# 13th Marathon of Parallel Programming

# WSCAD 2018

*October 2nd, 2018*

## Rules for Local Contest

For all problems, read carefully the input and output session. A sequential implementation is given, and it is against the output of those implementations that the output of your programs will be compared to decide if your implementation is correct. You can modify the program in any way you see fit, except when the problem description states otherwise.

You must upload a compressed file (zip) with your source code, the *Makefile* and an execution script. The script must have the name of the problem. You can submit as many solutions to a problem as you want. Only the last submission will be considered. The *Makefile* must have the rule *all*, which will be used to compile your source code. The execution script runs your solution the way you design it – it will be inspected not to corrupt the target machine.

All Teams have access to the target machine during the marathon. Your execution may have concurrent process from other teams. Only the judges have access to a non-concurrent environment.

The execution time of your program will be measured running it with *time* program and taking the real CPU time given. Each program will be executed at least three times with the same input and the mean time will be taken into account. The sequential program given will be measured the same way. You will earn points in each problem, corresponding to the division of the sequential time by the time of your program (*speedup*). The team with the most points at the end of the marathon will be declared the winner.

# General Information

## Execution

You should use any necessary commands inside your scripts. For instance:

```
export OMP_NUM_THREADS=10
./sum
```

Note that it reads the default input (`stdin`), and can write to the default output (`stdout`).

## Compilation

You must use `CC` or `CXX` inside your *Makefile.* Do not redefine them! Example:

```
FLAGS=-O3
EXEC=sum

all: $(EXEC)
$(EXEC):
    $(CXX) $(FLAGS) $(EXEC).cpp -c -o $(EXEC).o
    $(CXX) $(FLAGS) $(EXEC).o -o $(EXEC)
```

## Test machine

Check the contest website for more information.

# Problem A

## Average minimum distance

Given a directed weighted graph $G = (V, E, w)$ with $V$ the set of vertices, $E$ the set of edges (ordered pairs of vertices), and $w : E \to \mathbb{N} \in [1, |V|[$ the weight of the edges, we want to compute its average minimum distance (AMD) between all pairs of vertices with a path connecting them.

The average minimum distance equation is

$$AMD(G) = \frac{\sum_{v \in V} \sum_{u \in V \wedge v_1 \neq v_2 \wedge p(v,u)} md(v, u)}{\sum_{v \in V} \sum_{u \in V \wedge v \neq u \wedge p(v,u)} 1} \tag{1}$$

where the function $md : V \times V \to \mathbb{N}$ provides the minimum distance between two vertices with an existing path between them and the function $p : V \times V \to$ {true,false} informs if there exists a path from one vertice to another ($v$ to $u$ in Equation 1).

### Input

The input contains only one test case read from a file.

The first line contains two values: the first is the number of vertices $|V|$ and the second is the number of edges $|E|$.

The other lines contain three values: two vertices (whose ids are in the interval $[0, |V|[$) composing an edge and the weight of said edge (in the interval $[1, |V|[$).

The number of vertices will never be bigger than 5000.

### Output

The output contains only one line printing the average minimum distance truncated to an integer.

### Example

| Input file | Output |
|---|---|
| 10 7<br>0 4 3<br>4 7 4<br>4 8 6<br>7 0 7<br>7 4 1<br>8 3 8<br>9 3 2 | 7 |

# Problem B

## Levenshtein Distance

The Levenshtein Distance, named after the Soviet mathematician Vladimir Levenshtein, is a metric for measuring the difference between two strings. The difference between two strings is the minimum number of single-character edits required to change one string into the other. A single-character edit is either an insertion, deletion, or substitution one one character in a string.

The Levenshtein Distance between two strings $a$ and $b$ (with respective lenghts $|a|$ and $|b|$) is $\mathrm{lev}_{a,b}(|a|,|b|)$, where

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{2}$$

where $1_{(a_i \neq b_j)}$ equals 0 when $a_i = b_j$ and 1 otherwise. Thus, $\mathrm{lev}_{a,b}(i,j)$ may be interpreted as the distance between the first $i$ characters of $a$ and the first $j$ characters of $b$.

You shall parallelize a sequential algorithm that computes the Levenshtein Distance between two strings. The provided sequential algorithm is a recursive implementation following the mathematical definition presented above. Its inputs are the compared strings and its output is an integer, which is the Levenshtein distance separating the two strings.

The Levenshtein distance between "chicken" and "kicking" is 4, since the following four single-character edits change one into the other, and there is no way to do it with fewer than four edits:

1. **c**hicken → hicken (removes first letter);
2. **h**icken → **k**icken (replaces first letter);
3. kick**e**n → kick**i**n (replaces second to last letter);
4. kickin → kickin**g** (inserts last letter).

## Input

The input must be read from standard input. It is composed of two lines, each of them an input string.

## Output

The output must be written to standard output. It is an output integer indicating the computed distance.

## Example

| Input | Output |
|---|---|
| chicken<br>kicking | 4 |

# Problem C

## Himeno Benchmark

The Himeno benchmark was developed by Dr. Ryutaro Himeno in 1996 at the RIKEN Institute in Japan. It is highly memory intensive, bound by memory bandwidth on modern processors, in contrast to the highly compute intensive Linpack.

The Himeno benchmark focuses on the solution of a 3D Poisson equation in generalized coordinates on a structured curvilinear mesh. With the processing time dominated by the Poisson solution, it makes the Poisson procedure a good measure of overall performance. Using finite differences, the Poisson equation is dicretized in space yielding a 19-point stencil.

Your task is to improve performance of the source-code using parallel strategies.

## Input

The input file contains only one test case. The first three lines contain the size of a matrix *imax*, *jmax* and *kmax* ($0 < N \leq 10^4$). Next, *nn* is the number of iterations.

*The input must be read from the standard input.*

## Output

The output must be the *Gosa number* that is the residual to measure convergence. It must have 6 (six) digits of precision.

*The output must be written from the standard input.*

## Example

| Input | Output |
|---|---|
| `64`<br>`64`<br>`128`<br>`10` | `0.003069` |

# Problem D

## Cholesky Decomposition[1]

Every symmetric, positive matrix $A$ can be decomposed into a product of an unique lower triangular matrix $L$ and its transpose:

$$A = LL^T \tag{3}$$

$L$ is called the *Cholesky factor* of $A$, and can be interpreted as a generalized square root of $A$.

Your task is to improve performance of the source-code using parallel strategies. We are not interested in finding out which decomposition is better, therefore is not allowed to change the Cholesky decomposition algorithm.

## Input

The input file contains only one test case. The first line contains the size of a square matrix ($0 < N \leq 10^4$). Next, $N$ lines are the rows of the matrix, $N$ real numbers per row.

*The input must be read from a file named cholesky.in.*

## Output

The output must have the lower Cholesky factor $L$ from the symmetric matrix $A$.

*The output must be written to a file named cholesky.out.*

## Example

| Input | Output |
|---|---|
| 4<br>4.84019 0.39438 0.78310 0.79844<br>0.39438 4.19755 0.33522 0.76823<br>0.78310 0.33522 4.47740 0.62887<br>0.79844 0.76823 0.62887 4.91620 | 2.20004 0.39438 0.78310 0.79844<br>0.17926 2.04094 0.33522 0.76823<br>0.35595 0.13298 2.08159 0.62887<br>0.36292 0.34453 0.21804 2.14901 |

[1]Source: `https://rosettacode.org/wiki/Cholesky_decomposition`

# Problem E

## Mandelbrot Set

The Mandelbrot set is the set of complex numbers $c$ for which the function $f_c(Z) = Z^2 + c$ does not diverge when iterated from $Z = 0$, i.e., for which the sequence $f_c(0)$, $f_c(f_c(0))$, etc, remains bounded in absolute value. Its definition and name are due to Adrien Douady, in tribute to the mathematician Benoit Mandelbrot. The set is connected to a Julia set, and related Julia sets produce similarly complex fractal shapes. The Mandelbrot set is the set of values of $c$ in the complex plane for which the orbit of 0 under interation of the quadratic map

$$Z_{n+1} = Z_n^2 + c \tag{4}$$

remains bounded. That is, a complex number $c$ is part of the Mandelbrot set if, when starting which $Z_0 = 0$ and applying the iteration repeatedly, the absolute value of $Z_n$ remains bounded however large $n$ gets.

Create a parallel version of the given sequential algorithm that generates a textual approach for the Mandelbrot set.

## Input

The first line informs the maximum number of rows. The second line presents the maximum number of columns. Finally, the last one informs the number of iterations.

*The input must be read from the standard input.*

## Output

Textual representation of the Mandelbrot set.

*The output must be written to the standard output.*

## Example

Input

```
23
79
240
```

Output

```
...............................................................................
.......................................................#.......................
..................................................########.....................
..................................................##########...................
.....................................#......##.###############...#.............
.....................................###############################.####......
....................................###################################........
..................................#######################################......
............#......##..........############################################....
.............##############....##############################################..
..........###################.############################################.#...
.....####################################################################......
.....####################################################################......
..........###################.############################################.#...
.............##############....##############################################..
............#......##..........############################################....
..................................#######################################......
....................................###################################........
.....................................###############################.####......
.....................................#......##.###############...#.............
..................................................##########...................
..................................................########.....................
.......................................................#.......................
```