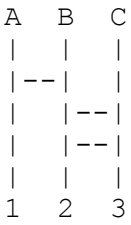
STICK PATH

Objectif

« Stick Path » commence par un certain nombre de lignes verticales. Entre les lignes se trouvent des connecteurs horizontaux aléatoires qui relient toutes les lignes dans un diagramme connecté, comme celui ci-dessous.



Pour jouer au jeu, le joueur choisit une ligne en haut et la suit vers le bas. Lorsqu'il rencontre un connecteur horizontal, il doit le suivre pour passer à une autre ligne verticale et continuer vers le bas. Répétez cette opération jusqu'à atteindre le bas du diagramme.

Dans l'exemple de diagramme, si vous partez de A, vous arriverez en 2. Si vous partez de B, vous arriverez en 1. Si vous partez de C, vous arriverez en 3. Il est garanti que chaque étiquette supérieure correspondra à une étiquette inférieure unique.

Avec un diagramme « Stick Path", découvrez quelle étiquette supérieure est reliée à quelle étiquette inférieure. Dressez la liste de toutes les paires connectées.

Entrées et sortie

Entrées
Line 1: Nombre entier **width** and **height** pour la largeur et la hauteur du diagramme ci-dessous.
Lignes **height** suivantes: Contenant un diagramme de « Stick Path » comme entrée.

Le diagramme lui-même est composé de caractères : "|" et "-", (et espace).
La ligne supérieure du diagramme comporte un certain nombre d'étiquettes **top**.
La ligne du bas contient les étiquettes **bottom**.

Chaque **top** et **bottom** est un caractère ascii unique qui peut avoir n'importe quelle valeur aléatoire. Ne supposez pas qu'ils seront toujours ABC ou 123.

En règle générale, les connecteurs horizontaux gauche et droit n'apparaîtront jamais au même endroit.

Tous les diagrammes ont le même style que les cas de test.

Sortie
Listez toutes les paires connectées entre les étiquettes supérieure et inférieure, **« top »« bottom »**, dans l'ordre des étiquettes supérieures de gauche à droite. Inscrivez chaque paire sur une ligne séparée.

Contraintes
3 < **width**
height ≤ 100

Exemples

Exemple 1
Entrées

```
7 7
A   B   C
|   |   |
|--|   |
|   |--|
|   |--|
|   |   |
1   2   3
```

Sortie

```
A2
B1
C3
```

Exemple 2
Entrée

```
P   Q   R   S   T   U   V   W
|   |   |   |   |--|   |   |
|   |   |--|   |   |   |--|
|   |--|   |--|   |   |   |
|--|   |--|   |   |   |--|
|--|   |   |   |   |--|   |
|   |--|   |   |--|   |--|
|   |   |   |--|   |--|   |
|--|   |   |   |--|   |   |
|   |   |--|   |   |   |   |
|   |   |   |--|   |   |--|
|   |   |   |   |--|   |   |
|--|   |   |   |   |   |   |
|--|   |--|   |   |   |--|
|   |--|   |   |--|   |   |
|   |   |--|   |   |   |--|
|--|   |--|   |   |--|   |
1   2   3   4   5   6   7   8
```

Sortie

```
P3
Q7
R8
S5
T6
U2
V4
W1
```

Comment commencer

Java

```
import java.util.*;
import java.io.*;
import java.math.*;

class Solution {

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int width = in.nextInt();
        int height = in.nextInt();
        if (in.hasNextLine()) {
            in.nextLine();
        }
        for (int i = 0; i < height; i++) {
            String line = in.nextLine();
        }

       System.out.println("answer");
    }
}
```

Javascript

```
var inputs = readline().split(' ');
const width = parseInt(inputs[0]);
const height = parseInt(inputs[1]);
for (let i = 0; i < height; i++) {
    const line = readline();
}

console.log('answer');
```

PHP

```
<?php
fscanf(STDIN, "%d %d", $width, $height);
for ($i = 0; $i < $height; $i++)
{
    $line = stream_get_line(STDIN, 1024 + 1, "\n");
}

echo("answer\n");
?>
```

C++

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    int width;
    int height;
    cin >> width >> height; cin.ignore();
    for (int i = 0; i < height; i++) {
        string line;
        getline(cin, line);
    }

    cout << "answer" << endl;
}
```

Rust

```
use std::io;

macro_rules! parse_input {
    ($x:expr, $t:ident) => ($x.trim().parse::<$t>().unwrap())
}

fn main() {
    let mut input_line = String::new();
    io::stdin().read_line(&mut input_line).unwrap();
    let inputs = input_line.split(" ").collect::<Vec<_>>();
    let w = parse_input!(inputs[0], i32);
    let h = parse_input!(inputs[1], i32);
    for i in 0..h as usize {
        let mut input_line = String::new();
        io::stdin().read_line(&mut input_line).unwrap();
        let line = input_line.trim_end().to_string();
    }

   println!("answer");
}
```