Queue Problem

1. BCQUEUE – Basic Queue

https://www.spoj.com/PTIT/problems/BCQUEUE/

Ban đầu cho một queue rỗng. Bạn cần thực hiện các truy vấn sau:

1. Trả về kích thước của queue
2. Kiểm tra xem queue có rỗng không, nếu có in ra "YES", nếu không in ra "NO".
3. Cho một số nguyên và đẩy số nguyên này vào cuối queue.
4. Loại bỏ phần tử ở đầu queue nếu queue không rỗng, nếu rỗng không cần thực hiện.
5. Trả về phần tử ở đầu queue, nếu queue rỗng in ra -1.
6. Trả về phần tử ở cuối queue, nếu queue rỗng in ra -1.

Input

Dòng đầu tiên chứa số nguyên n - lượng truy vấn (1 <= n <= 1000)

N dòng tiếp theo, mỗi dòng sẽ ghi loại truy vấn như trên, với truy vấn loại 3 sẽ có thêm một số nguyên, không quá 10^6.

Output

In ra kết quả của các truy vấn.

Example

Input	Output
14 3 1 3 2 3 3 5 6 4 4 4 4 4 3 5 3 6 5 1	1 3 5 2

2. ADAQUEUE – Ada and Queue

https://www.spoj.com/problems/ADAQUEUE/

Ada The Ladybug có rất nhiều việc phải làm. Cô ấy đưa tất cả chúng vào một hàng đợi. Dù vậy, cô ấy là người rất thiếu kiên định nên thỉnh thoảng cô ấy thực hiện từ phía trước hàng đợi, thỉnh thoảng là từ phía sau, và đôi khi, cô ấy đảo lộn lại nó (hàng đợi).

Input

Dòng đầu tiên chứa 1 số 1 <= Q <= 10^6, là số lượng truy vấn. Mỗi truy vấn sẽ được mô tả ở Q dòng tiếp theo và có thể là 1 trong các câu lệnh sau: back – in ra số nằm ở cuối hàng đợi và xóa nó đi sau đó.

front – in ra số nằm ở đầu hàng đợi và xóa nó đi sau đó.

reverse – đảo ngược tất cả các phần tử trong hàng đợi.

push_back N – Thêm số nguyên N vào cuối hàng đợi.

toFront N – thêm số nguyên N vào trước hàng đợi.

0 <= N <= 100

Output

Với mỗi truy vấn back/front, in ra số nguyên thích hợp.

Nếu nhận được dạng truy vấn này mà hàng đợi rỗng, in ra màn hình **“No job for Ada?”**.

Example

Input	Output
15 toFront 93 front back reverse back reverse toFront 80 push_back 53 push_back 50 front front reverse push_back 66 reverse front	93 No job for Ada? No job for Ada? 80 53 66

3. QUEUEEZ – Easy queue

https://www.spoj.com/problems/QUEUEEZ/

Bạn có một hàng đợi rỗng và sếp có một vài truy vấn. Những truy vấn này đều là các toán tử cơ bản của hàng đợi như Enqueue, Dequeue, và in ra một vài giá trị. Giờ đây, sếp yêu cầu bạn thực hiện các truy vấn của ông ấy.

Input

Dòng đầu là số lượng các truy vấn 1 <= T <= 10^6

Mỗi dòng trong T dòng tiếp theo là các truy vấn dựa theo cú pháp sau đây:

1 n : thêm n vào hàng đợi

2 : xóa phần tử đầu hàng đợi. Nếu hàng đợi rỗng thì bỏ qua.

3 : in ra phần tử đầu hàng đợi.

Output

Với mỗi truy vấn dạng 3, in ra màn hình giá trị đầu hàng đợi, thay thế bằng **"Empty!"** nếu hàng đợi rỗng.

Example

Input	Output
6 1 5 1 6 2 3 2 3	6 Empty!

Cảnh báo: Dữ liệu đầu vào rất lớn!

4. SSAM419B – Thuật toán BFS

https://www.spoj.com/PTIT/problems/SSAM419B/

Cho đồ thị vô hướng G=(V, E). Hãy thực hiện thuật toán duyệt đồ thị BFS bắt đầu tại một đỉnh u ∈ V.

Input

Dòng đầu tiên gồm một số nguyên T (1 ≤ T ≤ 20) là số lượng bộ test.

Tiếp theo là T bộ test, mỗi bộ test có dạng sau:

Dòng đầu tiên gồm 3 số nguyên N=|V|, M=|E|, u (1 ≤ N ≤ 103, 1 ≤ M ≤ 105, 1 ≤ u ≤ N).

M dòng tiếp theo, mỗi dòng gồm 2 số nguyên a, b (1 ≤ a, b ≤ N, a ≠ b) tương ứng cạnh nối hai chiều từ a tới b.

Dữ liệu đảm bảo giữa hai đỉnh chỉ tồn tại nhiều nhất một cạnh nối.

Output

Với mỗi bộ test, in ra trên một dòng theo thứ tự các đỉnh được duyệt trong quá trình duyệt đồ thị bằng thuật toán BFS bắt đầu tại đỉnh u.

Example

Input	Output
1 5 5 3 1 2 1 3 2 4 3 5 4 5	3 1 5 2 4

5. KNIGHT – Bước đi quân mã

https://www.spoj.com/PTIT/problems/SSAM419C/

Cho một quân mã trên bàn cờ vua tại vị trí **ST**. Nhiệm vụ của bạn là hãy tìm số bước di chuyển ít nhất để đưa quân mã tới vị trí **EN**.

Input

Dòng đầu tiên là số lượng bộ test T (T ≤ 20).

Mỗi test gồm 2 xâu dạng **"xy"** và **"uv"**, trong đó x, y là kí tự trong **"abcdefgh"** còn **y**, **v** là số thuộc 1, 2, 3, 4, 5, 6, 7, 8.

Output

Với mỗi test, in ra đáp án tìm được trên một dòng.

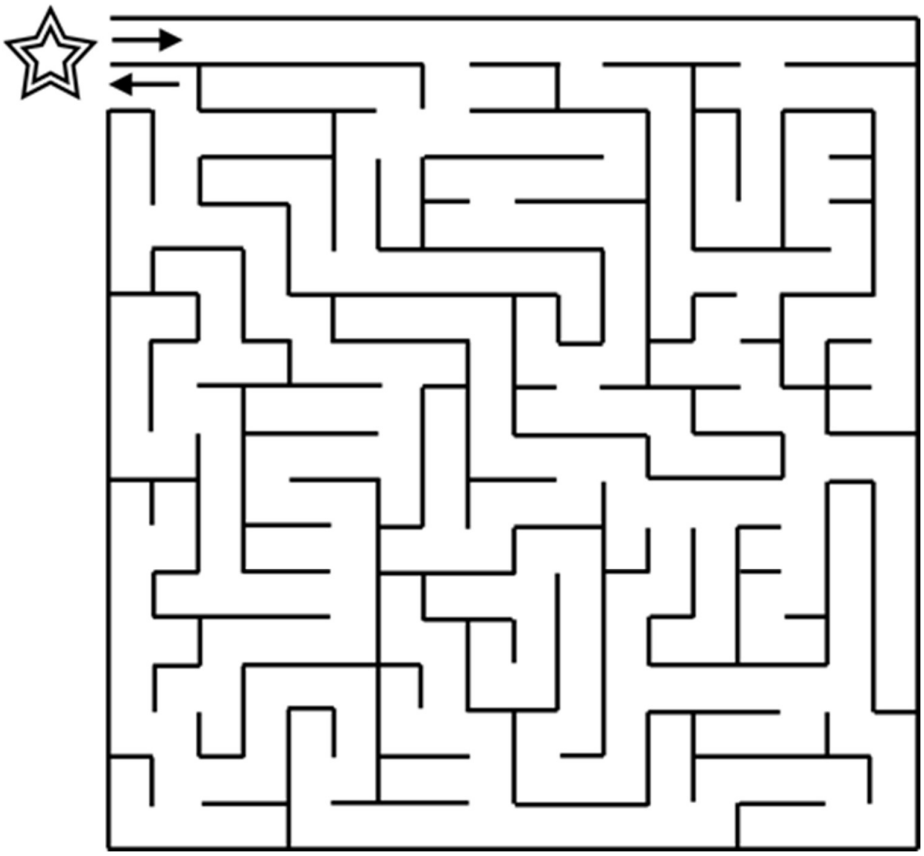
Example

Input	Output
8 e2 e4 a1 b2 b2 c3 a1 h8 a1 h7 h8 a1 b1 c3 f6 f6	2 4 2 6 5 6 1 0

6. MAKEMAZE – Mê cung hợp lệ

https://www.spoj.com/problems/MAKEMAZE/

Có rất nhiều thuật toán để tạo ra mê cung. Sau khi tạo ra một mê cung, ta cần kiểm tra xem mê cung này có hợp lệ hay là không. Một mê cung hợp lệ là mê cung có một và chỉ một lối vào cũng như một và chỉ một lối ra (nói cách khác là chỉ có 2 cổng vào/ra trên tất cả các cạnh) và phải có ít nhất một đường đi từ lối vào đến lối ra.



Đưa ra một mê cung, chỉ cần đưa ra mê cung đó có hợp lệ hay không.

Input

Dòng đầu tiên chứa một số nguyên dương t – số lượng test case. Sau đó, với mỗi test case, dòng đầu tiên chứa 2 số nguyên m và n lần lượt là số hàng và cột của mê cung. Sau đó là mô tả về mê cung trên 1 ma trận kí tự m x n. M[i][j] = '#' mô tả 1 bức tường; M[i][j] = '.' mô tả một khoảng trống.

Output

Với mỗi test case, in ra **"valid"** hoặc **"invalid"** trên một dòng thể hiện mê cung có hợp lệ hay không.

Input Constraints

1<=t<=10000

1<=m<=20

1<=n<=20

Example

Input	Output
6 4 4 #### #... #.## #.## 5 5 #.### #..## ##..# #.#.# ###.# 1 1 . 5 1 # # . . # 2 2 #. .# 3 4 #..# #.## #.##	valid valid invalid valid invalid invalid

7. KOZE – Bảo vệ nông trại

https://www.spoj.com/problems/KOZE/

Mirko có một đàn cừu nằm trên một mảnh đất có hàng rào. Trong khi anh ta đang ngủ, bầy sói có thể lẻn vào tấn công bầy cừu.

Mảnh sân có dạng hình chữ nhật với các ô đất được sắp xếp theo hàng và cột.

'.' biểu diễn một ô trống.

'#' biểu diễn một hàng rào.

'k' biểu diễn một con cừu.

'v' biểu diễn một con sói.

Các ô đất được coi là cùng một khu (sector) nếu có thể đi lại giữa chúng theo các hướng dọc hoặc ngang mà không cần vượt qua hàng rào (không thể đi chéo).

Nếu ta có thể thoát từ ô đất A ra khỏi mảnh đất, ô đất A này sẽ không thuộc về bất cứ một khu nào.

May mắn là Mirko đã dạy bọn cừu vài chiêu Kung-fu, do đó chúng có thể tự phòng thủ trước bầy sói nếu chúng có số lượng nhiều hơn trong cùng một khu. Khi đó tất cả sói sẽ bị đấm không còn con nào và cừu thì giữ nguyên quân số :v Nếu không thì cừu sẽ chết và sói bảo toàn lực lượng. Nếu ô đất đó không nằm trong khu nào thì cừu có thể chạy trốn và sói thì chỉ không kiếm được mồi, vì vậy tất cả đều còn sống sót.

Viết một chương trình xác định xem có bao nhiêu cừu và sói sẽ sống sót sau đêm máu me này.

Input

2 số N và M, là số hàng và cột biểu diễn mảnh đất của Mirko.

Trong mỗi N dòng, có M kí tự biểu diễn vị trí của các ô đất trống, cừu, sói, hàng rào.

Constraints

3 <= M, N <= 250

Output

In trên một dòng số cừu và sói sống sót, phân biệt bằng khoảng trắng.

Input	Output
8 8 .######. #..k...#	3 1

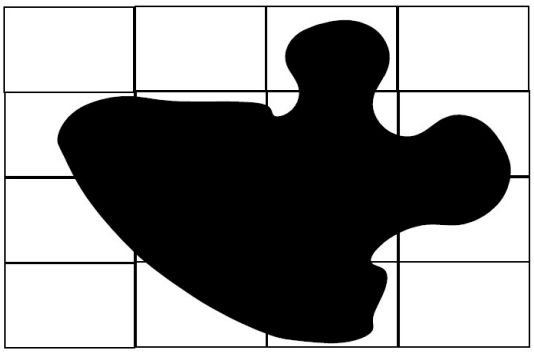
#.####.# #.#v.#.# #.#.k#k# #k.##..# #.v..v.# .######.	

8. UCV2013H – Slick

https://www.spoj.com/problems/UCV2013H/

Một tai nạn hàng hải đã khiến dầu tràn ra vùng biển Felipistonia, đây là một thảm họa thiên nhiên lớn. Chính phủ Felipistonia muốn dọn dẹp đống hỗn độn này trước khi có thêm thiệt hại xảy ra. Để làm được điều này, trước tiên họ phải biết mức độ nghiêm trọng của vụ tai nạn và lượng dầu đã đổ ra biển. Công cụ duy nhất mà chính phủ Felipistonia có để có được thông tin về mức độ nghiêm trọng của thảm họa này, là việc sử dụng các hình ảnh vệ tinh. Với những hình ảnh này họ có thể ước tính được mình phải bỏ ra bao nhiêu tiền để dọn dẹp đống bừa bộn này. Đối với điều này, số lượng vết loang trong biển và kích thước của mỗi vết loang phải được biết. Vết loang là một mảng dầu nổi trên mặt nước. Thật không may, người của Felipistonia có hơi …, vì vậy họ đã thuê bạn để giúp họ xử lý hình ảnh.

Ví dụ về hình ảnh do vệ tinh thu được được thể hiện trong Hình 1 (a). Hình ảnh này có thể được chuyển đổi thành 0 và 1 như trong Hình 1 (b). Với ma trận nhị phân này, công việc của bạn là đếm số lượng vết loang trong đại dương và kích thước tương ứng của chúng. Hai pixel liền kề trong hình ảnh được coi là có cùng một vết loang nếu chúng nằm trong cùng một hàng hoặc cùng một cột.



(a)

0	0	1	0
1	1	1	1
1	1	1	1
0	1	1	0

(b)

Input

Chứa 1 vài test case, mỗi cái tương ứng với một hình ảnh vệ tinh khác nhau

Dòng đầu tiên của mỗi trường hợp chứa hai số nguyên cho biết số hàng (N) và số cột (M) trong hình ảnh (1 <= N, M <= 250). Sau đó N dòng tiếp theo với M số nguyên, mỗi dòng chứa thông tin của hình ảnh.

Kết thúc của đầu vào được đánh dấu bằng test case dạng 0 0. Không xử lý test case này.

Output

Đối với mỗi hình ảnh, xuất số lượng vết loang trên biển. Ngoài ra, xuất ra kích thước của mỗi vết theo thứ tự tăng dần và số lượng vết của kích thước đó.

Example

Input	Output
10 10 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0	7 1 2 2 1 6 1 10 2 20 1

9. CLEANRBT – Cleaning Robot

https://vn.spoj.com/problems/CLEANRBT/

Ở đây chúng ta giải quyết vấn đề lập đường đi cho robot hút bụi trên một sàn phòng hình chữ nhật được đặt các đồ nội thất.

Phòng được lát bằng loại gạch vuông kích thước 1x1, phù hợp với kích thước của robot. Các ô gạch có thể sạch hoặc bẩn và robot có thể làm sạch các ô bẩn bằng cách đi đến chúng. Robot chỉ di chuyển mỗi bước đến 1 trong 4 ô gần kề ô hiện tại theo hướng Đông, Tây, Nam, Bắc. Robot có thể ghé thăm 1 ô gạch 2 lần hoặc nhiều hơn.

Nhiệm vụ của bạn là xác định số bước di chuyển tối thiểu để robot có thể làm sạch tất cả các ô bẩn nếu có thể.

Input

Đầu vào bao gồm nhiều bản đồ, mỗi bản đồ đại diện cho kích thước và cách sắp xếp của căn phòng. Một bản đồ được đưa ra ở định dạng sau.

w h

c11 c12 c13 ... c1w

c21 c22 c23 ... c2w

...

ch1 ch2 ch3 ... chw

Các số nguyên w và h là chiều dài của hai cạnh sàn của căn phòng theo chiều rộng của gạch lát nền. w, h <= 20. Ký tự cyx đại diện cho trạng thái ban đầu của những ô gạch:

'.' : ô gạch sạch

'*' : ô gạch bẩn

'x' : một phần nội thật (vật cản)

'o' : robot (vị trí ban đầu)

Trong bản đồ số ô bẩn không vượt quá 10. Chỉ có một robot. Kết thúc đầu vào được biểu thị bằng một dòng chứa hai số 0.

Output

Đối với mỗi bản đồ, chương trình của bạn sẽ xuất ra một dòng chứa số lần di chuyển tối thiểu. Nếu bản đồ bao gồm ô bẩn mà rô bốt không thể tiếp cận, chương trình của bạn sẽ xuất ra -1.

Example

Input	Output
7 5 ..... .o...*. ..... .*...*. ..... 15 13 ....x....... ...o...x....*.. ....x....... ....x....... ....x....... .... xxxxx.....xxxxx ....x....... ....x....... ....x....... ..*....x....*.. ....x....... 10 10 ....o....... ....xxxxx .....x.... .....x.*.. .....x.... .....x.... 0 0	8 49 -1