1. **BFS:**

BFS also known as breadth first search works on the principle of exploring nodes that are visited first. To implement this algorithm using python, following pseudo code is used.

*BFS Pseudo code:*

*add start to queue*

*loop:*

*item🡨pop first element in the queue*

*for node in graph(item)*

*If node not visited*

*add node to visited*

*add node to queue*

*keep track of parent of node*

*if node is goal*

*break*

*until the goal is found or no item in the queue*

Using the above pseudo code, program is implemented in python. Steps are counted every time a node is explored. It is simply the size of visited. Later the path is constructed using the following pseudo code

*Path pseudo code:*

node *🡨 goal*

*if found*

*while node is not start*

*node = parent(node)*

*add node to path*

*else*

*path is empty*

*reverse the path*

In this pseudo code, I am backtracking parents of each node starting from the goal to start node. If the path is not found, an empty path is returned.

BFS is tested with various test cases. Results are shown in the following figures.

Chart, bar chart

Description automatically generated

For this given grid and obstacles, the BFS takes 10 steps to find the path.

Chart

Description automatically generated

BFS takes 64 steps to find the path in the above grid.

Chart

Description automatically generated with medium confidence

For another test map shown in the above picture, BFS finds path in 188 steps.

1. **DFS:**

DFS is similar to BFS, but the order of exploration is a little different. We can achieve the functionality of DFS by popping the last element of the queue instead of the first element. We need to make sure of the correct queue formation as well.

Following pseudo code explains the DFS implementation.

*DFS Pseudo code:*

*add start to the queue*

*loop*

*item🡨pop last element in the queue*

*if item not visited*

*index=-1*

*for node in graph(item)*

*insert node to the queue at index*

*decrease index*

*keep track of parent*

*if node is goal*

*break*

*until the goal is reached and or no item in the queue*

The path is constructed using the same method that is explained for BFS.

Results with DFS are shown in the below pictures.

Chart, bar chart

Description automatically generated

It takes 9 steps for DFS to find the path in above grid.

Chart

Description automatically generated

It takes 41 steps for DFS to find the path for the above grid.

A picture containing text, crossword puzzle

Description automatically generated

It takes 127 steps for DFS to find the path in the above grid.

1. **Dijkstra:**

In Dijkstra, we use cost to come as the criterion to select the best possible node to explore. In many ways Dijkstra is same as BFS, but here we consider cost as well. Pseudo code for it is explained below.

*Pseudo Code:*

*initialize all the nodes’ cost as infinity*

*change start node cost as 0.*

*add start to the queue*

*loop*

*item🡨 minimum cost node out of the queue*

*remove item from queue*

*for node in graph(item)*

*if node not visited*

*update the cost for node*

*add node to visited*

*add node to queue*

*keep track of parent*

*if node is goal*

*break*

*until goal is found or no element in queue*

*Results with Dijkstra are shown below.*

*Chart, bar chart

Description automatically generatedChart

Description automatically generatedChart

Description automatically generated with medium confidence*

It takes 10, 64 and 188 steps respectively for Dijkstra to find the path in the above grids.

1. **A\*:**

A star is same as Dijkstra but while computing the best possible node to explore, we use heuristic distance as well. Here Manhattan distance is used as heuristic function.

*Pseudo Code:*

*initialize all the nodes’ cost as infinity*

*change start node cost as 0.*

*add start to the queue*

*loop*

*item🡨 minimum (cost+heuristic) node out of the queue*

*remove item from queue*

*for node in graph(item)*

*if node not visited*

*update the cost for node*

*add node to visited*

*add node to queue*

*keep track of parent*

*if node is goal*

*break*

*until goal is found or no element in queue*

For computing heuristic, following pseudo code is used.

*Initialize node*

*Initialize goal*

*x\_goal,y\_goal 🡨 goal*

*x\_node,y\_node 🡨 node*

*heuristic🡨 abs(x\_goal-x\_node)+abs(y\_goal-y\_node)*

Results of A\* are shown below.

Chart, bar chart

Description automatically generatedChart

Description automatically generatedChart

Description automatically generated

It takes 7, 50 and 156 steps respectively for A\* to find the path in the above grids.