Please write a Python script to solve the following problem. It should read the input from a file input.txt, which has the same format as the example.

You and The Historians look a lot more pixelated than you remember. You're inside a computer at the North Pole!

Just as you're about to check out your surroundings, a program runs up to you. "This region of memory isn't safe! The User misunderstood what a pushdown automaton is and their algorithm is pushing whole bytes down on top of us! Run!"

The algorithm is fast - it's going to cause a byte to fall into your memory space once every nanosecond! Fortunately, you're faster, and by quickly scanning the algorithm, you create a list of which bytes will fall (your puzzle input) in the order they'll land in your memory space.

Your memory space is a two-dimensional grid with coordinates that range from 0 to 70 both horizontally and vertically. However, for the sake of example, suppose you're on a smaller grid with coordinates that range from 0 to 6 and the following list of incoming byte positions:

```
5,4
4,2
4,5
3,0
2,1
6,3
2,4
1,5
0,6
3,3
2,6
5,1
1,2
5,5
2,5
6,5
1,4
0,4
6,4
1,1
6,1
1,0
0,5
1,6
2,0
```

Each byte position is given as an X,Y coordinate, where X is the distance from the left edge of your memory space and Y is the distance from the top edge of your memory space.

You and The Historians are currently in the top left corner of the memory space (at 0,0) and need to reach the exit in the bottom right corner (at 70,70 in your memory space, but at 6,6 in this example). You'll need to simulate the falling bytes to plan out where it will be safe to run; for now, simulate just the first few bytes falling into your memory space.

As bytes fall into your memory space, they make that coordinate corrupted. Corrupted memory coordinates cannot be entered by you or The Historians, so you'll need to plan your route carefully. You also cannot leave the boundaries of the memory space; your only hope is to reach the exit.

In the above example, if you were to draw the memory space after the first 12 bytes have fallen (using . for safe and # for corrupted), it would look like this:

```
...#...
..#..#.
....#..
...#..#
..#..#.
.#..#..
#.#....
```

You can take steps up, down, left, or right. After just 12 bytes have corrupted locations in your memory space, the shortest path from the top left corner to the exit would take 22 steps. Here (marked with O) is one such path:

```
OO.#OOO
.O#OO#O
.OOO#OO
...#OO#
..#OO#.
.#.O#..
#.#OOOO
```

Simulate the first kilobyte (1024 bytes) falling onto your memory space. Afterward, what is the minimum number of steps needed to reach the exit?

Here’s a Python script to solve the problem. The script reads the input from a file named `input.txt`, simulates the memory corruption as bytes fall, and finds the shortest path from the top-left corner (0,0) to the bottom-right corner (6,6) using Breadth-First Search (BFS).

Simulate Memory

Save this script as `solve_problem.py` and make sure the `input.txt` file is in the same directory. The `input.txt` file should contain the byte positions, one per line, formatted as `x,y`.

How It Works

1. **Input Parsing**:
   - Reads the input file and parses it into a list of `(x, y)` coordinates.
2. **Memory Corruption Simulation**:
   - Updates the grid to mark positions as corrupted based on the first 1024 bytes.
3. **Pathfinding with BFS**:
   - Uses BFS to find the shortest path from the top-left corner to the bottom-right corner while avoiding corrupted cells.
4. **Output**:
   - Prints the minimum number of steps required to reach the exit. If no path exists, it will return `-1`.

Run the script with `python solve_problem.py` to get the result.

I get this error:

```
Traceback (most recent call last):
  File "/Users/jroberts/repos/advent-of-code-2024/day-18/gpt_jack/part1.py", line 75, in <module>
    main()
    ~~~~^^
  File "/Users/jroberts/repos/advent-of-code-2024/day-18/gpt_jack/part1.py", line 64, in main
    corrupted_grid = simulate_corruption(byte_positions, grid_size, steps_to_simulate)
  File "/Users/jroberts/repos/advent-of-code-2024/day-18/gpt_jack/part1.py", line 18, in simulate_corruption
    grid[y][x] = True
    ~~~~~~~^^^
IndexError: list assignment index out of range
```

The `IndexError: list assignment index out of range` suggests that one of the `(x, y)` coordinates in your input exceeds the bounds of the grid size. This could happen if the grid size or input positions were mismatched.

Steps to Fix

1. **Verify Grid Size**: The grid size is defined as `grid_size = 7` (0 to 6 inclusive). Ensure that all input coordinates fit within this range.
2. **Debugging Input**: Add a debug print to check the `(x, y)` values and ensure they fit within the expected grid range before updating the grid:

   python Copy code

```
def simulate_corruption(byte_positions, grid_size, steps_to_simulate):
    \"\"\"Simulates the memory corruption based on the falling bytes.\"\"\"
    grid = [[False] * grid_size for _ in range(grid_size)]

    for i, (x, y) in enumerate(byte_positions):
        if i >= steps_to_simulate:
            break
        if not (0 <= x < grid_size and 0 <= y < grid_size):
            print(f"Error: Coordinate ({x}, {y}) is out of bounds for grid size
{grid_size}.")
            continue  # Skip out-of-bound coordinates
        grid[y][x] = True

    return grid
```

3. **Check Input**: Inspect your `input.txt` file to ensure all `(x, y)` coordinates are within the valid range.
4. **Adjust Grid Size**: If the coordinates in the input file exceed the current grid size, you can adjust the `grid_size` to a larger value to accommodate them. For example, if you need a `70x70` grid, change `grid_size = 7` to `grid_size = 71`.
5. **Handle Out-of-Bounds Errors Gracefully**: Update the function to skip invalid coordinates without crashing.

After these updates, re-run the program. If the issue persists, let me know!

I get lots of printed error messages like:

Error: Coordinate (37, 5) is out of bounds for grid size 7.

The grid size should be determined based on the input file.

To dynamically determine the grid size based on the input file, we can calculate the maximum x and y coordinates from the input and adjust the grid size accordingly. Here's the updated script:

Updated Script

python Copy code

```python
from collections import deque

def read_input(file_path):
    """Reads the input file and returns a list of (x, y) coordinates."""
    with open(file_path, 'r') as f:
        lines = f.readlines()
    return [tuple(map(int, line.strip().split(','))) for line in lines]
```

```python
def get_grid_size(byte_positions):
    """Determines the required grid size based on the input byte positions."""
    max_x = max(x for x, _ in byte_positions)
    max_y = max(y for _, y in byte_positions)
    return max_x + 1, max_y + 1  # +1 to include the boundary

def simulate_corruption(byte_positions, grid_width, grid_height, steps_to_simulate):
    """Simulates the memory corruption based on the falling bytes."""
    grid = [[False] * grid_width for _ in range(grid_height)]

    for i, (x, y) in enumerate(byte_positions):
        if i >= steps_to_simulate:
            break
        if 0 <= x < grid_width and 0 <= y < grid_height:
            grid[y][x] = True

    return grid

def find_shortest_path(grid, start, end):
    """Finds the shortest path using BFS on the grid from start to end."""
    grid_height = len(grid)
    grid_width = len(grid[0])
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    queue = deque([(start[0], start[1], 0)])  # (x, y, steps)
    visited = set()
    visited.add((start[0], start[1]))

    while queue:
        x, y, steps = queue.popleft()

        # If we reach the end, return the number of steps
        if (x, y) == end:
            return steps

        # Explore neighbors
        for dx, dy in directions:
            nx, ny = x + dx, y + dy

            if 0 <= nx < grid_width and 0 <= ny < grid_height and not grid[ny][nx] and (nx, ny) not in visited:
                visited.add((nx, ny))
                queue.append((nx, ny, steps + 1))

    # If no path is found
    return -1

def main():
    input_file = 'input.txt'
    steps_to_simulate = 1024  # Simulate first kilobyte of bytes
    byte_positions = read_input(input_file)

    # Determine grid size dynamically
    grid_width, grid_height = get_grid_size(byte_positions)

    # Simulate corruption
    corrupted_grid = simulate_corruption(byte_positions, grid_width, grid_height, steps_to_simulate)

    # Find shortest path
    start = (0, 0)
    end = (grid_width - 1, grid_height - 1)
    shortest_path = find_shortest_path(corrupted_grid, start, end)

    print(f"The minimum number of steps needed to reach the exit is: {shortest_path}")

if __name__ == "__main__":
    main()
```

Key Changes

1. **Dynamic Grid Size**:
    - The `get_grid_size` function calculates the grid dimensions based on the maximum x and y values in the input file.
2. **Error-Free Simulation**:
    - Out-of-bounds errors are avoided as the grid is dynamically sized to accommodate all input positions.
3. **Flexible Handling**:
    - The script adjusts automatically to any valid input size.

Usage

1. Save the updated script as `solve_problem.py`.
2. Ensure your `input.txt` file is in the same directory.
3. Run the script with:

    bash Copy code

    ```bash
    python solve_problem.py
    ```

Let me know if you encounter any further issues!

Please update the script for part 2:

The Historians aren't as used to moving around in this pixelated universe as you are. You're afraid they're not going to be fast enough to make it to the exit before the path is completely blocked.

To determine how fast everyone needs to go, you need to determine the first byte that will cut off the path to the exit.

In the above example, after the byte at 1,1 falls, there is still a path to the exit:

```
O..#OOO
O##OO#O
O#OO#OO
OOO#OO#
###OO##
.##O###
#.#OOOO
```

However, after adding the very next byte (at 6,1), there is no longer a path to the exit:

```
...#...
.##..##
.#..#..
...#..#
###..##
.##.###
#.#....
```

So, in this example, the coordinates of the first byte that prevents the exit from being reachable are 6,1.

Simulate more of the bytes that are about to corrupt your memory space. What are the coordinates of the first byte that will prevent the exit from being reachable from your starting position? (Provide the answer as two integers separated by a comma with no other characters.)

Simulate Memory

```python
from collections import deque

def read_input(file_path):
  """Reads the input file and returns a list of (x, y) coordinates."""
  with open(file_path, 'r') as f:
    lines = f.readlines()
  return [tuple(map(int, line.strip().split(','))) for line in lines]

def get_grid_size(byte_positions):
  """Determines the required grid size based on the input byte positions."""
  max_x = max(x for x, _ in byte_positions)
  max_y = max(y for _, y in byte_positions)
  return max_x + 1, max_y + 1 # +1 to include the boundary

```

Edited

The script has been updated to determine the first byte that blocks the path to the exit. It simulates the corruption process and checks after each byte if the path to the exit is still reachable. If a byte blocks the path, its coordinates are printed as the result. Run the script and let me know if further adjustments are needed!

Message ChatGPT