Question 1:

Graph to be stored in memory as a Adjacency List.
I will use 3 lists one to store distances between two nodes, another to store neighbors, and another to store directions

::PseduoCode to Build Adjacency Lists

```
for each row, col in rows, cols:
    graph_direction[row,col] = direction //this stores the direction of each node

//Code below maps out distances between neighbours
looks up keys in the graph_direction dictionary then stores relevant neighbors
for key in graph_direction_matrix.keys():
        if graph_direction_matrix[key] != 'X':
            if graph_direction_matrix[key] == 'S': ##down
                graph_distances[key] = [{(key[0]+3,key[1]) : 3}] + [{(key[0]+4,key[1]) : 4}]
            if graph_direction_matrix[key] == 'N': ##up
                graph_distances[key] = [{(key[0]-3,key[1]) : 3}] + [{(key[0]-4,key[1]) : 4}]
            if graph_direction_matrix[key] == 'E':
                graph_distances[key] = [{(key[0],key[1]+3) : 3}] + [{(key[0],key[1]+4) : 4}]
            if graph_direction_matrix[key] == 'W':
                graph_distances[key] = [{(key[0],key[1]-3) : 3}] + [{(key[0],key[1]-4) : 4}]
            if graph_direction_matrix[key] == 'NE':
                graph_distances[key] = [{(key[0]-3,key[1]+3) : 3}] + [{(key[0]-4,key[1]+4) : 4}]
            if graph_direction_matrix[key] == 'NW':
                graph_distances[key] = [{(key[0]-3,key[1]-3) : 3}] + [{(key[0]-4,key[1]-4) : 4}]
            if graph_direction_matrix[key] == 'SE':
                graph_distances[key] = [{(key[0]+3,key[1]+3) : 3}] + [{(key[0]+4,key[1]+4) : 4}]
            if graph_direction_matrix[key] == 'SW':
                graph_distances[key] = [{(key[0]+3,key[1]-3) : 3}] + [{(key[0]+4,key[1]+4) : 4}]
            if graph_direction_matrix[key] == 'J':
                graph_distances[key] = [{(key[0],key[1]): 0}]


//This builds out the Adjacency List for Neighbors in the following formart
graph(row, col): [neighbor1, neighbor2 etc]
Also stores out the goal
```

```
for key in graph_direction_matrix.keys():
        if graph_direction_matrix[key] != 'X':
            if graph_direction_matrix[key] == 'S':  ##down
                graph[key] = [(key[0] + 3, key[1])] + [(key[0] + 4,
key[1])]
            if graph_direction_matrix[key] == 'N':  ##up
                graph[key] = [(key[0] - 3, key[1])] + [(key[0] - 4,
key[1])]
            if graph_direction_matrix[key] == 'E':
                graph[key] = [(key[0], key[1] + 3)] + [(key[0], key[1]
+ 4)]
            if graph_direction_matrix[key] == 'W':
                graph[key] = [(key[0], key[1] - 3)] + [(key[0], key[1]
- 4)]
            if graph_direction_matrix[key] == 'NE':
                graph[key] = [(key[0] - 3, key[1] + 3)] + [(key[0] -
4, key[1] + 4)]
            if graph_direction_matrix[key] == 'NW':
                graph[key] = [(key[0] - 3, key[1] - 3)] + [(key[0] -
4, key[1] - 4)]
            if graph_direction_matrix[key] == 'SE':
                graph[key] = [(key[0] + 3, key[1] + 3)] + [(key[0] +
4, key[1] + 4)]
            if graph_direction_matrix[key] == 'SW':
                graph[key] = [(key[0] + 3, key[1] - 3)] + [(key[0] +
4, key[1] + 4)]
            if graph_direction_matrix[key] == 'J':
                graph[key] = (key[0], key[1])

//To find the goal: we just look up the graph_direction dict for a
value of 'J' and we return the row, col coordinates

::Algorithm to Solve the Puzzle


def find_path(graph, start, goal, path = []):
    path = path + [start]
    if start == goal:
        return path
    if not graph.has_key(start):
        return None
    for neighbor in graph[start]:
        if neighbor not in path:
            newpath = find_path(graph, neighbor, goal, path)
            if newpath: return newpath
    return None



The vertices will be start  position of the vines e.g [5,0]
```

The edges will be distances between vines e.g 3 or 4
Model will also utilize map to store the direction of a node, distances between vines and neighbors of vines

data structures utilized: dictionaries

graph_direction_matrix = { (0,0): 'S' } //stores direction of vine at that position
graph_matrix(1,3) = { (4,3): 3 } //stores distances