Midsem Report

OS Lab COM301P

(Contains C Program, Algorithm explanation, Output Screenshots and Efficiency analysis against the serial version of the solution)

Animesh Kumar

CED18I065

1st November, 2020

Question:

Develop a C program that checks if a user keyed in square matrix is a magic square or not. Have 3 processes in all, one to check for column condition, one for row sum checking and one for main and trailing diagonal check. Compare the efficiency of the multiprocessing version over its equivalent serial version.

Algorithm:

Input:

- The size 'n' of the Square Matrix(nxn)
- The Square Matrix(nxn)

Output:

Notifies on the stdout whether the input Square Matrix is a square matrix.

Assumptions:

The magic square shall contain only the integers from 1, 2, 3, … n^2 where n is the size of the matrix. (n x n square matrix). The input square satisfies all the criteria that a magic square satisfies, only then it is a magic square.

Steps:

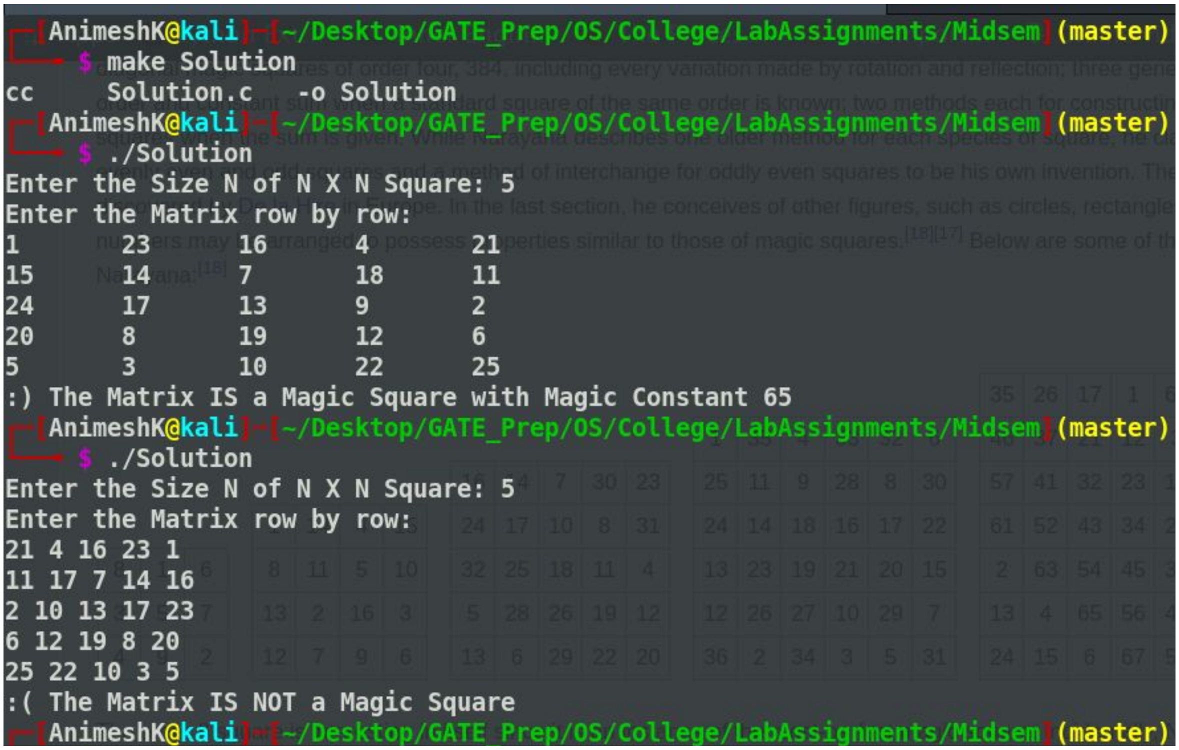
- The main process takes the 'size' and the 'Matrix[size][size]' as input.
- The main process calls the `is_magic()` method.

  Steps involved in the `is_magic()` are as below:

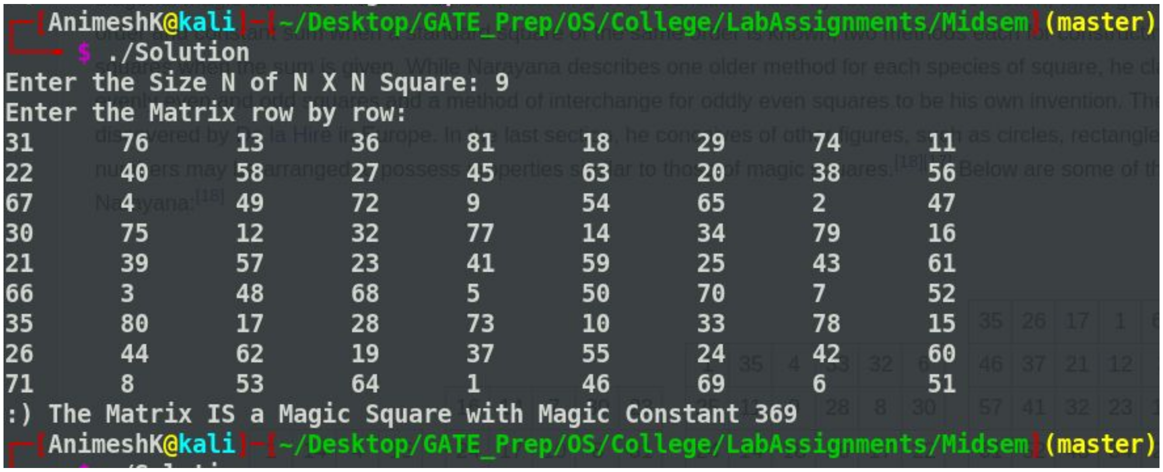
- Two pipes are opened.
- Two child processes are forked
    - First child process steps:
        - This process internally makes calls to `get_row_sum()` to calculate the sum of a single row.
        - If all the row_sum's are the same, it passes the row_sum to the parent via pipe.

        - otherwise it passes -1 to the parent via pipe.
        - Terminates successfully.
    - Second child process steps:
        - This process internally makes calls to `get_column_sum()` to calculate the sum of a single column.
        - If all the column_sum's are the same, it passes the column_sum to the parent via pipe.
        - otherwise it passes -1 to the parent via pipe.
        - Terminates successfully.
- Make a call to `DiagonalSum()` for calculating the sum of MAIN_DIAGONAL.
- Make a call to `DiagonalSum()` for calculating the sum of TRAILING_DIAGONAL.
- Receives the diagonal_1_sum and diagonal_2_sum from the child processes via pipes.
- If all the above 4 sums are equal only then
    - It returns one of the sums as the magic constant
- else
    - The user is notified by a message that the input square is not a magic square.

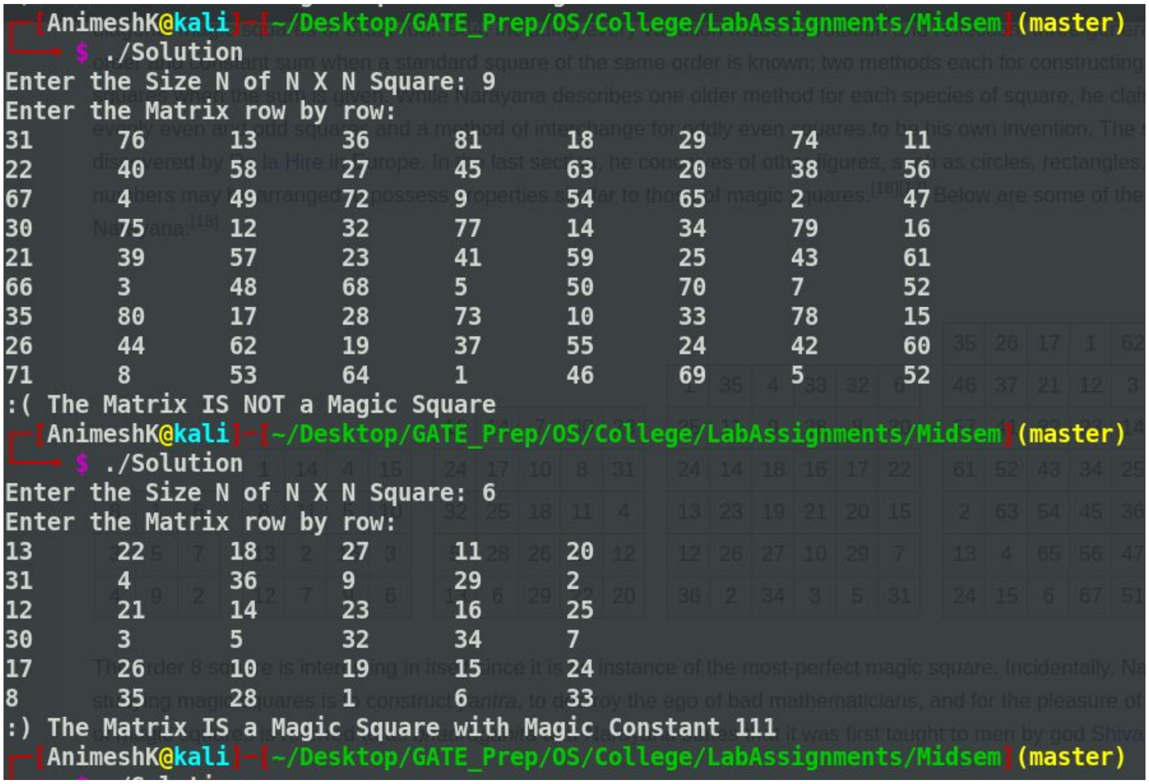
Output:



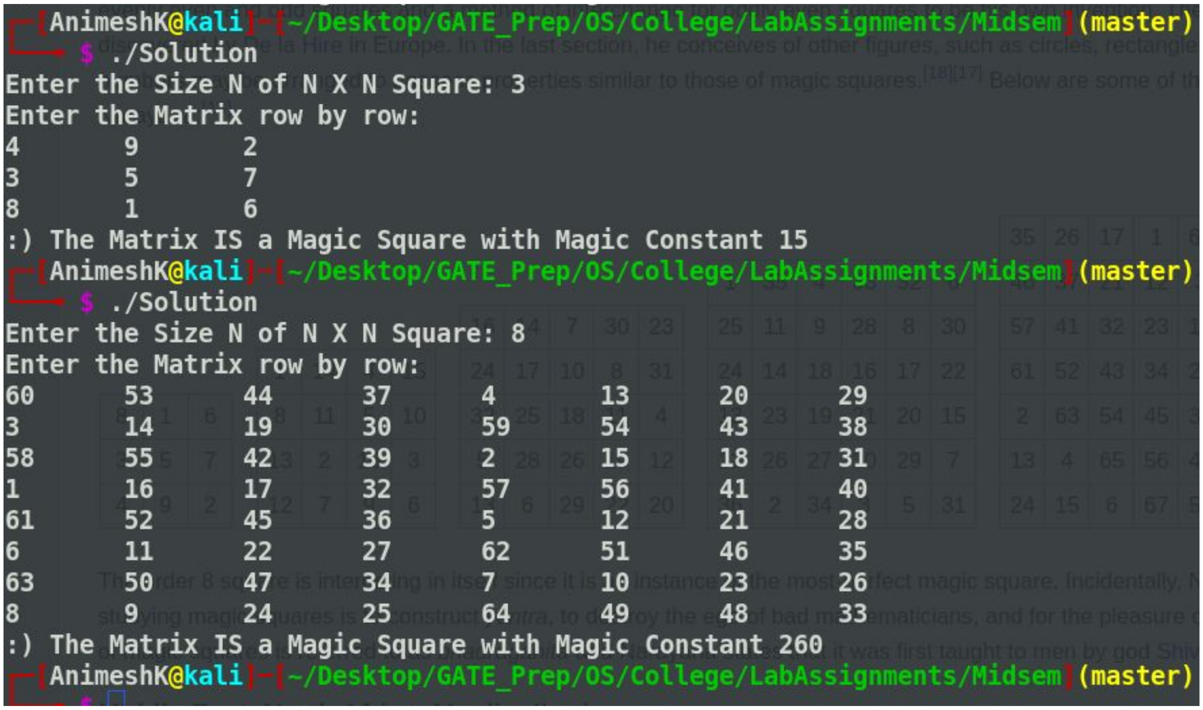
```
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ make Solution
cc     Solution.c   -o Solution
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 5
Enter the Matrix row by row:
1       23      16      4       21
15      14      7       18      11
24      17      13      9       2
20      8       19      12      6
5       3       10      22      25
:) The Matrix IS a Magic Square with Magic Constant 65
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 5
Enter the Matrix row by row:
21 4 16 23 1
11 17 7 14 16
2 10 13 17 23
6 12 19 8 20
25 22 10 3 5
:( The Matrix IS NOT a Magic Square
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
```



```
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 9
Enter the Matrix row by row:
31      76      13      36      81      18      29      74      11
22      40      58      27      45      63      20      38      56
67      4       49      72      9       54      65      2       47
30      75      12      32      77      14      34      79      16
21      39      57      23      41      59      25      43      61
66      3       48      68      5       50      70      7       52
35      80      17      28      73      10      33      78      15
26      44      62      19      37      55      24      42      60
71      8       53      64      1       46      69      6       51
:) The Matrix IS a Magic Square with Magic Constant 369
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
```



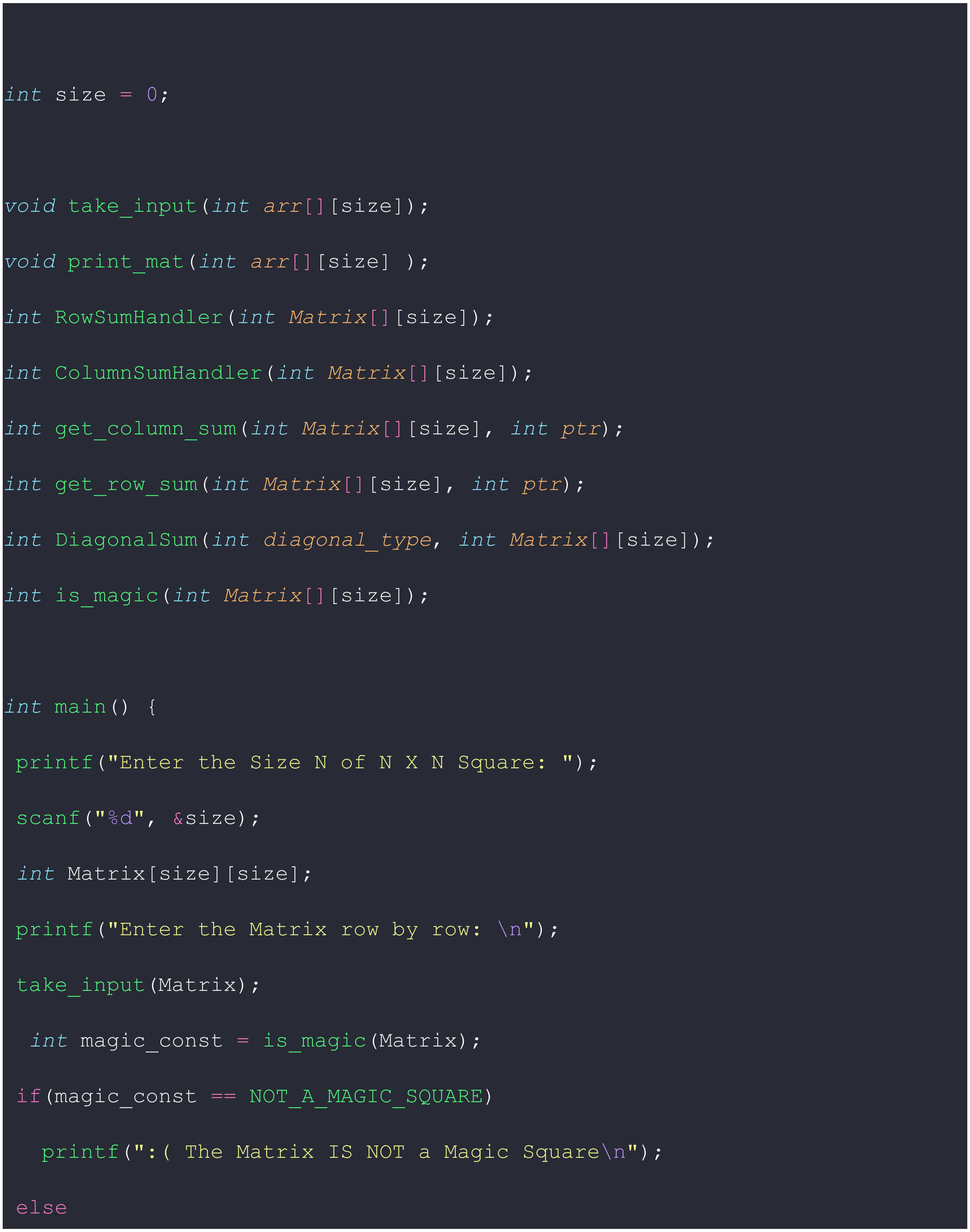
```
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 9
Enter the Matrix row by row:
31      76      13      36      81      18      29      74      11
22      40      58      27      45      63      20      38      56
67      4       49      72      9       54      65      2       47
30      75      12      32      77      14      34      79      16
21      39      57      23      41      59      25      43      61
66      3       48      68      5       50      70      7       52
35      80      17      28      73      10      33      78      15
26      44      62      19      37      55      24      42      60
71      8       53      64      1       46      69      5       52
:( The Matrix IS NOT a Magic Square
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 6
Enter the Matrix row by row:
13      22      18      27      11      20
31      4       36      9       29      2
12      21      14      23      16      25
30      3       5       32      34      7
17      26      10      19      15      24
8       35      28      1       6       33
:) The Matrix IS a Magic Square with Magic Constant 111
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
```



```
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 3
Enter the Matrix row by row:
4       9       2
3       5       7
8       1       6
:) The Matrix IS a Magic Square with Magic Constant 15
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $ ./Solution
Enter the Size N of N X N Square: 8
Enter the Matrix row by row:
60      53      44      37      4       13      20      29
3       14      19      30      59      54      43      38
58      55      42      39      2       15      18      31
1       16      17      32      57      56      41      40
61      52      45      36      5       12      21      28
6       11      22      27      62      51      46      35
63      50      47      34      7       10      23      26
8       9       24      25      64      49      48      33
:) The Matrix IS a Magic Square with Magic Constant 260
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Midsem](master)
└──╼ $
```

C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>



#define MAIN_DIAGONAL 0

#define TRAILING_DIAGONAL 1

#define NOT_A_MAGIC_SQUARE -1
```



```c
int size = 0;


void take_input(int arr[][size]);

void print_mat(int arr[][size] );

int RowSumHandler(int Matrix[][size]);

int ColumnSumHandler(int Matrix[][size]);

int get_column_sum(int Matrix[][size], int ptr);

int get_row_sum(int Matrix[][size], int ptr);

int DiagonalSum(int diagonal_type, int Matrix[][size]);

int is_magic(int Matrix[][size]);


int main() {

 printf("Enter the Size N of N X N Square: ");

 scanf("%d", &size);

 int Matrix[size][size];

 printf("Enter the Matrix row by row: \n");

 take_input(Matrix);

  int magic_const = is_magic(Matrix);

 if(magic_const == NOT_A_MAGIC_SQUARE)

   printf(":( The Matrix IS NOT a Magic Square\n");

 else
```

```
   printf(":) The Matrix IS a Magic Square with Magic Constant %d \n",
magic_const);

 return 0;

}



// This function calculates the magic constant of the square(If it is
Magic Sq)

int is_magic(int Matrix[][size]) {

 pid_t pid1, pid2;

 int fd1[2], fd2[2];

 pipe(fd1);

 pipe(fd2);

 // Forking 2 child processes

 ((pid1 = fork()) && (pid2 = fork()));

 if (pid1 == 0)  {    // Child 1 block

   int sum = RowSumHandler(Matrix);

   // Send the 'sum' to parent process via pipe

   close(fd1[0]);

   write(fd1[1], &sum, 4);

   close(fd1[1]);

   // Succesful termination

   exit(0);

 }

 else if (pid2 == 0)  { // Child 2 block
```

```
  int sum = ColumnSumHandler(Matrix);
  // Send the 'sum' to parent process via pipe
  close(fd2[0]);
  write(fd2[1], &sum, 4);
  close(fd2[1]);
  // Succesful termination
  exit(0);
}
else {   // The Parent Block
  int row_sum, column_sum;
  int diagonal_1_sum = DiagonalSum(MAIN_DIAGONAL, Matrix);
  int diagonal_2_sum = DiagonalSum(TRAILING_DIAGONAL, Matrix);

  // Recieving the Row Sum from First Child
  close(fd1[1]);
  read(fd1[0], &row_sum, 4);
  close(fd1[0]);

  // Recieving the Column Sum from Second Child
  close(fd2[1]);
  read(fd2[0], &column_sum, 4);
  close(fd2[0]);
```

```
   // Comparing the Sum values
   if ((row_sum == column_sum) && (column_sum == diagonal_1_sum) && (diagonal_1_sum == diagonal_2_sum))
     return row_sum; // Returning the Magic Constant of this Magic Square
   return NOT_A_MAGIC_SQUARE;  // Returning the signal of not being a magic square
 }
}


// This function calculates the Diagonal Sum(TRAILING_DIAGONAL or MAIN_DIAGONAL) based on the 'diagonal_type' value
int DiagonalSum(int diagonal_type, int Matrix[][size]) {
 int sum = 0;
 for (int i = 0; i < size; i++) {
   if (diagonal_type == MAIN_DIAGONAL)  // In case of main diagonal
     sum += Matrix[i][i];
   else                // In case of trailing diagonal
     sum += Matrix[size - 1 - i][i];
 }
 return sum;
}


// This function compares the sum of all the rows and returns final row_sum(if any)
int RowSumHandler(int Matrix[][size]) {
 int prev_sum = 0;
 int sum;
 for(int i = 0; i < size; i++) {
   sum = get_row_sum(Matrix, i); // Calculates Row sum of ith row of the Matrix
   if ((prev_sum != 0) && (prev_sum != sum)) // When the consecutive sums don't match
     return NOT_A_MAGIC_SQUARE;
   prev_sum = sum;
 }
 return prev_sum;
}


// This function compares the sum of all the columns and returns final column_sum(if any)
int ColumnSumHandler(int Matrix[][size]) {
 int prev_sum = 0;
 int sum;
 for(int i = 0; i < size; i++) {
   sum = get_column_sum(Matrix, i); // Calculates Column sum of ith Column of the Matrix
   if ((prev_sum != 0) && (prev_sum != sum)) // When the consecutive sums don't match
     return NOT_A_MAGIC_SQUARE;
   prev_sum = sum;
 }
 return prev_sum;
}


// Returns the sum of ptr'th row in the Matrix[][size]
int get_row_sum(int Matrix[][size], int ptr) {
 int sum = 0;
 for (int i = 0; i < size; i++)
   sum += Matrix[ptr][i];
 return sum;
}


// Returns the sum of ptr'th column in the Matrix[][size]
int get_column_sum(int Matrix[][size], int ptr) {
 int sum = 0;
 for (int i = 0; i < size; i++)
   sum += Matrix[i][ptr];
 return sum;
}


// A simple utility to take the Matrix as input
void take_input(int arr[][size]) {
 for (int i = 0; i < size; i++) {
   for (int j = 0; j < size; j++)
     scanf("%d", &arr[i][j]);
 }
}

// A simple utility to print out the matrix
void print_mat(int arr[][size]) {
 for (int i = 0; i < size; i++) {
   for (int j = 0; j < size; j++)
     printf("%d ", arr[i][j]);
   printf("\n");
 }
 printf("\n");
}
```

Efficiency Analysis:

I am using ‘n’ as the size of the input square matrix n x n. The function `int is_magic(int Matrix[][])` is of our interest, it makes calls to some crucial functions. Below are the few functions, which are of the most significance to contribute to the time complexity of the program:

Module#	Function Name	Total #(operations): T(Module#)
A	get_row_sum()	n + 2
B	get_column_sum()	n + 2

C	DiagonalSum()	n + 2
D	RowSumHandler()	n * (n + 1) + 2 = n^2 + n + 2
E	ColumnSumHandler()	n * (n + 1) + 2 = n^2 + n + 2

Efficiency of the parallelised algorithm:*(Multiprocessing using fork())*

The parallelised algorithm as described in previous sections, makes call to below modules in given order:

- Makes call to C for MAIN_DIAGONAL // T(C)
- Makes call to C for TRAILING_DIAGONAL // T(C)
- Two child processes are forked
    - First child process makes call to D and terminates // T(D)
    - Second child process makes call to E and terminates // T(E)
- Then the parent also terminates after some constant time operation.

Analysis

Assuming perfect parallelism, that is, the two child processes are executed on different processors altogether. In this condition the D and E modules will run perfectly parallely. Therefore we can choose the longest of T(D) and T(E) for the total time taken. Since both of them are same, we may choose any one of them

Total number of operations: 2*T(C) + T(D) = 2n + 4 + n^2 + n + 2 = **n^2 + 3n + 6**

Efficiency of the serialised version of the algorithm:

The serial version would have made call to below modules in given order:

- Makes call to C for MAIN_DIAGONAL // T(C)
- Makes call to C for TRAILING_DIAGONAL // T(C)
- Makes call to D // T(D)
- Makes call to E // T(E)
- Then the process terminates after some constant time operation.

Total number of operations: 2*T(C) + T(D) + T(E) = 2n + 4 + 2n^2 + 2n + 4 = **2n^2 + 4n + 8**

Thanks,

Animesh Kumar

CED18I065