Heuristic Discussion

Mohamed Yassine EL Aatabi

Saad Driouech

Younes Bennani

Important note:

We managed the separation between what is specific to the search shell, the problem specific code, and the test file by using 3 python files. The first file is the search file, it contains the class “Node”, expand function, make queue function, queuing function, and general search function. The second file is the problem specific file, it contains a general abstract class problem and 3 subclasses that inherit from the problem class: ProblemMissionaries class, PegProblem, and Eight_PuzzleProblem. The last file is the testing file where we prompt the user to choose a puzzle and a search strategy.

Peg Solitaire

Manhattan Distance

Finding a heuristic for peg solitaire was somewhat challenging. We tried a panoply of heuristics but seldom were admissible. The heuristic we settled for at the beginning was the Manhattan distance. The way we implemented it for Pegs is the following:

- *Step 1*: Calculate the Manhattan distance from the first peg with respect to every other peg
- *Step 2*: Calculate the Manhattan distance of every peg with respect to the center
- *Step 3:* Average both values outputted by step 1 and step 2

The idea behind using the *Manhattan distance* is to check for clusters. The lower the distance, the less isolated clusters of pegs we have. Note that the more isolated peg clusters we have, the more likely the occurrence of a dead-end is. In addition to the cluster checking provided by step1, our heuristic checks the existence of a cluster around the center. Note that a cluster around the center will increase the probability of reaching the goal state.

Manhattan distance seems from a logical perspective a fairly reliable heuristic, however we decided against using it because it is not admissible. When we use *MD* with A* or Greedy best-first, we can clearly notice that it overestimates the distance to the goal state.

The *MD* did not perform well for Greedy best-first and A*. The algorithm expanded 20000 nodes without being able to find a solution.

Counting the number of Isolated pegs

Intuitively, counting the number of standalone isolated pegs is not the best heuristic. Arguably, *the Manhattan Distance* seems like a better heuristic. However, one of the perks of using the number of isolated pegs as a heuristic is the fact that it is admissible and it indeed never overestimates the distance to the goal-state. The heuristic works by checking whether a particular peg has neighbors. If he has no neighbors, then the counter of isolated pegs is incremented.

This heuristic has many downsides. The first downside is that it expands a gargantuan number of nodes, which makes it quite slow from a time complexity perspective. The reason is the fact that there are no isolated pegs at the beginning, the value of the heuristic is thereby 0.

As mentioned previously, using this heuristic was very slow. Therefore, we decided to stop the search after expanding a huge number of nodes. It is needless to point out the fact that this heuristic did not find a solution. Neither for A* nor greedy best first.

```
It seems that the search is taking a lot of time. If you want to continue press 1 otherwise press 0: 0
Total number of loops is:  0
Total number of nodes expanded is:  7000
Search has failed

Process finished with exit code 0
```

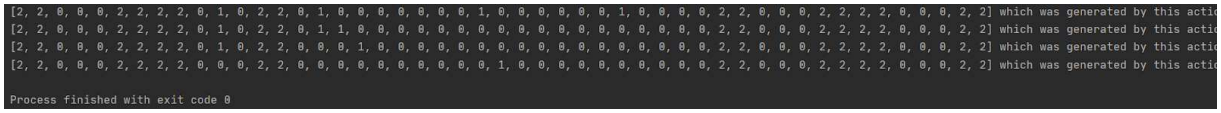
DFS

We would like to point out the fact, that it is easier to find a solution using a blind search more specifically the DFS algorithm. DFS was able to find an optimal solution, by expanding 1066 nodes. Therefore, we can confidently conclude that blind search is better suited for the peg solitaire puzzle.

```
Choose on of the following games:
1- Missionaries and Cannibals
2. Peg solitaire
3. 8-puzzle
Enter your choice: 2
The state of Peg Solitaire is represented using a list that has 49 cells. Each cell can be one of thefollowing 3 values: 0 1 and 2. 0 means that there is no peg and the correspondi
Please input the values of the initial state. Make sure to enter 49 values!
Enter the numbers seperated by space: 2 2 1 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 2 2 1 1 1 2 2
Please input the values of the goal state. Make sure to enter 49 values!
Enter the numbers seperated by space: 2 2 0 0 0 2 2 2 2 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 2 2 2 2 0 0 0 2 2
Please choose one of the following strategies (your choice must be the corresponding number in the menu below:
1. Breadth-First search
2. Depth-First search
3. Greedy Best-First search
4. A* search
Enter your choice: 2
```

```
Total number of loops is:  1252
Total number of nodes expanded is:  1066
Goal has been achieved
[2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2]
The path cost is:
31
The solution path is:
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2]
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 1, 0, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 2, 2, 1, 0, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2, 2, 1, 0, 0, 2, 2, 2, 2, 1, 1, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2, 2, 1, 0, 0, 2, 2, 2, 2, 0, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 2, 2, 1, 0, 0, 2, 2, 2, 2, 0, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 2, 2, 1, 0, 0, 2, 2, 2, 2, 0, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 1, 0, 1, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
```

```
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 0, 1, 2, 2, 2, 2, 1, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 0, 1, 1, 2, 2, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 1, 2, 2, 2, 2, 0, 1, 1, 2, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 0, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 1, 1, 0, 2, 2, 2, 2, 0, 1, 1, 2, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 0, 0, 1, 2, 2, 2, 2, 0, 1, 1, 2, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
[2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2] which was generated by this actic
```



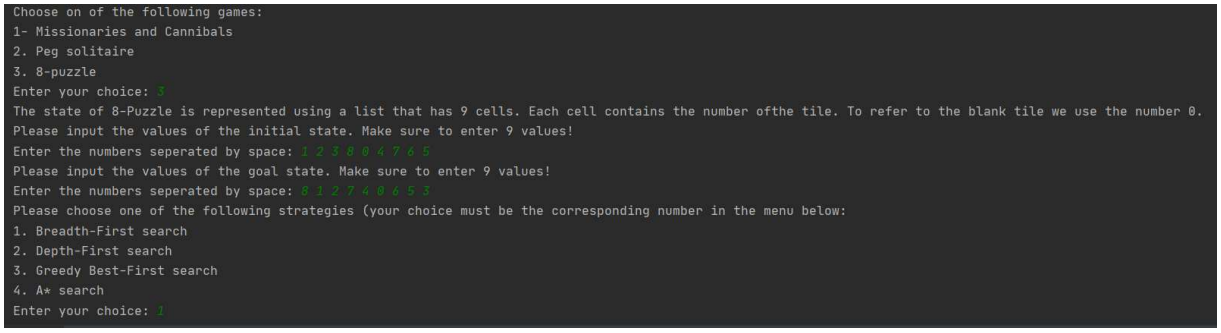
8-puzzle

Improved Manhattan

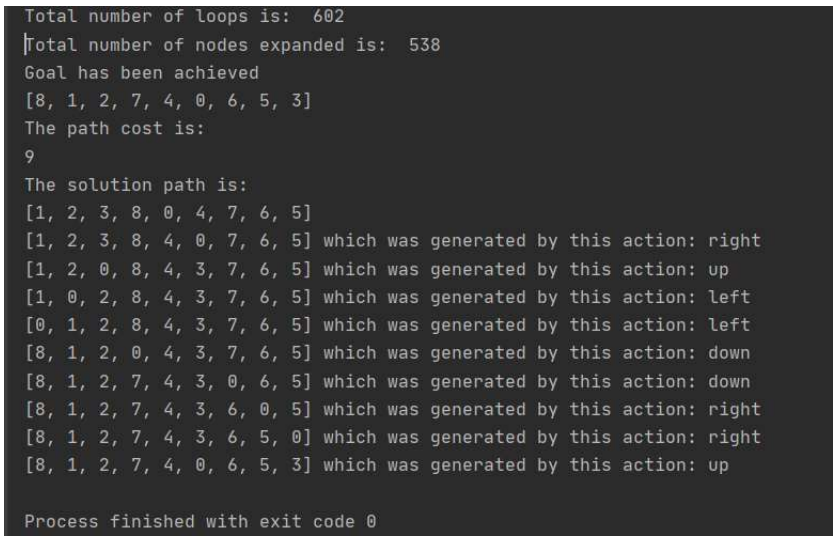
We decided to opt for a customized heuristic that improves the Manhattan Distance heuristic discussed in class. Our heuristic similarly to Manhattan Distance calculates the vertical and horizontal distances. However instead of summing them up, it takes *Max(Horizontal distance, Vertical distance)* and multiplies it by 2. We can therefore say that the improved Manhattan heuristic will always yield a value that is greater or equal to the value returned by the Manhattan distance while being not admissible*. Our heuristic is also not consistent, if we take an example where the goal state (n_2) is 2 steps far away from the initial state (n). Let n_1 be the parent of n_2. $h(n_2)$ will be equal to 4 and $h(n_1)$ will be equal to 2. It is clear that $h(n_2)$ is not less than $h(n_1) + c(n1,n2)$, therefore it is not consistent. The following is a proof of the dominance of our heuristic:

$$X + Y <= 2*Max(X,Y)$$

To test the heuristic, we opted for a goal-state at depth 9.



As a benchmark, the BFS algorithm expanded 538 nodes. It returned the optimal solution that has a path cost of 9.



As another benchmark, the DFS algorithm got stock into an infinite loop, so we decided to stop the algorithm after the expansion of 6000 nodes.

```
current generated node
[1, 4, 6, 8, 2, 0, 3, 5, 7]
current generated node
[1, 4, 6, 8, 2, 7, 3, 0, 5]
A loop has been detected
Number of loops is:  6420
node to expand
[1, 4, 6, 8, 2, 0, 3, 5, 7]
It seems that the search is taking a lot of time. If you want to continue press 1 otherwise press 0: 0
Total number of loops is:  6420
Total number of nodes expanded is:  6000
Search has failed

Process finished with exit code 0
```

Whereas for Greedy best first search, the algorithm expanded 9 nodes and returned an optimal solution that has a path cost of 9. We can therefore conclude that our heuristic is very reliable.

```
Total number of loops is:  8
Total number of nodes expanded is:  9
Goal has been achieved
[8, 1, 2, 7, 4, 0, 6, 5, 3]
The path cost is:
9
The solution path is:
[1, 2, 3, 8, 0, 4, 7, 6, 5]
[1, 2, 3, 8, 4, 0, 7, 6, 5] which was generated by this action: right
[1, 2, 0, 8, 4, 3, 7, 6, 5] which was generated by this action: up
[1, 0, 2, 8, 4, 3, 7, 6, 5] which was generated by this action: left
[0, 1, 2, 8, 4, 3, 7, 6, 5] which was generated by this action: left
[8, 1, 2, 0, 4, 3, 7, 6, 5] which was generated by this action: down
[8, 1, 2, 7, 4, 3, 0, 6, 5] which was generated by this action: down
[8, 1, 2, 7, 4, 3, 6, 0, 5] which was generated by this action: right
[8, 1, 2, 7, 4, 3, 6, 5, 0] which was generated by this action: right
[8, 1, 2, 7, 4, 0, 6, 5, 3] which was generated by this action: up

Process finished with exit code 0
```

We didn't encounter any challenges during the implementation nor during the utilization of the *Improved Manhattan*.

Missionaries and Cannibals

We settled for a heuristic that only takes into account the number of people (Missionaries + Cannibals) in the initial state. Given the constraint that suggests that missionaries should never be outnumbered by cannibals and that only two people can cross to the other side, with this is it brings to light that in order for the boat to return to the initial side of the river bank, one person will involuntarily have to return to the initial side of the river bank. Thereby, we can conclude that the heuristic function for this problem will be:

Heuristic = (Number of Missionaries* + Number of cannibals*) -1

*Number of Missionaries or Cannibals within the initial side

This heuristic is admissible, since the maximum number of people that can be carried by the boat is 2 and our heuristic function is a computation of the number of people. This function will always yield 1 except for the last trip where it is 2. As such, the heuristic function never overestimates the number of people that can be carried on the boat over the river.

This heuristic is also consistent in the way that from one trip to another the value number of people left in the initial side decreases by 1 or 0. In other words, me move from node n to node n_2 via node n_1, $h(n_2) = h(n_1) - 1$ or $h(n_2) = h(n_1)$. This means that $h(n_2) <= h(n_1)$ which implies that $h(n_2) <= h(n_1)+1$.

Our heuristic performs quite well. Similarly, to previous heuristic, we will benchmark greedy best first against DFS and BFS.

```
Choose on of the following games:
1- Missionaries and Cannibals
2. Peg solitaire
3. 8-puzzle
Enter your choice: 1
The state of the Missionaries and cannibals problem are represented using a list that has 3 cells.
The first and second values indicated the number of missionaries and cannibals respectively on the right side and the last value indicates where the boat is (1 if it is on the right
Please input the number of missionaries and the number of cannibals on the right side and a value torepresent where the boat is on the initial state:
Enter the numbers seperated by space: 3 3 1
Please input the number of missionaries and the number of cannibals on the right side and a value torepresent where the boat is on the goal state:
Enter the numbers seperated by space: 0 0 0
Please choose one of the following strategies (your choice must be the corresponding number in the menu below:
1. Breadth-First search
2. Depth-First search
3. Greedy Best-First search
4. A* search
Enter your choice: 1
```

The BFS algorithm was able to find an optimal solution (Total cost=11) by expanding 24 nodes.

```
Total number of loops is:  43
Total number of nodes expanded is:  24
Goal has been achieved
[0, 0, 0]
The path cost is:
11
The solution path is:
[3, 3, 1]
[3, 1, 0] which was generated by this action: [0, 2, 1]
[3, 2, 1] which was generated by this action: [0, 1, 1]
[3, 0, 0] which was generated by this action: [0, 2, 1]
[3, 1, 1] which was generated by this action: [0, 1, 1]
[1, 1, 0] which was generated by this action: [2, 0, 1]
[2, 2, 1] which was generated by this action: [1, 1, 1]
[0, 2, 0] which was generated by this action: [2, 0, 1]
[0, 3, 1] which was generated by this action: [0, 1, 1]
[0, 1, 0] which was generated by this action: [0, 2, 1]
[0, 2, 1] which was generated by this action: [0, 1, 1]
[0, 0, 0] which was generated by this action: [0, 2, 1]
```

The DFS algorithm was also able to find an optimal solution by expanding 13 nodes.

```
Total number of loops is:  13
Total number of nodes expanded is:  13
Goal has been achieved
[0, 0, 0]
The path cost is:
11
The solution path is:
[3, 3, 1]
[2, 2, 0] which was generated by this action: [1, 1, 1]
[3, 2, 1] which was generated by this action: [1, 0, 1]
[3, 0, 0] which was generated by this action: [0, 2, 1]
[3, 1, 1] which was generated by this action: [0, 1, 1]
[1, 1, 0] which was generated by this action: [2, 0, 1]
[2, 2, 1] which was generated by this action: [1, 1, 1]
[0, 2, 0] which was generated by this action: [2, 0, 1]
[0, 3, 1] which was generated by this action: [0, 1, 1]
[0, 1, 0] which was generated by this action: [0, 2, 1]
[1, 1, 1] which was generated by this action: [1, 0, 1]
[0, 0, 0] which was generated by this action: [1, 1, 1]
```

The greedy best first search algorithm was able to find an optimal solution by also expanding 13 nodes. We can conclude that our heuristic is a reliable heuristic.

```
Total number of loops is:  13
Total number of nodes expanded is:  13
Goal has been achieved
[0, 0, 0]
The path cost is:
11
The solution path is:
[3, 3, 1]
[3, 1, 0] which was generated by this action: [0, 2, 1]
[3, 2, 1] which was generated by this action: [0, 1, 1]
[3, 0, 0] which was generated by this action: [0, 2, 1]
[3, 1, 1] which was generated by this action: [0, 1, 1]
[1, 1, 0] which was generated by this action: [2, 0, 1]
[2, 2, 1] which was generated by this action: [1, 1, 1]
[0, 2, 0] which was generated by this action: [2, 0, 1]
[0, 3, 1] which was generated by this action: [0, 1, 1]
[0, 1, 0] which was generated by this action: [0, 2, 1]
[0, 2, 1] which was generated by this action: [0, 1, 1]
[0, 0, 0] which was generated by this action: [0, 2, 1]
```

We didn’t encounter any challenges during the implementation nor during the utilization of this heuristic.