Lab 8 Solutions

Exercise 1: You are in a 2D-maze. Find the shortest way to get from the start location to the end location. The starting location is specified by 'S', and the ending location by 'E'. A cell with '.' represents an empty square, and you can walk through it. A cell with '#' represents a wall, and you cannot walk through it. From any given cell, you can walk either up, down, left, or right, provided that you don't walk outside the maze or into a wall. Find the shortest path from the starting location to the end location.

Example solution: Create a graph where each cell is a vertex, and two vertices have an edge between them if the corresponding cells are adjacent. Then, we do a BFS starting from the cell containing 'S' to find the distance to the cell containing 'E'. The primary function here is `shortestPath`.

```
def shortestPath(maze, n, m):
    for i in xrange(n):
        for j in xrange(m):
            if maze[i][j] == 'S':
                sx = i
                sy = j
            elif maze[i][j] == 'E':
                ex = i
                ey = j
    # BFS
    distance = []
    for i in xrange(n):
        distance += [[-1]*m]
    queue = deque()
    visit(sx, sy, queue, distance, 0)
    dx = [1, -1, 0, 0]
    dy = [0, 0, 1, -1]
    while len(queue) > 0:
        b = queue.popleft()
        x = b[0]
        y = b[1]
        if x==ex and y==ey:
            return distance[x][y]
        # there are at most 4 vertices adjacent to (x,y)
        for i in xrange(4):
            nx = x + dx[i]
            ny = y + dy[i]
            if nx>=0 and nx<n and ny>=0 and ny<m and maze[nx][ny]!='#':
                if distance[nx][ny] == -1:
                    visit(nx, ny, queue, distance, distance[x][y] + 1)
```

```
    if distance[ex][ey] == -1:
        return -1

def visit(x, y, queue, distance, D):
    distance[x][y] = D
    queue += [[x,y]] # push x
```

Exercise 2: You are given a 2D-image drawn using 10 colors. Each color is represented by an integer from 0 to 9. The image is described by giving the color of each pixel in the image. Two pixels are adjacent if one is immediately to the left of the other or immediately above the other. The image consists of many objects, and two pixels are part of the same object if they are adjacent and have the same color. The area of an object is the number of pixels it has. Find the area of the largest object in the image.

Example solution: Create a graph where each pixel is a vertex, and two vertices have an edge between them if the corresponding pixels are adjacent. Then, we are looking for the size of the largest connected component. The `dfs` function below is recursive and returns the number of pixels in the connected component of (x, y) which have not already been visited in the variable `visited`. In the `main` function, we loop over all pixels (the two nested `for` loops), i.e. all vertices, and `dfs` on vertices which are in connected components that we haven't already visited. We return the maximum size over all such `dfs` calls.

Here is a recursive solution; the primary function is `largestObjInImage`:

```
def dfs(x, y, visited, image):
dx = [1, -1, 0, 0]
dy = [0, 0, 1, -1]
n = len(image)
m = len(image[0])
visited[x][y] = True
ans = 1
for i in xrange(4):
nx = x + dx[i]
ny = y + dy[i]
if nx>=0 and nx<n and ny>=0 and ny<m and image[x][y]==image[nx][ny]:
if not visited[nx][ny]:
ans += dfs(nx, ny, visited, image)
return ans


def largestObjInImage(image):
n = len(image)
m = len(image[0])
visited = []
for i in xrange(n):
visited += [[False]*m]
```

```
ans = 0
for i in xrange(n):
for j in xrange(m):
if not visited[i][j]:
ans = max(ans, dfs(i, j, visited, image))
return ans
```