Name: Toghrul Tahirov Student number: G47609664

Midterm Problems -- Information Retrieval CSCI 6452

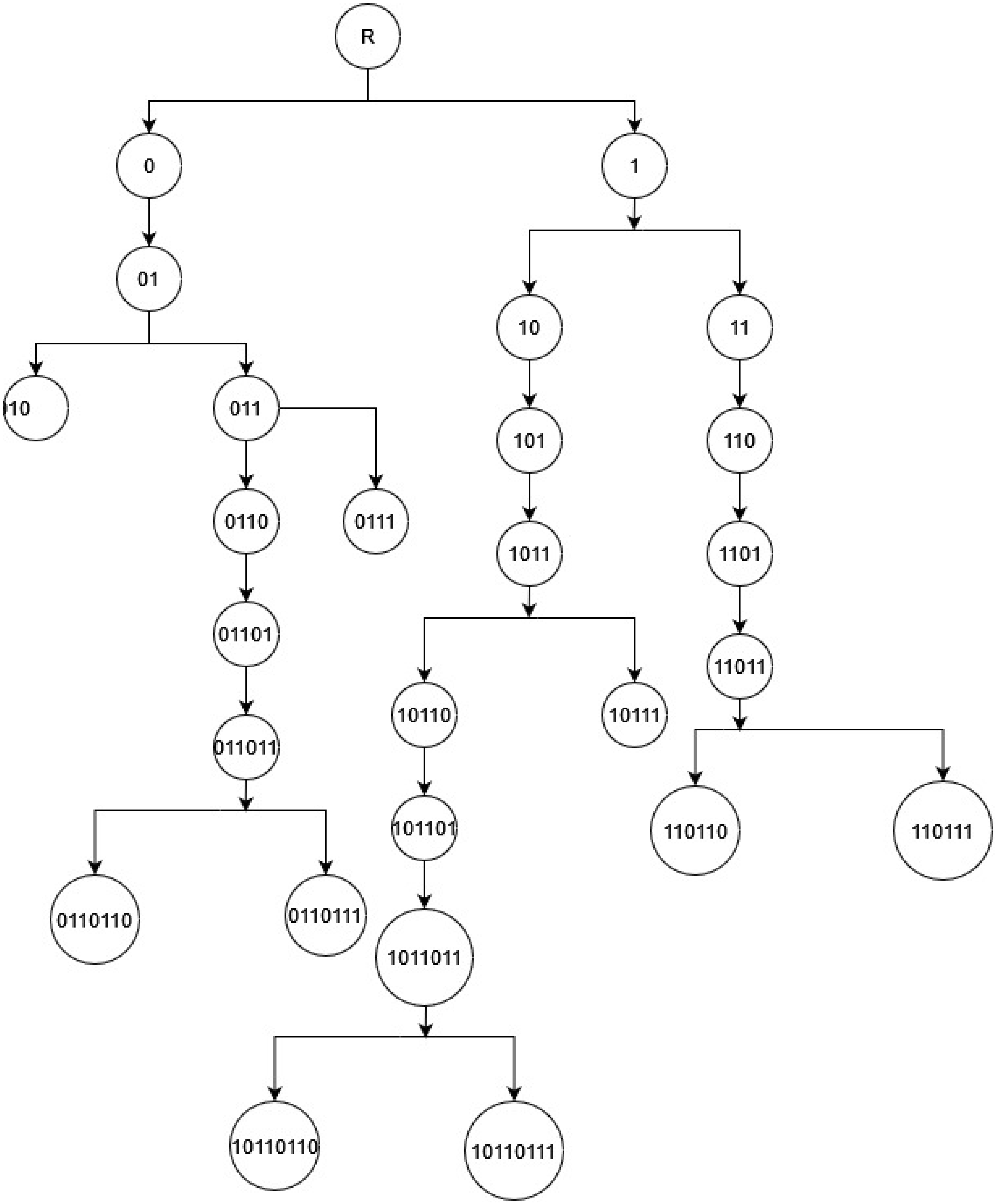
1. Given the following: 0101101101110111 (**16 points**)

A. Generate 9 sistrings – the first is given below. Underline or circle the unique prefix for each sistring (**3 points**)

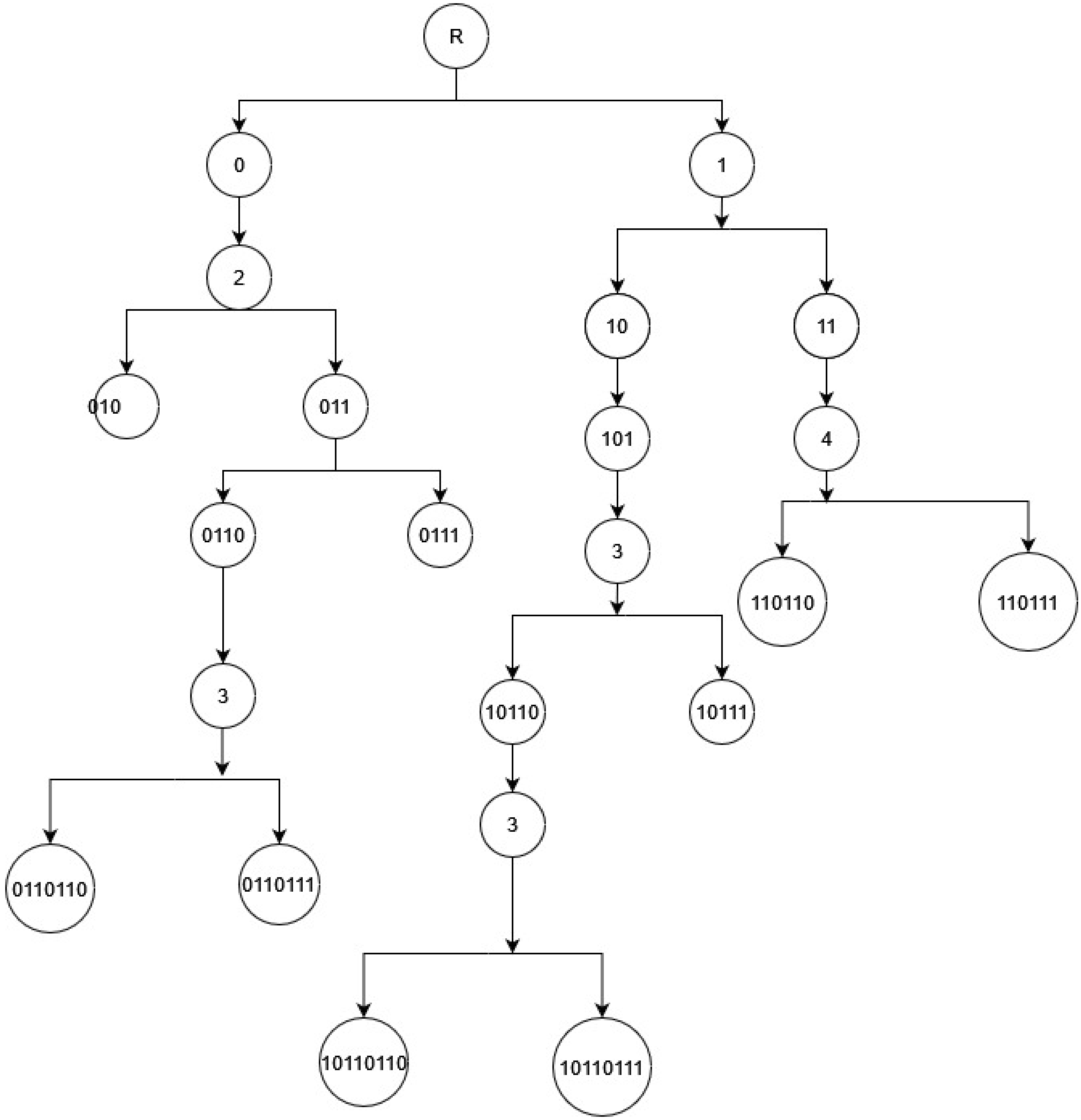
0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1
0101101101110111
101101101110111
01101101110111
1101101110111
101101110111
01101110111
1101110111
101110111
01110111

Unique Prefixes:
010
10110110
0110110
110110
10110111
0110111
110111
10111
0111

B. Create the PAT Trie (4 points)



C. Create the reduced PAT trie (**4 points**)



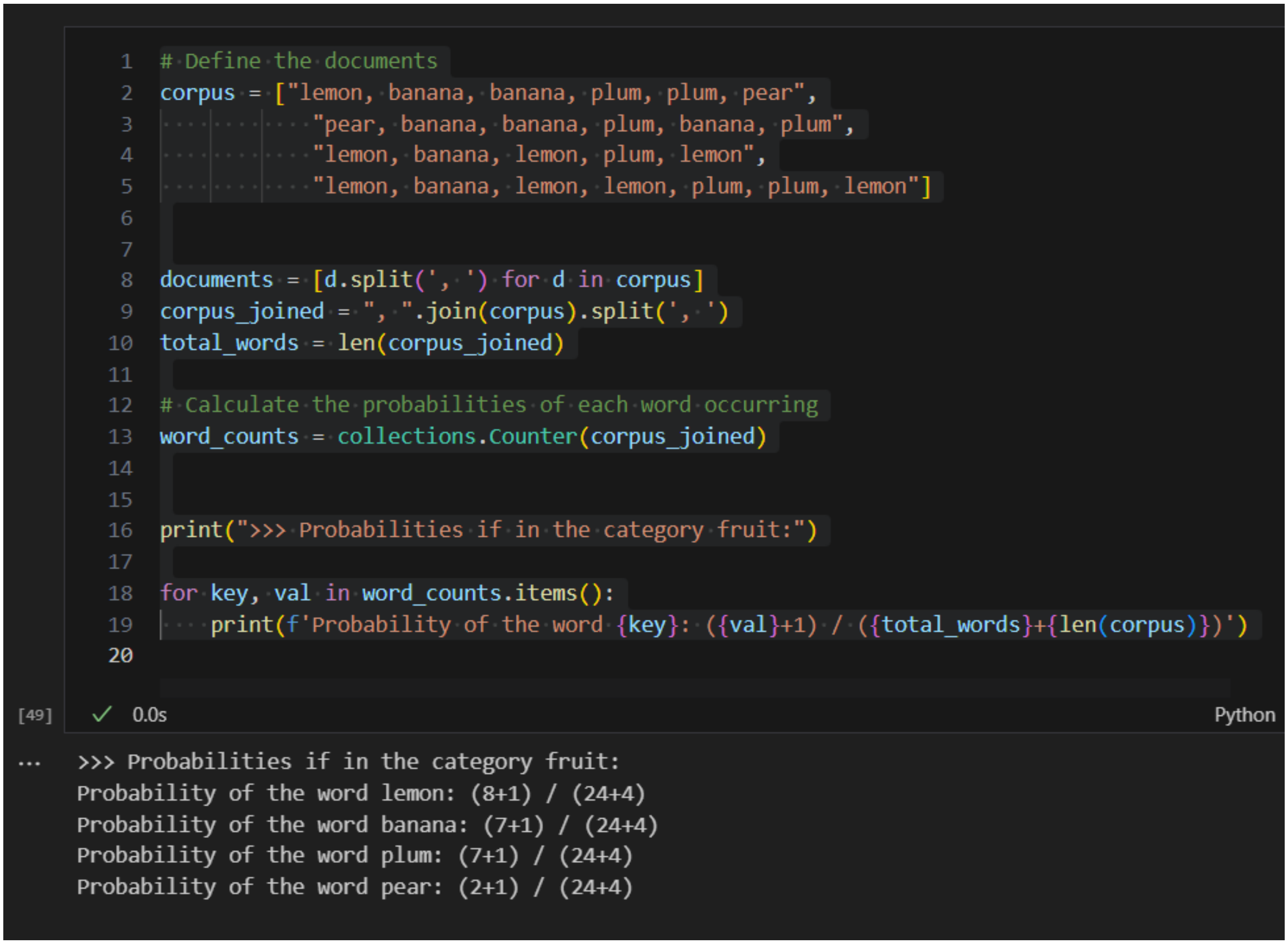
D. Given following search string show number of compares using PAT and reduced PAT trie and result of search from above: 101101111 and what the result of the search – sistring satisfies it or not. (**5 points)**

This string would require 8 compares when searched in the full PAT trie whilst requiring only 4 compares in the case of reduced PAT trie. Even though the prefix is found 1011011110111 in the trie, the next set of characters do match. Thus, the sistring does not satisfy the search

2. Given the following documents determine the weights for Naïve Bayesian category for document to be about "fruits". Calculate the probabilities for all words for both in and not in the category – you can leave it as a fraction. Given the new document listed determine if it should be given the category or not. **LEAVE ALL CALCULATIONS AS A FRACTION (10 points - 7 points calculate weights; 3 points new document)**

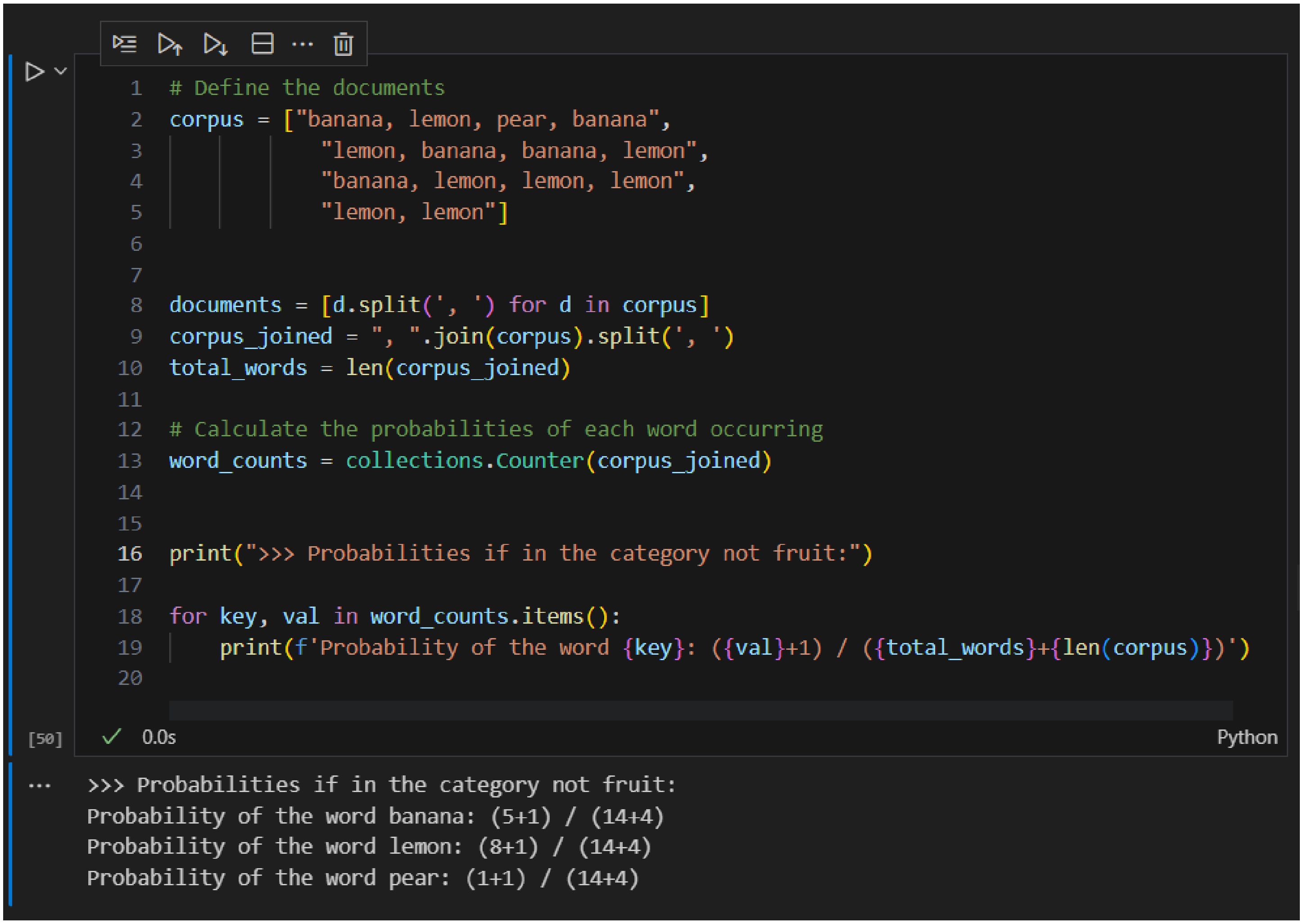
Doc1	lemon, banana, banana, plum, plum, pear	is member of fruits
Doc2	pear, banana, banana, plum, banana, plum	is member of fruits
Doc3	lemon, banana, lemon, plum, lemon	is member of fruits
Doc4	lemon, banana, lemon, lemon, plum, plum, lemon	is member of fruits
Doc5	banana, lemon, pear, banana	is NOT member of fruits
Doc6	lemon, banana, banana, lemon	is NOT member of fruits
Doc7	banana, lemon, lemon, lemon	is NOT member of fruits
Doc8	lemon, lemon	is NOT member of fruits

NEW DOCUMENT: banana, lemon, banana



```python
# Define the documents
corpus = ["lemon, banana, banana, plum, plum, pear",
            "pear, banana, banana, plum, banana, plum",
            "lemon, banana, lemon, plum, lemon",
            "lemon, banana, lemon, lemon, plum, plum, lemon"]


documents = [d.split(', ') for d in corpus]
corpus_joined = ", ".join(corpus).split(', ')
total_words = len(corpus_joined)

# Calculate the probabilities of each word occurring
word_counts = collections.Counter(corpus_joined)


print(">>> Probabilities if in the category fruit:")

for key, val in word_counts.items():
    print(f'Probability of the word {key}: ({val}+1) / ({total_words}+{len(corpus)})')

```

```
>>> Probabilities if in the category fruit:
Probability of the word lemon: (8+1) / (24+4)
Probability of the word banana: (7+1) / (24+4)
Probability of the word plum: (7+1) / (24+4)
Probability of the word pear: (2+1) / (24+4)
```



```python
# Define the documents
corpus = ["banana, lemon, pear, banana",
          "lemon, banana, banana, lemon",
          "banana, lemon, lemon, lemon",
          "lemon, lemon"]


documents = [d.split(', ') for d in corpus]
corpus_joined = ", ".join(corpus).split(', ')
total_words = len(corpus_joined)

# Calculate the probabilities of each word occurring
word_counts = collections.Counter(corpus_joined)


print(">>> Probabilities if in the category not fruit:")

for key, val in word_counts.items():
    print(f'Probability of the word {key}: ({val}+1) / ({total_words}+{len(corpus)})')

```

```
>>> Probabilities if in the category not fruit:
Probability of the word banana: (5+1) / (14+4)
Probability of the word lemon: (8+1) / (14+4)
Probability of the word pear: (1+1) / (14+4)
```

For the new doc:
P(fruit) = 1 / 2 * [8/28 * 9/28 * 8/28] = 576 / 43904
P(not fruit) = 1 / 2 * [6/18 * 9/18 * 6/18] = 324 / 11664

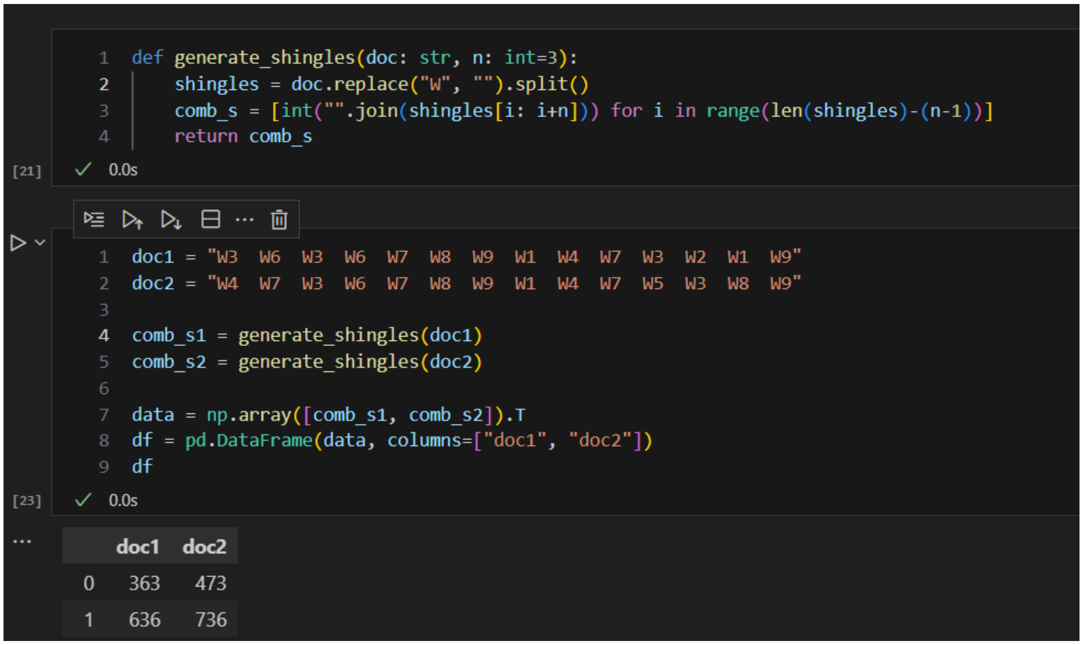
P(not fruit) > P(fruit) => new document is not a member of fruit

3. Given two documents – using 3 word shingles (e.g., w1w6w8 = 168) **(14 Pts: a and c = 5 points; b and d = 2 points)**.
    a. List the shingles for each document in 2 columns (one for each document)
    b. Determine the shingles in both documents and calculate the resemblance **leave as fraction**
    c. Select the 5 lowest shingle numbers from each document – write out them
    d. determine the resemblance – **leave as fraction**

W3 W6 W3 W6 W7 W8 W9 W1 W4 W7 W3 W2 W1 W9

W4 W7 W3 W6 W7 W8 W9 W1 W4 W7 W5 W3 W8 W9

a. Generated the shingles using the below python code (results are given on the right)



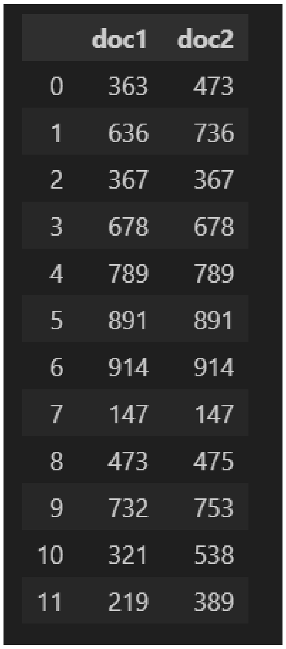
```python
def generate_shingles(doc: str, n: int=3):
    shingles = doc.replace("W", "").split()
    comb_s = [int("".join(shingles[i: i+n])) for i in range(len(shingles)-(n-1))]
    return comb_s
```

[21] ✓ 0.0s

```python
doc1 = "W3  W6  W3  W6  W7  W8  W9  W1  W4  W7  W3  W2  W1  W9"
doc2 = "W4  W7  W3  W6  W7  W8  W9  W1  W4  W7  W5  W3  W8  W9"

comb_s1 = generate_shingles(doc1)
comb_s2 = generate_shingles(doc2)

data = np.array([comb_s1, comb_s2]).T
df = pd.DataFrame(data, columns=["doc1", "doc2"])
df
```

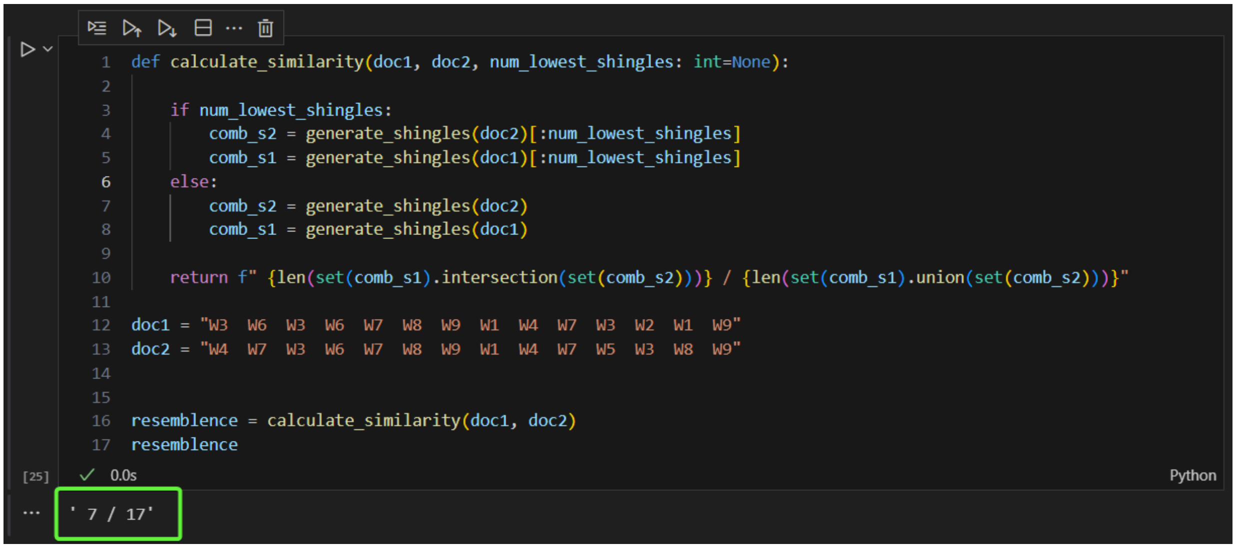
[23] ✓ 0.0s

	doc1	doc2
0	363	473
1	636	736



	doc1	doc2
0	363	473
1	636	736
2	367	367
3	678	678
4	789	789
5	891	891
6	914	914
7	147	147
8	473	475
9	732	753
10	321	538
11	219	389

b. Resemblance calculated using the below python code (results displayed at the output) – **7/17**:

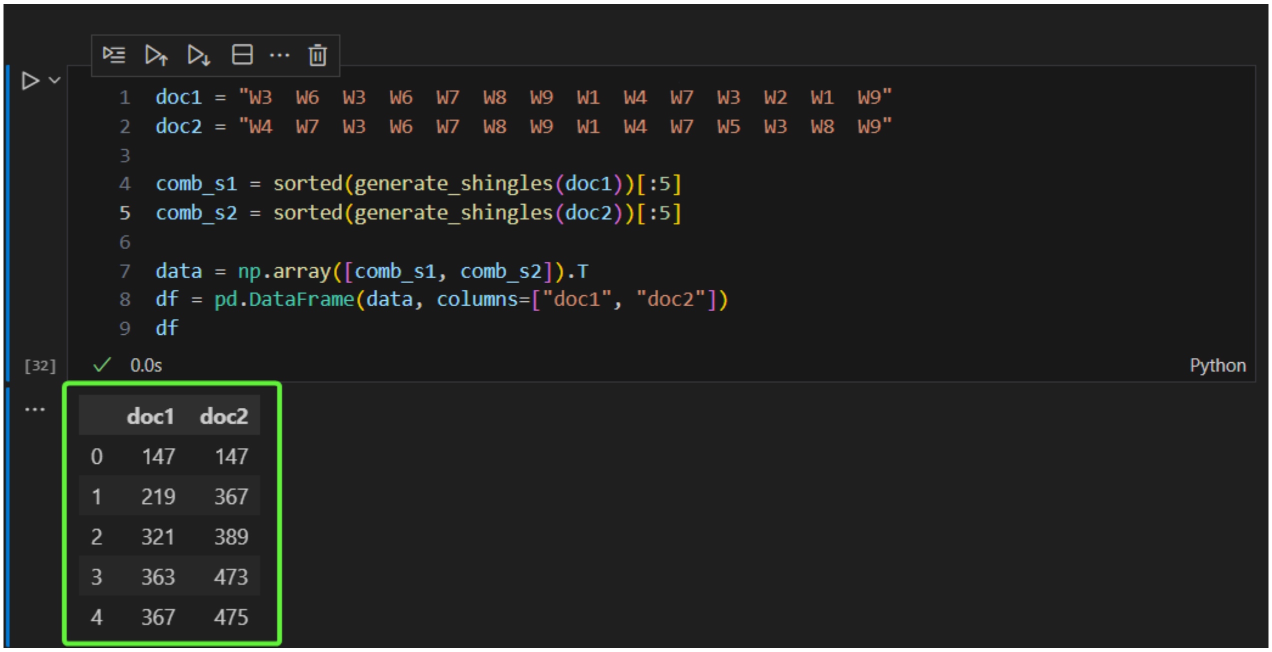


```python
def calculate_similarity(doc1, doc2, num_lowest_shingles: int=None):

    if num_lowest_shingles:
        comb_s2 = generate_shingles(doc2)[:num_lowest_shingles]
        comb_s1 = generate_shingles(doc1)[:num_lowest_shingles]
    else:
        comb_s2 = generate_shingles(doc2)
        comb_s1 = generate_shingles(doc1)

    return f" {len(set(comb_s1).intersection(set(comb_s2)))} / {len(set(comb_s1).union(set(comb_s2)))}"

doc1 = "W3  W6  W3  W6  W7  W8  W9  W1  W4  W7  W3  W2  W1  W9"
doc2 = "W4  W7  W3  W6  W7  W8  W9  W1  W4  W7  W5  W3  W8  W9"


resemblence = calculate_similarity(doc1, doc2)
resemblence
```

[25] ✓ 0.0s

```
' 7 / 17'
```

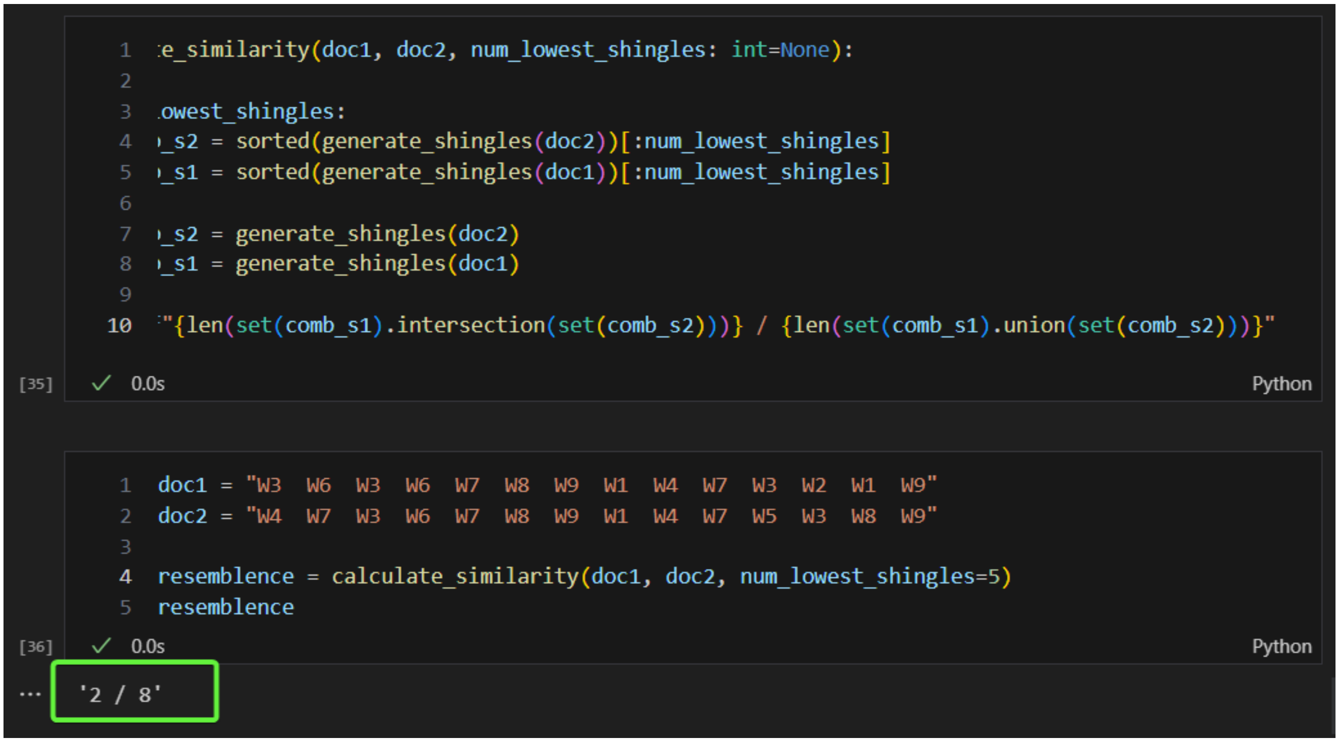
c. Generated using the below python code (results displayed at the output)



```python
doc1 = "W3  W6  W3  W6  W7  W8  W9  W1  W4  W7  W3  W2  W1  W9"
doc2 = "W4  W7  W3  W6  W7  W8  W9  W1  W4  W7  W5  W3  W8  W9"

comb_s1 = sorted(generate_shingles(doc1))[:5]
comb_s2 = sorted(generate_shingles(doc2))[:5]

data = np.array([comb_s1, comb_s2]).T
df = pd.DataFrame(data, columns=["doc1", "doc2"])
df
```

```
   doc1  doc2
0   147   147
1   219   367
2   321   389
3   363   473
4   367   475
```

d. Resemblance calculated using the below python code (results displayed at the output) – **2/8**

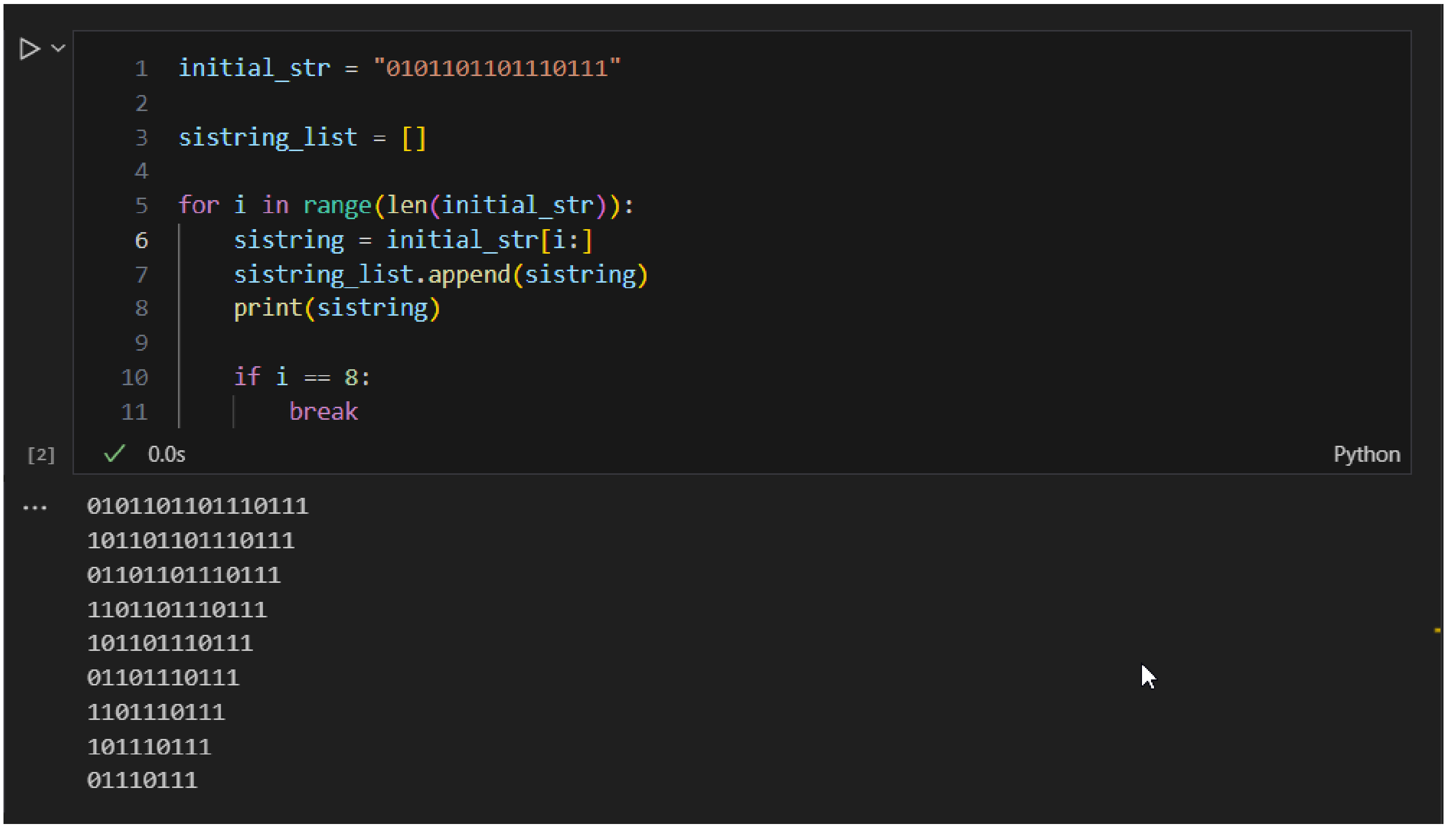


```python
e_similarity(doc1, doc2, num_lowest_shingles: int=None):

owest_shingles:
_s2 = sorted(generate_shingles(doc2))[:num_lowest_shingles]
_s1 = sorted(generate_shingles(doc1))[:num_lowest_shingles]

_s2 = generate_shingles(doc2)
_s1 = generate_shingles(doc1)

"{len(set(comb_s1).intersection(set(comb_s2)))} / {len(set(comb_s1).union(set(comb_s2)))}"
```

```python
doc1 = "W3  W6  W3  W6  W7  W8  W9  W1  W4  W7  W3  W2  W1  W9"
doc2 = "W4  W7  W3  W6  W7  W8  W9  W1  W4  W7  W5  W3  W8  W9"

resemblence = calculate_similarity(doc1, doc2, num_lowest_shingles=5)
resemblence
```

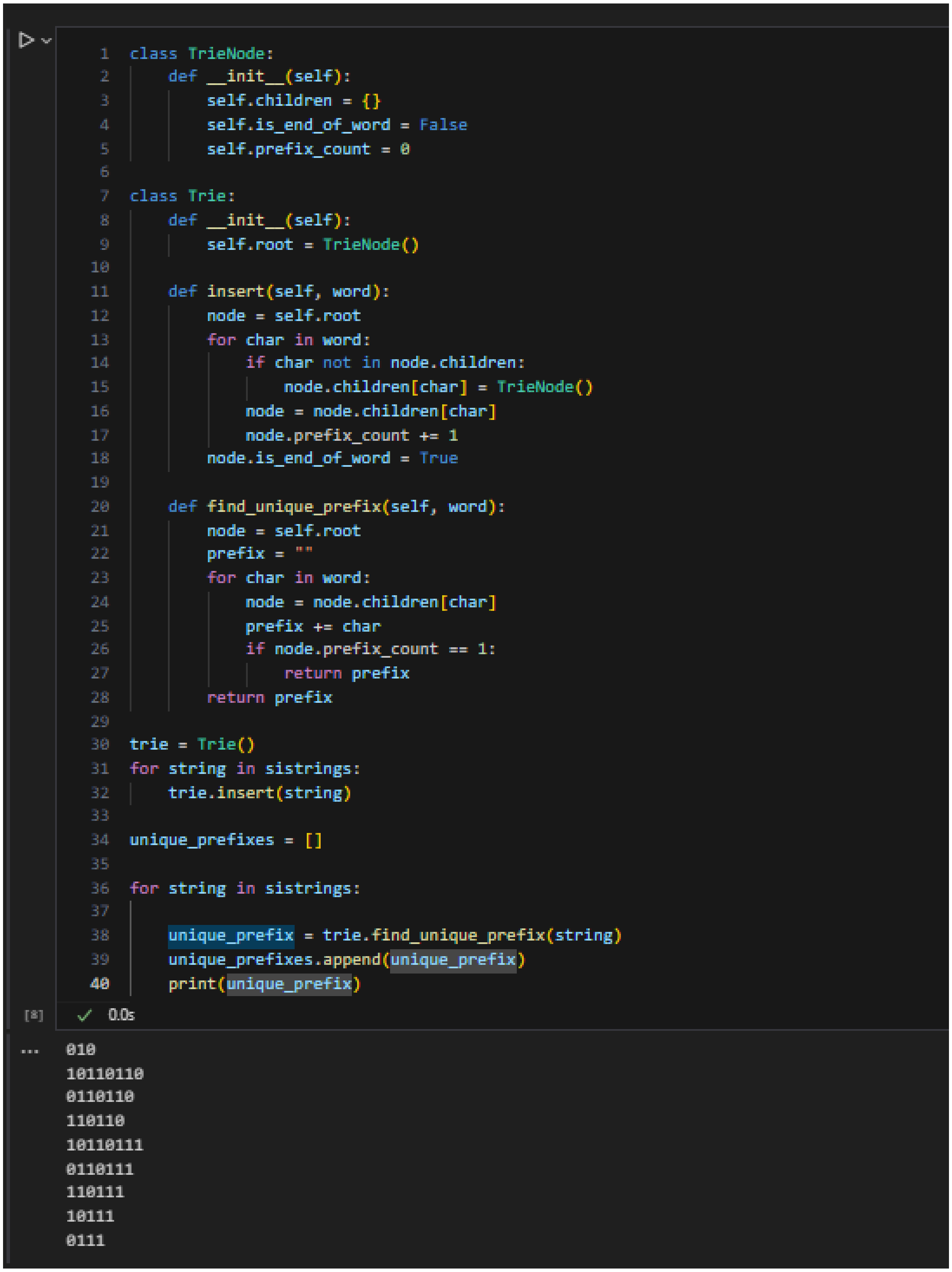
```
'2 / 8'
```

Code for task 1



```python
initial_str = "0101101101110111"

sistring_list = []

for i in range(len(initial_str)):
    sistring = initial_str[i:]
    sistring_list.append(sistring)
    print(sistring)

    if i == 8:
        break
```

```
0101101101110111
101101101110111
01101101110111
1101101110111
101101110111
01101110111
1101110111
101110111
01110111
```



```python
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False
        self.prefix_count = 0

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
            node.prefix_count += 1
        node.is_end_of_word = True

    def find_unique_prefix(self, word):
        node = self.root
        prefix = ""
        for char in word:
            node = node.children[char]
            prefix += char
            if node.prefix_count == 1:
                return prefix
        return prefix

trie = Trie()
for string in sistrings:
    trie.insert(string)

unique_prefixes = []

for string in sistrings:

    unique_prefix = trie.find_unique_prefix(string)
    unique_prefixes.append(unique_prefix)
    print(unique_prefix)
```

```
010
10110110
0110110
110110
10110111
0110111
110111
10111
0111
```