CSCI 630
LAB 1

For Implementing the shortest path using the A* star path finding algorithm, the following considerations were made:
A terrain dictionary with color tuple as a key and the corresponding tuple consisting of the type of terrain and the terrain constant.
A terrain constant is a value which is used to determine whether a terrain is difficult to walk on or not. Higher the value of the terrain constant more difficult is the path to traverse. For terrains that are impossible to walk on like a Lake,Swamp, or out of bounds as described in the image , the terrain constant is infinity.
The image is processed as a 2D matrix using the Image module from PIL. Since the image is in the 2 d form , a 2D array of same dimensions is created. Each element in the array consists of a Node object of Node class which consists of following attributes :-
x, y, terrain, elevation, gn, fn, parent
(x,y)- represents the coordinate of the Node in the Image file
Terrain- Represents the terrain tuple corresponding to the color of the pixel in the image file
Elevation=Elevation of the pixel after importing data from elevation file
Parent=is the previous node of the current node if it is visited in the A* algorithm. This helps us to backtrack the path file and append the nodes for printing.

For implementing the cost function, The coordinates and the corresponding elevation of the subsequent nodes are used for computing the straight line distance. This cost function is used in the A* for computing the f_n value.
For h(n), the distance value is calculated between the current node and the destination node , which in our scenario is the next co-ordinate in the path file.
For calculating the g_n , we consider the current node and the next neighbor which is selected because of the minimum f_n value. In the end we multiply the 1st node's terrain constant value with the distance we get.
The A* is implemented in the following steps:
1.Priority queue and the visited array are initialized.
2.we calculate the heuristic from initial node to goal.
3.The following happens until we find the goal node
4.We check for 4 neighbours of the current node and if it is not present in the visited array, we calculate the g_n ,h_n and f_n.

5.If we find a lower f value than the current one, we make the vertex's parent node to current node and add it to the visited and priority queue
6.Then we pop the priority queue and make the popped value the current.
This gives us the total path traversed by A*.

For Seasons we make the following adjustments:-
Fall:-We traverse the whole 2-d array of Nodes and if the terrain of the traversed node is a Slow run forest, then we increase the terrain constant and also for the neighboring vertexes

Spring:-As described in the problem, we first find the edge coordinates of all the water bodies and find 15 neighbours of each edge pixel using BFS which visits a node if and only if the the elevation difference between current and neighbor is less than 1.0 metre and the vertex's terrain is not water, since we want to modify the adjacent land to water edges with Mud. Mud's key is already stted in the dictionary.

Winter:-For winter , we find the 7 pixel neighbours of the water edges as we did in the Spring season, using BFS. So, we change the selected pixel's terrain to Frost and decrease the terrain constant from Infinity to 1000. So that a path can be etched out on it.

Summer:- No arrangement had to be made for Summer.

The path from the A* algorithm. The path array consists of every pixel that the algorithm finds while traversing from one coordinate to next.
Then using the PIL library we are putting the pixel that finally represents the path.