Thank you so much for your application. As part of the application process, we are asking candidates that will be engaged in some software engineering to complete **two coding challenges**, which are described in detail below. There's no time limit on it, though we ask for a response within a week.

Please solve the problem using your own code. You can use books, resources on the Internet, or even friends to figure out how to solve the problem-- just don't ask them to solve it for you or tell you what to type as you type things. **If a library exists to solve the specific problem, please do not use it.**

You are free to use any computer language you like so long as it's something that's freely available and that we will be able to compile and run on our Windows or Linux desktop computer for scoring.

You are also free to contact us to ask questions to clarify, or for additional pointers. As in the real world, good questions are a plus; excessive hand-holding will be a minus.

Please submit the following in an email to wilmes@kcse.com when you are ready:

1. Your program's source code, zipped or tarred for both challenges.
2. Build instructions if any are needed (e.g., compile instructions for a C++ code).
3. A short statement with your reasoning for using the language that you did for this particular challenge.

Thanks,

The K&C Hiring Committee

Challenge 1 – Super Word Search

Introduction

In a typical word search puzzle (http://en.wikipedia.org/wiki/Word_search), you are given an NxM grid of seemingly random letters and a list of P words that are in the grid. The words can be found going in any of the 8 directions in a two-dimensional grid:

- top to bottom
- bottom to top
- left to right
- right to left
- bottom left to top right
- bottom right to top left
- top left to bottom right
- top right to bottom left

You're a college professor (for English and Topology, of all things), and your students have become very good at traditional Word Search. Since you want them to continue spending time on academic games, you created a variant of Word Search (inventively) called Super Word Search.

Description

As with the standard word search, you get an NxM grid of letters, and P words that are to be found in the grid. You also get a "mode" flag with one of the following values: WRAP, NO_WRAP. The flag value indicates whether words can wrap-around after they hit a boundary of the grid.

Row numbers start at 0 (top row) and go to N-1 (bottom row). Column numbers start at 0 (leftmost column) and go to M-1 (rightmost column). Grid coordinates are specified as (row_num, column_num).

Here is an example to illustrate the difference between WRAP and NO_WRAP:

```
 012

 ---

0|ABC

1|DEF

2|GHI
```

"FED" is a word that starts at (1,2) and ends at (1,0).If we are in WRAP mode:

- "CAB" is a word that starts at (0,2) and ends at (0,1).
- "GAD" is a word that starts at (2,0) and ends at (1,0).
- "BID" is a word that starts at (0,1) and ends at (1,0).

If we are in NO_WRAP mode:

- "FED" is a word that starts at (1,2) and ends at (1,0).
- "CAB" is not a word since it requires wrapping in the horizontal direction.
- "GAD" is not a word since it requires wrapping in the vertical direction.
- "BID" is not a word since it requires wrapping in the horizontal and vertical directions.

A letter in the grid is not allowed to be in a word more than once. So, while technically "HIGH" can be found in the above grid in WRAP mode, we will not allow it because it uses the H at (2,1) twice.

Input Format

N M

N rows of M letters

"WRAP" or "NO_WRAP"

P

P words with 1 word per lines

Output Format

Your program should accept the name of an input file which will contain data in the above format.

For each of the P words, you are to output the start and end coordinates of that word in the format "(row_start, column_start) (row_end, column_end)". If the word cannot be found in the grid, output "NOT FOUND".

You are guaranteed that each word will occur at most once in the grid, so a word's start and end coordinates will always be unique (if the word is in the grid), and will never be ambiguous.

Your program can write its output to the screen/console.

Examples

Example Input	Example Output
3 3	(1,2) (1,0)
ABC	NOT FOUND
DEF	NOT FOUND
GHI	NOT FOUND
NO_WRAP	NOT FOUND
5	
FED	
CAB	
GAD	
BID	
HIGH	
4 3	(1,2) (1,0)
ABC	(0,2) (0,1)
DEF	(0,0) (2,0)
GHI	(3,2) (0,2)
JKL	NOT FOUND
WRAP	
5	
FED	
CAB	
AEIJBFG	
LGEC	
HIGH	

Challenge 2 – Nearest-Neighbors Algorithm

Description

For this challenge, please write a program that takes as input:

1. A floating point radius r,
2. The number of points N,
3. A series of lines of 3D points with each coordinate separated by a space

The ID for each point is generated sequentially starting from 0 and incrementing by 1. Your program should print to the screen for each node the node id, followed by a colon, followed by a comma separated list the id of each neighbor to that node. A neighbor of a node n is another node n_i whose distance from n is less than r. Formally, the neighbors of a node n is the set of nodes $\{n_i | \|n - n_i\| < r\}$. See the example output below.

Please discuss the time and space complexity of your solution and any implementation decisions that you feel were important when writing your program.

Input Format

r (input radius)

N (number of points)

x1 y1 z1

…

xN yN zN

Output Format

0: $n_0^0, \ldots, n_{m_0}^0$

...

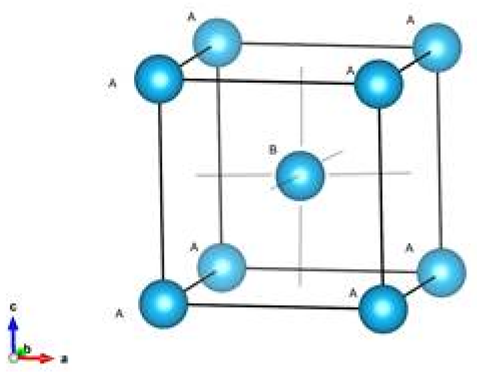
N-1: $n_0^{N-1}, \ldots, n_{m_{N-1}}^{N-1}$

Where n_j^i denotes node i's jth neighbor.

Constraints

- $1 \leq n \leq 10^5$, where n is the number of nodes.
- $\{(x, y, z) \in \mathbb{R}^3 : -10^7 \leq x, y, z \leq 10^7\}$
- The node IDs are enumerated from 1 to n.
- There are no other constraints on the input. You may format the input however you wish.

Example



We can consider the case where one corner is at (0, 0, 0) and the far corner is at (1, 1, 1). The input would then be:

```
1.0
9
0 0 0
1 0 0
0 1 0
0 0 1
1 1 0
0 1 1
1 0 1
1 1 1
0.5 0.5 0.5
```

The output should be:

```
0: 1, 2, 3, 8
1: 0, 4, 6, 8
2: 0, 4, 5, 8
3: 0, 5, 6, 8
4: 1, 2, 7, 8
5: 2, 3, 7, 8
6: 1, 3, 7, 8
7: 4, 5, 6, 8
8: 0, 1, 2, 3, 4, 5, 6, 7
```