```
 1 import copy
 2 import os
 3 import re
 4
 5 from progressbar import progressbar
 6
 7
 8 class Posting(object):
 9     """
10     A Posting is a document id with a list of offsets
   , for a particular token in the Vocabulary
11     """
12     def __init__(self, doc_id: int):
13         self.doc_id = doc_id
14         self.offsets = []
15
16     def count(self):
17         return len(self.offsets)
18
19     def __str__(self):
20         return "The token is found in doc_id:%d at %d
    locations"%(self.doc_id, self.count())
21
22     def add_offset(self, new_offset: int)->bool:
23
24         offset_exists = False
25         i = 0
26         for i, offset in enumerate(self.offsets):
27             if offset == new_offset:
28                 offset_exits = True
29                 return offset_exists
30             elif offset > new_offset:
31                 break
32         if len(self.offsets) == 0:
33             self.offsets.append(new_offset)
34         elif i >= len(self.offsets)-1:
35             self.offsets.append(new_offset)
36         else:
37             self.offsets.insert(i, new_offset)
38         return True
39
40     def first_offset(self):
41         return self.offsets[0]
42
43     def last_offset(self):
44         return self.offsets[0]
```

```
45
46      def get_offsets(self):
47          return copy.copy(self.offsets)
48
49      def get_doc_id(self):
50          return self.doc_id
51
52
53
54
55  class VocabularyItem(object):
56      def __init__(self, token: str):
57          self.token = token
58          self.postings = []
59
60      def __str__(self):
61          return ":%s is found in %d docs"%(self.token
    , self.count())
62
63      def add_posting(self, new_doc_id:int, new_offset
    : int)->bool:
64
65          posting_exists = False
66          i = 0
67          for i, posting in enumerate(self.postings):
68              if posting.get_doc_id() == new_doc_id or
    posting.get_doc_id() > new_offset:
69                  break
70          if len(self.postings) == 0:
71              posting = Posting(doc_id=new_doc_id)
72              self.postings.append(posting)
73          elif i >= len(self.postings)-1:
74              self.postings.append(posting)
75          else:
76              self.postings.insert(i, posting)
77          posting.add_offset(new_offset)
78          return True
79
80      def get_postings(self):
81          return copy.copy(self.postings)
82
83      def get_token(self):
84          return self.token
85
86      def get_doc_freq(self):
87          return len(self.postings)
```

```
 88
 89 class Vocabulary(object):
 90     def __init__(self):
 91         self.vocabulary = []
 92
 93     def add_token(self, new_token:str, new_doc_id:
    int, new_offset:int):
 94         token_exists = False
 95         i = 0
 96         for i, item in enumerate(self.vocabulary):
 97             if item.get_token() == new_token or item
    .get_token() > new_token:
 98                 break
 99         if len(self.vocabulary) == 0:
100             item = VocabularyItem(new_token)
101             self.vocabulary.append(item)
102         elif i >= len(self.vocabulary) - 1:
103             self.vocabulary.append(item)
104         else:
105             self.vocabulary.insert(i, item)
106         item.add_posting(new_doc_id=new_doc_id,
    new_offset=new_offset)
107         return True
108
109     def get_vocabulary_size(self):
110         return len(self.vocabulary)
111
112     def get_vocabulary(self):
113         return [item.get_token() for item in self.
    vocabulary]
114
115
116 class IRSystem(object):
117     def __init__(self, PATH_TO_CORPUS):
118         self.path = PATH_TO_CORPUS
119         self.doc_count = 0
120         self.vocabulary = Vocabulary()
121         self.raw_text = []
122
123     def create_index(self):
124         flnames = os.listdir(self.path)
125         for f, flname in progressbar(enumerate(
    flnames)):
126             with open(os.path.join(self.path, flname
    ), "r", encoding="latin") as fp:
127                 text = fp.read()
```

```
128             tokens = self.tokenize(text, 10)
129             for t, token in enumerate(tokens):
130                 self.vocabulary.add_token(token, f,
    t)
131             self.doc_count += 1
132             self.raw_text = tokens
133
134
135     def tokenize(self, text, min_length:int=3, lower
    :bool=True):
136         pattern = re.compile(r"\w+")
137         raw_tokens = re.findall(pattern, text)
138         return [token.lower() for token in
    raw_tokens if len(token)>=min_length]
139
140
141
142
143
144
```