주어진 임의의 문장 POS 태깅하는 프로그램 작성

방법1)

universal tagset 이용하여 tagging

```
In [2]: brown_tagged_sents = brown.tagged_sents(tagset='universal')
        brown_sents = brown.sents()

In [3]: # split train/test data
        size = int(len(brown_tagged_sents)*0.9)
        train_sents = brown_tagged_sents[:size]
        test_sents = brown_tagged_sents[size:]
        size
Out[3]: 51606

In [4]: t0 = nltk.DefaultTagger('UNK')
        t1 = nltk.UnigramTagger(train_sents, backoff=t0)
        t2 = nltk.BigramTagger(train_sents, backoff=t1)
        t3 = nltk.TrigramTagger(train_sents, backoff=t2)
```

Universal tagset을 이용하여 문장을 tagging 한다. 우선 성능을 테스트하기 위해, train data와 test data로 나누었으며 back off n-gram 태거에 기반하여 브라운 전체코퍼스를 트레이닝하여 개발하였다.

정확도

```
In [5]: t1.evaluate(test_sents)
Out[5]: 0.9156346262651662

In [6]: t2.evaluate(test_sents)
Out[6]: 0.9245510362314285

In [7]: t3.evaluate(test_sents)
Out[7]: 0.9230527440749355
```

T1(Unigram 까지) : 91.5 %

T2(Bigram 까지) : 92.45%

T3(Trigram 까지) : 92.3%

⇨ Trigram까지 back off 태깅한 것 보다 Bigram까지만 backoff 한것이 성능이 더 좋았다. 따라서 t2를 이용해 예제시문장들을 테스트해보자.

예시문장들

In [8]:
```
text = input('Enter a sentence: ')
text = nltk.word_tokenize(text)
t2.tag(text)
print('Pos Tagging result : ',end="")
for i in t2.tag(text):
    print(i[1], end=" ")
```

```
Enter a sentence: time flies like an arrow.
Pos Tagging result : NOUN VERB ADP DET NOUN .
```

-> time flies like an arrow 가 제대로 태깅된 것을 확인할 수 있다.

그밖의 예시문장들

In [9]:
```
text = input('Enter a sentence: ')
text = nltk.word_tokenize(text)
t2.tag(text)
print('Pos Tagging result : ',end="")
for i in t2.tag(text):
    print(i[1], end=" ")
```

```
Enter a sentence: We will see that the tag of a word depends on the word and its context within a sentence. Fo
r this reason, we will be working with data at the level of sentences rather than words.
Pos Tagging result : PRON VERB VERB ADP DET NOUN ADP DET NOUN VERB ADP DET NOUN CONJ DET NOUN ADP DET NOUN . A
DP DET NOUN . PRON VERB VERB VERB ADP NOUN ADP DET NOUN ADP NOUN ADP ADP NOUN .
```

n [10]:
```
text = input('Enter a sentence: ')
text = nltk.word_tokenize(text)
t2.tag(text)
print('Pos Tagging result : ',end="")
for i in t2.tag(text):
    print(i[1], end=" ")
```

```
Enter a sentence: Tagged corpora use many different conventions for tagging words.
Pos Tagging result : UNK UNK NOUN ADJ ADJ NOUN ADP VERB NOUN .
```

방법2)

기본 tagset을 이용하여 tagging

```
In [11]: from nltk.corpus import brown
         brown_tagged_sents = brown.tagged_sents()
         brown_sents = brown.sents()

In [12]: size = int(len(brown_tagged_sents)*0.9)
         train_sents = brown_tagged_sents[:size]
         test_sents = brown_tagged_sents[size:]
         size
Out[12]: 51606

In [13]: tt1 = nltk.UnigramTagger(train_sents, backoff=nltk.DefaultTagger('NN'))
         tt2 = nltk.BigramTagger(train_sents, backoff=tt1)
         tt3 = nltk.TrigramTagger(train_sents, backoff=tt2)
```

정확도

```
In [14]: tt1.evaluate(test_sents)
Out[14]: 0.8912742817627459

In [15]: tt2.evaluate(test_sents)
Out[15]: 0.9125751765470128

In [16]: tt3.evaluate(test_sents)
Out[16]: 0.9130466670857693
```

tt1(Unigram 까지) : 약 89.12 %

tt2(Bigram 까지) : 약 91.25%

tt3(Trigram 까지) : 약 91.30%

- ⇨ 앞서 1번에서 tagging한 프로그램보다는 성능이 떨어지는 것을 확인할 수 있다.
- ⇨ 이번에는 Trigram까지 back off 태깅한 것이 Bigram까지만 backoff 한것보다 성능이 더 좋았다. 따라서 tt3를 이용해 예제시문장들을 테스트해보자.

예시문장

```
In [17]: text = input('Enter a sentence: ')
         text = nltk.word_tokenize(text)
         tt3.tag(text)
         print('Pos Tagging result : ',end="")
         for i in tt3.tag(text):
             print(i[1], end=" ")

         Enter a sentence: time flies like an arrow.
         Pos Tagging result : NN NNS VB AT NN .
```

이번에는 원하는대로 태깅이 되지 않았다. 그렇다면 tt2를 이용하여 태깅해보자.

```
In [18]: text = input('Enter a sentence: ').split()
         tt2.tag(text)
         print('Pos Tagging result : ',end="")
         for i in tt2.tag(text):
             print(i[1], end=" ")
```

```
Enter a sentence: time flies like an arrow.
Pos Tagging result : NN NNS CS AT NN
```

tt2도 원하는 대로 태깅이 되지 않았다.

그밖의 예시문장들

```
In [19]: text = input('Enter a sentence: ').split()
         tt2.tag(text)
         print('Pos Tagging result : ',end="")
         for i in tt2.tag(text):
             print(i[1], end=" ")
```

```
Enter a sentence: In the beginning God created the heaven and the earth. And the earth was without form, and v
oid; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And
God said, Let there be light: and there was light. And God called the light Day, and the darkness he called Ni
ght. And the evening and the morning were the first day. And God made the firmament, and divided the waters wh
ich were under the firmament from the waters which were above the firmament: and it was so. Thus the heavens a
nd the earth were finished, and all the host of them. But there went up a mist from the earth, and watered the
whole face of the ground.
Pos Tagging result : IN AT NN NP VBD AT NN CC AT NN CC AT NN BEDZ IN NN CC NN CC NN BEDZ IN AT NN IN AT NN CC
AT NN-TL IN-TL NP-TL VBD IN AT NN IN AT NN CC NP NN VB RB BE NN CC EX BEDZ NN CC NP VBD AT NN NN CC AT NN PPS
VBD NN CC AT NN CC AT NN BED AT OD NN CC NP VBD AT NN CC VBN AT NNS WDT BED IN AT NN IN AT NNS WDT BED IN AT N
N CC PPS BEDZ NN RB AT NNS CC AT NN BED NN CC ABN AT NN IN NN CC EX VBD RP AT NN IN AT NN CC VBD AT JJ NN IN A
T NN
```

결론

1번 방법으로 한 tagging의 정확도가 더 좋았으며 time flies like an arrow도 제대로 태깅이 되었다.

2번 방법은 1번 방법보다 정확도가 떨어졌으며 time flies like an arrow가 제대로 태깅이 되지 않았다.