```
using PlutoUI
```

```
parse_line (generic function with 1 method)
```

```
function parse_line(line)
    return split(line, "") .|> x -> parse(Int8, x)
end
```

```
parse_file (generic function with 1 method)
```

```
function parse_file(io::IO)
    parsed_lines = [parse_line(line) for line in eachline(io)]
    return hcat(parsed_lines...)'
end
```

```
get_adjacent_indexes (generic function with 1 method)
```

```
function get_adjacent_indexes(row, col, size)
    (total_rows, total_cols) = size
    return ([
        (row, max(1, col-1)) # Left
        (min(total_rows, row+1), col) # Down
        (row, min(total_cols, col+1)) # Right
        (max(1, row-1), col) # Up
    ] |> indexes -> filter(i -> i != (row, col), indexes))
end
```

```
is_lowest_point (generic function with 1 method)
```

```
function is_lowest_point(floor_heights, row, col, size)
    for (adj_row, adj_col) in get_adjacent_indexes(row, col, size)
        if floor_heights[row, col] >= floor_heights[adj_row, adj_col]
            return false
        end
    end

    return true
end
```

Problem 1

```
find_low_points (generic function with 1 method)
```

```
function find_low_points(floor_heights)
    data_size = size(floor_heights)
    (total_rows, total_cols) = data_size

    risk_level = 0
    for row in 1:total_rows
        for col in 1:total_cols
            if is_lowest_point(floor_heights, row, col, data_size)
                risk_level += floor_heights[row, col] + 1
            end
        end
    end

    return risk_level
end
```

```
633
```

```
  0.001109 seconds (20.00 k allocations: 2.747 MiB)
```

```
with_terminal() do
    open("./Day9/prob_input.txt") do io
        floor_heights = parse_file(io)
        @time find_low_points(floor_heights)
    end
end
```

Problem 2

```
find_basin_size (generic function with 1 method)
```

```
# This is DFS
function find_basin_size((row, col), data_size, floor_heights; explored=zeros(Bool, data_size...))
    if explored[row, col] == true
        return 0
    end

    curr_val = floor_heights[row, col]
    explored[row, col] = true
    adjacent_indexes = filter(
        function(pos)
            (r, c) = pos
            (explored[r, c] == false &&
            floor_heights[r, c] < 9 &&
            floor_heights[r, c] > curr_val)
        end,
        get_adjacent_indexes(row, col, data_size))

    adj_basin_sizes = [
        find_basin_size(ai, data_size, floor_heights; explored)
            for ai in adjacent_indexes]
    return length(adj_basin_sizes) > 0 ? (sum(adj_basin_sizes) + 1) : 1
end
```

```
find_basin_sizes_simple (generic function with 1 method)
```

```
function find_basin_sizes_simple(floor_heights)
    data_size = size(floor_heights)
    (total_rows, total_cols) = data_size
    return [find_basin_size((row, col), data_size, floor_heights) for col in 1:total_cols for row in 1:total_rows]
end
```

The `find_basin_sizes_simple` is a basic Depth First Search so it makes for simpler code though its doesn't use memory efficiently. The reason is that we are exploring all the points and doing recursion at each point which consumes more memory due to maintaining call stack. Instead we can use the solution from previous problem and first find lowest_points and then do Depth First Search from that.

```
find_basin_sizes_optim (generic function with 1 method)
```

```
function find_basin_sizes_optim(floor_heights)
    data_size = size(floor_heights)
    (total_rows, total_cols) = data_size

    lowest_points = []
    for row in 1:total_rows
        for col in 1:total_cols
            if is_lowest_point(floor_heights, row, col, data_size)
                push!(lowest_points, (row, col))
            end
        end
    end

    return [find_basin_size(lp, data_size, floor_heights) for lp in lowest_points]
end
```

```
1050192
```

```
  0.045669 seconds (209.06 k allocations: 122.068 MiB, 35.74% gc time)
```

```
with_terminal() do
    open("./Day9/prob_input.txt") do io
        floor_heights = parse_file(io)
        @time basin_sizes = find_basin_sizes_simple(floor_heights) |> sort
        reduce(*, basin_sizes[end-2:end])
    end
end
```

```
1050192
```

```
  0.003636 seconds (49.54 k allocations: 8.808 MiB)
```

```
# Using optimized solution
```