CS532 Homework 9 Critique

Archana Machireddy

Question 1

def print\_tree(self,indent=""):

print("%s%s"%(indent,self.value))

next = self.child

while next is not None:

ret = next.print\_tree(indent+" "\*(len(self.value)))

if ret is not None:

return ret

next = next.sibling

return None

Question 1 Critique

I don’t remember why I was using return, but it is not necessary.

def print\_tree(self,indent=""):

print("%s%s"%(indent,self.value))

next = self.child

while next is not None:

next.print\_tree(indent+" "\*(len(self.value)))

next = next.sibling

Question 2

def split\_node(self,prefix):

b = self.value[len(prefix):]

self.value = prefix

c = self.child

self.child = chars(b)

self.child.parent = self

self.child.child = c

if c != None:

self.child.child.parent = self.child

next = self.child.child.sibling

while next is not None:

next.parent = self.child

next = next.sibling

Question 2 Critique:

The last few lines of the code can be simplified into one while loop.

def split\_node(self,prefix):

b = self.value[len(prefix):]

self.value = prefix

c = self.child

self.child = chars(b)

self.child.parent = self

self.child.child = c

n = self.child.child

while n is not None:

n.parent = self.child

n = n.sibling

Question 3

def find\_partial(self,string,indent=""):

if string.find(self.value) != 0:

return (None,string)

string = string[len(self.value):]

last = self

if string == "":

return (self,string)

next = self.child

while next is not None:

ret,sub = next.find\_partial(string,indent+" ")

if ret is not None:

return (ret,sub)

next = next.sibling

return (last,string)

Question 4

def add\_word(self,word):

node, sub = self.find\_partial(word)

next = node.child

while next is not None:

com = self.common\_sub(next.value,sub)

if com != "":

next.split\_node(com)

if sub[len(com):] != "":

next.add\_child(chars(sub[len(com):]))

self.set\_word(next,sub[len(com):])

else:

next.word = True

return

else:

next = next.sibling

if sub != "":

node.add\_child(chars(sub))

self.set\_word(node,sub)

return

def set\_word(self,next,sub\_added):

node1 = next.child

while node1 is not None:

if node1.value == sub\_added:

node1.word = True

break

else:

node1 = node1.sibling

def print\_tree(self,indent=""):

if self.word == False:

print("%s%s"%(indent,self.value))

else:

print("%s%s W"%(indent,self.value))

next = self.child

while next is not None:

ret = next.print\_tree(indent+" "\*(len(self.value)))

if ret is not None:

return ret

next = next.sibling

return None

def split\_node(self,prefix):

b = self.value[len(prefix):]

self.value = prefix

c = self.child

self.child = chars(b)

self.child.parent = self

if self.word == True:

self.child.word = True

self.child.child = c

if b != "":

self.word = False

if c != None:

self.child.child.parent = self.child

next = self.child.child.sibling

while next is not None:

next.parent = self.child

next = next.sibling

def \_\_init\_\_(self,value):

self.parent = None

self.child = None

self.sibling = None

self.value = value

self.word = False

Question 4 Critique

I am not returning the node that accounts for the last part of the word. I can get rid of set\_word function by just initializing a node and then adding it as a child making if easy to set the end statue of the word to true. I was checking for a finished word inside the while loop. This can be done outside. The modified code for add\_word is:

def add\_word(self,word):

node, sub = self.find\_partial(word)

if sub == "":

print('Word already in trie')

node.word = True

return node

next = node.child

while next is not None:

com = self.common\_sub(next.value,sub)

if com == "":

next = next.sibling

continue

next.split\_node(com)

if com != sub:

new =chars(sub[len(com):])

next.add\_child(new)

new.word = True

return new

next.word = True

return next

new = chars(sub)

node.add\_child(new)

new.word = True

return new

Question 5

def create\_trie():

l = lexicon()

start = ""

top = chars("")

while True:

words = l.next5(start)

if len(words) < 1:

break

for w in words:

top.add\_word(w)

start = words[-1]

return top

def tokenize(trie,string):

queue = [(0,[])]

str\_len = len(string)

answers = []

while queue:

start, tokens = queue.pop(0)

print(start,tokens)

if start == str\_len:

answers.append(tokens)

else:

node, rem = trie.find\_partial(string[start:])

done = len(string)-len(rem)

found = string[start:done]

if tokens == []:

new\_tokens=[found]

else:

new\_tokens = tokens+[found]

if node.word == True:

queue.append((done,new\_tokens))

print('Found word %s'%(found))

print('Adding to frontier %s %i'%(found,done))

next = node.parent

l = len(node.value)

while next is not None:

new\_l = len(found)-l

new\_ll = done - l

if next.word == True:

if tokens == []:

new\_tokens=[found[0:new\_l]]

else:

new\_tokens = tokens + [found[0:new\_l]]

queue.append((new\_ll,new\_tokens))

print('Adding to frontier %s %i'%(found[0:new\_l],new\_ll))

else:

print(' Node %s is not a word'%(found[0:new\_l]))

l = l+len(next.value)

next = next.parent

return answers

Question 5 Critique

In create\_trie my indent for start = words[-1] is wrong.

def create\_trie():

l = lexicon()

start = ""

top = chars("")

while True:

words = l.next5(start)

if len(words) < 1:

break

for w in words:

top.add\_word(w)

start = words[-1]

return top

Simplified tokenize code:

def tokenize(trie,string):

queue = [(0,[])]

answers = []

while queue:

start, tokens = queue.pop(0)

print(start,tokens)

if start == len(string):

answers.append(tokens)

continue

node, rem = trie.find\_partial(string[start:])

print('Found word %s'%(node.get\_word()))

while node is not None:

w = node.get\_word()

if node.word:

queue.append((start+len(w),tokens + [w]))

print('Adding to frontier %s %i'%(w,start+len(w)))

else:

print(' Node %s is not a word'%(w))

node = node.parent

return answers

def get\_word(self):

word = ""

node = self

while node is not None:

print(node.value)

word = node.value + word

node = node.parent

return word