You are not allowed to use any imported libraries.

All methods that output a string must **return** the string, not **print** it.

Make sure to document any helper methods you define

Before you implement the Anagrams class, you will need to modify some methods in the BinarySearchTree class to assist with storage and retrieval of anagrams (words that have the same letters, just rearranged).

- All Nodes should store a dictionary as its value, where the key is a string of the sorted order of a word (see below) and the value is a list of words that have been inserted so far with the same letters.
- Change the insert and _insert methods accordingly to comply with the new format.
    - If a new word is inserted that has a matching sorted order of its letters as an existing Node, do not insert a new Node. Instead, modify the existing Node and append the inserted word into the list.
- No changes should be made to the Node class or any method definitions in the BinarySearchTree class.

Python's sorted method may be useful here - it returns a new sorted list from the items in the sequence, for example:

```
>>> sorted('mama')
['a', 'a', 'm', 'm']
>>> sorted('john')
['h', 'j', 'n', 'o']
>>> sorted('STate')
['S', 'T', 'a', 'e', 't'] # Note that ASCII values for uppercase
                            are smaller than lowercase
```

Once all the changes are complete, you should be able to run the following lines of code and get the following output:

```
>>> x=BinarySearchTree()
>>> x.insert('mom')
>>> x.insert('omm')
>>> x.insert('mmo')
>>> x.root
Node({'mmo': ['mom', 'omm', 'mmo']})
```

Modified functions

insert(self, value)
Inserts a word into the binary search tree. Words that have the same letter are saved together in a list, in a dictionary. The key of the dictionary is the sorted order of all the letters in the word.

You will have to modify the _insert helper method as well.

Input		
str	value	Word to insert into the binary search tree

Now that your BinarySearchTree class inserts words into the tree, we move into the description of the Anagrams class. Instances of this class read words from a txt file and use a Binary Search Tree to sstore each word. When you read a word from the file, you must sort it and then insert both the sorted word and the original word in the tree.

Attributes

Type	Name	Description
BinarySearchTree	_bst	Binary Search Tree that holds the words dictionaries

Methods

Type	Name	Description
None	create(self, file_name)	Opens a text file and adds words to the BST
(many)	getAnagrams(self, word)	Finds all anagrams of a word saved in the BST

Special methods

Type	Name	Description
None	__init__(self, word_size)	Initializes an Anagrams object with given max word size

__init__(self, word_size)
Initializes an Anagram object with given max word size. Any words greater than this max word size should not be considered.

Input		
int	word_size	Maximum word size for this object

create(self, file_name)
Opens a text file and adds words that do not exceed the maximum word length to the Binary Search Tree _bst. Recall from Homework 1 the syntax to read the contents of a file using `read()`

```
with open(file) as f: # ensures the file closes after the file operation finishes
    contents = f.read() # reads the entire file, saving data in contents as string
```

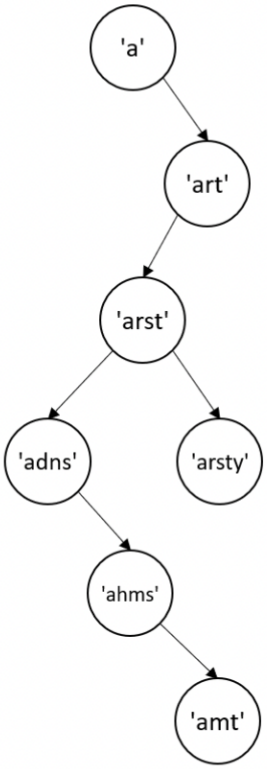
The string methods split() and splitlines() can also be helpful here.

Input		
str	file_name	File name of the txt file to read

```
>>> x = Anagrams(5)
>>> x.create('words_small.txt')
>>> x._bst.root.value
{'a': ['a']}
>>> x._bst.root.right.value
{'art': ['art', 'tar', 'rat']}
>>> x._bst.root.right.left.value
{'arst': ['arts', 'rats', 'star']}

# Result of the operation
   # 14 words inserted
   # 7 nodes in Tree
```

You might use encapsulation to define additional methods to retrieve and verify this information, that ensures more partial credit for your code if needed.



getAnagrams(self, word)

Finds all anagrams of a word saved in the Binary Search Tree. If there are no anagrams found, returns the string 'No match'

Input		
str	word	Word to find anagrams of

Output	
list	List of anagrams of this word found in _bst
str	'No match' if there are no anagrams of this word in the binary search tree

```
>>> x = Anagrams(5)
>>> x.create('words_small.txt')
>>> x.getAnagrams('ariel')
'No match'
>>> x.getAnagrams('mat')
['tam', 'mat']
>>> x.getAnagrams('art')
['art', 'tar', 'rat']
>>> x.getAnagrams('mom')
'No match'
>>> x.getAnagrams('sandshoes')
'No match'
>>> x.getAnagrams('samh')
['sham', 'hams']
```

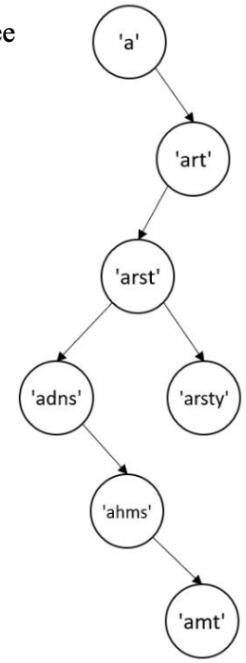
Section 3: Examples and doctests

Example #1: inserting all words with length 5 or less from words_small.txt
Order of insertion: a, art, arts, ands, sham, artsy, tam, rats, mat, tar, sand, rat, hams, star

Value of nodes:

```
{'a': ['a']}
{'art': ['art', 'tar', 'rat']}
{'arst': ['arts', 'rats', 'star']}
{'adns': ['ands', 'sand']}
{'ahms': ['sham', 'hams']}
{'arsty': ['artsy']}
{'amt': ['tam', 'mat']}
```

Binary search tree

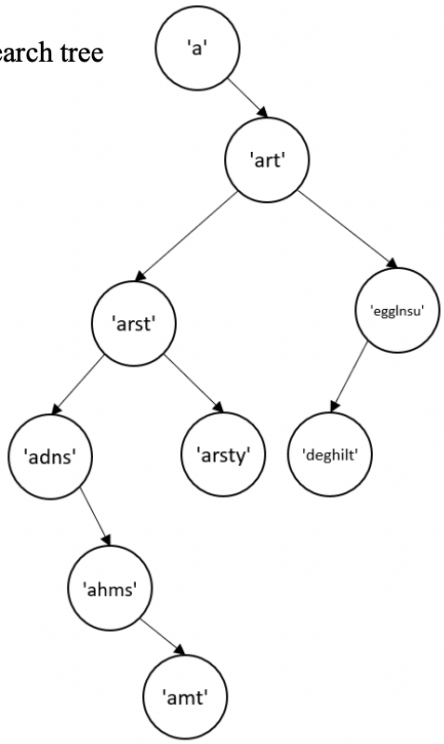


Example #2: inserting all words with length 7 or less from words_small.txt
Order of insertion: a, art, arts, ands, sham, artsy, tam, rats, mat, tar, sand, rat, hams, star, snuggle, lighted

Value of nodes:

```
{'a': ['a']}
{'art': ['art', 'tar', 'rat']}
{'arst': ['arts', 'rats', 'star']}
{'adns': ['ands', 'sand']}
{'ahms': ['sham', 'hams']}
{'arsty': ['artsy']}
{'amt': ['tam', 'mat']}
{'egglnsu': ['snuggle']}
{'deghilt': ['lighted']}
```

Binary search tree



Creating binary search trees from text files (not an exhaustive list):

Text file	Max length	Words inserted	Nodes inserted
words_small.txt	5	14	7
words_medium.txt	5	487	134
words_medium.txt	6	720	201
words_large.txt	3	1067	784
words_large.txt	5	13602	9747
words_large.txt	9	105175	89890

```
>>> x = Anagrams(5)
# starting with words_small.txt
>>> x.create('words_small.txt')
>>> x.getAnagrams('ariel')
'No match'
>>> x.getAnagrams('mat')
['tam', 'mat']
>>> x.getAnagrams('art')
['art', 'tar', 'rat']
>>> x = Anagrams(5)
# add more words from words_medium.txt
>>> x.create('words_medium.txt')
>>> x.getAnagrams('sale')
['ales', 'leas', 'sale', 'seal']
>>> x.getAnagrams('love')
'No match'
>>> x.getAnagrams('mean')
['amen', 'mane', 'mean', 'name']
>>> x = Anagrams(5)
# add more words from words_large.txt
>>> x.create('words_large.txt')
>>> x.getAnagrams('mart')
['mart', 'tram']
>>> x.getAnagrams('each')
['each', 'ache']
>>> x.getAnagrams('oval')
['oval']
>>> x.getAnagrams('rat')
['rat', 'tar', 'art']
```