*# -\*- coding: utf-8 -\*-***from** idna **import** unicode  
**from** mmseg **import** Algorithm, dict\_load\_defaults  
  
  
**def** cuttest(text):  
 wlist = [word **for** word **in** Algorithm(text)]  
 tmp = **"/"**.join(wlist)  
 print  
 tmp  
 print  
 **"================================"  
  
  
def** benchmark(text):  
 **import** time  
 dict\_load\_defaults()  
 print  
 **">>>> load dict done!"  
  
 for** i **in** range(100):  
 begin = time.time()  
 wlist = [word **for** word **in** Algorithm(text)]  
 end = time.time()  
 print  
 **">>>> times: %f"** % float(end - begin)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 cuttest(**u"请把手抬高一点儿"**)  
 cuttest(**u"长春市长春节致词。"**)  
 cuttest(**u"长春市长春药店。"**)  
 cuttest(**u"我的和服务必在明天做好。"**)  
 cuttest(**u"一次性交一百元。"**)  
 cuttest(**u"我发现有很多人喜欢他。"**)  
 cuttest(**u"我喜欢看电视剧大长今。"**)  
 cuttest(**u"半夜给拎起来陪看欧洲杯糊着两眼半晌没搞明白谁和谁踢。"**)  
 cuttest(**u"李智伟高兴兴以及王晓薇出去玩，后来智伟和晓薇又单独去玩了。"**)  
 cuttest(**u"一次性交出去很多钱。 "**)  
 cuttest(**u"这是一个伸手不见五指的黑夜。我叫孙悟空，我爱北京，我爱Python和C++。"**)  
 cuttest(**u"我不喜欢日本和服。"**)  
 cuttest(**u"雷猴回归人间。"**)  
 cuttest(**u"工信处女干事每月经过下属科室都要亲口交代24口交换机等技术性器件的安装工作"**)  
 cuttest(**u"我需要廉租房"**)  
 cuttest(**u"永和服装饰品有限公司"**)  
 cuttest(**u"我爱北京天安门"**)  
 cuttest(**u"abc"**)  
 cuttest(**u"隐马尔可夫"**)  
 cuttest(**u"雷猴是个好网站"**)  
 cuttest(**u"“Microsoft”一词由“MICROcomputer（微型计算机）”和“SOFTware（软件）”两部分组成"**)  
 cuttest(**u"草泥马和欺实马是今年的流行词汇"**)  
 cuttest(**u"伊藤洋华堂总府店"**)  
 cuttest(**u"中国科学院计算技术研究所"**)  
 cuttest(**u"罗密欧与朱丽叶"**)  
 cuttest(**u"我购买了道具和服装"**)  
 cuttest(**u"PS: 我觉得开源有一个好处，就是能够敦促自己不断改进，避免敞帚自珍"**)  
 cuttest(**u"湖北省石首市"**)  
 cuttest(**u"总经理完成了这件事情"**)  
 cuttest(**u"电脑修好了"**)  
 cuttest(**u"做好了这件事情就一了百了了"**)  
 cuttest(**u"人们审美的观点是不同的"**)  
 cuttest(**u"我们买了一个美的空调"**)  
 cuttest(**u"线程初始化时我们要注意"**)  
 cuttest(**u"一个分子是由好多原子组织成的"**)  
 cuttest(**u"祝你马到功成"**)  
 cuttest(**u"他掉进了无底洞里"**)  
 cuttest(**u"中国的首都是北京"**)  
 cuttest(**u"孙君意"**)  
 cuttest(**u"外交部发言人马朝旭"**)  
 cuttest(**u"领导人会议和第四届东亚峰会"**)  
 cuttest(**u"在过去的这五年"**)  
 cuttest(**u"还需要很长的路要走"**)  
 cuttest(**u"60周年首都阅兵"**)  
 cuttest(**u"你好人们审美的观点是不同的"**)  
 cuttest(**u"买水果然后来世博园"**)  
 cuttest(**u"买水果然后去世博园"**)  
 cuttest(**u"但是后来我才知道你是对的"**)  
 cuttest(**u"存在即合理"**)  
 cuttest(**u"的的的的的在的的的的就以和和和"**)  
 cuttest(**u"I love你，不以为耻，反以为rong"**)  
 cuttest(**u" "**)  
 cuttest(**u""**)  
 cuttest(**u"hello你好人们审美的观点是不同的"**)  
 cuttest(**u"很好但主要是基于网页形式"**)  
 cuttest(**u"hello你好人们审美的观点是不同的"**)  
 cuttest(**u"为什么我不能拥有想要的生活"**)  
 cuttest(**u"后来我才"**)  
 cuttest(**u"此次来中国是为了"**)  
 cuttest(**u"使用了它就可以解决一些问题"**)  
 cuttest(**u",使用了它就可以解决一些问题"**)  
 cuttest(**u"其实使用了它就可以解决一些问题"**)  
 cuttest(**u"好人使用了它就可以解决一些问题"**)  
 cuttest(**u"是因为和国家"**)  
 cuttest(**u"老年搜索还支持"**)  
 cuttest(**u"干脆就把那部蒙人的闲法给废了拉倒！RT @laoshipukong : 27日，全国人大常委会第三次审议侵权责任法草案，删除了有关医疗损害责任“举证倒置”的规定。在医患纠纷中本已处于弱势地位的消费者由此将陷入万劫不复的境地。 "**)  
  
*#-\*- coding: utf-8 -\*-  
  
###############################################################################  
# 基本数据结构  
###############################################################################***class** Word:  
 **def** \_\_init\_\_(self, text=**''**, freq=0):  
 self.text = text  
 self.freq = freq  
 self.length = len(text)  
  
**class** Chunk:  
 **def** \_\_init\_\_(self, w1, w2=**None**, w3=**None**):  
 self.words = []  
 self.words.append(w1)  
 **if** w2:  
 self.words.append(w2)  
 **if** w3:  
 self.words.append(w3)  
  
 *# 总长度* **def** total\_length(self):  
 length = 0  
 **for** w **in** self.words:  
 length += len(w.text)  
 **return** length  
  
 *# 平均长度* **def** average\_length(self):  
 **return** float(self.total\_length()) / float(len(self.words))  
  
 *# 标准平方差* **def** variance(self):  
 **import** math  
 avg = self.average\_length()  
 sqr\_sum = 0.0  
  
 **for** w **in** self.words:  
 tmp = len(w.text)  
 tmp = tmp-avg  
 sqr\_sum += float(tmp)\*float(tmp)  
 **return** math.sqrt(sqr\_sum)  
  
 *# 自由语素度* **def** degree\_of\_morphemic\_freedom(self):  
 freqs = 0  
 **for** w **in** self.words:  
 freqs += w.freq  
 **return** freqs  
  
 *# 调试输出* **def** printx(self):  
 print (**'========================================'**)  
 **for** w **in** self.words:  
 print (**'word: %s'** % w.text)  
 print (**'total\_length = %d'** % self.total\_length())  
 print (**'average\_length = %f'** % self.average\_length())  
 print (**'variance = %f'** % self.variance())  
 print (**'degree\_of\_morphemic\_freedom = %f'** % self.degree\_of\_morphemic\_freedom())  
 print (**'========================================'**)  
  
*###############################################################################  
# 字典的加载与解析  
###############################################################################*words\_dict = {}  
max\_word\_length = 0  
  
**def** dict\_get\_word(text):  
 entry = words\_dict.get(text)  
 **if** entry:  
 **return** Word(text, entry[1])  
 **return None  
  
  
def** file(filename):  
 **pass  
  
  
def** dict\_load\_chars(filename):  
 **global** max\_word\_length  
 f = file(filename)  
 **for** line **in** f.readlines():  
 freq, word = line.split(**' '**)  
 word = unicode(word.strip(), **'utf-8'**)  
 words\_dict[word] = (len(word), int(freq))  
 max\_word\_length = max\_word\_length < len(word) **and** len(word) **or** max\_word\_length  
 f.close()  
  
**def** dict\_load\_words(filename):  
 **global** max\_word\_length  
 f = file(filename)  
 **for** line **in** f.readlines():  
 word = unicode(line.strip(), **'utf-8'**)  
 words\_dict[word] = (len(word), 0)  
 max\_word\_length = max\_word\_length < len(word) **and** len(word) **or** max\_word\_length  
 f.close()  
  
**def** dict\_load\_defaults():  
 **from** os.path **import** join, dirname  
 dict\_load\_chars(join(dirname(\_\_file\_\_), **'data'**, **'chars.dic'**))  
 dict\_load\_words(join(dirname(\_\_file\_\_), **'data'**, **'words.dic'**))  
  
*###############################################################################  
# 过滤器  
###############################################################################***def** take\_highest(chunks, comparator):  
 i = 1  
 **for** j **in** range(1, len(chunks)):  
 rlt = comparator(chunks[j], chunks[0])  
 **if** rlt > 0:  
 i = 0  
 **if** rlt >= 0:  
 chunks[i], chunks[j] = chunks[j], chunks[i]  
 i += 1  
 **return** chunks[0:i]  
  
**def** mm\_filter(chunks):  
 **def** comparator(a, b):  
 **return** a.total\_length() - b.total\_length()  
 **return** take\_highest(chunks, comparator)  
  
**def** lawl\_filter(chunks):  
 **def** comparator(a, b):  
 **return** a.average\_length() - b.average\_length()  
 **return** take\_highest(chunks, comparator)  
  
**def** svwl\_filter(chunks):  
 **def** comparator(a, b):  
 **return** a.variance() - b.variance()  
 **return** take\_highest(chunks, comparator)  
  
**def** lsdmfocw\_filter(chunks):  
 **def** comparator(a, b):  
 **return** a.degree\_of\_morphemic\_freedom() - b.degree\_of\_morphemic\_freedom()  
 **return** take\_highest(chunks, comparator)  
  
*###############################################################################  
# 以下为算法实现  
###############################################################################***class** Algorithm(object):  
 match\_cache\_size = 3  
  
 **def** \_\_init\_\_(self, text):  
 **if** isinstance(text, unicode):  
 self.text = text  
 **else**:  
 self.text = unicode(text, **'utf-8'**)  
 self.text\_length = len(self.text)  
 self.pos = 0  
 self.match\_cache\_i = 0  
 self.match\_cache = []  
  
 **for** i **in** range(self.match\_cache\_size):  
 self.match\_cache.append([-1, Word()])  
  
 *# 确保有字典* **if not** words\_dict:  
 dict\_load\_defaults()  
  
 **def** \_\_iter\_\_(self):  
 **while True**:  
 tk = self.next\_token()  
 **if** tk **is None**:  
 **raise** StopIteration  
 **yield** tk  
  
 **def** next\_token(self):  
 **while** self.pos < self.text\_length:  
 **if** self.is\_cjk\_char(self.next\_char()):  
 tk = self.get\_cjk\_word()  
 **else**:  
 tk = self.get\_basic\_latin\_word()  
 **if** len(tk) > 0:  
 **return** tk  
 **return None  
  
 def** next\_char(self):  
 **return** self.text[self.pos]  
  
 **def** is\_cjk\_char(self, ch):  
 **return** 0x4e00 <= ord(ch) < 0x9fa6  
  
 **def** is\_latin\_char(self, ch):  
 **import** string  
 **if** ch **in** string.whitespace:  
 **return False  
 if** ch **in** string.punctuation:  
 **return False  
 return** ch **in** string.printable  
  
 **def** get\_basic\_latin\_word(self):  
 *# Skip pre-word whitespaces and punctuations* **while** self.pos < self.text\_length:  
 ch = self.next\_char()  
 **if** self.is\_latin\_char(ch) **or** self.is\_cjk\_char(ch):  
 **break** self.pos += 1  
  
 start = self.pos  
 **while** self.pos < self.text\_length:  
 **if not** self.is\_latin\_char(self.next\_char()):  
 **break** self.pos += 1  
 end = self.pos  
  
 *# Skip post-word whitespaces and punctuations* **while** self.pos < self.text\_length:  
 ch = self.next\_char()  
 **if** self.is\_latin\_char(ch) **or** self.is\_cjk\_char(ch):  
 **break** self.pos += 1  
  
 **return** self.text[start:end]  
  
 **def** get\_cjk\_word(self):  
 chunks = self.create\_chunks()  
 **if** len(chunks) > 1:  
 chunks = mm\_filter(chunks)  
 **if** len(chunks) > 1:  
 chunks = lawl\_filter(chunks)  
 **if** len(chunks) > 1:  
 chunks = svwl\_filter(chunks)  
 **if** len(chunks) > 1:  
 chunks = lsdmfocw\_filter(chunks)  
  
 **if** len(chunks) < 1:  
 **return ''** length = len(chunks[0].words[0].text)  
 token = self.text[self.pos:self.pos+length]  
 self.pos += length  
 **return** token  
  
 **def** create\_chunks(self):  
 chunks = []  
 orig\_pos = self.pos  
  
 words1 = self.find\_match\_words()  
 **for** w1 **in** words1:  
 self.pos += len(w1.text)  
 **if** self.pos < self.text\_length:  
 words2 = self.find\_match\_words()  
 **for** w2 **in** words2:  
 self.pos += len(w2.text)  
 **if** self.pos < self.text\_length:  
 words3 = self.find\_match\_words()  
 **for** w3 **in** words3:  
 **if** w3.length == -1:  
 chunk = Chunk(w1, w2)  
 **else**:  
 chunk = Chunk(w1, w2, w3)  
 chunks.append(chunk)  
 **elif** self.pos == self.text\_length:  
 chunk = Chunk(w1, w2)  
 chunks.append(chunk)  
 self.pos -= len(w2.text)  
 **elif** self.pos == self.text\_length:  
 chunk = Chunk(w1)  
 chunks.append(chunk)  
 self.pos -= len(w1.text)  
  
 self.pos = orig\_pos  
 **return** chunks  
  
 **def** find\_match\_words(self):  
 **for** i **in** range(self.match\_cache\_size):  
 **if** self.match\_cache[i][0] == self.pos:  
 **return** self.match\_cache[i][1]  
  
 orig\_pos = self.pos  
 n = 0  
 words = []  
  
 **while** self.pos < self.text\_length:  
 **if** n >= self.max\_word\_length():  
 **break  
 if not** self.is\_cjk\_char(self.next\_char()):  
 **break** self.pos += 1  
 n += 1  
  
 text = self.text[orig\_pos:self.pos]  
 word = dict\_get\_word(text)  
 **if** word:  
 words.append(word)  
  
 self.pos = orig\_pos  
 **if not** words:  
 word = Word()  
 word.length = -1  
 word.text = **'X'** words.append(word)  
  
 self.match\_cache[self.match\_cache\_i] = (self.pos, words)  
 self.match\_cache\_i += 1  
 **if** self.match\_cache\_i >= self.match\_cache\_size:  
 self.match\_cache\_i = 0  
  
 **return** words  
  
 **def** max\_word\_length(self):  
 **return** max\_word\_length  
  
*#coding=utf-8***import** re  
**import** xml.dom.minidom  
**from** mmseg **import** Algorithm, dict\_load\_defaults  
  
**class** Bayes:  
 **def** \_\_init\_\_(self):  
 self.message\_xml = open(**'message.xml'**, **"r"**)  
 self.message = self.message\_xml.read()  
 self.good = {}  
 self.bad = {}  
 **import** collections  
 self.total = collections.defaultdict(**lambda**: 1)  
 **return  
  
 def** get\_messages(self):  
 pattern\_message = re.compile(**'(<Body><\!\[CDATA\[)(.+?)(\]\]></Body>)'**)  
 self.message.decode(**'utf-8'**)  
 res = pattern\_message.findall(self.message)  
 message\_flag = open(**'message\_flag'**, **'w'**)  
 **for** mes **in** res:  
 tmp = unicode(mes[1], **'utf-8'**)  
 print(tmp)  
 flag = raw\_input()  
 message\_flag.write(tmp.encode(**'utf-8'**) + **' '** + flag + **'\n'**)  
 **for** word **in** Algorithm(tmp):  
 self.total[word] += 1  
 message\_xml.close()  
 **for** i **in** self.total.keys():  
 print i,self.total[i]  
  
 **def** cuttest(self, text):  
 wlist = [word **for** word **in** Algorithm(text)]  
 tmp = **"/"**.join(wlist)  
 print (tmp)  
 print (**"================================"**)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 bayes = Bayes()  
 bayes.get\_messages()