# CMPE300 PROJECT 2

# MPI PROGRAMMING PROJECT


## ALİ BAŞARAN 2020400357

## ÖMER ŞAFAK BEBEK 2019400180


## 20.12.2022

## Introduction:

The project is about calculating the data for a language model. A language model aims to calculate the probability distribution over a textual data. In this project we are calculating the data specifically for the bigram language model. A bigram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. We use adjacent words as bigrams in this project.

The project requests calculating the unigram and bigram frequencies in the textual data. The project also wants that calculations should be done by using the MPI framework. After calculating the frequencies, the program should use these frequencies to calculate the conditional probabilities for the given bigrams. The data will be in a .txt file whose name is specified in the arguments. The bigrams that are requested for calculating the probability are given in a .txt file whose name is specified in the arguments.

The rule of conditional probability is $P(A|B) = P(A \cap B)/P(B)$. For example, if a bigram is given in the test file, such as green turtle, the conditional probability of P(turtle | green) = P(green turtle) / P(green). This conditional probability can be estimated by using the two frequencies of the nominator and the denominator in a large textual data. So, the equation turns into P(turtle | green) = Freq(green turtle) / Freq(green).

Another constraint in the project is sharing the data among the worker processes. The number of processes is specified in the arguments. The data should be distributed among the worker processes equally. All processes should print their ranks and the number of sentences they received when they start to execute.

After execution of the worker processes, all the data should be gathered in the master process. There are two methods of gathering the data. One of the methods is that after each worker has finished calculating the data, they send the data directly to the master process. This method is named “MASTER”. Another method is implementing the data merging operation sequentially between the workers. The sequential merge occurs when the calculated data is not sent to the master process directly. Each worker process sends the calculated data to another worker process except the last worker process. In this project, each worker sends the data to the process that has a rank value one more than its rank. For example, a process that has rank 1 sends the data, which is calculated by itself, to the process that has rank 2. Process that has rank 2 sends the data, which is calculated by itself and the worker process that has rank 1, to the worker process that has rank 3. This process continues until all data is sent to the last worker process. After the last worker process finishes its calculations it sends all data to the master process. This merge method is called “WORKERS”. The merging method can be specified in the arguments.

When counting finishes in the worker processes, all data is sent to the master process. Then, the master process computes the conditional probabilities of the bigrams that are read from the input test file. The computation is made with the formula specified before. After computation, the master process should print bigrams and their conditional probabilities.

## Program Interface:

The program is run by the command “mpiexec -n <number of processes> python3 main.py --input_file <path of the input data file> --merge_method <merge method (MASTER or WORKERS)> --test_file <path of the test data file>”. The command includes mpiexec because the mpi is used in the project. Number of processes that the user wants to run parallel should be specified after the tag “-n”. One of these processes is called master, the others are called workers in this project. There must be exactly one master process and at least one worker process in the program, so the number of processes should be greater than one. The master process has the rank 0.

Oversubscription refers to the concept of allowing more ranks to be assigned to a host than the number of slots that are available on that host. There is an additional flag for oversubscription, which is called “- - oversubscribe”. If the user wants to use more slots than the device has, he should use the oversubscribe flag. Oversubscribe flag should be written after the command mpiexec and before the flag -n.

“python main.py” command is to run the program with python interpreter. The file that contains the source code is the main.py in the project, so the user should run the main.py with python interpreter. There are three arguments that are given to the source code.

The first argument is the path of the input data file. Input data file contains sample sentences. The program should calculate the number of bigrams and the unigrams the sentence includes. The path of the input data file should be given as an argument after the tag “- - input_file”.

The second argument that is given to source code is the merge method. There are two merge methods in this program. One of them is called “MASTER” while the other is called “WORKER”. The details of these execution methods will be discussed in the next sections. The brief execution of the merge method MASTER is that after each worker has finished calculating the data, they send the data directly to the master process. The brief execution of the merge method WORKERS is that data is sent to another worker process and the last worker process sends the whole data to the master process.

The last argument is the path of the test data file. Test data file contains bigrams that are requested to calculate their conditional probabilities. The program

calculates the conditional probabilities of the bigrams that are inside of the test data file. The path of the test data file should be given as an argument after the tag “- -test_file”.

In absence of one of these arguments, the program fails. In order to run the program error-free, these arguments should be given appropriately to the program.

## Program Execution:

The aim of the program is to calculate the conditional probabilities of the bigrams that the user specified. This calculation is made according to the data inside the input data file. First the program calculates the frequencies inside of the test input data, then it computes the conditional probability of the given bigrams inside of the input test file. The program outputs the result to the console after computations. In conclusion, the program takes two input files, which are the input data file and input test file, and gives output the conditional probabilities of the bigrams in the input test file to the terminal. The program uses parallel processes to calculate the frequencies of the unigrams and the bigrams in the input data file, which reduces the time that the user waits to the end of the program execution.

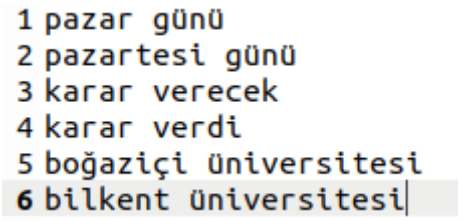
The program takes two input files and gives output to the terminal. The first input file is the input data file. Input data file should include the sentences that the user wants to calculate the frequencies of the unigrams and the bigrams. The path of the input file should be given in the arguments as stated in the interface section of the report. Sentences in the input data file should start with the <s> symbol and should end with </s> symbol. There must be at most one sentence in each line. The user must ensure these requirements in order to execute the program properly. There is a photo of the example input data file under this paragraph.

```
1 <s> türk halk müziği ve protest müziğin önemli isimlerinden selda bağcan 4 yıl aradan
  sonra çıkardığı albümünde 19 ocak 2007de silahlı saldırı sonucu hayatını kaybeden hrant
  dinki unutmadı bağcan albüme de adını veren güvercinleri de vururlar şarkısını hrant dinke
  adadı </s>
2 <s> güvercinleri de vururların söz ve müziği ise şehrazata ait </s>
3 <s> toplumsal duyarlılığı ve muhalifliğiyle tanınan selda bağcan daha önce uğur mumcuya
  ithafen de uğurlar olsunu seslendirmişti </s>
4 <s> meadow rain walker avukatı aracılığıyla verdiği dava dilekçesinde babasının
  çarpışmadan sonra hala hayatta olduğunu ancak hatalı tasarım yüzünden otomobilin alev
  alması sonucu öldüğünü ileri sürdü </s>
5 <s> meadow rain walkerın avukatı jeff milam yatpığı açıklamada porschenin walker
  tarafından kullanılan modelinin arka kısmı tehlikeli </s>
6 <s> caddelere uygun olarak tasarlanmamış diye konuştu </s>
7 <s> porsche firmasının kuzey amerika sorumlusu calvin kim ise yaptığı açıklamada daha önce
  de söylediğimiz gibi bizim firmamıza ait bir otomobilde canı yanan zarar gören herkes için
  üzgünüz </s>
8 <s> ancak inanıyoruz ki uzmanlar bu kazanın firmamızın üretimi olan otomobildeki tasarım
  hataları yüzünden değil tehlikeli ve aşırı süratlı hız yüzünden meydana geldiğini ortaya
  çıkaracaktır dedi </s>
```

Example input file

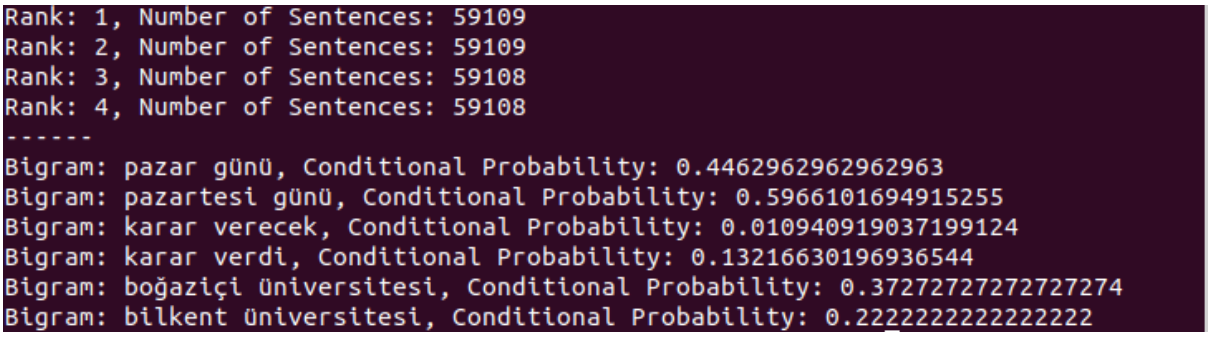
The second input file is the input test file. Input test file should include the bigrams that the user wants to learn the conditional probability of the occurrence of

these bigrams in the input test file. For example, if the user wants to learn the conditional probability of “pazar günü” bigram, the user should write the bigram to the test input data. The path of the test file should be given in the arguments as stated in the interface section of the report. There must be at most one bigram in each line. The first word of the bigram should be in at least one sentence. If there is no unigram of that word the program will give division by 0 error. There must be no bigram that the first word of the bigram is not in the input data file. The user must ensure these requirements in order to execute the program properly. There is a photo of the example test input file under this paragraph.




Example test input file

Output contains two parts. First part includes the rank of the processes and the number of sentences they received. This information is separated by a comma. The second part includes the bigrams that are specified in the input test file and the conditional probabilities of them. There is a “------” symbol between the first part and the second part of the output. There is a photo of the example output format under this paragraph.




Example output format

## Input and Output:

The program takes an input data file, test data file and gives output to the console. Input data file includes sample sentences. The path of the input file is given in the arguments as stated in the interface section of the report. Sentences in the input data file starts with the <s> symbol and ends with </s> symbol. For example, the sentence in the input file can be “<s> güvercinleri de vururların söz ve müziği ise şehrazata ait </s>”. The sentence “güvercinleri de vururların söz ve müziği ise şehrazata ait </s>” is not allowed because it does not start with “<s>” tag or the sentence “<s> güvercinleri de vururların söz ve müziği ise şehrazata ait” is not

allowed because it does not end with “</s>” tag. There is at most one sentence in each line.

The test data file includes the bigrams that the user wants to learn the conditional probability. For example, if the test data file includes the bigram “karar verecek”, that means the user wants to learn the conditional probability of the “karar verecek” bigram. The file includes at most one bigram in each line.

The program prints the rank of the worker processes and the number of sentences they received to compute frequencies of the bigrams and the unigrams. The rank of the worker processes and the number of sentences is separated by a comma. After printing these, processes calculate the frequencies and the conditional probabilities of the given bigrams in the test data file. The program prints the bigrams in the test data file and their conditional probabilities after the calculation. The bigrams and their conditional probability are separated by a comma. There is a "------" symbol between the first part of the output, which contains the rank of the processes and the number of the sentences they received, and the second part of the output, which contains the bigrams in the test input file and the conditional probabilities of them.

## Program Structure:

There are two types of processes inside of the program. One of them is the master process and the other is the worker process. Master process is the process that has the rank 0. All data is gathered into the master process after the calculations are made. Worker processes have a rank different from 0. They calculate the frequencies of the unigrams and bigrams inside of the input data file.

There are two merge methods in this program, which are MASTER and WORKERS. If the merge method is MASTER each worker process sends the frequency data they calculated to the master process directly. If the merging method is WORKERS, worker processes do not send the data directly to the master process. Each worker process sends data to the other worker processes except the worker that has the highest rank. The worker that has the highest rank sends the data to the master process. The other workers send the data they calculated and the data that was calculated by former processes to the worker process that has one more rank than their rank. For example, a process that has rank 1 sends the data, which is calculated by itself, to the process that has rank 2. The process that has rank 2 sends the data, which is calculated by itself or calculated by one of the former processes, to the process that has rank 3. This process continues until the process’s rank is the number of processes - 1. If the process’s rank is number of processes - 1, the worker process sends all data to the master process. The conditional probability calculation is made in the master process.

The program has three main parts, which are the common part, the part that only the master process executes, the part that only the worker processes execute. I

will call the second part the master part and the third part the workers part for the sake of simplicity.

Common part is the part that is common between the worker processes and the master process. It includes the rank of the processes, number of total processes, which is size, and the unigrams and the bigrams dictionaries. Rank of the process is needed for every process. The program should print the ranks of each worker process. The master process's rank is needed to check whether the process is master or not. Unigram and bigram dictionaries hold the bigrams and unigrams as the keys. They hold the frequencies of the unigrams and the bigrams as the values.

If the merge method is MASTER, each worker process has a unigrams and bigrams dictionaries. The dictionaries contain the unigrams and the bigrams inside of the sentences that they are given. They send these dictionaries to the master directly to the master process. If the merge method is WORKER each worker process has a unigrams and bigrams dictionaries. The dictionaries contain the unigrams and the bigrams inside of the sentences that they are given. They also contain the unigrams and bigrams inside of the sentences that the former processes were given. They send these dictionaries to the next worker process except the process that has the highest rank. The process that has the highest rank sends the dictionary, which contains all unigrams and bigrams, to the master process. Master process's unigrams and bigrams dictionary always holds all unigrams and bigrams. The merge method does not affect the master process's dictionary.

The master part of the program is executed when the rank is 0. If the rank is 0 the process must be the master process, so this part of the program is executed by only the master process. The data is read inside of the master part. The distribution of the data is also made in the master part of the program. The program creates a list containing lists. The list contains as many lists as the number of processes. Then, each line that is read from the input data file is added to the lists inside of the list one by one. The main reason that the distribution is made one by one is to distribute the data equally among the processes.

After the distribution is made, each list is sent to one worker process. When worker processes finish their job the master process receives the data from the worker processes. Receiving method differs according to the merge method. If the merge method is MASTER, the master process receives the data from the processes one by one with the help of a for loop. If the merge method is WORKER, the one receiving operation from the worker that has the highest rank is enough.

The conditional probability calculation is also made in the master part. When all the data is gathered into the master process, the master process starts the computation of conditional probabilities. The master process reads the input test file line by line and calculates the conditional probabilities of each bigram. Conditional probability calculation is made according to frequencies of the bigrams and unigrams of the first words of the bigrams. For example if the requested conditional probability is P(sky | blue) the value of this probability is freq(blue sky) / freq(blue). This approximation is made under the assumption that the input file is a large textual data. There can be division by 0 error if the denominator of the probability calculation is 0, which means there is no unigram of the first word in the bigram. If there is no

unigram of the first word in the bigram, that means the first word is not included in the input data file. The user must be sure that there are no such bigrams in the input test file. The master process prints each bigram and their conditional probability after the calculation.

The worker part of the program is executed when the rank is different than 0. First, the worker processes receive the data that is sent from the master process. After receiving the data they print their rank and the number of sentences inside of the data to the terminal. Each worker process calculates the frequencies of the unigrams and the bigrams inside of the sentences that they received.

If the merge method is MASTER, all worker processes send the unigrams and bigrams dictionaries to the master process and their job is done. If the merge method is WORKERS they first receive the data that was sent from former processes, except the worker process that has rank 1. The worker process that has rank 1 does not receive data from another worker process because there is no worker process before rank 1 process. After the receiving operation each worker process merges the data coming from previous processes and the data that they calculated. When the merge operation is done, each process sends the data to the next worker process, except the process that has the highest rank. The process that has the highest rank sends the data to the master process because there is no worker process after that process. The job of the worker processes is done when they finish the sending operations.

## Examples:

1)

Command:

mpiexec -n 3 python3 main.py --input_file data/sample_text.txt --test_file data/test.txt --merge_method WORKERS

Input file: same sample_text.txt file as the file provided on the moodle page

Test file:

meadow rain
pazartesi günü
tarihi rekor
ev fiyatları
boğaziçi üniversitesi
eski gazeteci
birinci transfer
kepek ekmeği

Output:

Rank: 1, Number of Sentences: 118217
Rank: 2, Number of Sentences: 118217
---
Bigram: meadow rain, Conditional Probability: 0.6666666666666666

Bigram: pazartesi günü, Conditional Probability: 0.5966101694915255
Bigram: tarihi rekor, Conditional Probability: 0.006060606060606061
Bigram: ev fiyatları, Conditional Probability: 0.0010235414534288639
Bigram: boğaziçi üniversitesi, Conditional Probability: 0.37272727272727274
Bigram: eski gazeteci, Conditional Probability: 0.0004952947003467063
Bigram: birinci transfer, Conditional Probability: 0.0015432098765432098
Bigram: kepek ekmeği, Conditional Probability: 0.0

Explanation: There are 2 worker processes for this execution. The 236434 sentences in the input file were distributed to 2 processes equally. The worker processes printed their ranks and the number of sentences sent by the master process. After that, these 2 processes calculated the number of bigrams and unigrams and the first worker process sent its calculation results to the second worker process. The second worker process merged the data sent by the first worker process to its own calculation results. The merged data was sent to the master process. The master process calculated the conditional probabilities of the given bigrams by dividing the total count of the bigram by the total count of the unigram which is the first word of the bigram. For example, the bigram “boğaziçi üniversitesi” has 41 instances in the input file and the unigram “boğaziçi” has 110 instances in the input file. So, the conditional probability of the bigram “boğaziçi üniversitesi” is 41/110 = 0.37272727272727274 .The bigram “kepek ekmeği” has 0 conditional probability since there is no instance of the bigram “kepek ekmeği”. The master process printed these bigrams and their conditional probabilities.

2)

Command:

mpiexec -n 4 python3 main.py --input_file data/sample_text.txt --test_file data/test.txt --merge_method MASTER

Input file: same sample_text.txt file as the file provided on the moodle page

Test file:

beşiktaş maçı
malatya kayısısı
bilgisayar mühendisliği
dana eti
şampiyonlar ligi

Output:

Rank: 1, Number of Sentences: 78812
Rank: 2, Number of Sentences: 78811
Rank: 3, Number of Sentences: 78811
---
Bigram: beşiktaş maçı, Conditional Probability: 0.011553273427471117
Bigram: malatya kayısısı, Conditional Probability: 0.013333333333333334
Bigram: bilgisayar mühendisliği, Conditional Probability: 0.05639097744360902
Bigram: dana eti, Conditional Probability: 0.18604651162790697
Bigram: şampiyonlar ligi, Conditional Probability: 0.4682274247491639

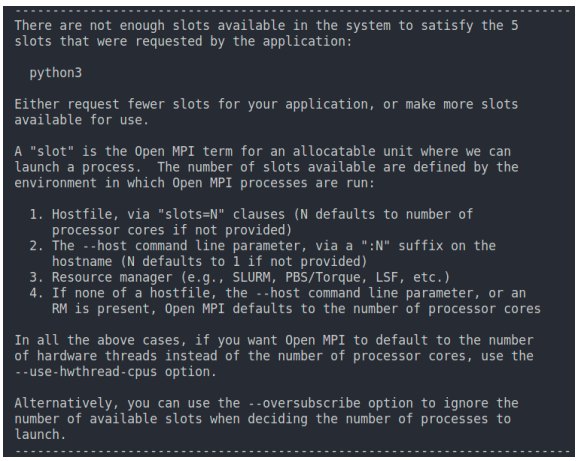
Explanation: There are 3 worker processes for this execution. The 236434 sentences in the input file were not distributed equally to the worker processes since 236434 is not divisible by 3. So, 78812 of these sentences were given to the first worker process and the other 157622 sentences were distributed to the other 2 workers equally. The worker processes printed their ranks and the number of sentences sent by the master process. The same counting procedure occurred again but the merging method is different for this execution since the specified merge method is “MASTER” instead of “WORKERS”. After the counting operation had been completed, the workers sent their data to the master process directly. Then, the master process merged all of the data sent by workers. The master calculated the conditional probabilities of the given bigrams and printed them as in the first example.

## Improvements and Extensions:

Since the project was not that complicated and the implementation of the program was simple - because the description of the project defines the features of the program in an exact manner- , it seems that there is no way of having major shortcomings. Maybe the control of an invalid bigram can be provided since if the first word of the bigram does not exist in the input text, the number of unigrams will be 0 and the division by 0 will result in a division by 0 error. However, this invalid situation is denoted in the program execution section. Also, thanks to the simplicity of the project, there was no deviation from the planned implementation features.

## Difficulties Encountered:

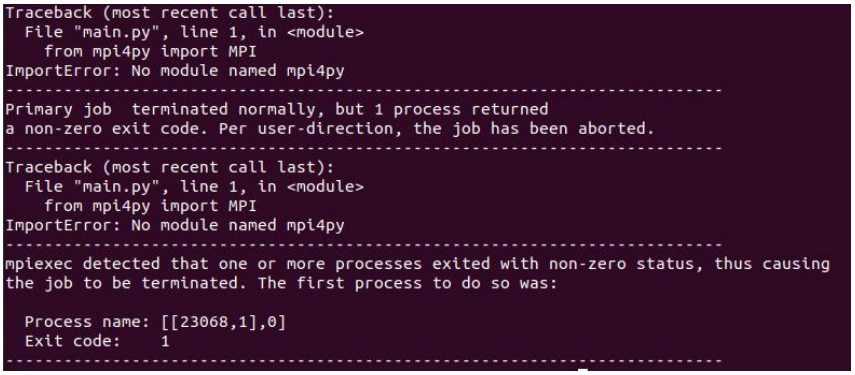
During the project, two problems are encountered. The first one was about the MPI framework. When the program was executed with n>4 processes, an error was occurring that said “There are not enough slots available in the system to satisfy the n slots that were requested by the application”.




Full text of the first error

This error was handled by using the “--oversubscribe” flag which is explained in the program interface section.

The second problem was about the installation of the “mpi4py” module. Even though this module was installed via the “pip install mpi4py” and “pip3 install mpi4py” commands, when the program was executed with the “python” command an error was occurring that said “no module named mpi4py”. This error was happening in both of the computers of the group members. It is guessed that this error is occurring since the environments of the computers are broken. This error was handled by using the “python3” command instead of “python” since it may take much time to fix the environment.




Full text of the second error

## Conclusion:

The main aim of the project is to use parallel programming. Parallel programming is implemented in the project by openmpi library. The program runs the specified amounts of processes in the parallel manner. The data is shared equally among the processes. The data sharing and processes running parallel manner made the whole process faster than the one process making all the operations.

The data passing between parallel processes is used in the project. The data passing is made with the send and receive operations. The different merge methods diversify the data passing. If the merge method is MASTER the worker processes send the data directly to the master process. If the merge method is WORKERS the data is sent between worker processes sequentially until the last worker. The last worker sends all the data to the master process.

## APPENDICES:

```
from mpi4py import MPI
import argparse


mpi = MPI.COMM_WORLD
rank = mpi.Get_rank()
size = mpi.Get_size()

parser = argparse.ArgumentParser()
parser.add_argument("--input_file", dest="inputfile",
                    help="Name of the input file",
                    required=True)

parser.add_argument("--test_file", dest="testfile",
                    help="Name of the test file",
                    required=True)

parser.add_argument("--merge_method", dest="mergemethod",
                    help="Merge method",
                    required=True)
args = parser.parse_args()
inputfile = args.inputfile
testfile = args.testfile
mergemethod = args.mergemethod

unigrams = dict()
bigrams = dict()

# Read the input file and distribute the data evenly to the worker
processes
if rank == 0:

    # Create 2-d list to distribute the data
    datalist = []
    for i in range(size):
        datalist.append([])
    file = open(inputfile)
    i = 0
    # Distribute the data to lists
    for line in file:
```

```
        datalist[i % (size-1)].append(line)
        i += 1
    # Distribute the data to worker process
    for i in range(size-1):
        mpi.send(datalist[i], i+1)

    if(mergemethod=="MASTER"):
        # Receive the datas from the worker processes and merge
them
        for i in range(1,size):
            recUnigrams=mpi.recv(None,i)
            recBigrams=mpi.recv(None,i)
            for key,value in recUnigrams.items():
                unigrams[key]=unigrams.get(key,0)+value
            for key,value in recBigrams.items():
                bigrams[key]=bigrams.get(key,0)+value
    # Receive the datas from the last worker process.
    elif(mergemethod=="WORKERS"):
        unigrams=mpi.recv(None,size-1)
        bigrams=mpi.recv(None,size-1)
    else:
        print("Unknown merge method")
    # Calculate the conditional probabilities for the given bigrams
    file = open(testfile)
    for line in file:
        line=line.rstrip()
        words=line.split()
        prob=bigrams.get(line,0)/unigrams.get(words[0],0)
        print("Bigram: {}, Conditional Probability:
{}".format(line,prob))

else:
    # Get the data from the master process
    sentencelist = mpi.recv(None, 0)
    # Print the rank of the process and length of the data
    print("Rank: {}, Number of Sentences: {}".format(rank,
len(sentencelist)))
    if(rank == size -1):
        print("------")
    # Iterate over the given sentences
    for sentence in sentencelist:
        wordlist = sentence.split()
```

```
        # Count unigrams and bigrams
        for i in range(len(wordlist)):
            word = wordlist[i]
            unigrams[word] = unigrams.get(word, 0)+1
            if (i != len(wordlist)-1):
                bigram = word+" "+wordlist[i+1]
                bigrams[bigram] = bigrams.get(bigram, 0)+1
    # Send the data to the master process
    if (mergemethod == "MASTER"):
        mpi.send(unigrams, 0)
        mpi.send(bigrams, 0)
    elif (mergemethod == "WORKERS"):
        # Send the data to the next worker process
        if(rank==1):
            if (size>2):
                mpi.send(unigrams,rank+1)
                mpi.send(bigrams,rank+1)
            else:
                mpi.send(unigrams,0)
                mpi.send(bigrams,0)
        # Take the data from the previous worker process and merge
that data with process' own data. Send the merged data to the next
worker
        else:
            recUnigrams=mpi.recv(None,rank-1)
            recBigrams=mpi.recv(None,rank-1)
            for key,value in recUnigrams.items():
                unigrams[key]=unigrams.get(key,0)+value
            for key,value in recBigrams.items():
                bigrams[key]=bigrams.get(key,0)+value
            if(rank!=size-1):
                mpi.send(unigrams,rank+1)
                mpi.send(bigrams,rank+1)
            # Send the merged data to the master process
            else:
                mpi.send(unigrams,0)
                mpi.send(bigrams,0)
    else:
        print("Unknown merge method")
```

Source code of the program