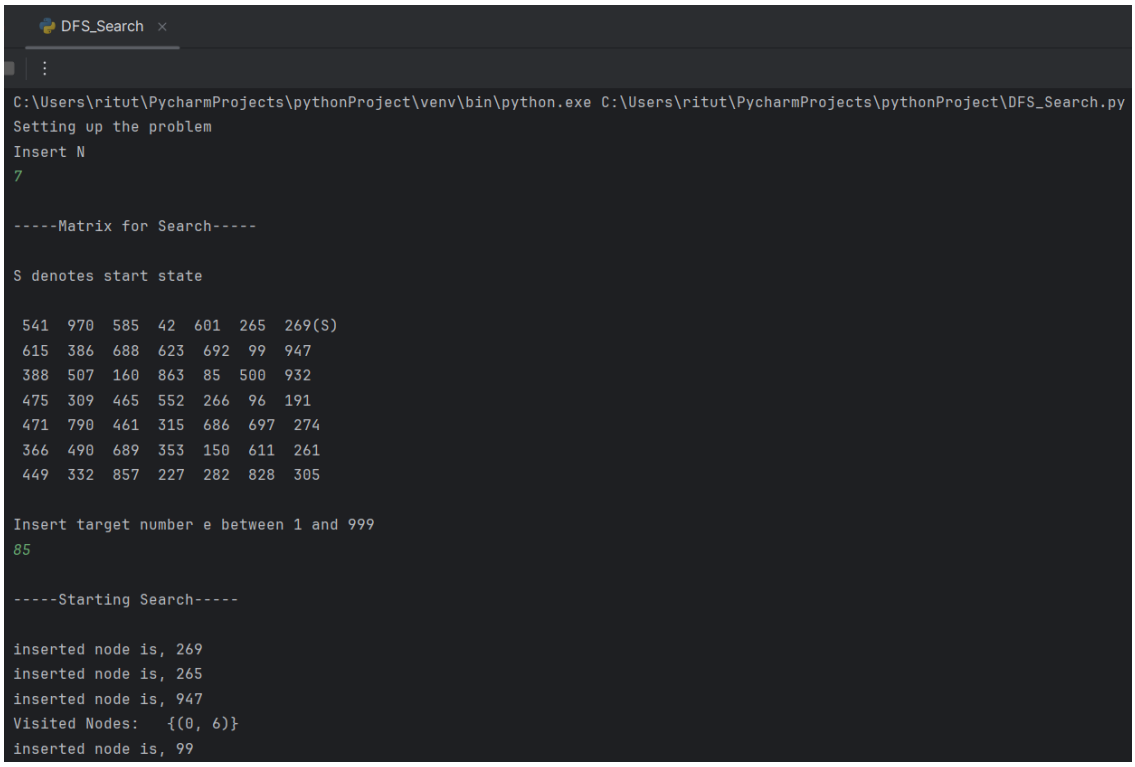
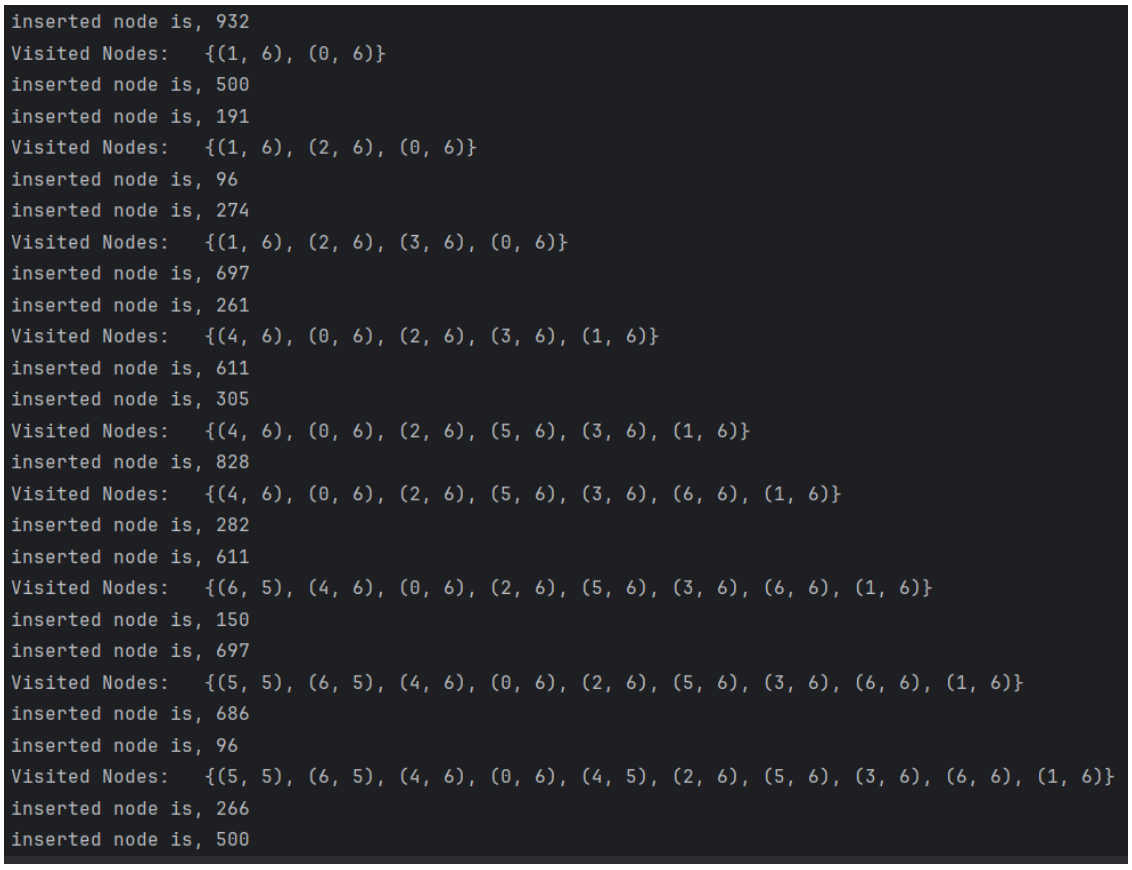
DFS Algorithm:-





```
Visited Nodes:   {(5, 5), (6, 5), (4, 6), (0, 6), (4, 5), (2, 6), (5, 6), (3, 6), (6, 6), (1, 6), (3, 5)}
inserted node is, 85
inserted node is, 99
Visited Nodes:   {(5, 5), (6, 5), (4, 6), (0, 6), (4, 5), (2, 6), (5, 6), (3, 6), (6, 6), (1, 6), (2, 5), (3, 5)}
inserted node is, 692
inserted node is, 265
Visited Nodes:   {(5, 5), (6, 5), (1, 5), (4, 6), (0, 6), (4, 5), (2, 6), (5, 6), (3, 6), (6, 6), (1, 6), (2, 5), (3, 5)}
inserted node is, 601
Visited Nodes:   {(5, 5), (6, 5), (1, 5), (4, 6), (0, 6), (4, 5), (2, 6), (5, 6), (0, 5), (3, 6), (6, 6), (1, 6), (2, 5), (3, 5)}
inserted node is, 42
inserted node is, 692
Visited Nodes:   {(5, 5), (0, 4), (6, 5), (1, 5), (4, 6), (0, 6), (4, 5), (2, 6), (5, 6), (0, 5), (3, 6), (6, 6), (1, 6), (2, 5), (3, 5)}
inserted node is, 623
inserted node is, 85
Visited Nodes:   {(5, 5), (0, 4), (6, 5), (1, 5), (4, 6), (1, 4), (0, 6), (4, 5), (2, 6), (5, 6), (0, 5), (3, 6), (6, 6), (1, 6), (2, 5), (3, 5)}
Search successful. Found at  (2, 4)

Path Solution: [(0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (6, 5), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (0, 5), (0, 4), (1, 4), (2, 4)]
```

We can see from the output that DFS keeps on expanding the deepest node first, without any information about how close the goal is. The visited node is popped from the stack, and all its neighbors are added to the fringe. The explored nodes are being tracked and stored in a set. In the worst case DFS might traverse the entire search matrix.

BFS Algorithm:-

```
C:\Users\ritut\PycharmProjects\pythonProject\venv\bin\python.exe C:\Users\ritut\PycharmProjects\pythonProject\BFS_Search.py
Setting up the problem
Insert N
7

-----Matrix for Search-----

S denotes start state

 267  657  674  374  255  799  556(S)
 94  91  115  263  947  907  493
 775  536  935  819  987  350  407
 857  294  951  592  782  295  402
 904  86  180  23  658  310  645
 367  194  19  854  132  744  789
 555  821  42  543  231  697  716

Insert target number e between 1 and 999
350

-----Starting Search-----

inserted node is, 556
inserted node is, 799
inserted node is, 493
Visited Nodes:   {(0, 6)}
```

```
inserted node is, 255
inserted node is, 907
Visited Nodes:   {(0, 5), (0, 6)}
inserted node is, 907
inserted node is, 407
Visited Nodes:   {(1, 6), (0, 5), (0, 6)}
inserted node is, 374
inserted node is, 947
Visited Nodes:   {(1, 6), (0, 4), (0, 5), (0, 6)}
inserted node is, 947
inserted node is, 350
Visited Nodes:   {(0, 4), (1, 5), (0, 6), (0, 5), (1, 6)}
inserted node is, 350
inserted node is, 402
Visited Nodes:   {(0, 4), (1, 5), (0, 6), (2, 6), (0, 5), (1, 6)}
inserted node is, 674
inserted node is, 263
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (0, 6), (2, 6), (0, 5), (1, 6)}
inserted node is, 263
inserted node is, 987
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (2, 6), (0, 5), (1, 6)}

Search successful. Found at  (2, 5)

Path Solution: [(0, 6), (0, 5), (1, 5), (2, 5)]
```

From the output it is clear that BFS keeps on expanding the shallowest node first, without any information about how close the goal is. It keeps on searching level wise. The visited node is dequeued from the queue, and all its neighbors are added to the fringe (queue). The explored nodes are being tracked and stored in a set.

UCS Algorithm:-

```
C:\Users\ritut\PycharmProjects\pythonProject\venv\bin\python.exe C:\Users\ritut\PycharmProjects\pythonProject\UCS_Search.py
Setting up the problem
Insert N
7

-----Matrix for Search-----

S denotes start state

 658  616  854  811  930  324  27(S)
 694  40  465  309  916  364  506
 251  720  540  451  670  162  81
 868  738  571  171  102  966  663
 884  643  710  189  291  989  433
 240  967  510  646  808  742  85
 559  700  125  107  678  880  730

Insert target number e between 1 and 999
40

-----Starting Search-----

inserted node is, 27
inserted node is, 324, cost: 1
inserted node is, 506, cost: 2
Visited Nodes:   {(0, 6)}
inserted node is, 930, cost: 2
inserted node is, 364, cost: 3
Visited Nodes:   {(0, 5), (0, 6)}
inserted node is, 811, cost: 3
inserted node is, 916, cost: 4
Visited Nodes:   {(0, 4), (0, 5), (0, 6)}
inserted node is, 364, cost: 3
inserted node is, 81, cost: 4
Visited Nodes:   {(1, 6), (0, 4), (0, 5), (0, 6)}
inserted node is, 854, cost: 4
inserted node is, 309, cost: 5
Visited Nodes:   {(0, 4), (0, 3), (0, 6), (0, 5), (1, 6)}
inserted node is, 916, cost: 4
inserted node is, 162, cost: 5
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (0, 6), (0, 5), (1, 6)}
inserted node is, 616, cost: 5
inserted node is, 465, cost: 6
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (0, 6), (0, 2), (0, 5), (1, 6)}
inserted node is, 309, cost: 5
inserted node is, 670, cost: 6
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (0, 5), (1, 6)}
inserted node is, 162, cost: 5
inserted node is, 663, cost: 6
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6)}
inserted node is, 658, cost: 6
inserted node is, 40, cost: 7
Visited Nodes:   {(0, 1), (0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6)}
inserted node is, 465, cost: 6
inserted node is, 451, cost: 7
Visited Nodes:   {(0, 1), (0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (1, 3)}
inserted node is, 670, cost: 6
inserted node is, 966, cost: 7
Visited Nodes:   {(0, 1), (0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (2, 5), (1, 3)}
```

```
Visited Nodes:   {(0, 1), (0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (2, 5), (1, 3)}
inserted node is, 694, cost: 8
Visited Nodes:   {(0, 1), (0, 4), (1, 5), (0, 0), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (2, 5), (1, 3)}
inserted node is, 40, cost: 7
inserted node is, 540, cost: 8
Visited Nodes:   {(0, 1), (1, 2), (0, 4), (1, 5), (0, 0), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (2, 5), (1, 3)}
inserted node is, 451, cost: 7
inserted node is, 102, cost: 8
Visited Nodes:   {(0, 1), (2, 4), (1, 2), (0, 4), (1, 5), (0, 0), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (1, 6), (2, 5), (1, 3)}
inserted node is, 966, cost: 7
inserted node is, 433, cost: 8
Visited Nodes:   {(0, 1), (2, 4), (1, 2), (0, 4), (1, 5), (0, 0), (0, 3), (1, 4), (0, 6), (0, 2), (2, 6), (0, 5), (3, 6), (1, 6), (2, 5), (1, 3)}

Search successful. Found at  (1, 1)

Path Solution: [(0, 6), (0, 5), (0, 4), (0, 3), (0, 2), (0, 1), (1, 1)]
```

Here, the visited node's neighbors along with its actual cost are added to the fringe. Using priority queue, on every deletion of the visited node from fringe, the minimum cost neighbor is explored first.

Greedy Search Algorithm:-

```
C:\Users\ritut\PycharmProjects\pythonProject\venv\bin\python.exe C:\Users\ritut\PycharmProjects\pythonProject\Greedy_Search.py
Setting up the problem
Insert N
7

-----Matrix for Search-----

S denotes start state

 346  171  85  816  913  11  752(S)
 133  561  199  336  620  630  320
 850  911  347  940  135  54  530
 360  655  131  509  167  316  885
 302  958  842  382  866  333  34
 817  828  991  874  565  164  212
 875  640  846  896  822  994  289

Insert target number e between 1 and 999
316
```

Here, we take the heuristic function as the sum of vertical and horizontal from current cell to element e, provided by the user. The traversal is prioritized w.r.t. how close the node is in comparison to the goal node. It greedily moves in that direction.

```
-----Starting Search-----

inserted node is, 752
inserted node is, 11, heuristic value: 3
inserted node is, 320, heuristic value: 3
Visited Nodes:   {(0, 6)}
inserted node is, 913, heuristic value: 4
inserted node is, 630, heuristic value: 2
Visited Nodes:   {(0, 5), (0, 6)}
inserted node is, 620, heuristic value: 3
inserted node is, 320, heuristic value: 3
inserted node is, 54, heuristic value: 1
Visited Nodes:   {(1, 5), (0, 5), (0, 6)}
inserted node is, 135, heuristic value: 2
inserted node is, 530, heuristic value: 2
inserted node is, 316, heuristic value: 0
Visited Nodes:   {(1, 5), (2, 5), (0, 5), (0, 6)}

Search successful. Found at  (3, 5)

Path Solution: [(0, 6), (0, 5), (1, 5), (2, 5), (3, 5)]
```

A* Search Algorithm:-

```
C:\Users\ritut\PycharmProjects\pythonProject\venv\bin\python.exe C:\Users\ritut\PycharmProjects\pythonProject\A_Star_Search.py
Setting up the problem
Insert N
7

-----Matrix for Search-----

S denotes start state

 423  855  901  490  353  604  196(S)
 560  340  418  24  952  437  905
 610  230  441  504  472  856  833
 784  117  529  510  339  918  803
 758  540  262  139  515  840  940
 743  859  530  687  842  976  351
 950  819  235  424  136  502  323

Insert target number e between 1 and 999
472

-----Starting Search-----
```

```
inserted node is, 604, cost (g): 1, heuristic value (h): 3, f: 4
inserted node is, 905, cost (g): 2, heuristic value (h): 3, f: 5
Visited Nodes:   {(0, 6)}
inserted node is, 353, cost (g): 2, heuristic value (h): 2, f: 4
inserted node is, 437, cost (g): 3, heuristic value (h): 2, f: 5
Visited Nodes:   {(0, 5), (0, 6)}
inserted node is, 490, cost (g): 3, heuristic value (h): 3, f: 6
inserted node is, 952, cost (g): 4, heuristic value (h): 1, f: 5
Visited Nodes:   {(0, 4), (0, 5), (0, 6)}
inserted node is, 24, cost (g): 5, heuristic value (h): 2, f: 7
inserted node is, 437, cost (g): 5, heuristic value (h): 2, f: 7
inserted node is, 472, cost (g): 6, heuristic value (h): 0, f: 6
Visited Nodes:   {(1, 4), (0, 4), (0, 5), (0, 6)}
inserted node is, 905, cost (g): 4, heuristic value (h): 3, f: 7
inserted node is, 856, cost (g): 5, heuristic value (h): 1, f: 6
Visited Nodes:   {(0, 4), (1, 5), (1, 4), (0, 6), (0, 5)}
inserted node is, 833, cost (g): 4, heuristic value (h): 2, f: 6
Visited Nodes:   {(0, 4), (1, 5), (1, 4), (0, 6), (0, 5), (1, 6)}
inserted node is, 901, cost (g): 4, heuristic value (h): 4, f: 8
inserted node is, 24, cost (g): 5, heuristic value (h): 2, f: 7
Visited Nodes:   {(0, 4), (1, 5), (0, 3), (1, 4), (0, 6), (0, 5), (1, 6)}
Search successful. Found at  (2, 4)

Path Solution: [(0, 6), (0, 5), (0, 4), (1, 4), (2, 4)]
```

For A* search, the node to be visited next is decided based on the sum, f(n) of forward h(n) and backward cost g(n). The minimum value f-value node is visited next from the fringe.