Problem 1: Family Time

It is Game Night and we are playing *The Game of Strife*. The rules are as follows:

- At the beginning of each game, each of two players start on tile 1 of the game track.
- Taking turns, each player draws a card. There are two types: **Life** and **Strife**.
- If a **Life** card is drawn, the player moves to the corresponding tile on the game track and places the card on the pile in front of themself.
- If a **Strife** card is drawn, the player discards the most recent **Life** card from the pile in from of themself, and moves to the tile that corresponds to the new number on top of their pile. If a player has no card to discard, they move to tile 1.
- The player with the least **Strife** cards wins.
- If both players have the same number of **Strife** cards, the player who is farthest along the game track wins.
- If both players occupy the same space on the game board, a tie occurs.

Create a program to print the results for The Game of Strife: `Player 1 wins`, `Player 2 wins`, or `Tie`.

The first line of input contains the total number of cards drawn in the game ($2 \leq N \leq 10^7$).

The following lines of input contain integers, separated by single spaces, representing the cards in the order drawn. A value of 0 represents a **Strife** card and all other values ($1 \leq M \leq 75$) are **Life** cards.

Sample input 1

```
10
7 54 65 38 2 0 37 28 63 26
```

Sample output 1

```
Player 1 wins
```

Player 2 received one Strife card while Player 1 received zero Strife cards.

Sample input 2

```
10
8 37 26 38 0 0 47 72 27 45
```

Sample output 2

```
Player 2 wins
```

Both players received the same number of Strife cards. Player 2 ended on space `45` while Player 1 ended on space `27`.

Sample input 3

```
10
0 0 0 0 0
0 0 0 0 0
```

Sample output 3

```
Tie
```

Both players received the same number of Strife cards. Both players ended on space `1`.

Problem 2: Knights Only

In a standard game of chess, each player begins with a set of six different pieces: one king, one queen, two rooks, two knights, and eight pawns, on an 8×8 grid. In this variant, the game involves only knights.

A knight moves as follows: two squares in one direction, either horizontally or vertically, followed by a move one square in a perpendicular direction. This results in an L-shaped move with a right-angled turn.

If a piece is moved onto another piece it "captures" that piece.

Create a program to determine the number of captures available.

The input consists of 8 lines representing each row of the game board. Each row consists of eight space-seperated characters representing each position in that row. Each position is either an `o` representing your knights, an `x` representing your opponent's pieces, or a period (`.`) representing an empty position. Each player may have up to 16 pieces on the board.

Output the number of different opponent pieces that can be immediately captured by one of your knights.

Sample input 1

```
x x . x x x x x
x x . x . x x x
. . . . . . . .
. . . o . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Sample output 1

```
0
```

Your knight, in the middle of the board, cannot capture any pieces in a single move.

Sample input 2

```
x x . x x x x x
x x . x x x x x
. . . . o . . .
. . . . . . . o
. . . . . . . .
. . . . . . . .
o o o o o o . o
o . o o o o . o
```

Sample output 2

```
3
```

None of the knights at the bottom of the board can capture any opponent pieces. Your knight in the middle of the board can capture 2 of the opponent pieces in the first row and one piece in the second row. Your knight on the right side can capture one piece on the second row, however it is the same piece the previous knight can capture. Thus, only 3 unique pieces can be captured.

Problem 3: Missing Number

Over the weekend, Max went to a party with his friends. Amidst the fun, he made a new friend and exchanged numbers. However, the next day, Max realized that he had misplaced the number for his new friend, and only remembered a few details. Max recalls some digits from the original phone number, and remembers that each digit is either the same or smaller than the next digit in the actual number.

Create a program to determine how many phone numbers Max must try to reach his friend.

Input consists on a sequence of exactly 10 characters; each character is a digit (`0` to `9`) or a pound symbol (`#`) representing an unknown digit.

Print the number of possible phone numbers, given what Max remembers.

Sample input 1	Sample output 1
`0123456#89`	`3`

The missing digit can be either `6` or `7` or `8`, thus only possible phone numbers are `0123456689`, `0123456789`, and `0123456889`.

Sample input 2	Sample output 2
`44#56#67#9`	`6`

The only possible phone numbers are `4445666779`, `4445666789`, `4445666799`, `4455666779`, `4455666789`, and `4455666799`.

Problem 4: Mystic Maze

In the heart of the enchanted realm of Eloria lay a legendary maze known as the "Mystic Maze". This labyrinth was not for the faint of heart; it was filled with traps, riddles, and untold treasures, said to be guarded by a mystical guardian. Among those brave enough to face the maze was Captain Lysandra, a renowned adventurer. Armed with her sword and a map, she entered the Mystic Maze, its walls formed from `o`'s and `x`'s, representing paths and barriers. Her quest was simple yet treacherous, find the fastest path that leads to the end of the maze.

The first line of input contains two integers indicating the number of rows and columns of the maze. Each dimension has a minimum value of 2 and a maximum value of 200. Each of the next lines contain space-separated characters consisting of `o` or `x`, where `o` represents an open space while `x` represents a maze barrier. Steps may only be taken in the four cardinal directions: North, South, East, or West.

Create a program to determine the minimum number of steps required to traverse the maze from the top-left to the bottom-right. If no such path exists, output `0`.

Sample input 1

```
8 9
o x x x x x o o o
o x x x x x o x x
o o o o o o o o o
x x o x x x x x o
x x o o o x x x o
x x x x o x x x o
x x x x x x x x o
```

Sample output 1

```
15
```

There are three paths available, however two of them are dead ends. The middle path, which extends horizontally across the entire maze reaches the bottom-right corner in 15 steps.

Sample input 2

```
5 5
o o o o o
o x x x o
o x o o o
o x o x x
o o o o o
```

Sample output 2

```
9
```

There are two paths available. One requires 13 steps while the other requires only 9 steps. Thus, the minimum is 9 steps.

Problem 5: Sweet Tooth

It's the Summer of 1957 and Brian and his friends are gathered for a delightful penny candy shopping spree at their local corner store. Each person knows how many pieces of candy they want to buy and how much they have to spend. They decide it will be simpler to just make a single collective purchase, so they want to know how much they will spend all together.

Input consists of three lines. The first line contains an integer (N) indicating the number of friends ($2 \leq N \leq 16$). The second line containing N integers indicating the number of pennies each friend has to spend. The third line contains N integers indicating the number of candies each friend wants to purchase. Numbers on the same line are separated by a single space and all amounts are between one and one hundred, inclusive.

Output the total number of penny candy that can be bought by the group of friends based on their desires and budgets.

Sample input 1

```
6
20 40 60 26 80 61
20 40 60 20 47 28
```

Sample output 1

```
215
```

Everyone has more pennies than the number of candies they want to buy.

Sample input 2

```
8
45 23 89 31 33 90 13 1
67 32 78 34 38 93 38 11
```

Sample output 2

```
325
```

The group collectively wants 391 penny candies, however they only have 325 pennies.

Problem 6: Park Placement

On your first day as an urban planner for your hometown government, your boss urgently assigns you the task of determining the best location for a new park in the city. You are provided with a city map, represented as a grid of numerical values indicating the type of each area. Additionally, you have a list of types suitable for building a park. Your goal is to identify the largest square park that can be established within the city.

The first line of input contains three integers (R, C, and N) indicating the number of rows, number of columns, and number of preferred locations, respectively ($1 \leq R,C \leq 1000$; $1 \leq N \leq 1000$). The second line of input contains N integers indicating the types of land suitable for a park. Finally, the next R lines contain C integers indicating the type of each block of land in the city. All values are separated by a single space.

Output the area of the largest possible park.

Sample input 1

```
5 15 4
1 2 3 4
9 9 9 9 9 9 1 3 1 3 9 9 9 9 9
9 9 9 9 9 9 3 3 3 2 9 9 9 9 9
9 9 9 9 9 9 2 2 3 1 9 9 9 9 9
9 9 9 9 9 9 2 8 3 3 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

Sample output 1

```
9
```

Two possible 3×3 parks are possible. If the eight on the fourth row were a suitable type, the area of the largest park would be 16.

Sample input 2

```
2 2 6
5 4 1 2 3 8
1 2
3 9
```

Sample output 2

```
325
```

Three possible 1×1 parks are possible, but no 2×2 park is possible.

Sample input 3

```
2 2 9
1 2 3 4 6 7 8 9 0
5 5
5 5
```

Sample output 3

```
0
```

There are no possible parks.

Problem 7: Sudoku Solver

Alex, a Sudoku enthusiast, dreams of one day participating in the prestigious World Sudoku Championship. Alex believes that mastering each type of deduction will be the key to success. So Alex practices different types each day.

Sudoku fills a 9×9 grid of cells with digits from 1 to 9. Each row of 9 must contain all digits from 1 to 9. Each column of 9 must contain all digits from 1 to 9. The grid is also divided into nine 3×3 subgrids. Each subgrid must contain all digits from 1 to 9.

A *neighbor* is any cell in the same row or same column or same subgrid. Since each digit must appear exactly once, the collective neighbors of a cell eliminate possible values for an empty cell.

An *elimination deduction* occurs when the possible values for an empty cell are reduced to one. When this happens, the cell can be filled with the unused value. For example, an empty cell could be deduced to be 1, if its neighbors include all the values from 2 to 9 (in any order).

Input consists of a single 9×9 grid representing the initial state of the puzzle. The grid includes prefilled numbers (digits `1` to `9`) and empty cells (represented by `0`).

Output the state of the grid after completing all possible elimination deductions.

Sample input 1

```
1 6 9 8 2 4 5 7 3
3 4 2 5 9 7 8 1 6
7 5 8 1 6 3 2 9 4
4 2 3 9 7 8 1 6 5
5 9 7 3 1 6 4 8 2
8 1 6 2 4 5 7 3 9
2 7 4 6 8 9 3 5 0
9 8 5 4 3 1 6 2 7
6 3 1 7 5 2 0 4 0
```

Sample output 1

```
1 6 9 8 2 4 5 7 3
3 4 2 5 9 7 8 1 6
7 5 8 1 6 3 2 9 4
4 2 3 9 7 8 1 6 5
5 9 7 3 1 6 4 8 2
8 1 6 2 4 5 7 3 9
2 7 4 6 8 9 3 5 1
9 8 5 4 3 1 6 2 7
6 3 1 7 5 2 9 4 8
```

An elimination deduction fills empty cell in row 7 with a `1`. Similarly, a `9` fills the empty cell in column 7. Once both of these deductions are made the final cell is filled with `8`.

Sample input 2

```
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 1 5 6 4 8 9 7
5 6 4 8 9 7 2 3 1
8 9 7 2 3 1 0 0 0
3 1 2 6 4 5 9 7 8
6 4 5 9 7 8 3 1 2
9 7 8 3 1 2 0 0 0
```

Sample output 2

```
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 1 5 6 4 8 9 7
5 6 4 8 9 7 2 3 1
8 9 7 2 3 1 0 0 0
3 1 2 6 4 5 9 7 8
6 4 5 9 7 8 3 1 2
9 7 8 3 1 2 0 0 0
```

No elimination deductions are possible; each of the empty cells has three possible values. Thus, the grid does not change.

Problem 8: Edit Distance

The *edit distance* is a string metric quantifying how dissimilar two strings are from each other. The "distance" is the sum of the cost of each edit operation required to transform one string into the other.

In this problem, the edit operations are:

- insert a single character,
- delete a single character, and
- replace one character with another character.

The cost of each operation depends on the ASCII value of the characters involved. Insert and delete operations cost the ASCII value of the character. The replacement cost is the absolute difference of the ASCII values of the characters involved.

The following pseudocode defines a recursive function to compute the ASCII edit distance, given two strings.

```
function first(s) → integer
  if s is empty then
    return 0
  return the ASCII value of the first character of s

function rest(s) → string
  if s is empty then
    return empty string
  return all of s except the first character

function distance(a,b) → integer
  if a is empty and b is empty then
    return 0
  if a is empty then
    return first(b) + dist(a,rest(b))
  if b is empty then
    return first(a) + dist(rest(a),b)
  if first(a) = first(b) then
    return distance(rest(a),rest(b))
  return minimum of
    first(a) + distance(rest(a),b), or
    first(b) + distance(a,rest(b)), or
    | first(a) - first(b) | + distance(rest(a),rest(b))
```

Create a program to determine the ASCII edit distance between strings.

The first line of input contains a single integer ($1 \leq N \leq 100$) indicating the number of distances to compute. The next N line-pairs contain pairs of strings to use. Each string contains at most 100 printable ASCII characters whose values are between 33 and 126, inclusive.