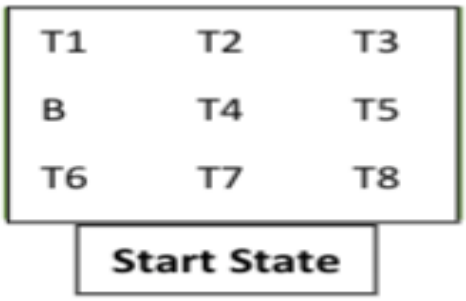
CS/AI 2203
ARTIFICIAL & COMPUTATIONAL INTELLIGENCE
ASSIGNMENT-1 Report

Thribhuvan Rapolu
SE21UCSE231
CSE-4

Index:

Initial and Final State Example for Best first search and A* search

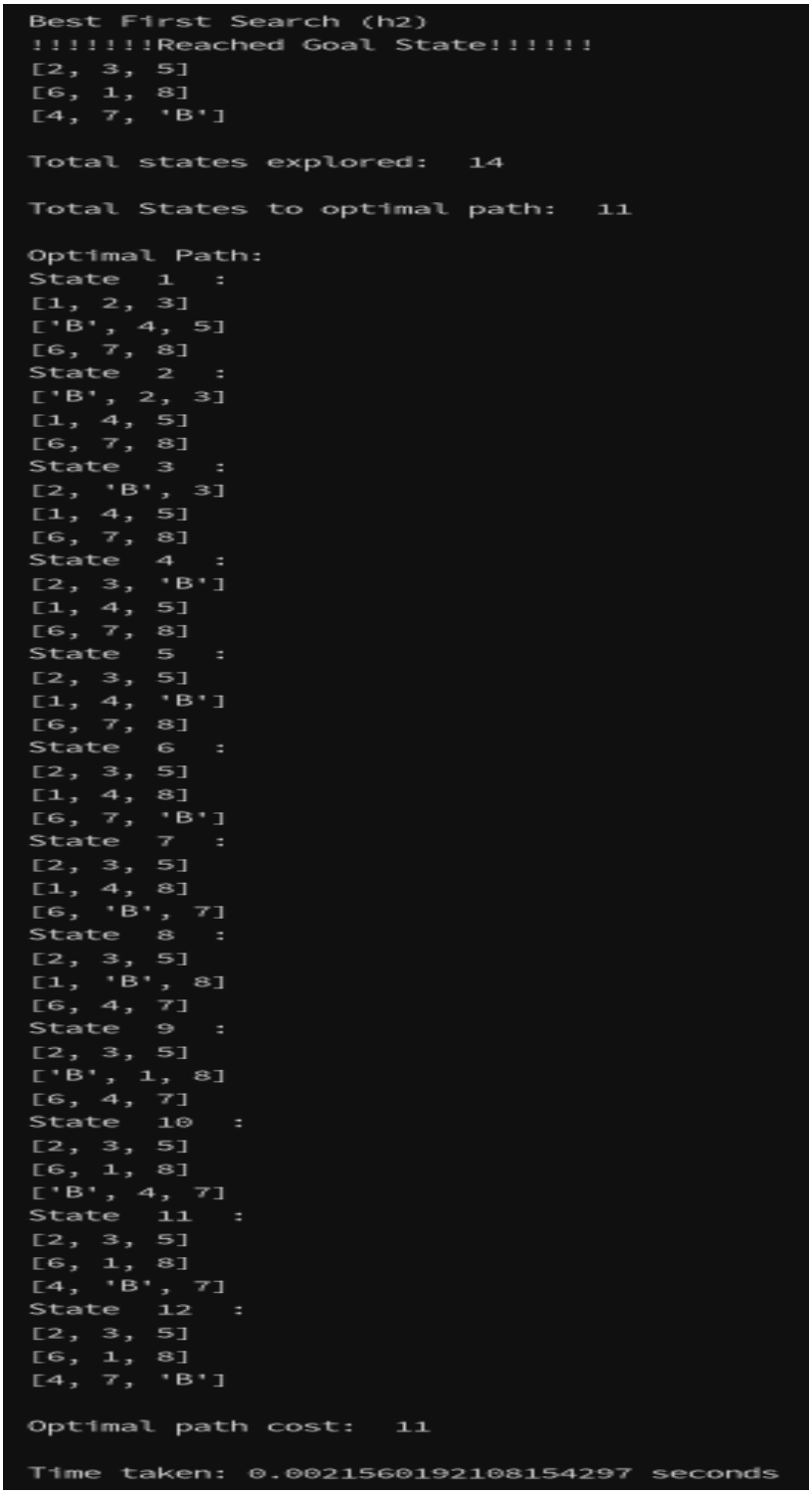


1) Using Best First Search with h1(misplaced tiles) heuristic for above example output is:

```
Best First Search (h1)
Failed :<
**State:**
[2, 3, 5]
[6, 1, 4]
['B', 7, 8]
Total states explored:  101


Time taken: 0.17032766342163086 seconds
```

2) Using Best First Search with h2(total manhattan distance) heuristic for above example output is:



3) Using A* Search with h1(misplaced tiles) heuristic for above example output is (Cost function is 1 for each state explored) :

```
A* Search (h1)
!!!!!!!Reached Goal State!!!!!!
[2, 3, 5]
[6, 1, 8]
[4, 7, 'B']

Total states explored:  84

Total States to optimal path:  11

Optimal Path:
State  1  :
[1, 2, 3]
['B', 4, 5]
[6, 7, 8]
State  2  :
['B', 2, 3]
[1, 4, 5]
[6, 7, 8]
State  3  :
[2, 'B', 3]
[1, 4, 5]
[6, 7, 8]
State  4  :
[2, 3, 'B']
[1, 4, 5]
[6, 7, 8]
State  5  :
[2, 3, 5]
[1, 4, 'B']
[6, 7, 8]
State  6  :
[2, 3, 5]
[1, 4, 8]
[6, 7, 'B']
State  7  :
[2, 3, 5]
[1, 4, 8]
[6, 'B', 7]
State  8  :
[2, 3, 5]
[1, 'B', 8]
[6, 4, 7]
State  9  :
[2, 3, 5]
['B', 1, 8]
[6, 4, 7]
State  10  :
[2, 3, 5]
[6, 1, 8]
['B', 4, 7]
State  11  :
[2, 3, 5]
[6, 1, 8]
[4, 'B', 7]
State  12  :
[2, 3, 5]
[6, 1, 8]
[4, 7, 'B']

Optimal path cost:  11

Time taken: 0.05627179145812988 seconds
```

4) Using A* Search with h2(total manhattan distance) heuristic for above example output is(Cost function is 1 for each state explored) :

```
 A* Search (h2)
!!!!!!!Reached Goal State!!!!!!
[2, 3, 5]
[6, 1, 8]
[4, 7, 'B']

Total states explored:  38

Total States to optimal path:  11

Optimal Path:
State  1  :
[1, 2, 3]
['B', 4, 5]
[6, 7, 8]
State  2  :
['B', 2, 3]
[1, 4, 5]
[6, 7, 8]
State  3  :
[2, 'B', 3]
[1, 4, 5]
[6, 7, 8]
State  4  :
[2, 3, 'B']
[1, 4, 5]
[6, 7, 8]
State  5  :
[2, 3, 5]
[1, 4, 'B']
[6, 7, 8]
State  6  :
[2, 3, 5]
[1, 4, 8]
[6, 7, 'B']
State  7  :
[2, 3, 5]
[1, 4, 8]
[6, 'B', 7]
State  8  :
[2, 3, 5]
[1, 'B', 8]
[6, 4, 7]
State  9  :
[2, 3, 5]
['B', 1, 8]
[6, 4, 7]
State  10  :
[2, 3, 5]
[6, 1, 8]
['B', 4, 7]
State  11  :
[2, 3, 5]
[6, 1, 8]
[4, 'B', 7]
State  12  :
[2, 3, 5]
[6, 1, 8]
[4, 7, 'B']

Optimal path cost:  11


Time taken: 0.034511327743530 27 seconds
```

3) Which Search is Best?

1) Optimality:

In terms of Optimality, **A* Search is the preferred** As it definitely provides us with solution.

In the above example, the best first search with h1 heuristic failed to provide a solution as it got stuck in a loop whereas A* search with h1 and h2 heuristic provided a solution.

We don't prefer the best first search in terms of optimality because it's not complete. It might get struck in an infinite loop. In A* Search, We add **Cost** to heuristic which prevents from going to infinite loop, thus optimal.

2) Time And Space Complexity:

In terms of Time and Space Complexity, **Best First Search is the preferred**.

We don't prefer the A* search because it makes sure that the same node is not repeated again thus preventing infinite loop and leads to increase in time and space complexity.

In the above example, We can see that **Best first search with h2 heuristic** has the lowest time complexity compared to all(**14 states explored and 0.021 sec)** where as A* search had (**h1**: 84 states explored, 0.056 sec) (**h2**: 38 states explored,0.034 sec)

4) Implement Hill Climbing Search Algorithm:

1) h1:

```
******* Hill climbing using H1 *******
1
*************Start************
heurestic:  5
[3, 1, 2]
[4, 5, 8]
[6, 7, 'B']
*************End**************
2
*************Start************
heurestic:  4
[3, 1, 2]
[4, 5, 'B']
[6, 7, 8]
*************End**************
3
*************Start************
heurestic:  3
[3, 1, 2]
[4, 'B', 5]
[6, 7, 8]
*************End**************
4
*************Start************
heurestic:  2
[3, 1, 2]
['B', 4, 5]
[6, 7, 8]
*************End**************
5
*************Start************
heurestic:  1
['B', 1, 2]
[3, 4, 5]
[6, 7, 8]
*************End**************


Hill Climb Algorithm (h1)
!!!!!!!Reached Goal State!!!!!!
['B', 1, 2]
[3, 4, 5]
[6, 7, 8]

Total states explored:  6

Total States to optimal path:  6

Optimal path cost:  6
```

2)h2:

```
******* Hill climbing using H2 *******
1
*************Start************
heurestic:  5
[3, 1, 2]
[4, 5, 8]
[6, 7, 'B']
*************End**************
2
*************Start************
heurestic:  4
[3, 1, 2]
[4, 5, 'B']
[6, 7, 8]
*************End**************
3
*************Start************
heurestic:  3
[3, 1, 2]
[4, 'B', 5]
[6, 7, 8]
*************End**************
4
*************Start************
heurestic:  2
[3, 1, 2]
['B', 4, 5]
[6, 7, 8]
*************End**************
5
*************Start************
heurestic:  1
['B', 1, 2]
[3, 4, 5]
[6, 7, 8]
*************End**************


Hill Climb Algorithm (h2)
!!!!!!!Reached Goal State!!!!!!
['B', 1, 2]
[3, 4, 5]
[6, 7, 8]

Total states explored:  6

Total States to optimal path:  6

Optimal path cost:  6
```

If the hill Climbing Algorithm gets struck at local optimum, there is a way to get out by using various techniques.

They are:

1)**Simulated annealing**: Simulated annealing is a variant of hill climbing that allows for occasional moves that are "downhill" in the search space. This technique can help the algorithm escape from local maxima and explore other regions of the solution space

2)**Gradient descent**: If the hill climbing algorithm is getting stuck on a local maximum, you can try using gradient descent to move downhill towards a better solution. This technique involves calculating the gradient of the objective function and updating the solution in the opposite direction of the gradient.

XXXXX THE END XXXXX