AIN442 Practicum in Natural Language Processing
BBM497 Introduction to Natural Language Processing Laboratory

Programming Assignment 1 – Byte-Pair Encoding Subword Tokenization
Due Date: March 10, 2025 (Monday)

You have to write a Python program (you can only use ***re*** and ***codecs*** libraries) which will implement a variation of Byte-Pair Encoding (BPE) algorithm for subword tokenization. Your program should have the following two parts.

1. **Token Learner**: The token learner that takes a raw training corpus as an input and induces a vocabulary (a set of tokens), a list of merges that are learned while creating the vocabulary and the tokenized corpus. The top-level methods of the token learner should take the training corpus as a string (or a filename) and it should return a triple **(Merges,Vocabulary,TokenizedCorpus)** as an output. If the training corpus is given in a file (a filename), the token learner should read the corpus from this "utf-8" file. The top-level methods of the token learner may take the maximum merge count as an optional argument (you should assume that the default maximum merge count is 10).
2. **Token Segmenter**: The token segmenter takes a text (a string) and **Merges** that is learned by the token learner and returns the tokenized text according to the learned vocabulary using the learned merges.

Your python program should work as described below.

Token Learner:

- Your token learner should have the following two top-level methods:

  `def bpeCorpus(corpus,maxMergeCount = 10):` and

  `def bpeFN(fn,maxMergeCount = 10):`

- The method **`bpeCorpus`** takes the training corpus as a string (the first argument **`corpus`**) and the method **`bpeFN`** takes the training corpus as the name (the first argument **`fn`**) of the file holding the training corpus. The method **`bpeFN`** reads the content of the given file (a "utf-8" file) as a string which will be the training corpus.
- The second argument of these two methods is the optional maximum merge count **`maxMergeCount`** which indicates the amount of maximum merge operations will be performed by the token learner. The actual merge operations will be performed can be less than this maximum value if all possible merge operations are performed by the token learner. You should assume that the default value for this second argument is 10.
- First, these methods should separate the given corpus into a list of white-space separated tokens. Your token learner should NOT merge two subword tokens from two different white-space separated words. These methods should create an initialized tokenized corpus which a list of subword tokenized of words. Initial subword tokens of a word is a list of its characters together with special beginning (space character) and ending (underbar character) word characters. For example, the initial subword tokens of the training corpus string **"**`sos ses sus`**"** should be as follows before any merge operations.

```
[[' ', 's', 'o', 's', '_'],
 [' ', 's', 'e', 's', '_'],
 [' ', 's', 'u', 's', '_']]
```

- You can create the initial vocabulary by using the list below.

```
list("abcçdefgğhıijklmnoöprsştuüvyzwxq”+
     "ABCÇDEFGĞHIİJKLMNOÖPRSŞTUÜVYZWXQ”+
     "0123456789 !'^#+$%&/{([)]=}*?\\_-<>|.:´,;`@€¨~\"é”)
```

- Both of these methods should return a triple in the following form.

  (Merges,Vocabulary,TokenizedCorpus)

- **Merges** is a list of merges that are performed by the token learner. The merges must be in the learning order. Each merge in **Merges** is a pair of merged bigram and its count. For example, the merge `(('s','_'),6))` indicates that the subword tokens `'s'` and `'_'` are merged in order to obtain the subword token `'s_'` and the bigram `('s','_')` occurred 6 times in the tokenized corpus before this merge operation.
  - Remember that your token learner should merge the bigram with the highest count. If there are more than one bigram with the highest count, the alphabetically first bigram should be selected for the merge operation.
- **Vocabulary** is a list of all subword tokens that are learned by the token learner. It should contain initial character tokens followed by the learned tokens in the learning order.
- **TokenizedCorpus** is a list and each item in that is also a list holding the subword tokens of the white-space separated token.

- The following working examples describe how your token learner should work.

Example 1:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses",6)
print(Merges)
[((' ', 's'), 6), (('s', '_'), 6), ((' s', 'o'), 3), ((' so', 's_'), 3),
 ((' s', 'e'), 2), ((' se', 's_'), 2)]
print(Vocabulary)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z', 'w', 'x', 'q',
'A', 'B', 'C', 'Ç', 'D', 'E', 'F', 'G', 'Ğ', 'H', 'I', 'İ', 'J', 'K', 'L', 'M',
'N', 'O', 'Ö', 'P', 'R', 'S', 'Ş', 'T', 'U', 'Ü', 'V', 'Y', 'Z', 'W', 'X', 'Q',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '!', "'", '^', '#', '+',
'$', '%', '&', '/', '{', '(', '[', ')', ']', '=', '}', '*', '?', '\\', '_', '-',
'<', '>', '|', '.', ':', '´', ',', ';', '`', '@', '€', '¨', '~', '"', 'é', ' s',
's_', ' so', ' sos_', ' se', ' ses_']
print(TokenizedCorpus)
[[' sos_'], [' ses_'], [' sos_'], [' s', 'u', 's_'], [' sos_'], [' ses_']]
```

- Since 6 is given as the maximum merge count, the maximum 6 merge operations are performed in this example.

Example 2:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses")
print(Merges)
[((' ', 's'), 6), (('s', '_'), 6), ((' s', 'o'), 3), ((' so', 's_'), 3), ((' s',
'e'), 2), ((' se', 's_'), 2), ((' s', 'u'), 1), ((' su', 's_'), 1)]
```

```
print(Vocabulary)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z', 'w', 'x', 'q',
'A', 'B', 'C', 'Ç', 'D', 'E', 'F', 'G', 'Ğ', 'H', 'I', 'İ', 'J', 'K', 'L', 'M',
'N', 'O', 'Ö', 'P', 'R', 'S', 'Ş', 'T', 'U', 'Ü', 'V', 'Y', 'Z', 'W', 'X', 'Q',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '!', "'", '^', '#', '+',
'$', '%', '&', '/', '{', '(', '[', ')', ']', '=', '}', '*', '?', '\\', '_', '-',
'<', '>', '|', '.', ':', '´', ',', ';', '`', '@', '€', '¨', '~', '"', 'é', ' s',
's_', ' so', ' sos_', ' se', ' ses_', ' su', ' sus_']
print(TokenizedCorpus)
[[' sos_'], [' ses_'], [' sos_'], [' sus_'], [' sos_'], [' ses_']]
```

- Since the second argument is not given, the maximum merge count is assumed to be 10. Since the number of all possible merges is 8 for this training corpus, 8 merge operations (less than 10) are performed in this example. In this case, each word in the tokenized corpus is a singleton list containing a subword token consisting of characters of that whitespace separated token together with token beginning character (space character) and token ending character (_).

Example 3:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses",0)
print(Merges)
[]
print(Vocabulary)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z', 'w', 'x', 'q',
'A', 'B', 'C', 'Ç', 'D', 'E', 'F', 'G', 'Ğ', 'H', 'I', 'İ', 'J', 'K', 'L', 'M',
'N', 'O', 'Ö', 'P', 'R', 'S', 'Ş', 'T', 'U', 'Ü', 'V', 'Y', 'Z', 'W', 'X', 'Q',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '!', "'", '^', '#', '+',
'$', '%', '&', '/', '{', '(', '[', ')', ']', '=', '}', '*', '?', '\\', '_', '-',
'<', '>', '|', '.', ':', '´', ',', ';', '`', '@', '€', '¨', '~', '"', 'é']
print(TokenizedCorpus)
[[' ', 's', 'o', 's', '_'], [' ', 's', 'e', 's', '_'], [' ', 's', 'o', 's', '_'],
[' ', 's', 'u', 's', '_'], [' ', 's', 'o', 's', '_'], [' ', 's', 'e', 's', '_']]
```

- Since 0 is given as the maximum merge count, no merge operations are performed in this example. In this case, the vocabulary is the initial character set and each word in the tokenized corpus is the list of characters of that whitespace separated token together with token beginning character (space character) and token ending character (_).

Example 4:

- If the file "hw01_tiny.txt" contains the following text, your token learner should work as follows.

```
He came, I came.
I came and HE came.
They came, I came.
I did not come, they came.
```

```
(Merges,Vocabulary,TokenizedCorpus)=bpeFN("hw01_tiny.txt")
print(Merges)
[((' ', 'c'), 8), (('m', 'e'), 8), ((' c', 'a'), 7), ((' ca', 'me'), 7), ((' ',
'I'), 4), ((' I', '_'), 4), ((' came', '.'), 4), ((' came.', '_'), 4), ((',', '_'),
3), ((' ', 'H'), 2)]
```

```
print(Vocabulary)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z', 'w', 'x', 'q',
'A', 'B', 'C', 'Ç', 'D', 'E', 'F', 'G', 'Ğ', 'H', 'I', 'İ', 'J', 'K', 'L', 'M',
'N', 'O', 'Ö', 'P', 'R', 'S', 'Ş', 'T', 'U', 'Ü', 'V', 'Y', 'Z', 'W', 'X', 'Q',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '!', "'", '^', '#', '+',
'$', '%', '&', '/', '{', '(', '[', ')', ']', '=', '}', '*', '?', '\\', '_', '-',
'<', '>', '|', '.', ':', '´', ',', ';', '`', '@', '€', '¨', '~', '"', 'é', ' c',
'me', ' ca', ' came', ' I', ' I_', ' came.', ' came._', ',_', ' H']
print(TokenizedCorpus)
[[' H', 'e', '_'], [' came', ',_'], [' I_'], [' came._'], [' I_'], [' came', '_'],
[' ', 'a', 'n', 'd', '_'], [' H', 'E', '_'], [' came._'], [' ', 'T', 'h', 'e', 'y',
'_'], [' came', ',_'], [' I_'], [' came._'], [' I_'], [' ', 'd', 'i', 'd', '_'],
[' ', 'n', 'o', 't', '_'], [' c', 'o', 'me', ',_'], [' ', 't', 'h', 'e', 'y', '_'],
[' came._']]
```

Example 5:

- If the file "`hw01_tiny.txt`" contains the following text, your token learner should work as follows.

```
He came, I came.
I came and HE came.
They came, I came.
I did not come, they came.
```

```
(Merges,Vocabulary,TokenizedCorpus)=bpeFN("hw01_tiny.txt",100)
print(Merges)
[((' ', 'c'), 8), (('m', 'e'), 8), ((' c', 'a'), 7), ((' ca', 'me'), 7), ((' ',
'I'), 4), ((' I', '_'), 4), ((' came', '.'), 4), ((' came.', '_'), 4), ((',', '_'),
3), ((' ', 'H'), 2), ((' came', ',_'), 2), (('d', '_'), 2), (('e', 'y'), 2), (('ey',
'_'), 2), (('h', 'ey_'), 2), ((' ', 'T'), 1), ((' ', 'a'), 1), ((' ', 'd'), 1), ((' ',
'n'), 1), ((' ', 't'), 1), ((' H', 'E'), 1), ((' H', 'e'), 1), ((' HE', '_'),
1), ((' He', '_'), 1), ((' T', 'hey_'), 1), ((' a', 'n'), 1), ((' an', 'd_'), 1),
((' c', 'o'), 1), ((' came', '_'), 1), ((' co', 'me'), 1), ((' come', ',_'), 1),
((' d', 'i'), 1), ((' di', 'd_'), 1), ((' n', 'o'), 1), ((' no', 't'), 1), ((' 
not', '_'), 1), ((' t', 'hey_'), 1)]
print(Vocabulary)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z', 'w', 'x', 'q',
'A', 'B', 'C', 'Ç', 'D', 'E', 'F', 'G', 'Ğ', 'H', 'I', 'İ', 'J', 'K', 'L', 'M',
'N', 'O', 'Ö', 'P', 'R', 'S', 'Ş', 'T', 'U', 'Ü', 'V', 'Y', 'Z', 'W', 'X', 'Q',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '!', "'", '^', '#', '+',
'$', '%', '&', '/', '{', '(', '[', ')', ']', '=', '}', '*', '?', '\\', '_', '-',
'<', '>', '|', '.', ':', '´', ',', ';', '`', '@', '€', '¨', '~', '"', 'é', ' c',
'me', ' ca', ' came', ' I', ' I_', ' came.', ' came._', ',_', ' H', ' came,_',
'd_', 'ey', 'ey_', 'hey_', ' T', ' a', ' d', ' n', ' t', ' HE', ' He', ' HE_', '
He_', ' They_', ' an', ' and_', ' co', ' came_', ' come', ' come,_', ' di', ' did_',
' no', ' not', ' not_', ' they_']
print(TokenizedCorpus)
[[' He_'], [' came,_'], [' I_'], [' came._'], [' I_'], [' came_'], [' and_'], ['
HE_'], [' came._'], [' They_'], [' came,_'], [' I_'], [' came._'], [' I_'], ['
did_'], [' not_'], [' come,_'], [' they_'], [' came._']]
```

- Since the number of all possible merges is 37 for this training corpus, 37 merge operations (less than 100) are performed in this example.

Example Output Files:

- You can see the token learner results for the file "`hw01_bilgisayar.txt`" in the following files.
- "`hw01-output1.txt`" which obtained by the following method call and writing obtained results `(Merges,Vocabulary,TokenizedCorpus)` separately into that file.

  ```
  bpeFNToFile("hw01_bilgisayar.txt", 1000, "hw01-output1.txt")
  ```

- "`hw01-output2.txt`" which obtained by the following method call and writing obtained results `(Merges,Vocabulary,TokenizedCorpus)` separately into that file.

  ```
  bpeFNToFile("hw01_bilgisayar.txt", 200, "hw01-output2.txt")
  ```

NOTE: `bpeFNToFile(infn, maxMergeCount=10, outfn="output.txt")` function is the version of `bpeFN()` that does not return anything and instead writes to the output file.

Token Segmenter:

- Your token segmenter should have the following top-level method:

  `def bpeTokenize(str,merges):`
- Its first argument is a string to be subword tokenized and its second argument is the merges that are learned by the token learner.
- The following working examples describe how your token learner should work.

Example 6:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses",6)
print(Merges)
[((' ', 's'), 6), (('s', '_'), 6), ((' s', 'o'), 3), ((' so', 's_'), 3),
 ((' s', 'e'), 2), ((' se', 's_'), 2)]
tokenizedStr=bpeTokenize("sos sus ses sel fes araba",Merges)
print(tokenizedStr)
[[' sos_'], [' s', 'u', 's_'], [' ses_'], [' se', 'l', '_'], [' ', 'f', 'e', 's_'],
[' ', 'a', 'r', 'a', 'b', 'a', '_']]
```

Example 7:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses")
print(Merges)
[((' ', 's'), 6), (('s', '_'), 6), ((' s', 'o'), 3), ((' so', 's_'), 3), ((' s',
'e'), 2), ((' se', 's_'), 2), ((' s', 'u'), 1), ((' su', 's_'), 1)]
tokenizedStr=bpeTokenize("sos sus ses sel fes araba",Merges)
print(tokenizedStr)
[[' sos_'], [' sus_'], [' ses_'], [' se', 'l', '_'], [' ', 'f', 'e', 's_'], [' ',
'a', 'r', 'a', 'b', 'a', '_']]
```

Example 8:

```
(Merges,Vocabulary,TokenizedCorpus)=bpeCorpus("sos ses sos sus sos ses",0)
print(Merges)
[]
tokenizedStr=bpeTokenize("sos sus ses sel fes araba",Merges)
print(tokenizedStr)
[[' ', 's', 'o', 's', '_'], [' ', 's', 'u', 's', '_'], [' ', 's', 'e', 's', '_'],
[' ', 's', 'e', 'l', '_'], [' ', 'f', 'e', 's', '_'], [' ', 'a', 'r', 'a', 'b',
'a', '_']]
```

Hand in:

- You will submit your programming assignment using the HADI system. You have to upload a single zip file (.zip, gzip or .rar file) holding a single python program (.py file).
- The name of your python file should be **hw01-*NameLastname*.py** by replacing ***NameLastname*** with your actual first name and your last name. Similarly, the name of your zip file name should be the same name with .zip (.gzip or .rar) extension.
- You have to make sure that your python program contains the following four top-level method definitions and they should work as described as above examples.

```
def bpeCorpus(corpus,maxMergeCount = 10):
def bpeFN(fn,maxMergeCount = 10):
def bpeTokenize(str,merges):
def bpeFNToFile(infn, maxMergeCount=10, outfn="output.txt"):
```

- Your programming assignments will be tested the training corpora (including `hw01_tiny.txt` file) at above examples. But, your assignments will be also tested with other training corpora (different strings and different files) which are not given here.
- You can use `hw01_template.py` file as a starting reference or write your own code from scratch. However, make sure to have `bpeFNToFile()` function from the template file as is, and ensure that your output files match the sample output files.

Late Policy:

- You must submit your programming assignment before its due date.
- You may submit your assignment up to three days late, but with a penalty. A 10% penalty will be applied for each day late (Penalties: 1 day late: 10%, 2 days late: 20%, 3 days late: 30%)

DO YOUR PROGRAMMING ASSIGNMENTS YOURSELF!

- **Do not share your programming assignments with your friends.**
- **Do not use AI-tools (such as chatgpt) to do your programming assignments.**
- **Cheating will be punished.**