SI507 Final

Project Document

Find the Best Restaurant in a City

Author: Jingbo Li

December 12, 2022

Contents

1 Project Code

1.1 Link

The source code can be seen at https://github.com/Quiet-ljb/SI507_fproj, which includes description of scripts.

1.2 Instruction

1. **Get an API key**
    - Go to link https://www.yelp.com/developers/v3/manage_app
    - In the create new app form, enter information about your app, then agree to Yelp API Terms of Use and Display Requirements. Then click the Submit button.
    - Your API key is generated
2. **Load data (If city_data.json doesn't exist or needs to be refreshed)**
    - Download the file load_data.py
    - Replace the variable `api\_key` with your API key generated in the first step
    - Run command `python load\_data.py`
    - You will see file city_data.json is generated or refreshed
3. **Run the server**
    - Download the file fproj.py
    - Run command `python fproj.py`
    - Open your web browser, enter url "localhost:5000" or"127.0.0.1:5000" to enter the website
4. **Interact with the program**
    - You will see four options after entering the website
    - Click on the options you want to choose
    - For the first two options, enter the query you want to search and click the search button
    - For the last two options, click on the result you are interested at

1.3 Required Python Packages

`flask,requests,beautifulsoup`

2 Data Sources

2.1 Yelp Fusion

Origin: It's a Web API and requires an API key. The url ofe this API is https://api.yelp.com/v3/businesses/search. The documentation of the API is https://docs.developer.yelp.com/docs/fusion-intro

Format: json file

How to access: First, I generate an API key according to the description in the documentation. Then I use requests package to access the API based on the format given by the documentation. I adjusted the parameter like "attributes", "limit" and "sort_by" to make sure that it returns the most popular restaurant for each city. Caching was used for this dataset.

Summary: The records available and retrieved are both about 331, which is the number of cities in the U.S. The import data fields are:

- **url:** the url of the restaurant
- **display_address:** the address of the restaurant
- **display_phone:** the phone number of the restaurant

2.2 Web Page

Origin: The url of the web page is https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population, and there is no documentation.

Format: html file

How to access: I use beautiful soup to parse and scrape the data, which contains all cities in the U.S. Those cities are not directly stored in the cache since it will be used as the searching query for the first data source. (Actually, it is stored together with the first data source)

Summary: The records available and retrieved are both about 331, which is the number of cities in the U.S. Since it's a web page, and the cities are stored in a table, the important data field is a tag "tr", which means the table in html. And the first column stores the city name.

3 Data Structure

3.1 Prefix Tree

All data was stored in a prefix tree, where all children of a node have common prefix stored in their parent node. Below shows an example about the prefix tree. Instead of storing those words in the figure, we store city name in the tree. Also, this tree is not case-sensitive and ignores the white space (e.g. "Ann Arbor" will be stored as "annarbor" in this tree

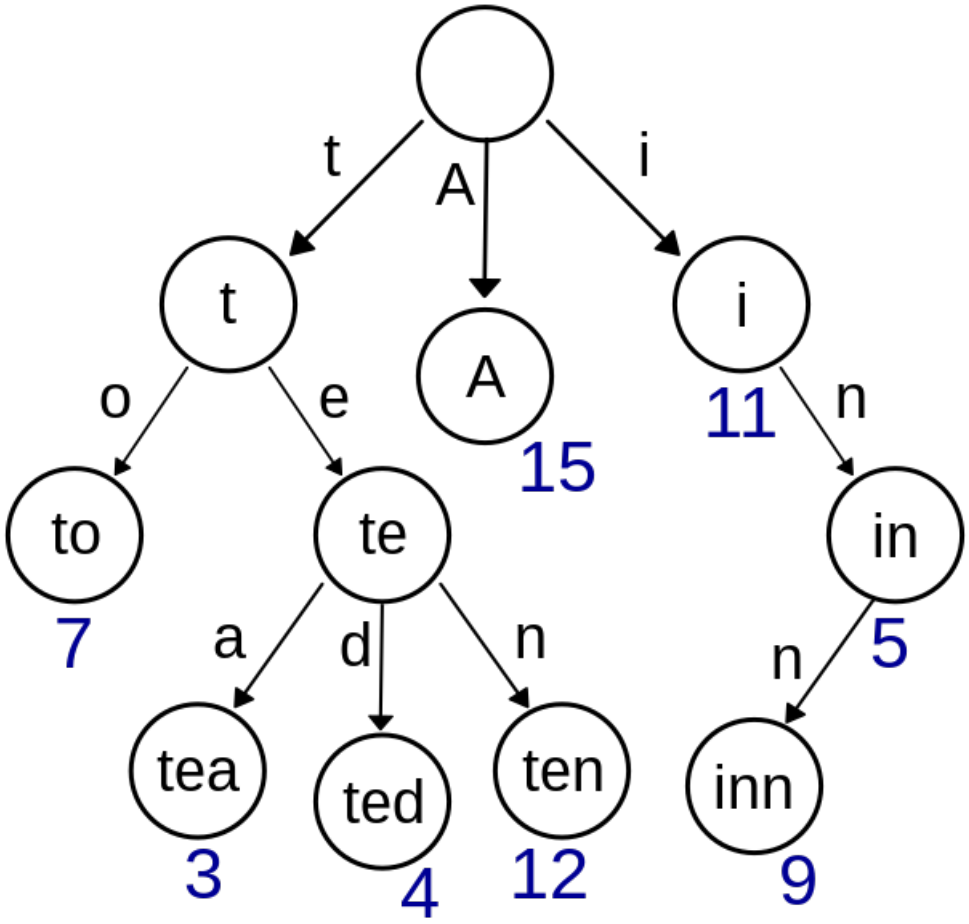

Figure 1: Prefix tree example

3.2 Tree Construction

The python script to construct the tree using classes is shown below.

```
class city_letter_node:
    def __init__(self, letter):
        self.letter = letter
        self.children = {}
        self.info = None

    def insert_city(self, city_name, city_info):
        if city_name=='':
            self.info = city_info
            return
        letter = city_name[0]
        if letter not in self.children:
            self.children[letter] = city_letter_node(letter)
```

```
14          self.children[letter].insert_city(city_name[1:], city_info)
15
16  top_node = city_letter_node('')
17
18  for city in all_city_data:
19      try:
20          city_name = city['businesses'][0]['location']['city']
21          city_name = city_name.lower()
22          city_name = city_name.replace(" ","")
23          top_node.insert_city(city_name, city)
24      except:
25          pass
```

3.3 JSON File

Below shows a json example that stores the data:

{

"businesses": [{"name": "Una's Cupcakes",

"url": "https://www.yelp.com/biz/unas-cupcakes-new-york?adjust_creative=jZ1AzlbsMQ0MFeKzJ662tA&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=jZ1AzlbsMQ0MFeKzJ662tA",

"location": {"city": "New York", "display_address": ["New York, NY 10035"]}, "display_phone": "(929) 955-6078"}]

}

3.4 Read the JSON File

The corresponding code is shown below:

```
1  all_city_data = []
2  f = open("city_data.json","r")
3  lines = f.readlines()
4  for line in lines:
5      data = json.loads(line)
6      all_city_data.append(data)
7  f.close()
```

3.5 Screenshots of Data and Data Structures

Below is the part of the data stored in the cache:

Figure 2: Data

Below is the prefix tree node class structure:

```python
class city_letter_node:
    def __init__(self, letter):
        self.letter = letter
        self.children = {}
        self.info = None

    def insert_city(self, city_name, city_info):
        if city_name=='':
            self.info = city_info
            return
        letter = city_name[0]
        if letter not in self.children:
            self.children[letter] = city_letter_node(letter)
        self.children[letter].insert_city(city_name[1:], city_info)

    def get_city_info(self, city_name):
        if city_name=='':
            return self.info
        letter = city_name[0]
        if letter not in self.children:
            return None
        return self.children[letter].get_city_info(city_name[1:])

    def get_city_info_by_prefix(self, prefix):
        if prefix=='':
            return self.get_all_city_info()
        letter = prefix[0]
        if letter not in self.children:
            return []
        return self.children[letter].get_city_info_by_prefix(prefix[1:])

    def get_all_city_info(self):
        if self.info is not None:
            return [self.info]
        res = []
        for child in self.children.values():
            res.extend(child.get_all_city_info())
        return res

    def show(self,level=0):
        print("-"*level + self.letter)
        for child in self.children.values():
            child.show(level+1)
```

Figure 3: Data Structure

4 Interaction & Presentation Options

4.1 Description

There are four options

1. **Search by Name**: You can enter a city's name into the searching box, then the website will return the information of the restaurant it recommends in this city

2. **Search by Prefix**: You can enter a city's prefix name into the searching box, then the website will return a list of the city that has the prefix. If you click on one city's name, it will return the information of the restaurant it recommends in this city

3. **Recommend Restaurant**: Return five restaurants it recommends

4. **All city**: Return all cities' name that can be searched in this system. If you click on one city's name, it will return the information of the restaurant it recommends in this city

4.2 Technology used

Only Flask is used.

4.3 Instructions

After entering the website

- Click on the options you want to choose
- For the first two options, enter the query you want to search and click the search button
- For the last two options, click on the result you are interested at

5 Demo Link

https://youtu.be/oYUJDp2ATfs