

CS F372 Operating Systems

1st Semester 2024 - 25

Assignment 1

Marks: 45

- **Release Date: 11th September, 2024**
- **Deadline: 19th September, 2024, 23:59:00 hours**
- **Max free late days per student for the entire semester: 3 days (the last updated timestamp will be treated as the submission date, even if that involves changing only a single character)**
- *Use Piazza to clear your doubts*

Problem Statement

- You will be given an **NxN matrix** with strings in each cell, as well as a text file containing a large number of strings. Your task is to only write a POSIX compliant C program that finds the number of times each string in the matrix occurs in the given text file *as efficiently as possible*.
- **Catch:** Each string in the matrix (except the one in the very first cell) will be encoded with a [Caesar Cipher](#), and must be decoded using a key before its occurrences in the file can be counted.
- The key to decode the strings can be obtained by communicating with another process, **the helper process**, using a **message queue**.
- The helper process will already be running and ready to communicate with your program. You don't have to write the helper process.
- All the strings belonging to a specific **right diagonal** of the matrix will have the same key, which may be received by sending the sum of the occurrences of all the strings in the previous diagonal to the helper process.
- The matrix will be present in a **shared memory** that your program will have to connect to. More details for the same are given below.

The Input: Your program will receive exactly one command line argument, which will be a number with at most two digits. This number will be used to access the files your program needs to read, i.e., the input file and the words file. These files will be named as **input[t].txt** and **words[t].txt**, where **t** is the command line argument received (The square braces are not part of the name). For instance, if the command line argument received is "3", these files will be named **input3.txt** and **words3.txt**.

The first of these, the input file, will be present in the same directory where your program will be run. It will have four lines of text. Each line will contain a single integer, in the following format:

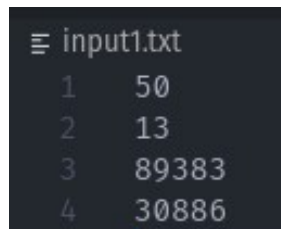
Size of the matrix (N)

Maximum length of a word in the words file

The key for connecting to the shared memory containing the NxN matrix

The key for connecting to the message queue to communicate with the helper

A sample input file may look as so:



```
≡ input1.txt
1 50
2 13
3 89383
4 30886
```

The Matrix: This will be an NxN matrix stored in the shared memory whose key is given in the input file described above. You may connect to the same via the `shmget` function. Considering that the shared memory is a 2D array of strings you may refer to the function signatures provided below to access the shared memory correctly. Your program is allowed to make changes to the matrix if needed.

Function Signatures to work with the 3D matrix (third dimension is for the string, which is a character array in C)

```
char (*shmptr) [N][stringSize];
```

```
shmget(key_t key, size_t sizeof(char[N][N][stringSize]), int shmflg)
```

```
shmat(shmId, NULL, 0);
```

Here, N is the side length of the matrix, `stringSize` refers to the maximum string length, as given in the input file.

With this signature, after connecting to the shared memory, `shmptr[i][j]` will address the word present at the *i*th row and *j*th column of the matrix ($0 \leq i, j < N$)

The Text File: There will be a text file named **words[t].txt** present in the same directory as the one your code is run from, containing a large number of strings separated by spaces. This is the file from which you must count the number of occurrences of specific words, as given in the matrix. Note that each word, both in this file and in the matrix, will contain only lowercase English characters.

The Helper Process: A helper process will automatically be started when your code is executed. Your program can communicate with it using a message queue, which can be connected to using the key provided in the input file described above. You will be required to send one message to it for each right diagonal of the matrix. The **ith** message must correspond to the sum of the number of occurrences of each word in the **ith** right diagonal of the matrix (Hint: *Tutorial 2: Q3*). If the correct sum is input, the helper process will reply on the same message queue with an integer representing the **key** for the Caesar Cipher for the next right diagonal of the matrix. This key will range from **0 to 10⁵**. Once the final message (i.e., the number of occurrences of the word present in the last cell of the matrix) is sent, the execution will be considered complete (*Hint: Tutorial 2: Q3*). The Helper replies to the last value with **0** if it is correct, and **-1** if it isn't. After receiving this final value, ensure that your process follows the guidelines for cleanup activities needed for the shared memory and message queue as mentioned below, and exit as well.

The Helper will listen to messages of **mtype 1** from your program, and will send responses with **mtype 2**.

Note that every time the Helper Process receives a message in its message queue, it will assume this message to be the solution for the next diagonal. As such, the Helper will always exit after having received messages equal to the number of diagonals, i.e., **2N-1**. Moreover, if at any step an incorrect value is received by the Helper (eg., the correct sum for the 11th diagonal is 24, but 22 is received as the 11th value) the Helper will return **-1** instead and exit immediately, regardless of how many diagonals are remaining. This case of returning a -1 is not expected to occur during correct functioning of your program, and is only provided to help you with debugging purposes.

Also note that words that appear as substring in other words should not be counted. For instance, when counting the number of occurrences of the word 'abc', occurrences inside words like 'h**ab**cp' should not be considered.

A step by step example

Consider the following 3x3 Matrix as your input:

abcd	efef	ttto
ghi	rtty	ybl
plk	xvxx	qwerty

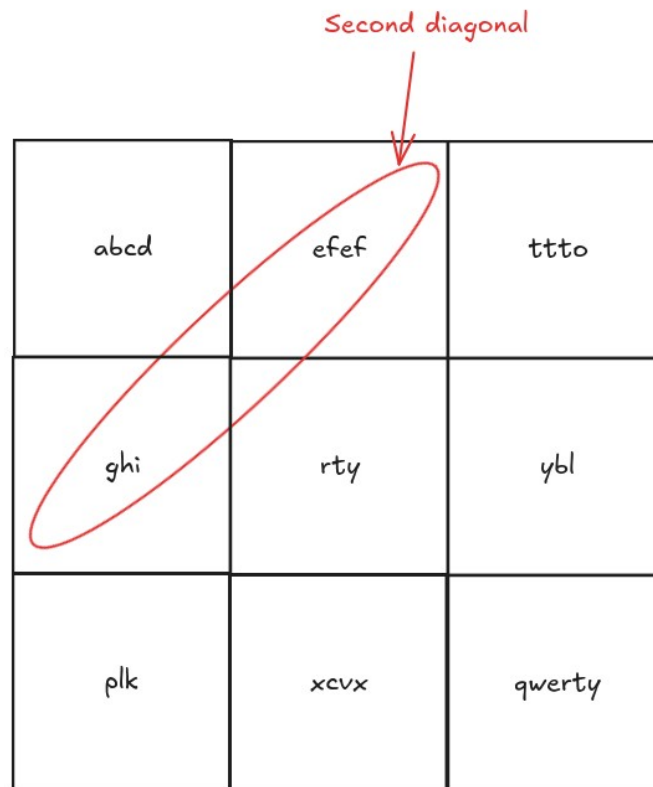
Now, the first right diagonal of the matrix contains just the first cell. As stated before, the string here is not encoded; therefore, your first step is to find the number of times the word 'abcd' appears in the given text file, words[t].txt.

First diagonal

abcd	efef	ttto
ghi	rtty	ybl
plk	xvxx	qwerty

Suppose you count it and find that it appears 13 times. Your next step is to send this answer, 13, to the helper process using the message queue. Upon sending this message, you'll get a reply containing the key for the next diagonal; suppose this is 4. Thus, each character in all the strings in the second diagonal will be rotated right by 4 characters, to obtain the strings whose occurrences have to be counted.

Second diagonal



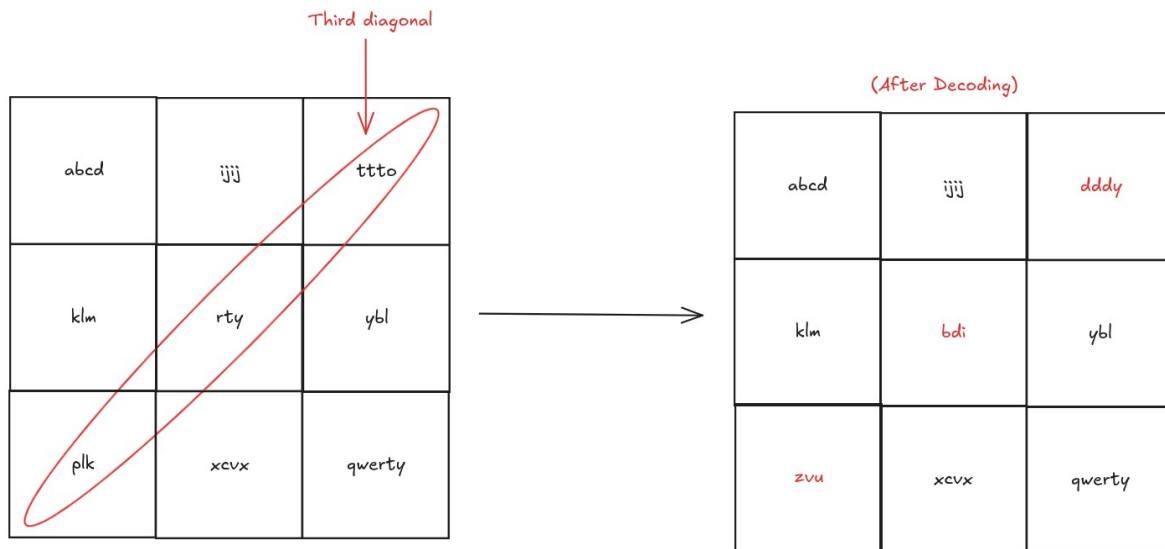
abcd	efef	ttto
ghi	rty	ybl
plk	xcvx	qwerty

(After Decoding)

abcd	ijij	ttto
klm	rty	ybl
plk	xcvx	qwerty

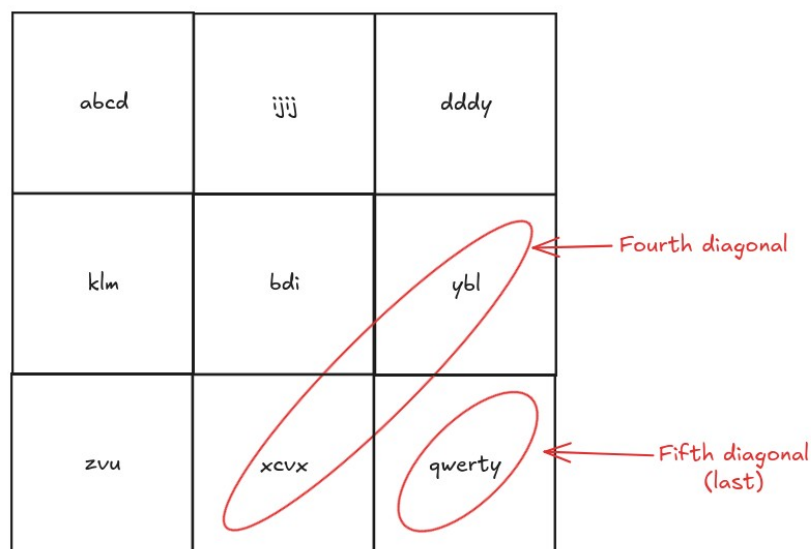
With this, you must count the number of occurrences of the strings 'ijij' and 'klm' in words[t].txt. Suppose these are 21 and 4, respectively; then you must sum them up and send the answer, 25, to the helper process. It will then return the key to decode the next diagonal; suppose this is 10. Thus, the strings in the third diagonal all need to be rotated by 10 characters in order to get the

decoded words:



This process continues. Next you will send the sum of the number of occurrences of 'dddy', 'bdi', and 'zvu' in the text file. On receiving this, the helper will reply with the key for the fourth diagonal.

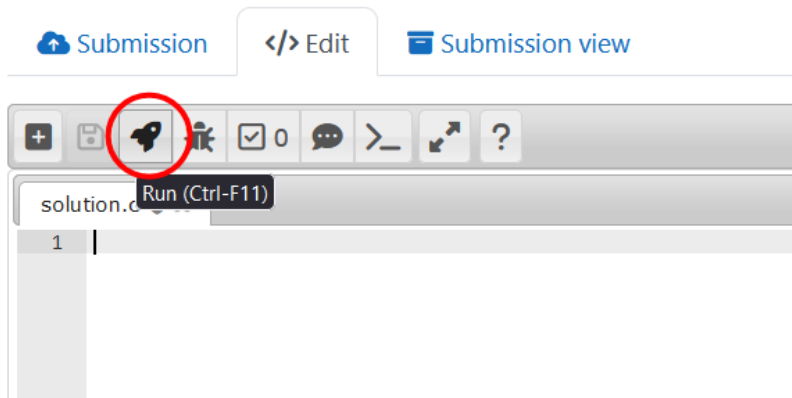
Finally, after decoding the word in the final diagonal (in this case the fifth one), counting its occurrences, and sending this number to the helper, the helper will respond back with 0 to indicate it got the correct final answer, and then wait for your solution program to exit, before exiting itself.



Submitting and Testing your solution

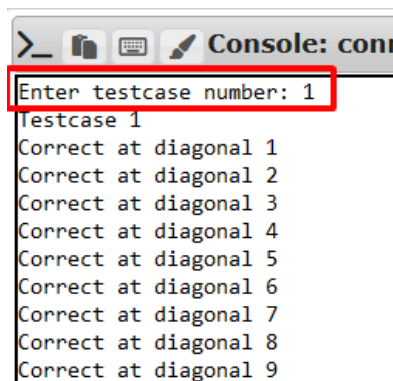
- The assignment is to be submitted on the portal: <http://responsible-tech.bits-hyderabad.ac.in/moodle/>
 - To login, Your username can be derived from your ID number thus:
 - If you ID Number is 2022A7PS1234H, then your username is 20221234
 - The password can be obtained from <https://lms.erp.bits-pilani.ac.in/moodle>. Go to the OS Course -> gradebook and look at the 'Feedback' column against the grade item 'RTMoodleLogin'

- On the RTMoodle portal, navigate to [Operating Systems Autumn 2024](#) -> Assignment 1
- Assignment submission is available from outside the campus, *but testing is only available from within the campus*
- In the assignment, you can submit your code in the Submission tab, or directly write/paste it into the editor in the Edit tab.
- To test your code against the available test cases, you can click the Run button as shown in the image below on the Edit tab. The helper process and the other files will already be present and running on the platform, ready to communicate with your process.



After clicking the Run button, a terminal window will pop up with your code's output. Before running, though, it will first ask you to input which test case you wish to test your code against. Currently, there are four test cases, so you can enter a number from 1 to 4.

Important: The time limit for execution on VPL counts down even while it waits for you to enter which test case you wish to run, so enter it quickly.



- The platform will evaluate your program and show you the test cases which pass or fail and the time required for each test case. Please note that this time may vary based on how many students are simultaneously submitting on the server. Currently there is a time limit of 4 seconds for each test case and *efficient* solutions to all the test cases should run within 4 seconds
- There is also an 'evaluate' button that attempts to run all the test cases within the four second time limit. *Please ignore this button*, as even faster code may time out with it. Moreover, the final grading will be done later as it will involve hidden test cases and efficiency parameters.

- You may print values to stdout or stderr for testing purposes while writing your code, but please ensure the final solution you submit **does not print anything**, as evaluation will be done against the output printed by the helper program.
- **Guidelines for Cleanup:** *Only detach from the shared memory in your process, do not use `shmctl` or `msgctl` to delete the shared memory and message queue.*

Operational Guidelines

- The assignment is to be done individually
- The assignment needs to be done using the C language. Your solution should be POSIX compliant and run on an Ubuntu System (≥ 22.04). You are allowed to use either multiple processes or threads or both for *efficiency*
- After the assignment deadline is over, the solutions will be evaluated against a few more hidden test cases to arrive at the final score. The submission will be timed after the deadline and part of the score will depend on how fast / efficient your solution is.
- Part of the score will depend on the code efficiency determined by the run time of the code for the test cases
- **The submissions will be checked for plagiarism. Any hints of plagiarism will attract a summary 0 score. There is no partial dishonesty**
 - Lifting code directly from the Internet, including from Stack Overflow and/or github and/or other platforms including ChatGPT, is plagiarism
 - Discussions are encouraged, but copying is not. One thumb rule is: after a discussion do not take away any written or soft copy notes. Instead, watch something mind-numbing, and then implement what was discussed from memory. Look at the Cheating Vs. Collaborating Guidelines here: <https://www.cse.iitd.ac.in/~mausam/courses/col772/spring2024/>
 - The purpose of the course and the assignment is for you to learn, shortcuts might fetch you grades but you will learn nothing and will have to bear the burden of being a cheat for the rest of your life
- Every student gets 3 free late days in the semester. You are free to distribute the late days across the two assignments as you see fit. Beyond the free late days, every additional late day will incur a 10% penalty