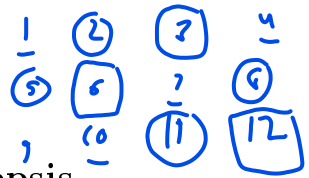
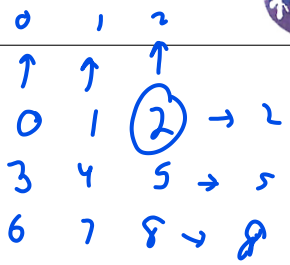


Assignment 3

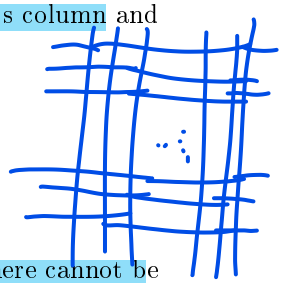


1 Synopsis

Please read the document, ***General Requirements Programming Assignments***, posted on the course website and make sure you follow them when you do this assignment. This assignment is to write an MPI program to find the ***saddle point*** of a matrix, if it exists.

A saddle point of a two-dimensional matrix is defined as an element that is the smallest in its column and the largest in its row. For example in the matrix `A` below

```
6  4  5  9
7  3  8  1
0  2  0  1
6  5  8  3
```



`A[2,1]` is a saddle point. Not all matrices have saddle points, but if one does, it is unique - there cannot be two saddle points in a single matrix.

2 Program Invocation, Input, and Output

In this project, you are to write an MPI program that expects as its only ccommand line argument the pathname of a file that contains a two dimensional array of floating point numbers in binary format in the same format as is used in the parallel Floyd algorithm. Specifically, the file is a ***binary file*** and it starts with two integers specifying the number of rows followed by the number of columns of the matrix, after which the elements of the matrix are stored in row-major order as ***double-precision floating point numbers*** (C type `double`). The matrix may be of any size.

Therefore, if the program is named `saddle_point`, and the file `matrix1` is in the proper form, it would be used as follows

```
saddle_point matrix1
```

The program will output a single line, which is either the coordinates of the saddle point in the matrix, first the row index and then the column index, separated by a single tab character, or it will output the message, "`No saddle point`".

2.1 Error Handling

If one or more arguments are missing or if the file can not be opened for reading, the program should print an error message on ***standard error*** that includes how to use it correctly, and it should exit. If during processing the program results in an error such as being unable to allocate memory or other programmatic failures, it should print a message on ***standard error*** that it failed and it should clean up all MPI processes and exit. The standard error stream in C can be written to using `fprintf`.

3 Program Implementation Requirements

- Only the root process can perform input and output and error handling. It is an error if any other process does so.

- The program must produce correct results regardless of the size of the matrix, provided that there is memory for it.
- The program must work correctly regardless of how many tasks are run.
- The program must be documented and written to comply with the requirements stated in the ***General Requirements Programming Assignments*** referred to above
- The program is supposed to be immune to incorrect usage. This means that if the first argument is anything other than a file in the correct format that can be read by the program, it should detect this and exit with an error message.
- The program must run correctly on any `cslab` host.

4 Testing

The program should be tested with very large arrays and arrays that are not square. Use the tools I have written in the directory

```
/data/biocs/b/student.accounts/cs493.65/mpi_demos/common
```

to create matrices by hand or automatically and convert them to binary form. The `README` in that directory explains how touse these tools.

5 Program Grading Rubric

The program will be graded based on the following rubric out of 100 points:

- The program must compile and run on any `cslab` host. If it does not compile and link on any `cslab` host, it loses 80 points.
- **Correctness** (70 points)
  - The program should do exactly what is described above. Incorrect output, incorrectly formatted output, missing output, or output containing other characters are all errors.
  - It must process arrays of arbitrary size.
  - The program should produce the same output no matter how many processes it is given, except for the elapsed time.
  - It should handle errors correctly.
- **Performance** (10 points)

  The program should be as efficient as possible. There are ways to solve this problem that are more efficient than others. When run with successively greater numbers of processes, the elapsed time should decrease, except that if the number gets too large (larger than the available processors generally), it will start to increase again. Check that this behavior occurs on average. On any one run, it may not be exactly like this, but over many runs it should behave like this.
- **Compliance with the Programming Rules** (20 points)

  Are all of the rules stated in that document observed? Programs that violate them will lose points accordingly.

6 Submitting the Homework

This project is due by 19:00 on October 27, 2021. Follow these instructions precisely to submit it.

Your program should be a single source code file, written in C. It must have a `.c` suffix. It does not matter what base name you give it because the `submithwk_cs49365` program will change its base name anyway to the form `hwk3_username.c` (or more accurately, `hwk3_username.`*XXX*, where *XXX* is the suffix you gave it.)

In these instructions, assume your program file is named `myprog.c`.

1. You will use the `submithwk_cs49365` command to submit the program source code file. **To submit your file, type the command**[1]

   ```
   $ /data/biocs/b/student.accounts/cs493.65/bin/submithwk_cs49365  -t 3  myprog.c
   ```

   The program will copy your program into the directory

   ```
   /data/biocs/b/student.accounts/cs493.65/hwks/hwk3/
   ```

   and if it is successful, it will name it `hwk3_username.c` and display the message, "`File hwk3_`*`username.c`* `successfully submitted.`" where `username` is your actual username.

   You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

   ```
   ls -l /data/biocs/b/student.accounts/cs493.65/hwks/hwk3
   ```

   and making sure you see a non-empty file named `hwk3_`*`username.c`*.

2. ***You can do the preceding step as many times as you want. Newer versions of the file will overwrite older ones.***

[1] If you have modified your `PATH` variable to include the directory `/data/biocs/b/student.accounts/cs493.65/bin`, then you can just type `submithwk_cs49365`.