CS532: Homework 9

Trie Tree

These questions will continue the trie tree that you did in Homework 7 and 8. You must start with the solution given in homework 7 for the methods for __init__, add_child, find and walk, and in homework 8 for the methods split_node and find_partial.

Question 1: Add Word

Write a method add_word that adds a word to a trie tree. If the word is already in the tree, it should not change the tree. The method should return the node that accounts for the last part of the word.

Your method must use the two methods you just added along with add_child. You can also use commonprefix, which is in the os module. Simply add to the beginning of your code:

```python
from os.path import commonprefix as commonprefix
```

For the example tree, your routine should be able to successfully add a word like 'rombus', where it does not need to split a node, and a word like 'romaine' where it first needs to split a node before adding in the new suffix.

Note: for the words 'rose' and 'roses', the node that 'rose' ends at should have a child with a value of 's'. In other words, nodes that have children can also be valid word endings.

Since some nodes that have children might be words and some might not be words, add a boolean variable word to the object to indicate whether a valid word ends at this node. You need to change split node so that it updates this value correctly, and change print_tree so that it indicates if a node is the ending for a word.

You should test your code on a variety of testcases to make sure it is correct. In particular, it should give the following result for the following sequence of commands.

```python
top = chars"
top.add_word("roses")
top.add_word("romane")
top.add_word("romanus")
top.add_word("rome")
```
Hand in the code for make_word, and the code any other routines that you hanged, which should be print_tree, split_node, and __init__.

Question 2: Tokenizing

Many languages use spaces to mark the beginning and end of words in the language. Some languages do not mark the end of words, and rely on people to determine what the boundaries are. Build a tokenizer to find all possible word segmentations.

Use the lexicon from homework 2, and build a Trie tree for it, using your prior code to retrieve the next 5 words at a time from your ‘web resource’. Call this routine create_trie. It should return the top node of the trie tree.

Create a routine called tokenize that takes as input the top node of a trie tree, and a string of words (with no spaces). It should return an array of different tokenizations. Each tokenization should be a list of words, where each word is in the lexicon from hw2_data.txt.

The routine tokenize will do a breadth-first search in which the frontier will be a list of pairs: (a) an index into the input string indicating how much of the input has been processed so far, and (b) a list of words that have been tokenized up to that point. You will not be using the version of breadth-first search that is in the textbook (which operates on graphs); instead you will make your own version specifically for this task. Your code should take the top element from the frontier and find all possible words to extend it by using the find_partial routine. That routine will find the longest match, and then you just need to add an entry into the frontier for each ancestor of it in the trie tree that is a valid word ending. Any time your code has a solution, you should add it to a list of answers, which you will output at the end. The routine should be iterative rather than recursive.

For the string ‘uponthewarmeat’, I get 5 tokenizations. These are:

```
[‘upon’, ‘the’, ‘warm’, ‘eat’]
[‘upon’, ‘the’, ‘war’, ‘meat’]
[‘upon’, ‘the’, ‘war’, ‘me’, ‘at’]
[‘up’, ‘on’, ‘the’, ‘warm’, ‘eat’]
[‘up’, ‘on’, ‘the’, ‘war’, ‘meat’]
[‘up’, ‘on’, ‘the’, ‘war’, ‘me’, ‘at’]
```

For the string ‘iabvolunteer’, I get 0 tokenizations.
Here is a copy of the output of my internal print statements when processing ‘warmeattastegood’. The non-indented lines are each item I get from the top of the frontier. The first indented line is what I search in for using find_partial. The second lines is the result. The next lines are walking up the trie tree and checking if the prefix is marked as a word or not. The first word found is ‘warm’. ‘warm’ is a valid word; its parent is ‘war’, which is also a valid word. It’s ancestors are ‘wa’, ‘w’ and ‘’, none of which are valid words. So, entries corresponding to ‘warm’ and ‘war’ are added to the frontier.

0 []
Search trie tree for: warmeattas...
Maximum found is: warm
  Checking node: warm
    Word: adding to frontier
  Checking node: war
    Word: adding to frontier
  Checking node: wa
    Not a word
  Checking node: w
    Not a word
  Checking node:
    Not a word
4 ['warm']
Search trie tree for: eattastego...
Maximum found is: eat
  Checking node: eat
    Word: adding to frontier
  Checking node: ea
    Not a word
  Checking node: e
    Not a word
  Checking node:
    Not a word
3 ['war']
Search trie tree for: meattasteg...
Maximum found is: meat
  Checking node: meat
    Word: adding to frontier
  Checking node: mea
    Not a word
  Checking node: me
    Word: adding to frontier
  Checking node: m
    Not a word
  Checking node:
    Not a word
...

Hand in your code for create_trie and tokenize.

**All About Eve**

In homework 7, you created a class and some methods for the Eve world. You will be using that class for this assignment. Use the solution from homework 7. Do not modify that class.
**Question 3a: Min Priority Heap**

In the next question, you will be implementing Dijkstra’s algorithm to find shortest paths from a source system. Dijkstra’s algorithm uses a min priority heap to store the distances, and updates these distances as it visits the adjacent systems from the system it just extracted.

Start from the solution that was given from Homework 4. You will need to add a method to the min heap class to decrease a key.

In addition to storing the distances, you will also need to store the index of the star. This way, when you extract the minimum distance, you can extract the star index as well. Make this a generic field, so that this routine is useful for other tasks as well. I referred to my two arrays as .key and .val. (.key replaces .lst in my previous answer key).

There is another issue as well. When you need to decrease a key, you will need to find the star's index in the heap. You could search the entire heap for this, but that will turn the \( O(\log(n)) \) time to decrease a key into \( O(n) \) time. Instead, have your min queue keep track of what index in the array each value is stored at. I used a python dictionary (hash table). Any time I move a value to a new place, I updated it’s entry. When an item is extracted, its entry should be updated so that it is obvious that it is no longer in the heap.

Hand in your code for your min priority heap.

**Question 3: Dijkstras Algorithm**

In a previous homework assignment, using breadth first search, we were able to find the shortest path between two points on the graph as we assumed each edge weight was 1. For this implementation, you will calculate the edge weights as the Euclidean distance between two systems. Add a method `distance` that works on objects that will calculate the distance between itself and another object (use the common Python style of calling the other object `other`). The `System` class is already reading in the x, y, and z coordinate from the CSV file. Here is the calculation:

\[
\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}
\]

Similar to the previous homework, we will be only traveling using stargates.

Implement Dijkstras algorithm, and stay faithful to the books pseudo-code as much as reasonable. It is fine to incorporate the initialization and relax code directly in your function. The algorithm should not access the system class at all. Use the implementation of a min priority heap from the previous homework. You will need to add a method to the min heap class to decrease a key.

Have another function (also not part of the class) that takes in two strings, that represent the start and destination systems, and it should return a list of system names representing a path. This method should first create the adjacency list (a weighed version of the one used in homework 7). Then it should call your implementation of Dijkstra’s algorithm.

Hand in your code for `distance`, Dijkstra’s algorithm, and the function that calls Dijkstra’s algorithm.

Here is my solution for the shortest path from Jita to Dodixie:

```
Jita -> Perimeter -> Iyen-Oursta -> Faurent -> Ambeke -> Glettiers -> Dodenvale
    -> Chinelant -> Unel -> Auberulle -> Pettinck -> Adreland -> Grinacanne -> Erme
    -> Botane -> Dodixie
```
Question 4: Life of Crime (Not the Plagiarism Kind)

You have a bounty on your head. The higher the security of the system, the more law enforcement is on the lookout for you.

Implement a version of Dijkstra’s algorithm, where it will give you the best path from a start to a destination, such that you are least likely to run into any form of law enforcement. The likelihood of your running into law enforcement is directly proportional to the security_status of the system.

Your edge weights will now be dependent on the security_status value of the system you’re going into. You are not looking for the path with the lowest minimum sum of the security_status values, but looking for a path with the lowest maximum security_status rating along the way. In other words, if you have a path with weights 5, 6, 10, the cost of that path is 10 (the maximum weight) rather than 21 (the sum of the weights).

Again, you will only be traveling via stargates.

Hand in your code for your modified version of Dijkstra’s algorithm, and the function that calls it.

Question 5: Correctness

Explain why the previous question has the optimal substructure property.