PROBLEM-1

In [1]:

```
def read_file(path):
    tokens = []
    tags = []
    with open(path, 'r') as file:
        recent_tok = []
        recent_tag = []
        for line in file:
            line = line.strip()
            if line:
                parts = line.split('\t')
                if len(parts) == 2:
                    token, tag = parts
                    recent_tok.append(token)
                    recent_tag.append(tag)
            else:
                if recent_tok and recent_tag:
                    tokens.append(recent_tok)
                    tags.append(recent_tag)
                recent_tok = []
                recent_tag = []
        if recent_tok and recent_tag:
            tokens.append(recent_tok)
            tags.append(recent_tag)
    return tokens, tags

# Read the train.tsv and test.tsv files
train_tokens, train_tags = read_file('C:/Users/mano2/Downloads/train.tsv')
test_tokens, test_tags = read_file('C:/Users/mano2/Downloads/test.tsv')
#print(test_tokens)
#print(test_tags)
#print(train_tokens)
#print(train_tags)
print(f"Number of sequences in train: {len(train_tokens)}")
print(f"Number of sequences in test: {len(test_tokens)}")

# Print the tokens and tags of the first sequence in the training dataset
print("First sequence of tokens in training dataset:")
```

```
print(train_tokens[0])
print("First sequence of tokens in training dataset:")
print(train_tags[0:1])
```

```
Number of sequences in train: 5432
Number of sequences in test: 940
First sequence of tokens in training dataset:
['Identification', 'of', 'APC2', ',', 'a', 'homologue', 'of', 'the', 'adenomatous', 'polyposis', 'coli', 'tumour', 'suppresso
r', '.']
First sequence of tokens in training dataset:
[['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'O', 'O']]
```

PROBLEM-2

In [2]:

```
from collections import Counter

# Count the occurrences of each tag in the training dataset
tag_counts = Counter(tag for tags in train_tags for tag in tags)
print("Tag counts in the training data:")
print(tag_counts)

# Create a list of (token, tag) pairs for words associated with "B-Disease" or "I-Disease"
#disease_tokens = [(train_tok[i][j], train_tag[i][j]) for i in range(len(train_tok)) for j in range(len(train_tok[i]))
                # if train_tag[i][j] in ["B-Disease", "I-Disease"]]
disease_tokens = [(token, tag) for token_seq, tag_seq in zip(train_tokens, train_tags) for token, tag in zip(token_seq, tag_se
# Count the occurrences of tokens associated with "B-Disease" or "I-Disease"
disease_token_counts = Counter(token for token, tag in disease_tokens)
print("\nThe most common 20 tokens with tag 'B-Disease' or 'I-Disease':")
common_tokens_list = [token for token, _ in disease_token_counts.most_common(20)]
print(common_tokens_list)

#for i in range(4):
   # print(train_tokens[i],"\n")
   # print(train_tags[i],"\n")
```

```
Tag counts in the training data:
Counter({'O': 124819, 'I-Disease': 6122, 'B-Disease': 5145})

The most common 20 tokens with tag 'B-Disease' or 'I-Disease':
['-', 'deficiency', 'syndrome', 'cancer', 'disease', 'of', 'dystrophy', 'breast', 'ovarian', 'X', 'and', 'DM', 'ALD', 'DMD', 'A
PC', 'disorder', 'muscular', 'G6PD', 'linked', 'the']
```

PROBLEM-3

In [3]:

```python
def token_features(tokens, position):
    features = []
    current_word = tokens[position]

    features.append(f'w0.lower ={current_word.lower()}')

    features.append(f'w0.suffix3 = {current_word[-3:]}')

    previous_word = tokens[position - 1] if position > 0 else "BOS"
    features.append(f'w0.previous ={previous_word}')

    next_word = tokens[position + 1] if position < len(tokens) - 1 else "EOS"
    features.append(f'w0.next = {next_word}')

    features.append(f'w0.length = {str(len(current_word))}')
    features.append(f'w0.isdigit = {current_word.isdigit()}')
    features.append(f'w0.istitle = {current_word.istitle()}')

    return features

for i in range(3):
    features = token_features(train_tokens[0], i)
    print(features)

# print("\n test data features")
# for i in range(3):
#     features = extract_features(test_tokens[0], i)
#     print(features)
```

```
['w0.lower =identification', 'w0.suffix3 = ion', 'w0.previous =BOS', 'w0.next = of', 'w0.length = 14', 'w0.isdigit = False', 'w
0.istitle = True']
['w0.lower =of', 'w0.suffix3 = of', 'w0.previous =Identification', 'w0.next = APC2', 'w0.length = 2', 'w0.isdigit = False', 'w
0.istitle = False']
['w0.lower =apc2', 'w0.suffix3 = PC2', 'w0.previous =of', 'w0.next = ,', 'w0.length = 4', 'w0.isdigit = False', 'w0.istitle = F
alse']
```

PROBLEM-4

In [4]:

```python
import pycrfsuite
from sklearn.metrics import classification_report

def train_crf(train_tokens, train_tags):

    train_data = zip(train_tokens, train_tags)

    trainer = pycrfsuite.Trainer(verbose=False)

    for tokens, labels in list(train_data):
        x_seq = [token_features(tokens, i) for i in range(len(tokens))]
        trainer.append(x_seq, labels)

    trainer.set_params({
        'c1': 1.0,  # Coefficient for L1 penalty
        'c2': 1e-3,  # Coefficient for L2 penalty
        'max_iterations': 100,  # Maximum number of iterations
        'feature.possible_transitions':False
    })

    model_file = 'crf_model.crfsuite'

    trainer.train(model_file)
    return model_file

# Train the CRF model using your training data
model_file = train_crf(train_tokens, train_tags)

# Trained CRF model to  test dataset
tagger = pycrfsuite.Tagger()
tagger.open(model_file)
```

```
predicted_tags = []
for tokens in test_tokens:
    features = [token_features(tokens, i) for i in range(len(tokens))]
    predicted_tags.append(tagger.tag(features))


# Flatten true and predicted tags
true_tags = [tag for tag_seq in test_tags for tag in tag_seq]
flat_predicted_tags = [tag for tag_seq_v2 in predicted_tags for tag in tag_seq_v2]

# Create a classification report
report = classification_report(true_tags, flat_predicted_tags, target_names=["B-Disease", "I-Disease", "O"])
print(report)
```

```
              precision    recall  f1-score   support

   B-Disease       0.86      0.72      0.78       960
   I-Disease       0.85      0.75      0.80      1087
           O       0.98      0.99      0.99     22450

    accuracy                           0.97     24497
   macro avg       0.90      0.82      0.86     24497
weighted avg       0.97      0.97      0.97     24497
```

PROBLEM-5

In [5]:

```
from collections import Counter
transitions = tagger.info().transitions
# print(info)
for trans, weigth in transitions.items():
    print(f"transitions: {trans}, Weigth: {weigth}")

features = tagger.info().state_features

print("\n State features weights:")
for feature, weigth in features.items():
    if(feature[0][:10] == 'w0.istitle'):
        print(f"Feature: {feature}, Weigh: {weigth}")
```

```
transitions: ('O', 'O'), Weigth: 3.031935
transitions: ('O', 'B-Disease'), Weigth: 3.484415
transitions: ('B-Disease', 'O'), Weigth: -4.225638
transitions: ('B-Disease', 'B-Disease'), Weigth: -5.413926
transitions: ('B-Disease', 'I-Disease'), Weigth: 3.826891
transitions: ('I-Disease', 'O'), Weigth: -4.930928
transitions: ('I-Disease', 'B-Disease'), Weigth: -4.695163
transitions: ('I-Disease', 'I-Disease'), Weigth: 3.179538

 State features weights:
Feature: ('w0.istitle = True', 'O'), Weigh: -0.00035
Feature: ('w0.istitle = True', 'B-Disease'), Weigh: -0.004644
Feature: ('w0.istitle = True', 'I-Disease'), Weigh: -0.728516
Feature: ('w0.istitle = False', 'O'), Weigh: -0.019489
Feature: ('w0.istitle = False', 'B-Disease'), Weigh: -0.40535
Feature: ('w0.istitle = False', 'I-Disease'), Weigh: -0.62905
```

PROBLEM-6

In [6]:

```python
from sklearn.metrics import precision_score, recall_score

def document_level_precision_recall(test_tags):
    p=[]
    key_words = ['B-Disease', 'I-Disease']

    def check_existence(sublist):
        return any(word in sublist for word in key_words)

    for sublist in test_tags:
        if check_existence(sublist):
            p.append(1)
        else:
            p.append(0)

    return p

true_tags = document_level_precision_recall(test_tags)
predicted_tags = document_level_precision_recall(predicted_tags)

precision =  precision_score(true_tags, predicted_tags)
```

```
recall =  recall_score(true_tags, predicted_tags)

print(f"Document-level Precision: {precision:.2f}")
print(f"Document-level Recall: {recall:.2f}")
```

```
Document-level Precision: 0.97
Document-level Recall: 0.88
```