

```
1 from utils import *
2
3
4 def lower_and_pad(file_data):
5     """
6         :param file_data: a file to be made lower case and each
7             sentence to padded
8         :return: tuple(list[][], dict{})
9             each list[i] is a sentence, and list[i][j] is a
10            word
11            dict{} a counter dictionary
12            """
13
14    processed_data = []
15    word_count = {}
16
17    line = file_data.readline().split()
18    while line:
19        processing_line = ['<s>']
20        for token in line:
21            token = token.lower()
22            if token in word_count:
23                word_count[token] += 1
24            else:
25                word_count[token] = 1
26            processing_line.append(token)
27        processing_line.append('</s>')
28        processed_data.append(processing_line)
29        line = file_data.readline().split()
30
31    return processed_data, word_count
32
33
34 def get_words_appeared_once_only(word_count: dict) -> set:
35     words_appeared_once = set()
36     for word in word_count.keys():
37         if word_count[word] == 1:
38             words_appeared_once.add(word)
39
40     # fancy one liner
41     # words_appeared_once = set(dict(filter(lambda x: x[1]
42     ] == 1, word_count.items())).keys())
43
44     return words_appeared_once
45
46
47 def replace_words_appeared_once_with_unk(data: list[list],
48     words_appeared_once: set):
49     for sentence in data:
50         for i in range(len(sentence)):
51             if sentence[i] in words_appeared_once:
52                 sentence[i] = '<unk>'
```

```
47
48 def write_data_to_file(data: str, file_obj=None, file_path=None):
49     if not file_obj:
50         if not file_path:
51             raise Exception('No path and file object provided')
52         file_obj = open(file_path, 'w')
53
54     file_obj.write(data)
55     file_obj.close()
56
57
58 def pre_process_training_data(training_data_path: str,
59                               pre_processed_data_path: str):
60     with open(training_data_path, 'r') as training_corpus:
61         with open(pre_processed_data_path, 'w') as pre_processed_training_corpus:
62             processing_data, word_count = lower_and_pad(
63                 training_corpus)
64             words_appeared_once =
65             get_words_appeared_once_only(word_count)
66             replace_words_appeared_once_with_unk(
67                 processing_data, words_appeared_once)
68             pre_processed_training_corpus.write(
69                 data_list_to_str(processing_data))
70
71
72
73
74 def lower_and_pad_file(data_path: str, save_path: str):
75     with open(data_path) as file:
76         with open(save_path, 'w') as output_file:
77             lower_and_padded_data, word_count =
78             lower_and_pad(file)
79             output_file.write(data_list_to_str(
80                 lower_and_padded_data))
81
82
83
84 def pre_process_test_data(data_path: str, save_path: str,
85                           pre_processed_training_path):
86     with open(data_path) as file:
87         with open(save_path, 'w') as output_file:
88             with open(pre_processed_training_path) as
89                 training_file:
90                     training_token_counter = count_token(
91                         file_to_list_list(training_file))
92                     lower_and_padded_data, word_count =
93                     lower_and_pad(file)
94
95                     for sentence in lower_and_padded_data:
96                         for i in range(len(sentence)):
```

```
83             if sentence[i] not in
84                 training_token_counter:
85                     sentence[i] = '<unk>'
86             output_file.write(data_list_to_str(
87                 lower_and_padded_data))
88
89 # pre_process_test_data('./Data/test.txt', './1.txt', './
90 # pre_processed_training_data.txt')
91 # pre_process_data('./Data/train-Spring2023.txt', './Data/
92 # pre_processed_training_data.txt')
93 # pre_process_data('./Data/test.txt', './Data/
94 # pre_processed_test_corpus.txt')
95
```

```

1 from utils import *
2 from bigram_model import *
3 from unigram_model import *
4
5 def q1():
6     """
7         How many word types (unique words) are there in the
8             training corpus? Please include the end-of-sentence padding
9                 symbol </s> and the unknown token <unk>. Do not include
10                the start of sentence padding symbol <s>.
11        :return:
12        """
13        with open('./Data/pre_processed_training_data.txt') as
14            training_corpus:
15                training_data = file_to_list_list(training_corpus)
16                token_counter = count_token(training_data)
17                del token_counter['<s>']
18                unique_token_total = len(token_counter.keys())
19                print('Q1. Total word types in training corpus is:'
20 , unique_token_total)
21                return unique_token_total
22
23    def q2():
24        """
25            How many word tokens are there in the training corpus?
26            Do not include the start of sentence padding symbol <s>.
27        :return:
28        """
29        with open('./Data/pre_processed_training_data.txt') as
30            training_corpus:
31                training_data = file_to_list_list(training_corpus)
32                token_counter = count_token(training_data)
33                del token_counter['<s>']
34                token_total = sum(token_counter.values())
35                print('Q1. Total word tokens in training corpus is
36 :', token_total)
37                return token_total
38
39    def q3():
40        """
41            What percentage of word tokens and word types in the
42            test corpus did not occur in training (before you mapped
43            the
44            unknown words to <unk> in training and test data)?
45            Please include the padding symbol </s> in your calculations
46            .
47            Do not include the start of sentence padding symbol <s
48 >.

```

```

39     :return:
40     """
41     with open('./Data/padded_and_lowered_training_corpus.
42         txt') as training_corpus:
43         with open('./Data/padded_and_lowered_test_corpus.
44             txt') as test_corpus:
45             training_data = file_to_list_list(
46                 training_corpus)
47             test_data = file_to_list_list(test_corpus)
48             training_token_counter = count_token(
49                 training_data)
50             test_token_counter = count_token(test_data)
51
52             del training_token_counter['<s>']
53             del test_token_counter['<s>']
54
55             test_token_type_total = len(test_token_counter.
56                 keys())
57             token_type_list_in_test_not_training = [token
58                 for
59                 token in test_token_counter.keys()
60                 if
61                 token not in training_token_counter]
62
63             percentage_token_type_in_test_not_training =
64                 len(
65                     token_type_list_in_test_not_training) /
66                 test_token_type_total
67
68             test_token_total = sum(test_token_counter.
69                 values())
70             token_count_in_test_not_training = sum([
71                 test_token_counter[token]
72                     for
73                     token in token_type_list_in_test_not_training])
74             percentage_token_in_test_not_training =
75                 token_count_in_test_not_training / test_token_total
76
77             print("The percentage of word tokens in test
78                 corpus but not in training corpus is:",
79                 percentage_token_in_test_not_training *
80                 100)
81             print("The percentage of word types in test
82                 corpus but not in training corpus is:",
83                 percentage_token_type_in_test_not_training *
84                 100)
85
86             return percentage_token_in_test_not_training,
87                 percentage_token_type_in_test_not_training
88
89
90

```



```

105         not_occurred_type_percentage = (
106             total_bigram_test_type - occurred_types) /
107             total_bigram_test_type
108         not_occurred_token_percentage = (
109             total_bigram_test_token - occurred_tokens) /
110             total_bigram_test_token
111
112
113
114 def q5():
115     data = [['i', 'look', 'forward', 'to', 'hearing',
116             'your', 'reply', '.', '</s>']]
117     unigram_prob = unigram_evaluate_log_prob(data, './
118 Models/unigram_model1.json')
119
120     print()
121     bigram_prob = bigram_evaluate_log_prob(data, './Models
122 /bigram_model1.json')
123
124     print()
125     print('Unigram Log Probability:', unigram_prob)
126     print('Bigram Log Probability:', bigram_prob)
127     print('Bigram with add one smoothing Log Probability:'
128 , bigram_prob_add_one_smoothing)
129
130
131 def q7():
132     with open('./Data/test_token_not_in_training_to_unk.
133 txt') as test_file:
134         test_data = file_to_list_list(test_file)
135
136         unigram_prob = unigram_evaluate_log_prob(test_data
137 , './Models/unigram_model.json')
138         bigram_prob = bigram_evaluate_log_prob(test_data,
139 './Models/bigram_model.json')
140         bigram_prob_add_one_smoothing =

```

```
137 bigram_evaluate_log_prob_add_one_smoothing(test_data, './  
Data/pre_processed_training_data.txt')  
138  
139     print('Unigram Log Probability:', unigram_prob)  
140     print('Bigram Log Probability:', bigram_prob)  
141     print('Bigram with add one smoothing Log  
Probability:', bigram_prob_add_one_smoothing)  
142  
143     total_token_test_data = sum([len(sentence) - 1 for  
sentence in test_data])  
144     unigram_ppl = calculate_ppl(unigram_prob,  
total_token_test_data)  
145     bigram_ppl = calculate_ppl(bigram_prob,  
total_token_test_data)  
146     bigram_add_one_ppl = calculate_ppl(  
bigram_prob_add_one_smoothing, total_token_test_data)  
147  
148     print('Unigram Perplexity:', unigram_ppl)  
149     print('Bigram Perplexity:', bigram_ppl)  
150     print('Bigram add one Perplexity:',  
bigram_add_one_ppl)  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179
```

180

181

```

1 import collections
2 import json
3 from collections import defaultdict
4
5
6 def file_to_list_list(corpus):
7     return [sentence.split(' ') for sentence in corpus.read()
8         ().split('\n')]
9
10 def count_token(data: list[list], subsequence=1, skip=0
11 ) -> collections.defaultdict:
12     token_counter = defaultdict(int)
13     if subsequence == 1:
14         for sentence in data:
15             for i in range(skip, len(sentence)):
16                 token = sentence[i]
17                 token_counter[token] = token_counter.get(
18                     token, 0) + 1
19     else:
20         return ngram_count_token(data, subsequence)
21     return token_counter
22
23
24 def ngram_count_token(data: list[list], ngram=2, skip=0
25 ) -> collections.defaultdict:
26     """
27     You should always do defaultdict.get(key, 0) for all
28     ngram tuple count retrieval because the returned
29     defaultdict
30     does not contain the ngram parameters that have 0 count
31     for saving memory purpose
32     :param skip: number of token to be skipped in each
33     sentence, mainly used to ignore <s>
34     :param data:
35     :param ngram: 2 for bigram, 3 for trigram
36     :return: a defaultdict counter
37     """
38     token_counter = defaultdict(int)
39     for sentence in data:
40         for i in range(ngram - 1 + skip, len(sentence)):
41             token_seq = tuple([sentence[j] for j in range(i
42 - ngram + 1, i + 1)])
43             token_counter[token_seq] = token_counter.get(
44                 token_seq, 0) + 1
45     return token_counter
46
47
48 def load_language_model(model_path):

```

```
41     with open(model_path) as model:
42         return defaultdict(int, json.load(model))
43
44
45 def save_language_model(model, model_path):
46     with open(model_path, 'w') as model_file:
47         json.dump(model, model_file)
48
49
50 def data_list_to_str(data: list[list]):
51     return '\n'.join([' '.join(sentence) for sentence in
52                     data])
53
54 def map_test_not_in_training_unk(test_path, training_path):
55     with open(test_path, 'r') as test_file:
56         with open(training_path) as training_file:
57             training_token_counter = count_token(
58                 file_to_list_list(training_file))
59             test_data_list = file_to_list_list(test_file)
60
61             for sentence in test_data_list:
62                 for i in range(len(sentence)):
63                     if sentence[i] not in
64                         training_token_counter:
65                         sentence[i] = '<unk>'
66
67
68
69 def calculate_ppl(log_prob, total_token):
70     return 2 ** (-1 * (log_prob / total_token))
71
```

```

1 import math
2
3 from utils import *
4
5
6 def train_unigram(preprocessed_data_path, model_save_path,
7     skip=0):
8     with open(preprocessed_data_path) as training_corpus:
9         training_data = file_to_list_list(training_corpus)
10        token_counter = count_token(training_data, skip=
11            skip)
12        token_total = sum(token_counter.values())
13        # unigram_prob = {token: f'{token_counter[token] / token_total}' for token in token_counter.keys()}
14        unigram_prob = {token: token_counter[token] / token_total for token in token_counter.keys()}
15        save_language_model(unigram_prob, model_save_path)
16
17
18 def unigram_evaluate_log_prob(data_list_list: list[list],
19     unigram_model_path):
20     unigram_model = load_language_model(unigram_model_path)
21     prob = 0
22     for sentence in data_list_list:
23         for token in sentence:
24             print(token)
25             print(unigram_model.get(token, 0))
26             print(math.log2(unigram_model.get(token, 0)))
27             prob += math.log2(unigram_model.get(token, 0))
28
29
30
31 # data_list_list = [
32 #     ['a', 'unigram', 'maximum', 'likelihood', 'model',
33 #     '...', '</s>'],
34 #     ['a', 'bigram', 'maximum', 'likelihood', 'model',
35 #     '...', '</s>'],
36 #     ['a', 'bigram', 'model', 'with', 'add-one', 'smoothing',
37 #     '...', '</s>']
38 # ]
39
40 # train_unigram('./Test/test_training.txt', './Test/
41 # test_model.json')
42 train_unigram('./Data/pre_processed_training_data.txt', './
43 Models/unigram_model1.json')
44
45

```

```

1 ## Instructions
2
3 - For questions in 1.3, there is python file named 'questions.py' which include the code for each question. For example, the method q1() is the code for question 1 in section 1.3
4
5 - utils.py contains some utility methods to train language models, such as `count_token()`, `ngram_count_token()`, `file_to_list_list()`, etc.. Majority of the methods have docstring written.
6
7 - While reading my code, you will see `defaultdict` rather than regular `dict` because the method `get()` in `defaultdict` is very useful and convenient to prevent parameters not shown in the training or test corpus to occupy memories
8
9 - In other word, my code handles the parameters with zero probability dynamically rather than save it into counter or model before using it
10
11 ## To Run the Program
12
13 ### Preprocessing
14
15 You need to preprocess training file and test file before training and evaluating, and the file `pre_process.py` can help you do that
16
17 - <b>Preprocess Training data: </b>
  pre_process_training_data(training_data_path: str,
  pre_processed_data_path: str)
18 - <b>Preprocess Test data: </b> pre_process_test_data(
  data_path: str, save_path: str, pre_processed_training_path
)
19 - <b>Only lower and pad the file (used for q3): </b>
  lower_and_pad_file(data_path: str, save_path: str)
20
21 ### Train Langauge Model
22
23 ###### Unigram Model
24
25 Use method in `unigram_model.py`: train_unigram(
  preprocessed_data_path, model_save_path, skip=0), skip is used to skip the words in each sentence, mainly used to skip `<s>`
26
27 ##### Bigram Model
28

```

```
29 Use method in `bigram_model.py`: train_bigram(  
    preprocessed_data_path, model_save_path, skip=0)  
30  
31 ##### Bigram Model with add one smoothing  
32  
33 Use method in `bigram_model.py`:  
    bigram_evaluate_log_prob_add_one_smoothing(data_list_list:  
        list[list], training_data_path: str, skip=0), this one does  
        not return model, but create it during evaluation.  
34  
35  
36 #### Section 1.3  
37  
38 As stated above, `questions.py` is for section 1.3  
    questions. Each method in the file contains specific steps  
39  
40  
41  
42
```

```

1 import math
2
3 from utils import *
4 from collections import defaultdict
5
6
7 def train_bigram(preprocessed_data_path, model_save_path,
8     skip=0):
9     """
10         train bigram language model, the start sentence padding
11         <s> is ignored
12         :param skip: use to skip first token, mainly for <s>
13         :param preprocessed_data_path:
14         :param model_save_path:
15         :return:
16         """
17
18     with open(preprocessed_data_path) as training_corpus:
19         training_data = file_to_list_list(training_corpus)
20         token_counter = count_token(training_data)
21         bigram_token_counter = ngram_count_token(
22             training_data, skip=skip)
23
24         bigram_prob = {}
25         for token_seq in bigram_token_counter.keys():
26             bigram_prob[f'{token_seq[0]} {token_seq[1]}'] =
27                 bigram_token_counter[token_seq] / token_counter[
28                     token_seq[0]]
29             # bigram_prob[f'{token_seq[0]} {token_seq[1]}'] =
30                 f'{bigram_token_counter[token_seq]} / {token_counter[
31                     token_seq[0]}''
32
33         save_language_model(bigram_prob, model_save_path)
34
35
36 def bigram_evaluate_log_prob(data_list_list: list[list],
37     bigram_model_path):
38     bigram_model = load_language_model(bigram_model_path)
39     prob = 0
40     for sentence in data_list_list:
41         for i in range(1, len(sentence)):
42             bigram_token = f'{sentence[i - 1]} {sentence[i]}'
43
44             try:
45                 # print(bigram_token, bigram_model.get(
46                 bigram_token, 0), math.log2(bigram_model.get(bigram_token,
47                     0)))
48                 prob += math.log2(bigram_model.get(
49                     bigram_token, 0))
50             except:
51                 # print(bigram_token, 0, 0)

```

```

40             prob += 0
41
42     return prob
43
44
45 def bigram_evaluate_log_prob_add_one_smoothing(
46     data_list_list: list[list], training_data_path: str, skip=0
47 ):
48     training_data = None
49     with open(training_data_path) as training:
50         training_data = file_to_list_list(training)
51     bigram_token_counter = ngram_count_token(training_data
52 , skip=skip)
53     unigram_token_counter = count_token(training_data)
54     del unigram_token_counter['<s>']
55
56     total_token_types = len(unigram_token_counter.keys())
57     total_bigram_token_types = total_token_types ** 2
58     # print(total_token_types, total_bigram_token_types)
59     prob = 0
60
61     for sentence in data_list_list:
62         for i in range(1, len(sentence)):
63             bigram_token = (sentence[i - 1], sentence[i])
64
65             token_prob = (bigram_token_counter.get(
66                 bigram_token, 0) + 1) \
67                 / \
68                 (unigram_token_counter.get(
69                 sentence[i - 1], 0) + total_bigram_token_types)
70             # print(bigram_token_counter.get(bigram_token,
71             # 0) + 1)
72             # print((unigram_token_counter.get(sentence[i
73             - 1], 0) + total_bigram_token_types))
74             # print(bigram_token, token_prob, math.log2(
75             token_prob))
76             prob += math.log2(token_prob)
77
78     return prob
79
80
81
82 train_bigram('./Data/pre_processed_training_data.txt', './
83 Models/bigram_model1.json', skip=1)
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
790
791
792
793
794
795
796
797
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1490
1491
1492
1493
1494
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1590
1591
1592
1593
1594
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1890
1891
1892
1893
1894
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207

```