CODECON

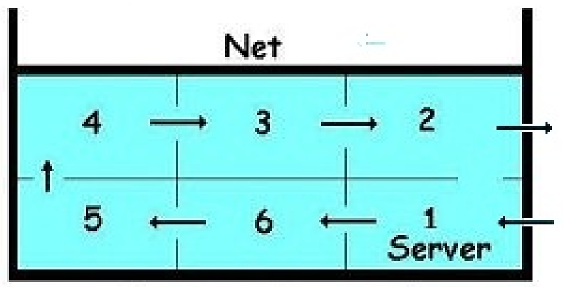
Memory Limit: 1024 MB
Time Limit: 5 s

Volleyball Rotations (100 points)

Introduction

You are the Captain of the **Bloomberg Volleyball Team**. In order to make an effective lineup you want to know who will be on the court after a certain number of **rotations** as well as **where on the court they will be**.

In volleyball, 6 players are on the court at a time and they **rotate clockwise one position** every time their team gains the right to serve. A volleyball court is referenced with numbers 1 - 6. Starting from the lower right corner is position 1 going all the way around the court in a counter-clockwise direction to position 6 at the lower middle.



Court positions and rotation direction

You have **will have anywhere between 7 and 26 people on your team**. Because your team is > 6, there will be players waiting off court on the **sidelines** while six play. Every **rotation**, one of these **sideline** players will come into the court at position 1 (server position), while one player will leave the court at position 2 and join the **sideline at the back of the line**.

The **setter** will **always** be in the **starting lineup** and will not be rotated off the court. This means that when the **setter** is at position 2, when it comes time for a rotation instead of leaving the court and joining the sideline, they will instead go directly to position 1.

Input Specifications

Your program will take from **STDIN:**

- The first line will be the number of rotations that the team will undergo `1 ≤ N ≤ 200`
- The second line will be the name of the **setter**. Names will only contain alphabetical characters.
- The third line will be the starting lineup. The setter will always be in the starting lineup. Each player name will be a **unique string**. The names are **space delimited**. There will always be **6** people in the starting lineup.
- The fourth line will contain the players on the sidelines. The first token will be the number of players on the sideline `1 ≤ S ≤ 20`, followed by that many space-delimited unique player names.The first name will be the first to rotate in.

Output Specifications

Based on the input, print out the **lineup** after the **N number of rotations** ordering them based on which **position on the court** they will end up.

Sample Input/Output

Input

```
3
A
A B C D E F
4 G H I J
```

Output

```
I E F A G H
```

Explanation

The starting lineup was:
######<-Net
D C B
E F A
[G H I J]<- Sideline Players

Notice how A is in position 1, B position 2 etc. The input order will correspond to where they are on the court, the first player is in position 1, second position 2 etc.
After 3 rotations, the lineup looks like so:
######<-Net
A F E
G H I
[J B C D]<- Sideline Players

Therefore the answer, starting with player at position 1, is I E F A G H

Input

```
6
A
A B C D E F
4 G H I J
```

Output

```
A G H I J B
```

Explanation

This demonstrates what happens to A (the setter). Since they do not come off the court they rotated back to position 1 on the sixth rotation.

The starting lineup was:
######<-Net
D C B
E F A
[G H I J]<- Sideline Players

After 5 rotations, the lineup looks like so:
######<-Net
H G A
I J B
[C D E F]<- Sideline Players

The last rotation takes A back to position 1 and none of the Sideline Players get to come in. This results in the lineup:

######<-Net
I H G
J B A
[C D E F]<- Sideline Players

Therefore the answer, starting with player at position 1, is A G H I J B

CODECON

Memory Limit: 1024 MB
Time Limit: 5 s

Sum Sum Cryptography (150 points)

Introduction

Alice and Bob are sending encrypted messages to each other. Your task is to intercept and decode the message that Alice has received.

Bob sends Alice a positive integer `N`) as the message, and Alice will decode each message by solving the equation $x^2 + y^2 + z^2 = N$ where x, y, z are positive non-zero integers.

To create the decoded output, Alice sums up all components of all unique `(x,y,z)` tuples together.

For example, (1,2,3) and (2,3,4) are unique tuples; (1,2,3) and (2,3,1) are not unique tuples.

Note that an entirely naive solution will timeout!

Input Specifications

The only input line will be a number $3 \le N \le 3000000$.

Output Specifications

The output will contain one positive integer as the decoded result.

Sample Input/Output

Input

```
42
```

Output

```
10
```

Explanation

For 42, the unique (x, y, z) would be (1, 4, 5), and 1 + 4 + 5 is 10.

Input

```
99
```

Output

```
47
```

Explanation

For 99, the unique (x, y, z) could be (1, 7, 7), (3, 3, 9), (5, 5, 7), and 1 + 7 + 7 + 3 + 3 + 9 + 5 + 5 + 7 = 47

Input

```
1234566
```

Output

```
478112
```

Memory Limit: 1024 MB
Time Limit: 5 s

Dependency Gambit (350 points)

Introduction

In a world of ever-growing software stacks, building the latest-and-greatest libraries is starting to get complicated! To help simplify the build process, you've decided to write a tool to help generate a library link-line from a dependency list. Given a list of libraries & their dependencies, and a target library we want to build, print out the perfect linkline according to the rules listed in the Output Specs.

Input Specifications

The first line of input is the name of the Library we're looking to build. Library names will *only* contain lowercase letters `a-z` and hyphens `-`. This second line will be an integer `0 < N < 25`, followed by `N` lines of libraries & their dependencies. Each library-dependency line will be a series of space-delimited tokens: `lib-name Q dep-1 dep-2 ... dep-Q`. The first token is the name of the library; the second token `0 ≤ Q ≤ 25` represents the number of dependencies to follow, which **all** must be built in order for that library to compile.

Note that the input will never contain cycles.

Output Specifications

The output should be the linkline for the target library. Each library required to build the target (including the target itself) should appear exactly once, according to the following criteria:

1. The linker will run in rounds. In each round, it will build *all* libraries that have no remaining unbuilt dependencies at the start of that round. You may assume that the linker can infinitely parallelize, and that the round completes only once all libraries in that round have built.
2. Whenever the linker does a round, print out all the libraries from that round in lexicographical order (space-delimited), and move on to the next round.
3. The very last round will always be a single library: the target

Sample Input/Output

Input

```
lib-x
4
lib-x 2 lib-b lib-c
lib-a 0
lib-b 1 lib-a
lib-c 1 lib-a
```

Output

```
lib-a lib-b lib-c lib-x
```

Explanation

`lib-x` is dependent on `lib-b` & `lib-c`, both of which are dependent on `lib-a`. Therefore, `lib-a` is built by itself in the first round (as it is the only one with no unbuilt dependencies); `lib-b` & `lib-c` are both built in the second round (and are sorted alphabetically when printed); and finally `lib-x` is printed in the 3rd round.

Input

```
lib-y
4
lib-y 0
lib-a 0
lib-b 1 lib-a
lib-c 1 lib-a
```

Output

```
lib-y
```

Explanation

There are many other libraries with dependencies, but our target is `lib-y`, which has none. Therefore the only round is `lib-y`.

Input

```
lib-z
7
lib-a 0
lib-b 0
lib-c 0
lib-d 0
lib-e 1 lib-d
lib-f 1 lib-e
lib-z 4 lib-a lib-b lib-c lib-f
```

Output

```
lib-a lib-b lib-c lib-d lib-e lib-f lib-z
```

Input

```
bloom-lib
23
lib-a 0
lib-b 1 lib-a
lib-c 1 lib-b
lib-d 1 lib-c
lib-e 0
lib-f 1 lib-e
lib-g 1 lib-e
lib-h 1 lib-e
lib-i 0
lib-j 1 lib-i
bloom-lib 7 lib-b lib-j lib-r lib-s lib-t lib-u lib-v
```

```
lib-k 0
lib-l 0
lib-m 0
lib-n 0
lib-o 0
lib-p 0
lib-q 0
lib-r 7 lib-k lib-l lib-m lib-n lib-o lib-p lib-q
lib-s 7 lib-k lib-l lib-m lib-n lib-o lib-p lib-q
lib-t 7 lib-k lib-l lib-m lib-n lib-o lib-p lib-q
lib-u 7 lib-k lib-l lib-m lib-n lib-o lib-p lib-q
lib-v 7 lib-k lib-l lib-m lib-n lib-o lib-p lib-q
```

Output

```
lib-a lib-i lib-k lib-l lib-m lib-n lib-o lib-p lib-q lib-b lib-j lib-r lib-s lib-t
lib-u lib-v bloom-lib
```

CODECON

Memory Limit:	1024 MB
Time Limit:	5 s

Halloween Candy (350 points)

Introduction

It's Halloween and you are going trick-or-treating in your neighborhood. You want to maximize the candy that you can get by the end of the night. Your neighborhood is a grid of houses, and every house gives out a different amount of candy. You can move horizontally or vertically between houses in the grid, but not diagonally, and you cannot visit the same house more than once. You begin at a designated house and must end at another designated house and you have a limit on the total number of houses you can visit.

Input Specifications

The first line of input will be of the form (space delimited):

`N M MAX` X_1 Y_1 X_2 Y_2 where:

- `0 < N < 10` : Number of rows of houses
- `0 < M < 10` : Number of columns of houses
- `0 < MAX ≤ 100` : max houses you can visit *(does not include the start house)*
- X_1 Y_1 : coordinates of the start house
- X_2 Y_2 : coordinates of the end house
    - X coordinates are constrained to `0 ≤ X < M`
    - Y coordinates are constrained to `0 ≤ Y < N`

The following `N` lines of input will be the matrix of houses, each line with `M` space-delimited integers representing the amount of candy at that house.

Output Specifications

You will need to return the integer value of the maximum amount of candy that you can pick up between (and including) the starting house and ending house. Given that you must always finish at the End House, note that the route that yields the optimal of candy may not always be `MAX` length long.

Sample Input/Output

Input

```
2 2 2 0 0 1 1
3 4
1 1
```

Output

```
8
```

Explanation

We begin at house (0, 0) and we get 3 pieces of candy from there for free. We can then visit up to 2 more houses. In this case, we go right to house (0, 1) because it has 4 pieces of candy, compared to only 1 candy at house (1, 0). Finally, we proceed down to the end house at (1, 1) and collect 1 more piece of candy, for a total of 8.

Input

```
2 2 1 0 0 1 0
3 4
1 1
```

Output

```
4
```

Explanation

This is the same house matrix as sample (1) and once again we begin at house (0, 0), but this time we can only visit 1 additional house, so we must proceed directly to the end house at (1, 0). Our total candy is simply the candy from the start house (3) plus the candy from the end house (1), for a total of 4.

Input

```
3 3 4 0 0 1 0
1 2 3
5 5 5
11 11 11
```

Output

```
13
```

Explanation

In this example, the limiting factor is the number of houses we can visit, which is only 4. While our algorithm would love to go for the 11's at the bottom, we do not have enough houses remaining to go down to the last row and back up. Instead, our only possible course is to go right to (0, 1), down to (1, 1), then left to (1, 0). We only use 3 of our available 4 houses, but note that using exactly 4 is not possible in this case.

Input

```
3 3 15 0 0 1 0
1 2 3
5 5 5
11 11 11
```

Output

```
51
```

Explanation

This final example is the same as test case 3, but the number of houses we can visit is not at all limiting this time. Instead, we try to trace the longest possible path without visiting the same house twice. Starting from (0, 0), we move right to (0, 1), down to (1, 1), right to (1, 2), and down to (2, 2). Next, we sweep across the row of 11's at the bottom through (2, 1), and back to (2, 0). Finally, we move up to (1, 0), the end point. In total, we grabbed 51 pieces of candy, and note the only house not visited was house (0, 2), worth 3 pieces of candy.

Memory Limit: 1024 MB
Time Limit: 5 s

TokenChain (550 points)

Introduction

You manage a network of block chain nodes that receive crypto-token data packets. Each node will receive a set of packets (or hashes) from a particular chain. A chain is identified by the first and last packets it receives. For example:

Node A: S1 → 1 → 2 → E1

Node B: S1 → 1 → E1

Node C: S2 → 4 → 5 → E2

In the above example, Nodes A and B received packets from the same chain (identified as S1→E1). Node C, however, received packets from a different chain (identified as S2→E2).

Additionally, some of the nodes in your network contain spoofed (incorrect) information. Your first task is to identify and eliminate such chains. Consider the following example.

Node A: S1 → 1 → 2 → 3 → E1

Node B: S1 → 1 → 2 → 3 → E1

Node C: S1 → 2 → 4 → 5 → E1

Node D: S1 → 1 → 3 → 4 → E1

Note that Nodes A, B, C, and D all received packets for the same block chain (identified as S1→E1). However, Nodes A, B, and D believe that the correct second packet in the sequence was '1', whereas Node C believes it to be '2'. Because there are three pieces of evidence to support '1', but only one piece to support '2', we conclude that '1' must be correct and the data from Node C is discarded entirely. Similarly, Node D defers from Nodes A and B regarding what the third packet in the sequence should be. Because two nodes (A and B) believe the packet to be '2', while only one node (D) believes the packet to be '3', we conclude that '2' is in fact correct and discard the data from Node D.

It's also worth noting that a node does not conflict with its peers from the same chain merely if data is omitted, provided the sequence is still correct. Return your attention to the first example, above: nodes A and B do not present conflicting information. They both believe the second packet in the sequence to be '1'. Additionally, while Node A believes the third packet in the sequence to be '2', Node B has the end node third, and thus does not have a vote. The two nodes do not conflict with each other, but we assume that the full sequence must contain '2' as the third packet because we have 1 vote to the positive and no dissenting votes.

Your goal is to find the longest valid block chain, once all spoofed data is removed. In the event of a tie, favor the chain with the most valid nodes in support. Referencing the first example yet again, the full sequences of chains S1→E1 and S2→E2 both have a length of 4. However, there are two valid nodes that received data for chain S1→E1 (nodes A and B), but only one valid node that received data for chain S2→E2 (node C). Thus, chain S1→E1 should be output.

Input Specifications

The first line is the number of events `0 < L < 300` followed by L lines, each representing an event. Each event line will be pairs of node ids & hash ids, in the order in which the events were recorded in the network. Remember that the first and last packets a node receives are special and indicate the identitiy of the chain that node is receiving data for. Each line is two space-separated tokens: `node_id hash_id`. Each token will be a string that can contain only alphanumeric characters & underscores.

Output Specifications

Output the longest, most reported hash chain that was computed, delimited by spaces.

packet_id1 packet_id2 packet_id3 ... packet_idn

Sample Input/Output

Input

```
29
A S1
A E1
B S1
B 1
B E1
C S1
C 1
C 2
C E1
D S1
D 1
D 2
D 3
D E1
E S2
E 1
E 2
E 3
E E2
F S2
F 1
F 2
F 3
F E2
G S2
G 1
G 2
G 3
G E2
```

Output

```
S1 1 2 3 E1
```

Explanation

After processing, we have the following node chains:

- A: S1 → E1

- B: S1 → 1 → E1
- C: S1 → 1 → 2 → E1
- D: S1 → 1 → 2 → 3 → E1
- E: S2 → 1 → 2 → 3 → E2
- F: S2 → 1 → 2 → 3 → E2
- G: S2 → 1 → 2 → 3 → E2

The most agreed-upon non-spoofed chain is S1 → E1, and the longest version of that chain is S1 → 1 → 2 → 3 → E1.

Input

```
9
A S1
A 1
A E1
B S1
B 1
B E1
C S2
C 2
C E2
```

Output

```
S1 1 E1
```

Explanation

After processing, we have the following node chains:
A: S1 → 1 → E1
B: S1 → 1 → E1
C: S2 → 2 → E2
First note there is no spoofed data. Next, note that there is a tie for the longest chain. Both S1→E1 and S2→E2 have a length of 3. Chain S1→E1 wins, however, because it has two valid nodes (A and B), while S2→E2 has only one (Node C).

Input

```
9
A S1
A E1
B S2
B E2
C S2
C E2
D S1
D 1
D E1
```

Output

```
S1 1 E1
```

Explanation

Both chains have two valid nodes in support (A,D for S1→E1 and B,C for S2→E2). Chain S1→E1 has a chain length of 3 (S1 → 1 → E1), while chain S2→E2 has a chain length of just 2 (S2 → E2).

Input

```
26
H S2
A S2
B S2
A 3
E S3
D S2
C S3
J S2
D E3
C E4
J 4
K S2
E E4
A E5
K 4
G S2
I S2
F S3
F E4
G E5
H 3
H E6
I E4
J E5
K E5
B E5
```

Output

```
S2 4 E5
```

Explanation

Each node reported following chain of blocks:

A: S2 → 3 → E5
D: S2 → E3
C: S3 → E4
B: S2 → E5
E: S3 → E4
F: S3 → E4
G: S2 → E5
H: S2 → 3 → E6
I: S2 → E4
J: S2 → 4 → E5
K: S2 → 4 → E5

Two nodes (J, K) reported that the second packet of the S2→E5 chain was '4'. Only one node (A) reported that the second packet was '3'. We conclude that Node A is spoofed and discard it. The two remaining chains of maximum length are S2→E6 (reported by Node H) and S2→E5 (reported by nodes B, G, J, and K). With four valid nodes, chain S2→E5 is selected.

CODECON

Memory Limit:	1024 MB
Time Limit:	5 s

Ride Hailing Improved (650 points)

Introduction

You're developing a new ride hailing app that will feature a faster way to pick up passengers. The ride hailing app will send the passenger an optimal meeting location at which the taxi will pick them up. The meeting location will be somewhere between their current location and the location of the taxi, along the optimal path. Each user of the app will specify a maximum distance they are willing to walk to meet up with the taxi. Your task here is to compute that optimal pick up location such that the time it takes for the person and the taxi to meet is minimal.

You will be provided a map of the city in the form of a matrix of integers. Each cell in the matrix represents a potential meeting location. To move from one cell in the matrix to an adjacent cell (no diagonals), the person or car will pay a distance cost equal to the value in that cell. For example, if a person is standing on cell (0, 0) and would like to move to cell (0, 1), which contains a value of 4, then the person must travel a distance of 4 to make that transition. The actual time it will take to do the transition depends upon the speed at which the person is walking, which will be provided.

Input Specifications

You will be provided with the following values, one per line:

- `0 < SC < 100` - the speed of the car
- `0 < SP < 100` - the speed of the person
- `0 < MD < 100` - the maximum distance the person will walk
- X_1 Y_1 - the coordinates of the car to start (space-separated)
- X_2 Y_2 - the coordinates of the person to start (space-separated)
- `N M` - The dimensions of the matrix (rows and columns, space-separated). Each dimension is between 0 and 100
- `N` lines of `M` space-separated values. Each value indicates the cost of entering that cell from any of the adjacent cells

You can make the following assumptions:

- The distances in each cell will always be a multiple of the average speed of both the car and the person, so your answer will always be an integer
- The speed provided is measured as a function of distance per time unit
- The person and the car will never meet between cells. They will always travel the entire distance to the cell and meet on the cell.
- The car will never wait while the person walks further. For example, if the taxi and person are two cells away from each other after some period of time, and it would take the car 3 time units to travel the last two cells, but the person could take 2 time units to travel to the middle cell while the car takes 1 time unit to travel there, the person will still stay put. In this scenario, the car would be waiting 1 time unit on the final cell for the person to arrive, which is not allowed. Either both are moving, or the person is waiting for the car, but not vise verse.

Output Specifications

You need to output the minimum amount of time it will take for the taxi and passenger to meet, as a single integer.

Sample Input/Output

Input

```
2
1
2
0 0
1 1
2 2
2 2
2 2
```

Output

```
2
```

Explanation

In this simple example, the taxi and the person start on opposite sides of a 2x2 grid. It will take the car 2 time units to reach the person, in either direction. If the person attempts to walk one cell, it will take two time units. This will not save any time, and it will cause the car to have to wait while the person is walking, which is not allowed.

Input

```
5
1
10
2 0
6 3
7 4
5 10 15 25
5 5 5 5
5 5 5 5
10 10 10 10
25 25 25 5
5 10 25 5
10 10 5 20
```

Output

```
7
```

Explanation

The optimal path involves the car traveling all the way to the right side of the grid, from (2, 0) to (2, 3), then directly down to the bottom of the grid, from (2, 3) to (6, 3). If the car traveled the entire distance, it would take 11 time units. Instead, the person should walk one cell up, from (6, 3) to (5, 3), at a cost of 5 time units. This saves the car 4 time units, and reduces the optimal time from 11 to 7.

Input

```
4
2
7
4 3
0 0
5 5
16 4 4 32 4
4 8 4 4 8
12 8 8 16 16
4 24 4 24 28
4 32 4 4 64
```

Output

```
7
```

Explanation

The optimal path, beginning from (4, 3), moves left to (4, 2), straight up all the way to (0, 2), then left two spots to (0, 0). It would take the car 11 time units to drive it all. The person can walk 1 cell right at a cost of 2 time units, which reduces the car's time to 7 units. This ends up being the correct answer. Note that, had the person walked one more cell to the right, they would only have used 4 time units walking to the car's 6 units driving, but they will have walked a total of 8 distance units. Due to the person's maximum distance walked being 7 units, this is not an option.

Memory Limit:	1024 MB
Time Limit:	5 s

Food will be provided (800 points)

Introduction

Henry is a college student trying to save money. Therefore, he is trying to attend as many events that have free food as possible. **He can attend up to one event per day,** and can take any quantity of food from the event (up to the event's maximum), but at the end of each day he must eat one food. Each event has three attributes:

- `D` : the day that this event occurs (the first day is D = 0).
- E: the expiration day of the food provided at this event *(this is the absolute day, NOT a window relative to D)*). A food can only be eaten if its expiration day ≥ the current day *(otherwise it is expired and can no longer be eaten)*
- `Q` : the total quantity of food at this event

Henry only needs to eat one unit per day and can hold as much food as he wants. His goal is to go as many days in a row as possible eating free food (and not having to call his parents for money).

Input Specifications

On the first line, you will receive one integer value `0 <= N <= 500` which represents the total potential events Henry can attend.

This is followed by `N` lines, each representing a single event, containing the three space separated integer values `D E Q` (all >= 0, and <= 500).

Output Specifications

Return an integer value of the maximum days that Henry can eat once per day in a row.

Sample Input/Output

Input

```
11
0 15 8
2 10 6
0 10 30
2 13 5
5 5 1
15 25 3
15 22 10
17 22 8
25 10 15
25 15 10
30 50 50
```

Output

```
24
```

Explanation

This is a basic case that covers the basic functionality. You should take [0,15,8], [2,10,6], [5,5,1], [15,25,3], [17,22,8]

Input

```
10
2 10 0
1 2 3
3 0 10
4 5 5
4 5 5
500 500 500
0 0 0
4 5 6
4 6 5
1 10 10
```

Output

```
0
```

Explanation

Remember to consider edge cases when viewing schedules; this has no day 0 event, so this should return 0. However, this also has many other schedule edge cases that you should consider.

Input

```
130
0 45 7
83 106 5
292 294 1
58 88 3
218 266 6
140 148 3
165 212 1
122 149 5
33 73 1
172 209 4
53 61 4
172 173 4
221 231 6
207 236 5
285 302 6
263 289 6
204 231 7
210 257 6
205 250 6
235 272 5
0 20 3
247 276 1
297 338 3
```

```
255 264 1
289 318 4
235 248 1
139 160 1
3 16 2
228 268 5
16 40 7
3 21 1
216 224 1
104 136 4
2 21 1
116 161 1
170 200 3
155 168 2
173 194 2
281 318 3
41 85 1
165 185 5
79 105 4
176 209 3
91 139 2
19 37 1
175 224 5
42 43 6
131 131 4
23 56 6
28 61 5
69 73 6
20 35 2
257 287 6
2 3 2
261 264 6
216 231 6
262 268 2
32 66 6
159 180 7
238 278 7
34 84 5
270 294 3
176 214 4
178 192 7
297 330 7
28 36 4
73 97 5
131 143 3
144 167 3
12 25 3
44 88 6
56 77 2
199 238 3
58 90 1
126 156 6
252 267 1
25 74 1
```

```
3 40 5
282 307 1
291 330 7
293 328 4
297 343 2
282 302 5
265 280 1
108 147 3
166 182 6
100 121 7
198 213 3
4 39 2
31 48 1
8 37 2
48 98 4
1 23 4
182 218 4
206 222 5
173 202 5
230 277 3
254 303 1
276 321 6
262 286 7
135 180 2
270 302 2
283 304 1
58 100 1
270 319 5
149 155 2
270 276 6
255 276 1
162 204 7
67 97 1
53 75 5
232 269 5
112 161 4
98 133 3
8 42 5
152 178 7
283 319 2
81 91 1
255 265 5
207 254 1
169 216 3
32 62 3
114 118 3
72 100 7
59 76 4
32 61 6
107 142 3
281 325 1
139 151 7
298 313 6
```

Output

334

Explanation

This a typical example of schedules that you can expect.

Memory Limit: 1024 MB
Time Limit: 5 s

Pandemic vs. WHO (900 points)

Introduction

An epidemic broke out and WHO (World Health Organization) has to quarantine the area. Can WHO quarantine the epidemic? If so, what is the minimum # of walls required to quarantine all affected areas?

The rules for the epidemic and WHO are as follows:

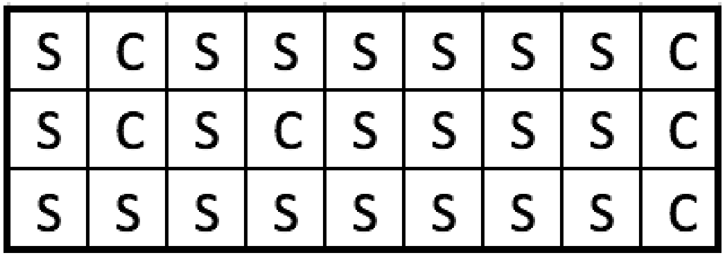
1. The world will be represented in a 2-D array
2. Every day, WHO sends quarantine officers ONLY to the affected area that endangers the most Safe areas (i.e. that will infect the most cells on the next turn), and installs walls surrounding that region. A single wall is used between a contaminated cell and a safe cell. If a wall is already installed from previous rounds, WHO won't need to add another wall there. *(Diagram provided in Input / Output specifications.)*
    - Note: There will never be a scenario where there will be a tie for the contaminated area with the largest number of threatened cells.
3. Every night, the epidemic spreads to all adjacent cells (North/South/East/West) UNLESS there is a wall installed to prevent the spread.

Will WHO save the world, and if so, what is the # of walls required?

Input Specifications

The first line of the input will represent the # of strings (N). The second input will represent length of the strings (M). This will be followed by N strings with M characters each ('S' or 'C' only) - where 'S' represents safe cells and 'C' represents contaminated cells.

For example :
3
9
SCSSSSSC
SCSCSSSC
SSSSSSSC
will represent the below 2-D array.



S	C	S	S	S	S	S	S	C
S	C	S	C	S	S	S	S	C
S	S	S	S	S	S	S	S	C

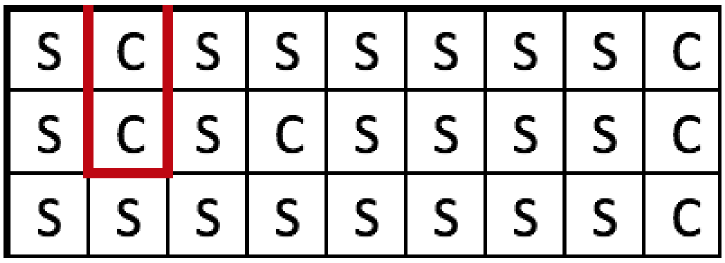
This grid represents 3 independent epidemic regions (represented as `(row,col)`):

Region 1: (1,1) + (2,1)
Region 2: (1, 3)
Region 3: (0,8) + (1,8) + (2,8)

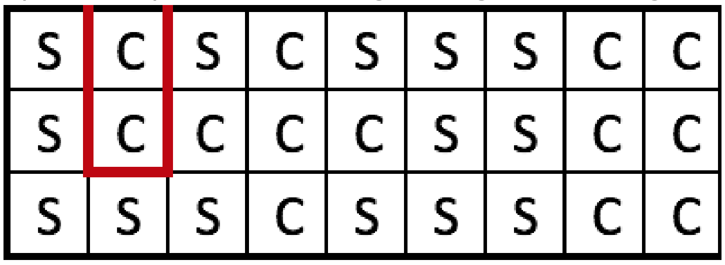
Output Specifications

Your output will be an int value indicating the # of walls used by WHO. If the world is fully contaminated, return the # of walls built by WHO so far.
For the above example, the expected output is 13

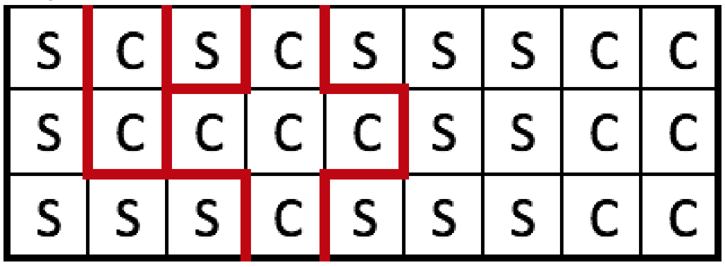
On Day 1,
WHO will quarantine Region 1 (though it is not the largest affected area, it threatens the most cells (5)) using 5 walls. Notice, WHO doesn't need to use the wall north of (0,1) since there's no cells that can be affected.



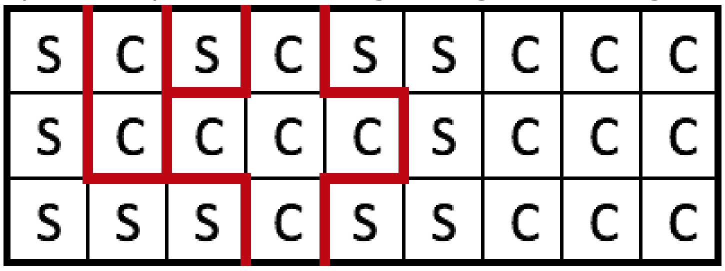
Epidemic spreads to its neighboring cells for Regions 2 & 3.



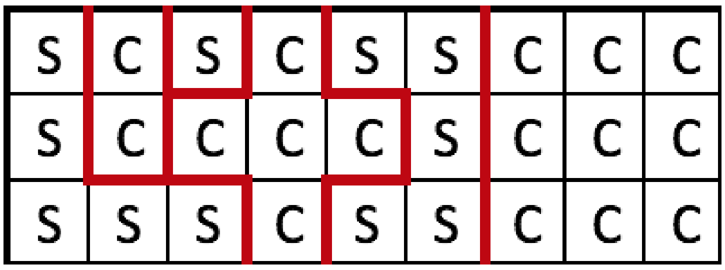
On Day 2,
WHO quarantines Region 2 (threatens 4 cells) using 9 walls. WHO used 5+9=14 walls so far. *(Note that they did not need to build a left wall for (1,2) because there was already a wall there from the first round.)*



Epidemic spreads to its neighboring cells for Region 3.



On Day 3,
WHO quarantines Region 3 with 3 additional walls. WHO used 17 walls in total.



Output: 17

Sample Input/Output

Input

```
3
9
SCSSSSSSC
SCSCSSSSC
SSSSSSSSC
```

Output

```
17
```

Explanation

As explained in Output Specifications.

Input

```
2
2
SC
SC
```

Output

```
2
```

Explanation

WHO can build 2 walls between (0,0)(0,1) and (1,0)(1,1)

Input

```
2
5
CCSCS
CSSCS
```

Output

```
6
```

Explanation

Two rounds of wall-building and we're done

Input

```
1
1
C
```

Output

```
0
```

Explanation

No wall for WHO to build

Input

```
5
5
CSCSS
SSSSS
CCCSC
SSSSC
SSSSC
```

Output

```
12
```

Explanation

A couple edge cases to consider.