LAB 1

Kumar Ayush - 2015eeb1060

Sumit Singh - 2015csb1036

Course Instructor - **Dr. Narayanan C. Krishnan**

Indian Institute of Technology, Ropar

1. Finding a Fixed Food Dot using Depth First Search

Method

Implemented using recursion.

Assessments and Measures

Tiny Maze:

- Path found with total cost of 8 in 0.0 seconds
- Search nodes expanded: 15

Medium Maze:

- Path found with total cost of 246 in 0.0 seconds
- Search nodes expanded: 269

Big Maze:

- Path found with total cost of 210 in 0.0 seconds
- Search nodes expanded: 466

Open Maze:

- Path found with total cost of 390 in 0.0 seconds
- Search nodes expanded: 683

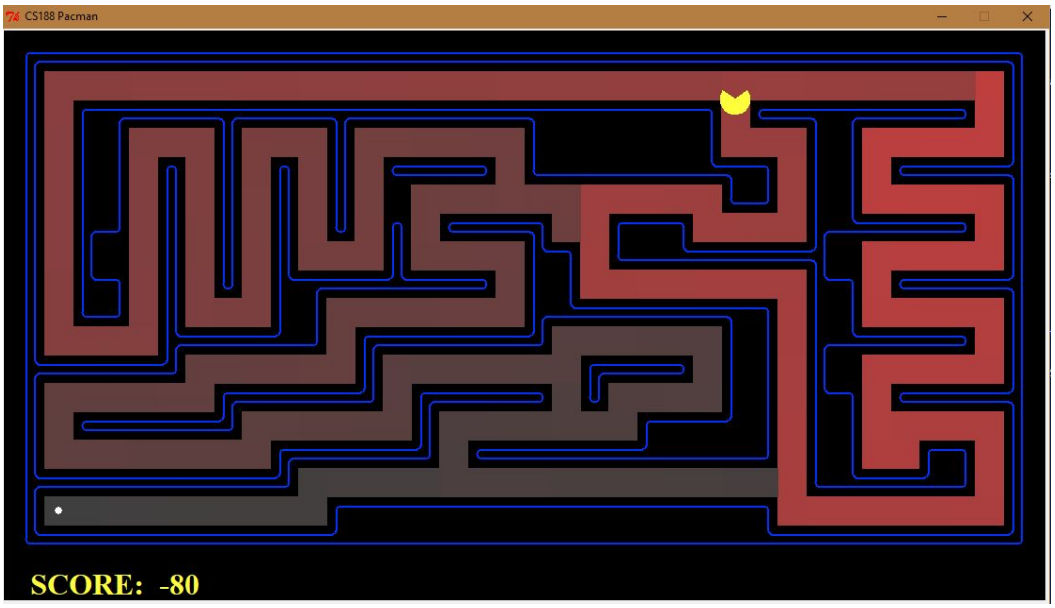

Fig 1: Medium maze using DFS

Observation

The exploration order is same as provided by *getSuccessor* function because recursion is used(no stacks were used). Pacman won't go to all the explored squares as the actions that the function returns are just backtracking the moves the agent took to reach the goal. The path, as already seen in Fig. 1, is not the least cost solution due to the obvious nature of DFS algorithm. It just goes to the first successor it receives while cycling the closest siblings until goal state is reached.

When implemented using stack, the children were explored in reverse order, though autograding script didn't give any marks.

2. Breadth First Search

Method

Implemented using Queue.

Assessments and Measures

Tiny Maze:

- Path found with total cost of 8 in 0.0 seconds
- Search nodes expanded: 15

Medium Maze:

- Path found with total cost of 68 in 0.0 seconds
- Search nodes expanded: 269

Big Maze:

- Path found with total cost of 210 in 0.0 seconds
- Search nodes expanded: 620

Open Maze:

- Path found with total cost of 54 in 0.0 seconds
- Search nodes expanded: 682

Eight Puzzle:

- BFS found a path of 13 moves: ['down', 'right', 'down', 'left', 'up', 'up', 'right', 'down', 'left', 'down', 'left', 'up', 'up']

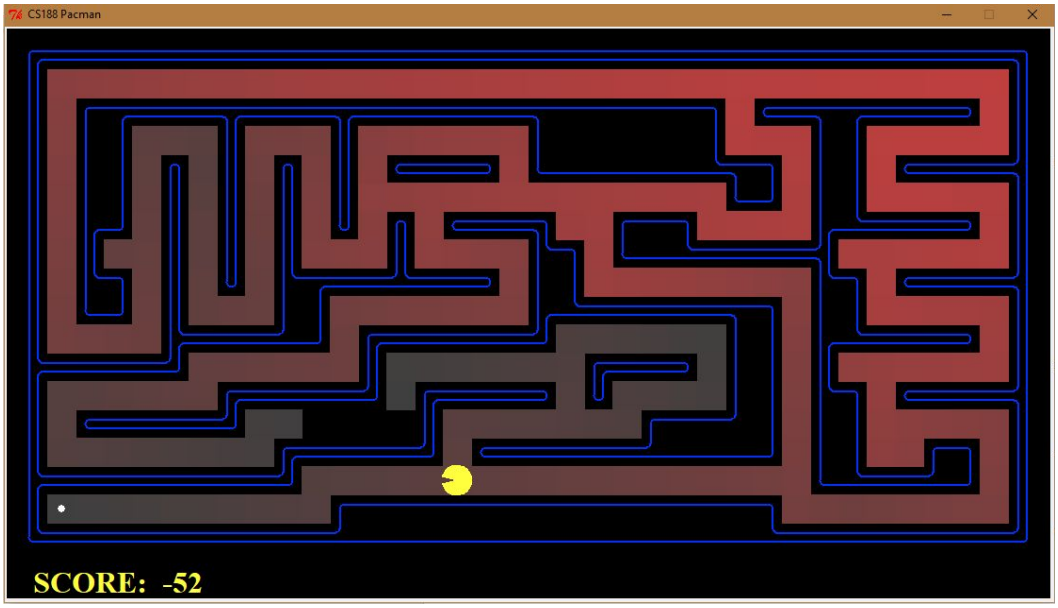

Fig 2: Medium maze using BFS

Observation

The order in queue is same as provided by *getSuccessor* function. In few cases we could see the search nodes that are expanded are quite more than when DFS was used. It's because few times the DFS algorithm, with better luck, could just expand the correct order of states to reach the goal. Even over a large number of cases, BFS would still not be clear winner against DFS because the exact path of goal state is unknown. It finds least cost solution because it always do level order traversal, where, closest nodes are explored first.

openMaze in BFS actually expands around same number of nodes that we got in DFS though BFS have fewer path cost.

3. Varying the Cost Function

Method

Implemented using PriorityQueue.

Assessments and Measures

Medium Maze:

- Path found with total cost of 68 in 0.0 seconds
- Search nodes expanded: 269

Big Maze:

- Path found with total cost of 210 in 0.0 seconds
- Search nodes expanded: 620

Medium Dotted Maze:

- Path found with total cost of 1 in 0.0 seconds
- Search nodes expanded: 186

Medium Scary Maze:

- Path found with total cost of 68719479864 in 0.0 seconds
- Search nodes expanded: 108

Open Maze:

- Path found with total cost of 54 in 0.0 seconds
- Search nodes expanded: 682

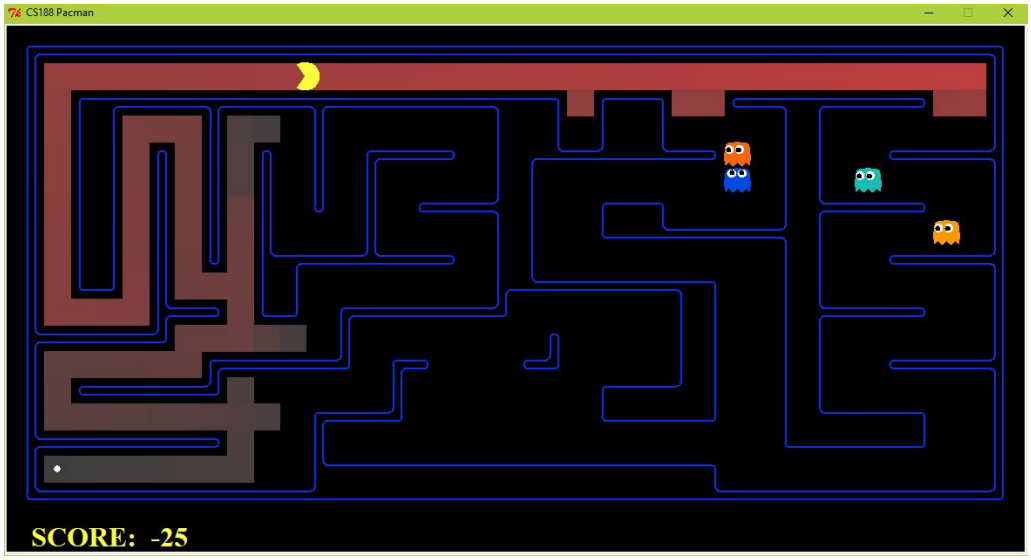

Fig 3: Medium Scary maze using UCS

Observation

Though in Fig. 3 the agent reaches the goal without encountering any ghosts, it's actually taking a predetermined path. The algorithm doesn't take actual ghosts into consideration but just the cost it takes to get to those areas(cost is higher in ghost-ridden areas). We could also see the cost in *mediumScaryMaze* is actually quite high. It's because our agent initially starts in ghost-ridden area and have to take even longer route than it would have taken without the ghosts.

Since the path cost is actually same as in BFS, UCS fair same in *openMaze* as the former. We'll later see the advantage in applying heuristics in *openMaze*.

4. A* Search

Method

Implemented by adding Manhattan Heuristics in UCS.

Assessments and Measures

Big Maze:

- Path found with total cost of 210 in 0.0 seconds
- Search nodes expanded: 549

Open Maze:

- Path found with total cost of 54 in 0.0 seconds
- Search nodes expanded: 535

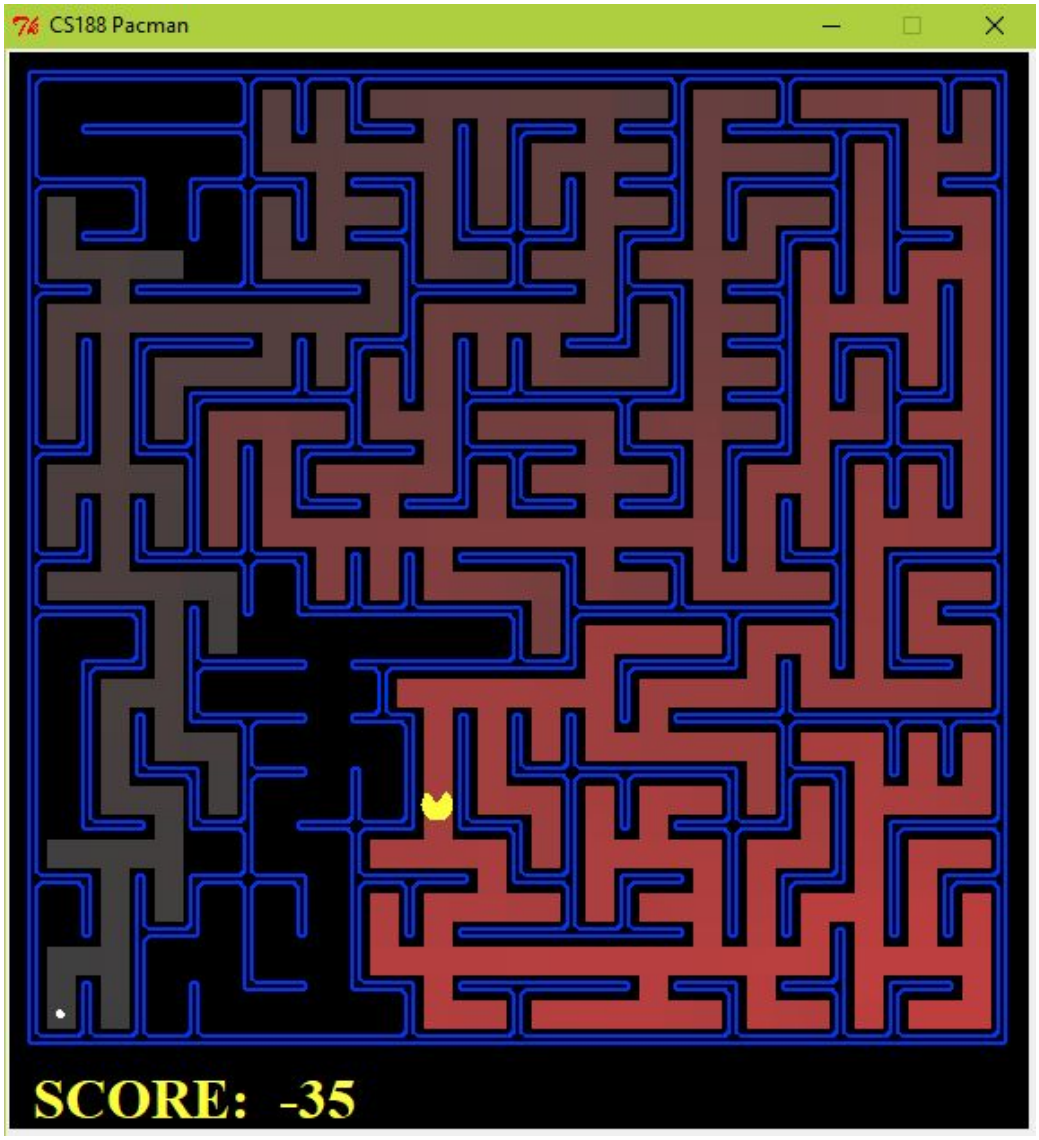

Fig 4: Big maze using A* (Manhattan heuristics)

Observation

We could see actually substantial improvement in just having a simple heuristic in place. About 20% less node are expanded in both *bigMaze* and *openMaze* than UCS. For more complex goals, even bigger improvements could be observed. In

openMaze, A* performed better than in UCS with respect to number of nodes expanded*.*

5. Finding All the Corners

Method

We changed the state format from *state* to *[state, cornerArray]*, where *cornerArray* is of the format *[0, 0, 1, 0]*, each index denoting whether the respective corner have been explored or not(*1* if explored, *0* if not). The goal state would be reached when all the values in corner array will become *1. getSuccessor* function is same as in position search problem.

Assessments and Measures

Tiny Corners:

- Path found with total cost of 28 in 0.0 seconds
- Search nodes expanded: 252

Medium Corners:

- Path found with total cost of 106 in 0.0 seconds
- Search nodes expanded: 1966

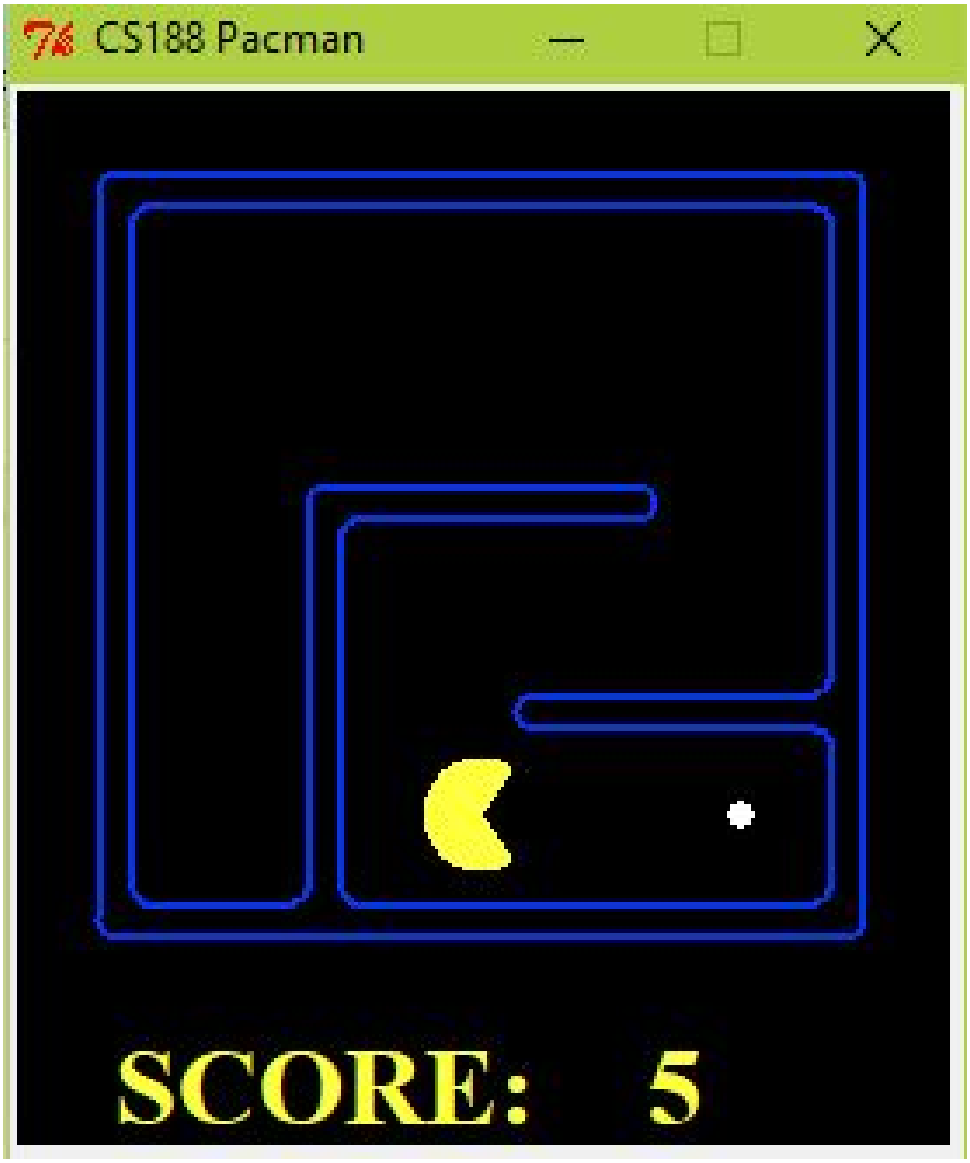

Fig 5: Tiny corners using BFS

Observation

Without heuristics, it's taking 1966 nodes in *mediumCorners* to get to the goal state. UCS would be the same because cost of single step is 1. We'll later see the real advantage of applying proper heuristics.

6. Corners Problem: Heuristic

Method

Since there are only four dots that we have to reach, the heuristic applied is just finding the permutation having minimum cost of reaching remaining corners from any given state. It's implemented using manhattan distance. The time complexity for finding the heuristic for single state in *O(4!)*, which is a constant.

Assessments and Measures

Tiny Corners:

- Path found with total cost of 28 in 0.0 seconds
- Search nodes expanded: 159

Medium Corners:

- Path found with total cost of 106 in 0.0 seconds
- Search nodes expanded: 741

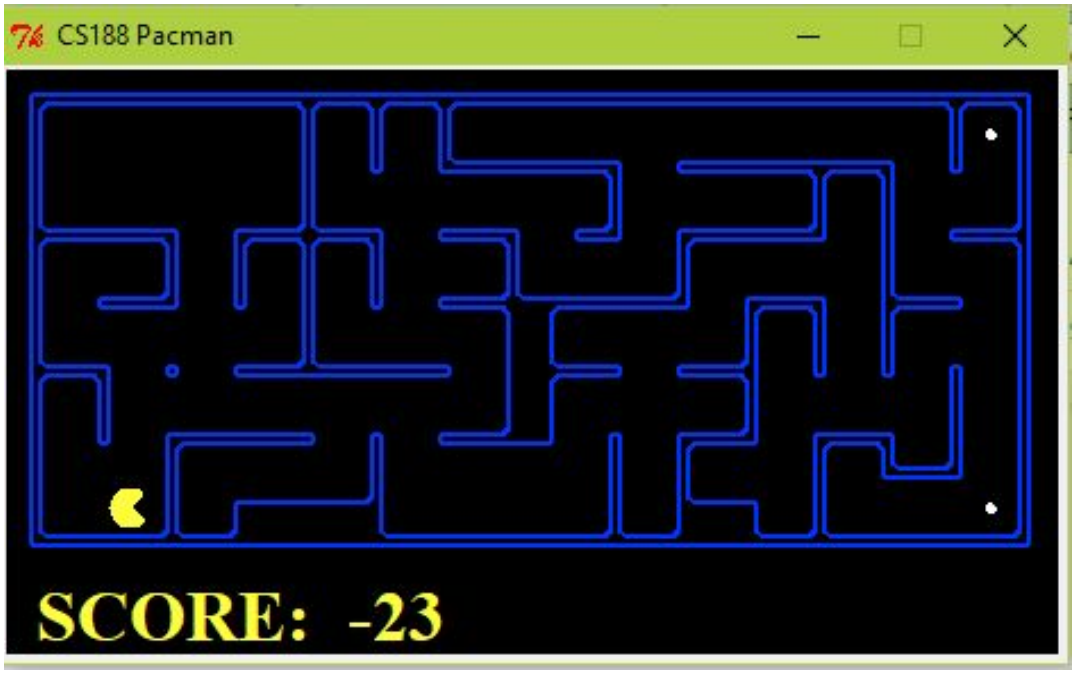

Fig 6: Medium corners using corner heuristics

Observation

The number of expanded nodes in *tinyCorners* is cut in half, while, in *mediumCorners* it's almost reduced to 3 times! The heuristic is admissible because it returns the absolute minimum for reaching the goal state for any given state.

This heuristic won't work in optimal time for a large number of dots because the time complexity for a single state heuristic is *O(factorial of dots)*, which increases exponentially. Next, we'll see another kind of heuristic that will work on any number of dots(It's not as optimal as this one though).

7. Eating All The Dots

Method

Since the permutation approach won't work anymore, we've to use faster heuristic.

Assessments and Measures

Tricky Search:

- Path found with total cost of 60 in 5.4 seconds
- Search nodes expanded: 7775

Medium Search:

- [No solution in optimal time.]

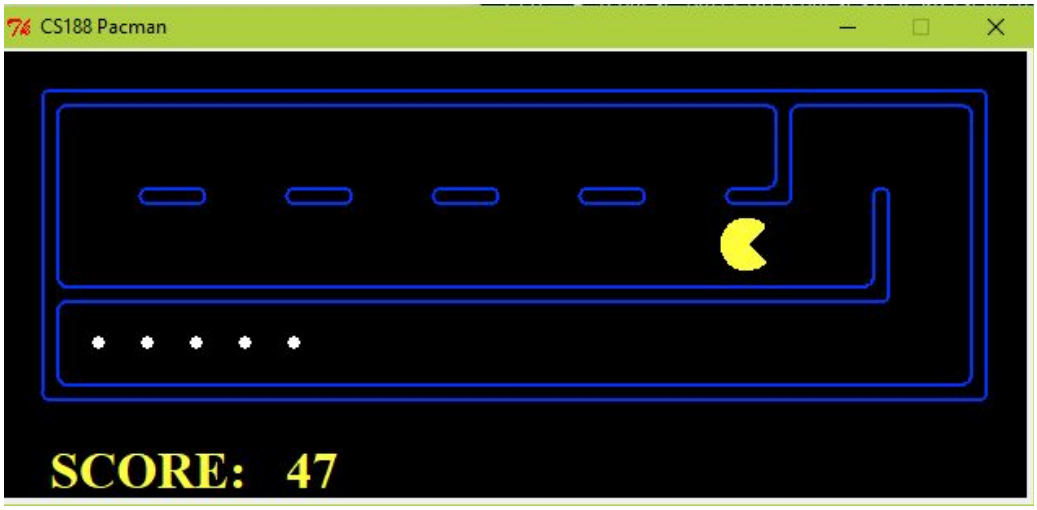

Fig 7: Tricky search using food heuristic.

Observation

The explored nodes was cut in half compared to BFS. Also, no solution for *mediumSearch* was found using same heuristic though we waited a long time(>10 minutes). Few points to note:

- In function *getClosestDot*, when comparing *currentMin* and *d, the* equality sign in ">=" is important. Without it, consistency would fail.
- In *foodHeuristic,* we subtracted *currentMin* from *totalDist* to have a lower bound so as not to fail admissibility test. Decrease *currentMin* (best with *alpha = 0.3*)would make the agent more optimal but it will tend towards inadmissibility while increasing it would just make it slower and will eventually become inconsistent.

We tried two other heuristics but both failed in some area. There's a heuristic that uses greedy approach to calculate summation of remaining foods by finding closest food to current state. But, though optimal, it was inconsistent. We also tried using minimum spanning tree heuristic using kruskal. It didn't work because it didn't took into account the number of times backtracking was needed to reach the goal state, thus, making it inconsistent.

8. Suboptimal Search

Method

Implemented using BFS.

Assessments and Measures

Tricky Search:

- Path found with cost 68.
- Search nodes expanded: 8460

Medium Search:

- Path found with cost 171

Big Search:

- Path found with cost 350.

Fig 8: Big Search using Suboptimal Search

Observation

As apparent in Fig. 8, suboptimal search takes substantially less time but failed miserably in total cost it took to reach the goal state. Main problem occurs when there are multiple shortest path routes from current state. When agent decides to go on to one state, it keeps expanding further from that state and had to come back at a later stage, thus, increasing cost unnecessarily . Best way is to visualise how a DFS algorithm might work, where children of current state are the closest dots to it.

Conclusion:

When designing heuristics, there's always a tradeoff in faster and admissible solution. Relaxing minimum number of conditions give more admissible and consistent but slower solution and vice versa. There could always be an absolute h(n) and minimum possible nodes would be expanded but that would be substantially slow.