Train Ticket Machine Documentation

Release 0.1

Diogo Pires

Dec 10, 2017

Contents:

1 Introduction 1

2 Implementation 1

3 Normal API usage 2

4 Tests 2

5 API documentation 2

Python Module Index 4

1 Introduction

Challenge where one is asked to develop a search algorithm to be used in a train ticket machine to help users entering the name of stations. For every character users type, the API computes the possible words from the characters already typed and the next possible characters the user can type.

2 Implementation

The autocomplete API is implemented using a Trie as a data structure to save all the possible words. The Trie is computed from these words in the loading phase and this ony happens once.

During runtime, when users are typing and the API is outputting the possible words and the next available characters, the search in the Trie is done using the BFS algorithm.

3 Normal API usage

The normal usage of the API should be done from the UI/UX interface by calling the Trie.*from_prefix* method for every character chosen by the user. Nonetheless, here is an example of computing the Trie from a list of words and getting the possible words and next characters from prefixes:

```
>>> import trie
>>> words = ["Dartford", "Dartmouth", "Tower Hill",
...          "Derby", "Liverpool", "Liverpool Line Street",
...          "Paddington", "Euston", "London Bridge", "Victoria"]
>>> t = trie.Trie()
>>> t.add_words(words)
>>> t.from_prefix("dart")
{'matches': ['Dartford', 'Dartmouth'], 'next_chars': ['m', 'f']}
>>> t.from_prefix("liverpool")
{'matches': ['Liverpool', 'Liverpool Line Street'], 'next_chars': [' ']}
```

4 Tests

The API was tested using three different sets of words, which differ in size. All the tests and the words set are in the **tests** directory in the project's GitHub page. The tests are done using the *unittest* Python module.

The first test is done using a small word set of train stations that were provided in the problem statement. To run the test, go to the **tests** directory and type:

```
python -m unittest test_stations
```

The second test is done using a larger word set taken from here. The set is composed by 25000+ words in English. To run the test, go to the **tests** directory and type:

```
python -m unittest test_25000_words
```

The third and last test is done using a big word set taken from here. This set is composed by 466000+ words in English. To run the test, go to the **tests** directory and type:

```
python -m unittest test_466000_words
```

The tests can also be run all at once. Go to the **tests** directory and type:

```
python -m unittest run_tests
```

5 API documentation

class `trie.`**`Node`**(*data=None*, *label=None*)

This class implements Nodes of a Trie.

Normal usage of the class is

```
>>> n = Node("d")
>>> n.add_child("da")
>>> n
<data --> d (children=['da'])>
```

```
>>> n["da"]
<data --> da (children=[])>
>>> n["daa"]
Traceback (most recent call last):
...
KeyError: 'daa'
```

add_child(*data*)
Add child to node.

Parameters **data** (*str*) – The identifier of the node in the Trie

class trie.**Trie**
This class implements a Trie data structure using BFS as the search algorithm within the Trie

Normal usage of the class should be done from the UI/UX interface by calling the Trie.from_prefix method for every character chosen by the user.

Nonetheless, an example is

```
>>> t = Trie()
>>> words = ["Dartford", "Dartmouth", "Tower Hill",
...          "Derby", "Liverpool", "Liverpool Line Street",
...          "Paddington", "Euston", "London Bridge", "Victoria"]
>>> t.add_words(words)
>>> t.from_prefix("Liverpool")
{'matches': ['Liverpool', 'Liverpool Line Street'], 'next_chars': [' ']}
>>> t.from_prefix("X")
{'matches': [], 'next_chars': []}
```

add_words(*words*)
Add list of words to Trie. Should be used during load period before Trie is used.

Parameters **words** – List of strings to be inserted as nodes in the Trie

Raises TypeError

from_prefix(*prefix*)
Check what are the possible words given a prefix. Uses BFS search.

Parameters **prefix** (*str*) – String from which to check the possible words in the Trie

Returns Dictionary with the possible words and the next available characters

Raises ValueError, TypeError

word_exists(*word*, *ignore_case=True*)
Check if a given word exists in the Trie.

Parameters

- **word** (*str*) – String to check for correspondences in the Trie
- **ignore_case** (*bool*) – Boolean to indicate if case should be taken into consideration

Returns True, False

Raises ValueError, TypeError

Python Module Index

t

trie, 2
Trie, 2