4th Marathon of Parallel Programming

SBAC’09

October 30th, 2009.

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 submit your source code 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 script must be a *Makefile* with two rules: *all* and *run*.
All teams have access to a test machine during the marathon. Your execution may have concurrent process from other teams. Only the judges have access to the target machine. The execution time of your program will be measured running it with time program and taking the user 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. The team with the most points at the end of the marathon will be declared the winner.

This problem set contains 6 problems; pages are numbered from 1 to 10.

Problem A

3SAT[1]

Satisfiability is a problem that takes an expression made up of the conjunction of disjunctions between Boolean variables. To solve the problem you must determine whether or not an assignment of TRUE or FALSE to the Boolean variables exists that will make the entire expression evaluate to TRUE. The Satisfiability belongs to the NP-complete class problem.

The 3SAT problem is a version of Satisfiability that restricts the size of the disjunction subexpressions to contain exactly three variables. An example of a 3SAT expression would be:

(X_1 | !X_2 | X_3) & (X_3 | X_2 | !X_1) & (X_2 | X_1 | !X_3) & (!X_1 | !X_2 | !X_3)

where '|' is OR, '&' is AND, and '!' signifies NOT. This expression is satisfied when the variables X_1 and X_2 are TRUE and X_3 is FALSE.

Write a parallel program that determines if there is an assignment of Boolean values that will satisfy the given 3SAT expression.

Input

The input contains only one test case. The first line contains two integers: the maximum number of variables that will be in the expression (N) and the number of disjunction subexpressions in the file (K), separated by a single space ($1 \leq N \leq 100$, $1 \leq K \leq 10^4$). The next K lines will contain three integers from $abs([1,N])$ separated by a space. These integers represent the subscript of a Boolean variable and a negative value represents the negation of the Boolean variable within that subexpression.

The input must be read from standard input

Output

If there is an assignment that satisfies the entire input expression, the output contains N lines. Each line corresponds to one of the N variables and the Boolean value needed.

[1] Based on 3SAT problem from 2009 Intel Threading Challenge.

The format of each line is an integer from [1,*N*], a space, and the character ‘T’ or ‘F’ for the assignment of TRUE or FALSE, respectively. The variables must be ordered. Otherwise, if there is no solution, the output contains only one message ‘Solution is not possible’.

The output must be written to standard output

Example 1

Input	Output for the input

`3 4`
`1 -2 3`
`3 2 -1`
`2 1 -3`
`-1 -2 -3`
	`1 T`
	`2 T`
	`3 F`

Example 2

Input	Output for the input

`1 2`
`1 1 1`
`-1 -1 -1`
	`Solution is not possible`

Problem B

DNA subsequences[2]

FASTA archive is a text-based format to store DNA/RNA in witch the bases are represented using single-letter codes. In bioinformatics, this file is used to sequence alignment and string matching.

Write a parallel program to find DNA subsequences (a.k.a. query string) in a FASTA database. If a query string matches within multiple sequences, each result must be reported. If a query string matches multiple locations in the same sequence, the earliest position that matches exactly must be reported.

Input

The input must be read from two different files. Both of them follow FASTA format.

The FASTA format represents many sequences. Each sequence contains one line with the DNA description followed by several lines with the bases. The description line begin with a greater-than character ('>'). The bases sequence are made up of only four characters ('A', 'T', 'C', 'G') and divided by line within 80 characters per line. The file ends with the EOF-mark. The base length is up to 1,000,000 bases.

The database must be read from a file named dna.in

The query file must be read from a file named query.in

Output

The output contains the string matching results. For each query string, the program must output its description in one line followed by its report. If the query string was found within the database, the report contains the sequences description followed by the position within the sequences that exactly match. If the query string is not found within any sequences, a 'NOT FOUND' message must be printed.

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

[2] Based on String Matchin problem from 2009 Intel Threading Challenge.

Example

FASTA database

```
>Escherichia coli partial genome (1)
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTG
TCTGATAGCAGCTTCTGAACTG
GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCA
>Escherichia coli partial genome (2)
CTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACA
CAACATCCATGAAACGCATTAG
CACCACCATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGACGCGTA
CAGGAAACACAGAAAAAAGCCC
GCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCG
AGTGTTGAAGTTCGGCGGTACA
```

Query file

```
>Query string #1
TATAGG
>Query string #2
TTTT
>Query string #3
ATCG
>Query string #4
AACTGG
```

Output

```
>Query string #1
>Escherichia coli partial genome (2)
17
>Query string #2
>Escherichia coli partial genome (1)
3
>Escherichia coli partial genome (2)
178
>Query string #3
NOT FOUND
>Query string #4
>Escherichia coli partial genome (1)
75
```

Problem C

Machin's π

In 1706, John Machin proposed a simple formula to compute the mathematical constant π that converges very quickly:

$$\frac{\pi}{4} = 4 \cdot arc\cot(5) - arc\cot(239)$$

In his formula, the arc cotangent function was calculated using an expansion Taylor series:

$$arc\cot(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1) \cdot x^{(2n+1)}}$$

Since π number has infinite decimal places, computational implementation reduces it to only some "trillion" places.

Write a parallel program that computes the π number.

Input

The input contains only one test case. The first line contains only one number (D) that represents the amount of decimal places ($1 \leq D \leq 10^{127}$).

The input must be read from standard input

Output

The output contains only one line printing the π number with exact D decimal places.

The output must be written to standard output

Example

Input	Output for the input

`100`	`3.14159238864170925052793894`
	`23399213277482179339167560 60`
	`53206800212881846333845186 03`
	`582058144592135732`

Problem D

MSD Sorting[3]

The Radix Sort algorithms can be classified in two basic groups: least significant digit (LSD) and most significant digit (MSD). The LSD approach examines the digits in the keys in a right-to-left order, working with the least significant digits. The other approach (MSD) examines the digits in the key in a left-to-right order. Figure D.1 show an example of LSD and MSD radix-sort using 3-digits integer keys.

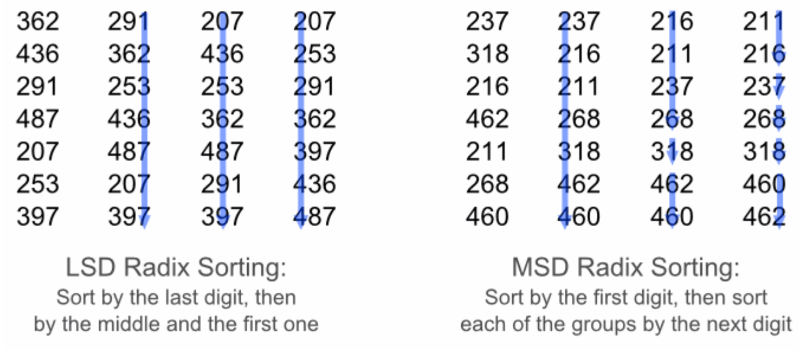

Figure D.1. – Summary of LSD and MSD approaches.

When a digit is chosen for MSD sorting, an internal sorting can be used to organize the keys, grouping them with the same digit. This internal sorting can use a "fat-pivot" quicksort algorithm (three-way partitioning), taking advantage of repeating digits.
Write a parallel program that uses a MSD algorithm to sort keys.

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 input must be read from a file named radix.in

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 radix.out

[3] Bases on Radix Sort problem from 2009 Intel Threading Challenge.

Example

Input	Output for the input

`11`
`SINAPAD`
`SbacPad`
`Wscad09`
`Sinapad`
`1234567`
`LADGRID`
`WEAC-09`
`CTDeWIC`
`sinaPAD`
`MPP2009`
`SINApad`
	`1234567`
	`CTDeWIC`
	`LADGRID`
	`MPP2009`
	`SINAPAD`
	`SINApad`
	`SbacPad`
	`Sinapad`
	`WEAC-09`
	`Wscad09`
	`sinaPAD`

Problem E

Software Testing

White box testing is a difficult task for engineering. They have to analyze the source code and build a control flow graph before define the test cases. In order to decrease the manual effort, some tools find that graph and show some information about the source code.

One of this information, applying to object oriented codes, is to find the relationship between classes/objects. When the number of relationship is very high, the software testing becomes prohibited.

Write a parallel version of the source code that calculates one information of a given control flow graph.

Input

The input contains only a test case. The first line contains the number (N) of nodes. The following lines contain pairs of X and Y that represents a vertex from node X to node Y and vice-versa ($1 \leq X, Y \leq N$).

The input must be read from standard input

Output

The output contains only one line with the number of relationship found on the control flow graph.

The output must be written to standard output

Example

Input	Output for the input

`5`
`1 2`
`1 3`
`1 4`
`1 5`
`2 3`
`2 4`
`3 4`
`4 5`
	`4`

Problem F

Fibonacci numbers

Fibonacci numbers is a well-kwon sequence of numbers that follows the recurrence relation $F_n = F_{n-1} + F_{n-2}$, with seeds values $F_0=0$ and $F_1=1$.
The Fibonacci numbers are used in many researches such as mathematics, computer science and physics. Besides, it can be found in nature too, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple among others.
Write a program that calculates the Fibonacci sequence.

Input

The input contains only one test case. The first line contains only one number (N) that will be used to search the F_n number on the Fibonacci sequence ($0 \leq N \leq 10^5$).
The input must be read from standard input.

Output

For the input test case, your program will output one line containing the number of the Fibonacci sequence.
The output must be written to standard output.

Example 1

Input	Output for the input
`11`	`89`

Example 2

Input	Output for the input
`20`	`6765`