**General**

All the points in a maze are represented as 2-dimensional array in python. Each element in this array is an object of custom class called Cell. Each cell in this maze can be either wall, path, starting cell or ending cell. At the beginning we are only given the starting node and function to get all the neighbours of a given cell. Using this function, we can get the neighbours of starting cell as well.

**Node**

The Node data structure contains current position (current cell) in the maze, the parent node from which this node was obtained and cost to move to this node

**Stack Data structure**

This is last in first out data structure. Any element that is recently added to the stack gets removed first. Stack is implemented in python using python list and pop method of python.

**Queue Data structure**

This is a first in first out data structure. Element added first will get removed first before other elements added after the first element. Queue is implemented in python using list and list slicing operations.

**Depth First Search**

This search algorithm uses the stack data structure. At the beginning of the search the stack will contain only the starting node. The starting node has the starting cell and no parent and no cost. The algorithm does the following (pseudo code):

1. If the stack is empty then no solution
2. Remove a node from the stack
3. Check if the node’s cell is the ending cell and if it is the ending cell then search is over. The path followed can be attained by backtracking using the parent property of the ending node
4. Add this node’s cell in the explored set
5. Get all the neighboring cells for this node’s cell
6. For every neighboring cell if they are not in the explored set and no node in stack has the cell as the neighboring cell then create a node of this cell with the parent as the previous node and this new node to the frontier.
7. Repeat

By running this algorithm (DFS) we either end up at the ending node or we find that we have no solution for the search problem. By our choice of the data structure which is a stack, we explore the newer nodes, we explore the nodes deeper and deeper till we reach the goal or a wall. If we reach a wall then we go back to the node at the top of the stack which will be neighbor to the node where we hit a wall on deeper searching.

This algorithm guarantees to find a solution as long as the maze is finite. Unlike BFS (next algorithm), DFS in most of the case explores lesser number of nodes and is in most of the cases is faster in finding the path. DFS also uses less memory and computation compared to DFS, Greedy Best First Search and A\*.

The main limitation of DFS is that it does not guarantee the shortest path and may possibly the longest path if multiple solutions are available for the maze.

DFS can be used in places where optimal solution is not necessary but memory and computation power is limited.

**Breadth First Search**

This algorithm uses the same procedure as DFS but instead of stack it uses a queue data structure. Due to the choice of the data structure the algorithm explores the shallowest nodes rather than deeper nodes and this way BFS guarantees shortest path. BFS may take up more space and computation compared to DFS but BFS guarantees the shortest path which is major benefit over DFS.

The only limitation of BFS being it takes up more memory and time to find the optimal solution, but it guarantees the shortest path.

DFS can be used in places where optimal solution is required where memory and computation power are abundant. DFS can be used in apps like Google Maps (Google Maps actually uses Dijkstra’s shortest path algorithm).

**Greedy Best First Search**

Unlike the other two algorithms previously mentioned which where uninformed search algorithm, this algorithm is informed search algorithm i.e., the algorithm knows some extra piece of information about the maze beforehand, in this case the algorithm knows the coordinates of all the points in the maze. In this algorithm we define a function called heuristic which returns the Manhattan distance between two cells in the maze. The Manhattan distance is defined as the |x1 – x2| + |y1 – y2|. This algorithm proceeds the same way as breadth first search but the difference being at the time of creation of the node, we also give it a cost which equals the Manhattan distance between the current cell and goal cell and while removing the node from the data structure, we remove the node such that the node being removed has the least cost among all the nodes in the data structure. By this way, when we have a choice to make, we make an informed choice and hence the informed algorithm.

This algorithm does not guarantee optimal solution but it does guarantee fewer exploration, lower memory usage and faster solution time compared to DFS and BFS

This algorithm can be used where time required to find the solutions is less and memory and computation is limited.

**Dijkstra’s Shortest Path**

This algorithm initially assigns a cost of infinity to every single node except the starting node which will have a cost of zero. This algorithm begins by getting all the neighbors of the current node and assigns the cost of all the neighboring node as cost of the current node + cost required to react the neighboring node if this new cost determined is lower than the present cost of the neighboring nodes. If the cost is to be changed then the parent property of the neighboring node is changed to the current node. After this step, from all the neighboring nodes we choose the node with the least cost and go to that node. We repeat this same process till we reach the ending cell.

This algorithm guarantees the shortest path. Dijkstra’s Algorithm is used by Google in Google Maps. Dijkstra’s algorithm gives the optimal solution but computationally intensive algorithm.

**Implementation Details**

BFS, DFS and Greedy Best First search algorithms are implemented in python. I have used only pygame as the dependency for rendering purposes. The program can be run in two modes. The manual mode where you have to press any keyboard key to move forward in the algorithm. In this mode the inner workings of the algorithm become clear as we can visually see each step made by the algorithm. The other mode is normal mode where on the press of a keyboard key the solution is presented instantly. For more info about how to run the program refer to the README.md in github