5th Marathon of Parallel Programming

SBAC'10

October 28th, 2010.

Rules

For all problems, read carefully the input and output session. For all problems, 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 *tar.gz* with your source code, the *Makefile* and an execution script. The program to execute should 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* and will be used to compile your source code before submit. The execution script must follow the *Cluster Altix-XE* submitting rules – that will be inspected not to corrupt the target machine.

All teams have access to the target machine during the marathon. Your execution is queued by the Sun Grid Engine (SGE) and does not have concurrent process. During the marathon, SGE will be examined to find any suspect execution.

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 twice with the same input and only the smaller 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.

This problem set contains 5 problems; pages are numbered from 1 to 8.

Problem A

TSP

The *Traveling Salesman Problem* –TSP – is a well known problem in computing area because of its complexity and time consuming (CPU-bound). Given a list of cities and their pairwise distances, the task is to find a shortest possible path that visits each city exactly once.

A TSP problem can be modeler as an undirected graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight.

Write a parallel version that find the solution to the symmetric TSP problem (for all pair $i, j, C_{ij} = C_{ji}$)

Input

The input contains only one test case. The first line contains two integers: the number of vertices in the problem (C) and the number of edges (E) separated by a single blank space ($1 \le C \le 50, 1 \le E \le 10^4$). The next E lines contain three integers that represent the path between two vertices (I, J) and its weight (W) separated by a single space ($1 \le I, J \le C, 1 \le W \le 10^5$). Remember that $C_{ij} = C_{ji}$.

The movie project must be read from a file named tsp.in

Output

The output contains only one line with the number of the shortest possible tour for a symmetric TSP.

The output must be written to a file named tsp.out

Example

Input	Output for the input
4 6 1 2 3 1 3 3 1 4 2 2 3 2 2 4 3 3 4 1	8

Problem B

Smooth

In image-processing area, there is a simple smooth algorithm that reduces noise and also reduces the amount of intensity variation between groups of pixels. This algorithm is a mean filtering and has the effect of eliminating pixel values which are unrepresentative of their surroundings.

Usually, this smooth algorithm takes a 3x3 group of pixels and calculates the average of the new pixel, as shown on Figure B.1.

x_0	x_1	x_2
x_3	x_4	x_5
x_6	x_7	x_8

$$x_4' = \frac{x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8}{9}$$

Figure B.1. x_4' is the new value for x_4 pixel in a 3x3 group.

Write a parallel version of this smooth algorithm. The sequential program uses it in a movie project.

Input

A movie project has up to 4000 images and all images have the same size (max. 2048x1536 pixels). On all images, a pixel is represented by a 32 bits word. The input file is in binary format: the first 32 bits word represents the width of an image; the next 32 bits word represents the height of an image; all the images come next, until the end of file.

The movie project must be read from a file named movie.in

Output

The output file must be in binary format, keeping the same structure from input file. This output file must have all the processed images.

The output must be written to a file named movie.out

Problem C

Matrix

Based on the mathematical concepts, matrix is a two-dimension array of numbers (or variables representing numbers). An *n* x *m* matrix has *n* rows and *m* columns of elements.

A multiplication of two matrices, *A* and *B*, produces the matrix *C*, whose elements, $c_{i,j}$, can be computed as follows:

$$c_{i,j} = \sum_{k=0}^{p-1} a_{i,k} b_{k,j}$$

where *A* is an *m* x *p* matrix and *B* is a *p* x *n* matrix ($0 \leq i < n$, $0 \leq j < m$). This multiplication is illustrated in Figure C.1.
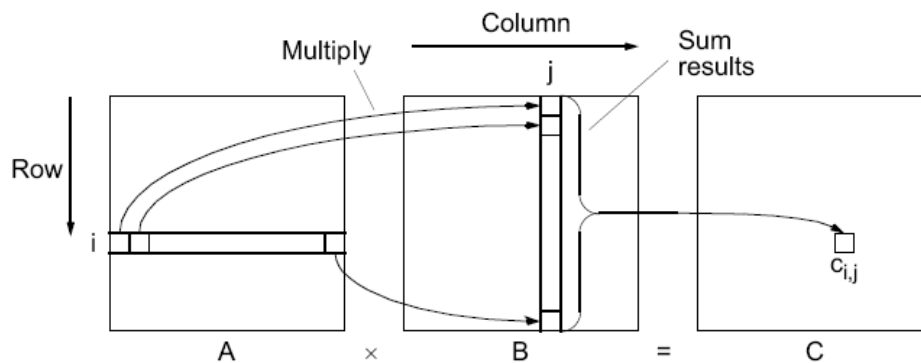

Figure C.1. – Matrix multiplication, *C* = *A* x *B*.

Write a parallel program that computes the multiplication of two matrices.

Input

The input contains only one test case. The first line contains two integers: the numbers of rows (*M*) and the number of columns (*P*) of a matrix *A* separated by a blank space ($0 \leq M, P < 5000$). The next *M* lines contain *P* integers in each line separated by a blank space representing the $a_{m,p}$ element of the matrix *A* ($0 \leq m < M, 0 \leq p < P$). The next line contains two integers: the numbers of rows (*P*) and the numbers of columns (*N*) of a matrix *B* separated by a blank space ($0 \leq N < 5000$). Notice that the same value *P* is guarantee in the input. The next *P* lines contain *N* integers in each line representing the $b_{p,n}$ element of the matrix *B* ($0 \leq n < N$).

The movie project must be read from a file named matrix.in

Output

The first output line contains two integers: the number of rows (*M*) and the number of columns (*N*) of the matrix *C*, separated by a single blank space. The next *M* lines contains *N* elements on each line separated by a single blank space representing the $c_{n,m}$ element of the matrix *C* ($0 \le n < N$, $0 \le m < M$).

The output must be written to a file named matrix.out

Example

Input	Output for the input
4 3 2 3 0 0 2 -1 1 0 2 3 1 4 3 3 2 2 -1 7 1 -4 8 1 3	25 7 -14 6 1 -11 18 4 5 45 11 5

Problem D

Gauss Elimination

Suppose the existence of a linear equation:

$$a_{0,0}x_0 + a_{0,1}x_1 + a_{0,2}x_2 + \quad \ldots \quad + a_{0,n-1}x_{n-1} \quad = b_0$$
$$a_{1,0}x_0 + a_{1,1}x_1 + a_{1,2}x_2 + \quad \ldots \quad + a_{1,n-1}x_{n-1} \quad = b_1$$
$$a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 + \quad \ldots \quad + a_{2,n-1}x_{n-1} \quad = b_2$$
$$\ldots \quad\quad\quad \ldots \quad \ldots \quad\quad\quad \ldots$$
$$a_{n-1,0}x_0 + a_{n-1,1}x_1 + a_{n-1,2}x_2 + \quad \ldots \quad + a_{n-1,n-1}x_{n-1} \quad = b_{n-1}$$

witch, in matrix form, is

$$Ax = b$$

The objective of solving this system of equations is to find values for the unknowns x_0, x_1, …, x_{n-1}, given values for $a_{0,0}$, $a_{0,1}$, …, $a_{n-1,n-1}$, and b_0, b_1, …, b_{n-1}.

The Gauss Elimination algorithm transforms this linear equation into a triangular system of equation. It uses the characteristic of linear equation that any row can be replaced by that row added to another row multiplied by a constant. This has the effect of making all the elements in the ith column below the ith row zero because

$$a_{j,i} = a_{j,i} + a_{i,i}\left(\frac{-a_{j,i}}{a_{i,i}}\right) = 0$$

Figure D.1 show the situation when row i is being considered.
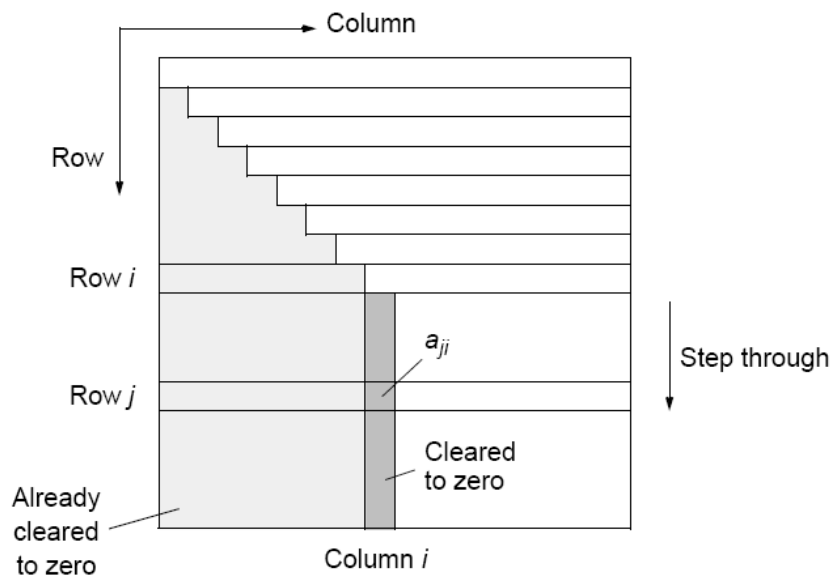

Figure D.1. Gaussian elimination.

To minimize decimal problems, this procedure is modified into so-called *partial pivoting* by swapping the *i*th row with the row below it that has the largest absolute element in the *i*th column of any of the rows below the *i*th row if there is one.

Write a parallel program that solves the linear equation using the Gauss Elimination.

Input

The input contains only one test case. The first line contains one integer (N) representing the number of unknowns and the number of rows/columns if matrix A ($1 \leq N \leq 4000$). The next N lines contains N integers that represent the $a_{i,j}$ element of the equation in the matrix separated by a single space ($1 \leq a_{i,j} \leq 20$). The last line contains N integers separated by a space representing the elements of the b_n vector.

The movie project must be read from a file named gauss.in

Output

The output contains only one line with N values separated by a single space representing the vector x_n, where each value has exactly two decimal precision.

The output must be written to a file named gauss.out

Example

Input	Output for the input
3 6 2 -1 2 4 1 3 2 8 7 7 13	1.00 1.00 1.00

Problem E

Quicksort

Quicksort is a popular sequential sorting algorithm that, on average, makes $O(n\log n)$ comparisons to sort n elements.

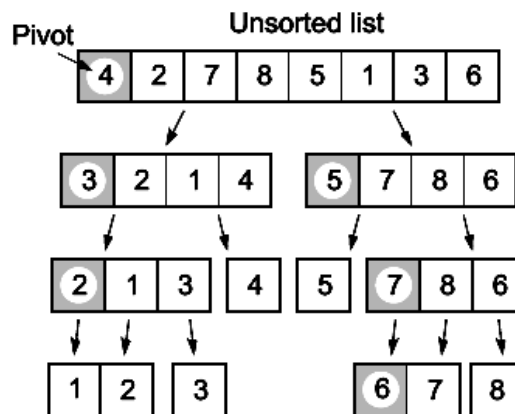Figure E.1 show a tree structure for an unsorted vector of integers.

Figure E.1. Quicksort algorithm.

Write a parallel program that uses the Quicksort as the main algorithm to sort keys – the solution can use other algorithms and Quicksort must be part of it.

Input

The input file contains only one test case. The first line contains the total number of keys (*N*) to be sorted ($1 \leq N \leq 10^{10}$). The following lines contain *N* keys, each key in a separate line. A key is a seven-character string made up of printable characters (0x21 to 0x7E – ASCII) not including the space character (0x20 ASCII).

The movie project must be read from a file named <u>quicksort.in</u>

Output

The output file contains the sorted keys. Each key must be in a separate line.

The output must be written to a file named <u>quicksort.out</u>

Example

Input	Output for the input
11 SINAPAD SbacPad Wscad10 Sinapad 1234567 LADGRID WEAC-10 CTDeWIC sinaPAD MPP2010 SINApad	1234567 CTDeWIC LADGRID MPP2010 SINAPAD SINApad SbacPad Sinapad WEAC-10 Wscad10 sinaPAD