Report

Assignment - 1

Q1: For N-Puzzle problem, I've implemented following search algorithms:

- BFS
  Complete? : Yes
  Complexity: $O(b^{d+1})$
  Space: $O(b^{d+1})$
  Optimal? : Yes

- DFS
  Complete? : No
  Complexity: $O(b^{m})$
  Space: $O(b^{m})$
  Optimal? : No

- A* (Heuristic used: Manhattan distance)
  Complete? : Yes
  Complexity: Exponential
  Space: $O(b^{d+1})$
  Optimal? : Yes

- IDA*
  Complete? & Optimality? : Same as A*
  Complexity: $O(b^{m})$
  Space: $O(b^{m})$

Some observations:

- A* algorithm is faster than Bfs, Dfs and IDA*.
  Time taken by A* is very less as compared to other search algorithms.

- Iterations: No of nodes visited by A* is also less as compared to other algorithms. Dfs and Bfs varies a lot for different configurations of graph. No of nodes visited by IDA* also varies a lot on graph, since it can contain a cycle in graph.

- Bfs is faster than Dfs uptil some Depth x. After this depth x number of nodes in visited array of bfs are increased enormously which makes lot of iterations to check for already visited node. Dfs is not affected by this since we add node to visited array after popping it from stack.

- Depth(Level Order): Lowest in A*, Bfs, IDA*.

- For N uptil 15 Algorithms are working fine and usually give answer within 1 min. For higher N value algorithms are taking lot of time.

- IDA* gets opportunity to explore graph in depth for each iterations. By setting the cost threshold. For some cases IDA* search time is better than A*. (*search result 2nd image)*

```
Kaustav@Alienware->Assignment-1$ pyt
hon3 Q1.py
Enter Your Puzzle N:
8
Enter Start State:
0 2 3
1 4 5
8 7 6
Enter Goal State:
1 2 3
8 0 4
7 6 5
-------Menu-------
1. BFS
2. DFS
3. A*
4. IDA*
5. All
6. Exit
5
No of Nodes visited:  59
Depth:  6
Found Match for Bfs
Total time taken by bfs:  3

No of Nodes visited:  1807
Depth:  1320
Found Match for Dfs
Total time taken by Dfs:  50

No of Nodes visited:  7
Depth:  6
Found Match for A*
Total time taken by A*:  5

Threshold:  6
No of Nodes visited:  7
Depth:  6
Found Match for IDA*
Total time taken by IDA*:  3
```

```
Kaustav@Alienware->Assignment-1$ pyt
hon3 Q1.py
Enter Your Puzzle N:
8
Enter Start State:
0 3 8
4 1 7
2 6 5
Enter Goal State:
1 2 3
4 5 6
7 8 0
-------Menu-------
1. BFS
2. DFS
3. A*
4. IDA*
5. All
6. Exit
5
No of Nodes visited:  120793
Depth:  24
Found Match for Bfs
Total time taken by bfs:  4970

No of Nodes visited:  75606
Depth:  52500
Found Match for Dfs
Total time taken by Dfs:  2066

No of Nodes visited:  1656
Depth:  24
Found Match for A*
Total time taken by A*:  172

Threshold:  16
Threshold:  18
Threshold:  20
Threshold:  22
Threshold:  24
No of Nodes visited:  1656
Depth:  24
Found Match for IDA*
Total time taken by IDA*:  141
```

Q2: For colored NxN board. I've implemented following search algorithms:

Algorithm analysis same as in Q1

- BFS
- DFS
- A*

Observations and Findings:

- **Iterations:** A* has less number of iterations than BFS and DFS, DFS has less number of iterations than BFS for majority of cases.

- **Time**: A* is the fastest algorithm. Then comes DFS with higher time. BFS is the slowest as branching factor is high for Q2.

- A* and DFS consume less memory, but in worst case All three consume equivalent memory.
- Branching factor is very high. Max possible new matrix configuration for a coordinate can be 3, So for a whole matrix it will be very high.

- All algorithms are taking huge amount of time for N > 7.

Heuristic Function: For a coordinate x,y in NxN board. Find number of adjacent neighbours with same color as of x,y point. Do this for every node and add them. This will give us the heuristic cost.
Why this heuristic works?: For all neighbours of a particular node-- we will select minimum f + h cost value(h is heuristic function cost; f is cost from root node).By selecting minimum we are reaching more closer to our goal state.

A* Algorithm- Finding Neighbours Configuration
For a point x,y in matrix, i check if its adjacent have same color as node x,y. If at least one adjacent has then i switch x,y point with adjacent point of different color(if it exist). Repeat this for each point in matrix and add all new matrix to children array of parent node.

```
Kaustav@Alienware->Assignment-1$ pyt
hon3 Q2.py
Enter width of the Board:
4
Enter Matrix:
1 2 3 3
2 3 4 4
3 2 3 3
1 2 3 4
-------Menu-------
1. BFS
2. DFS
3. A*
4. All
5. Exit
4
No of Nodes visited:  328
Depth:  3
Found Match for Bfs
Total time taken by bfs:  74

No of Nodes visited:  54
Depth:  39
Found Match for Dfs
Total time taken by Dfs:  10

No of Nodes visited:  17
Depth:  3
Found Match for A*
Total time taken by A*:  6
```

```
hon3 Q2.py
Enter width of the Board:
6
Enter Matrix:
1 4 3 2 4 3
2 1 1 3 1 2
3 3 1 2 4 3
4 1 2 4 3 4
2 1 4 2 4 1
3 4 3 3 2 4
-------Menu-------
1. BFS
2. DFS
3. A*
4. All
5. Exit
4
No of Nodes visited:  33494
Depth:  5
Found Match for Bfs
Total time taken by bfs:  23358

No of Nodes visited:  310
Depth:  240
Found Match for Dfs
Total time taken by Dfs:  667

No of Nodes visited:  390
Depth:  5
Found Match for A*
Total time taken by A*:  284
```