Mini-Project 2

Design Document

CMPUT 291

Ryan Furrer (rfurrer)

Tim Tran (ttran1)

November 26, 2018

Ryan Furrer (rfurrer)
Tim Tran (ttran1)

1. General Overview

1.1 Introduction

Mini-project 2 is a command line based python program that allows a user to input ads from kijiji stored in xml format into a database. Then the user can search for different ads based on keywords, dates locations, categories and prices. The system will respond by displaying found ads in the terminal.

1.2 User Guide

- To use the program, enter mini_project_2 directory from the terminal.
- Phase 1
    - Phase 1 generates text files to be inputted into a berkeley database, from an xml file containing kijiji ads.
    - To run phase 1, enter the command “python3 __main__.py --phase 1 -i *inputfile”*
        - where *inputfile* is the xml file containing the kijiji ads.
- Phase 2
    - Phase 2 allows loads up the berkeley database with files generated in phase 1.
    - To run phase 2, enter the command “python3 __main__.py --phase 2”
- Phase 3
    - Phase 3 allows the user to query for ads.
    - To start phase 3, use the command “python3 __main__.py --phase 3”
    - Then, you may enter a query. Queries are made up of term, date, location, category and price searches. Queries can be made up of a single search criteria, or a combination, by separating the different criteria by spaces.
        - Term searches consist of a single word. Words can consist of any alphanumeric characters as well as - (dash) and _ (underscore).
        - Date searches are formatted as such:
            - date <operator> <date_value>
                - Where operator is one of “=”, “<=”, “>=”, “<” or “>” and date_value is a date in the format “YYYY/MM/DD”
        - Location searches are formatted as:
            - location = <location_value>
                - where location_value is the name of a location
        - Category searches are formatted as:
            - cat = <category_value>
                - where category_value is the name of the desire category
        - Price searches are formatted as:

- price <operator> <price_value>
    - Where operator is one of “=”, “<=”, “>=”, “<” or “>” and price_value is a price as any number (do not include $).

2. Detailed Design

2.1 Phase 1

phase1.py

This module contains functions that will properly generate the ads.txt, prices.txt, pdates.txt and terms.txt, given a file in xml format.

2.2 Phase 2

phase2.py

This module contains functions that properly load the index files by making use of linux system commands.

2.3 Phase 3

class InputParser

This class parses and validates the user input. Then it will return a dictionary with search types (see below) as keys and search operators and values of that search type as tuples in a list as the key. I.e. {searchtype1: [(operator1, value1), (operator2, value2)], searchtype2: …}. The class also contains a method to check if the user inputted “output=brief” or “output=full”

search types

A string determining the type of search to be made. Types can be either “price”, “date”, “keyword”, “location”, “category”.:

search operators

A string of either “=”, “>=”, “>”, “<”, “<=”, “%”. This indicates whether the search should be an equality search, a range search, or a likeness search (‘%’)

search values

A string indicating the thing to search for. E.g. “2018/11/02” or “Edmonton” or “camera”

Example:

Query: *camera date>=2018/11/05 date<=2018/11/07 price > 20 price < 40*

InputParser() will parse the above query and return a dictionary called searches:

- searches[“keyword”][0] = (“=”, “camera”)
- searches[“date”][0] = (“>=”, “2018/11/05”)
- searches[“date”][1] = (“<=”, “2018/11/07”)
- searches[“price”][0] = (“>”, “20”)
- searches[“price”][1] = (“<”, “40”)

Query execution with class AdsDatabase

AdsDatabase tries to execute as little of the query as possible.

It first gets matching ad ids for price, location and category if a price is provided. Multiple price criteria are evaluated and only the most limiting are used. It searches the pr.idx for the lower bounds (which is the first entry if no lower bound) and iterates until the upper bounds is past (may be EOF), only adding to results if the price, location and category filters match. If there are no results it exits.
It then gets matching ad ids for date, location and category if a date is provided. Multiple date criteria are evaluated and only the most limiting are used. It searches the da.idx for the lower bounds and iterates until the upper bounds is past (may be EOF), only adding to results if the date, location and category filters match. If there are no results it exits.
If there were price and date criteria it intersects them here, with time complexity `O(min(len(price_aids), len(date_aids))`. If there are no results it exits.
If there are term criteria, it searches for the term or if an entry starts with the term and returns results.
If there were previous results it intersects them here. If there are no results it exits.
If results, it then looks in ad.inx for the aids and if there were no “price” or “date” criteria and filters any “location” or “category” criteria. If not filtered it prints the result based on mode.
If no price, date, or term criteria, it just goes through ad.inx and prints the ad based on mode if it matches any “location” or “category” criteria.

3. Testing Strategy

- Each of the classes and major methods will be unit tested using pytest
- Also test queries were made and their outputs were analyzed

4. Assumptions Made

- Ad ids for a query will fit in main memory when the user enters a date, price and/or keyword
- Queries are case insensitive
- Prices are under 13 digits long
- Idx file creation will be run on lab machines

4. Group Work Break-Down

We primarily coordinated through GitHub, text message and in person meetings. An organization was set up for the project (CMPUT291F18MP2) and each task or bug was turned into an issue ticket. To measure progress, and when an issue was done, the assigned member would close the issue.

- Ryan Furrer did phase 1 and the phase 3 index data searching and printing.
    - Time: 10 hrs
- Tim Tran did phase 2 and the phase 3 user input parsing.
    - Time: 8 hrs