and the second column has this content.
4. The widths of the header, footer and the content area stretch to fill the window.
5. The two columns of the content area split the window horizontally half and half. There are margins of length 1em on the left, right and in-between the columns.

6. The header and the footer is always displayed at the top and bottom of the window. The height of the content area is adjusted according to the window size.
7. If the text in either column overflows, a scroll bar should be displayed on the right end of the window. *Both* columns should move *together at one unit* when this scroll bar is manipulated.

The Web page should be named as css-layout.html. The CSS rules should be part of the HTML page itself, enclosed by the `<style>` tag.

In order to get full credit, your page must be displayed correctly within the most recent version of Google Chrome browser.

What to Submit

You should submit a single file **project2.zip** that has the following packaging structure.

```
project2.zip
 |
 +- team.txt
 |
 +- css-layout.html
 |
 +- build.xml
 |
 +- create.sql
 |
 +- drop.sql
 |
 +- load.sql
 |
 +- queries.sql
 |
 +- runLoad.sh
 |
 +- README.txt
 |
 +- src
 |   +- java source codes for parser (with your own naming/structure)
 |
 +- lib
     +- external java libraries used
```

Each file or directory is as following:

- **`team.txt`**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
  If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**
- **`css-layout.html`**: The HTML page that you created for Part F. Your page must be displayed correctly by **the most recent version of Google Chrome Browser**.
- Your MySQL batch files **`create.sql`, `drop.sql`, `load.sql`**, and **`queries.sql`**. *Do not submit the output produced by running these files.*
- A plaintext file named **`README.txt`**, containing your relational schema design, answers to the 3 items listed in Part B, as well as anything else you'd like to document.
- Your ant **`build.xml`** script. Your `build.xml` script **MUST** meet the following requirements:
  - The default target must compile your Java files.
  - The target "run-all" must invoke your parser and pass all files at `$EBAY_DATA/items-*.xml` to your parser as the parameter. The value of `$EBAY_DATA` MUST NOT be hard-coded. It should be obtained from the environment variable.
- Your **`runLoad.sh`** script developed in Section D.3 above. Your `runLoad.sh` script **MUST** meet the following

requirements:
  - Your script must delete all your temporary files (including your load file) at the end.
  - All tables must be created in `CS144` database.
  - Your script must use the default user account `cs144` to connect to MySQL.
  - Your script must contain ONLY relative paths. DO NOT use any absolute paths in your script.
- `src` directory: Your parser source code file(s). *Do not submit the binary version of your parser.*
- `lib` directory: Any external Java library files (.jar files) that you use and are not available in our VM.

Important Notes:

- It is an **EXTREMELY GOOD IDEA** to test your parser compilation and invokation steps and ensure that they are self-sufficient (and that you are not relying on anything that have been installed by you separately).
- Please do NOT submit the output of your parser code and scripts, the binary version of your parser, or the eBay data.
- As always, remember to allow sufficient time to prepare your submission once your work is complete.

Testing of Your Submission

In order to help you ensure the correct packaging of your submission, we have made a "grading script" p2_test available. In essence, the grading script unzips your submission to a temporary directory and executes your files to test whether they are likely to run OK on the grader's machine. Download the grading script and execute it in the VM like:

```
cs144@class-vm:~$ ./p2_test project2.zip
```

(if your project2.zip file is not located in the current directory, you need to add the path to the zip file before project2.zip. You may need to use "chmod +x p2_test" if there is a permission error.)

You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. When evertything runs properly, you will see an output similar to the following from the grading script:

```
Running your runLoad.sh script...
   ...
   <output from your runLoad.sh>
   ...
Running your query.sql script...
COUNT(*)
13422
COUNT(*)
103
COUNT(*)
8365
ITEMID
1046740686
COUNT(*)
3130
COUNT(*)
6717
COUNT(DISTINCT CATEGORY)
150


Finished running your queries.
Please make sure that you got correct results from your queries
```

Submitting your files

Once your submission zip file is properly created and passes the grading script, submit it via our submission page at CCLE. You may resubmit as many times as you like, however only the latest submission will be saved, and those are what we will use for grading your work and determining late penalties. *Submissions via email will not be accepted.*

Grading Criteria

Breakdown

- Relation schema desigin (10%)
- Transformation runs without error: java compilation, import file generation, database load are done without any error (20%)
- Transformation correctness: testing with queries.sql, using various XML data (50%)
- CSS Layout(20%)

Correctness vs Efficiency

We grade your project based on the "correctness" of your work. Efficiency of your work is not a primary concern, but your codes MUST complete in a "reasonable" time:

- Project 2 (including answering 7 queries) should complete in 20~30 seconds. No penalty if your work completes in 2 minutes; after that, 20% off of your entire project score. Your code will be considered as a failed work if it takes 5 minutes or more.