Homework

Mariam Vardishvili - mariamv2 (4)
Sidhartha Satapathy - ss46 (4)
Shunping Xie - sxie12 (4)

October 3, 2017

1 Maze

We solve the first part using the following search routines: BFS, DFS, Greedy best first and A*.

1.1 Single Pellet

For part 1.1, we use Manhattan distance for greedy best first and A* (as the assignment specified). The code ran in under a second for every algorithm and every input. We see that bfs and A* is optimal and that A* expands fewer nodes than bfs.

1.1.1 Medium Maze

BFS cost: 94 and Expanded nodes: 608

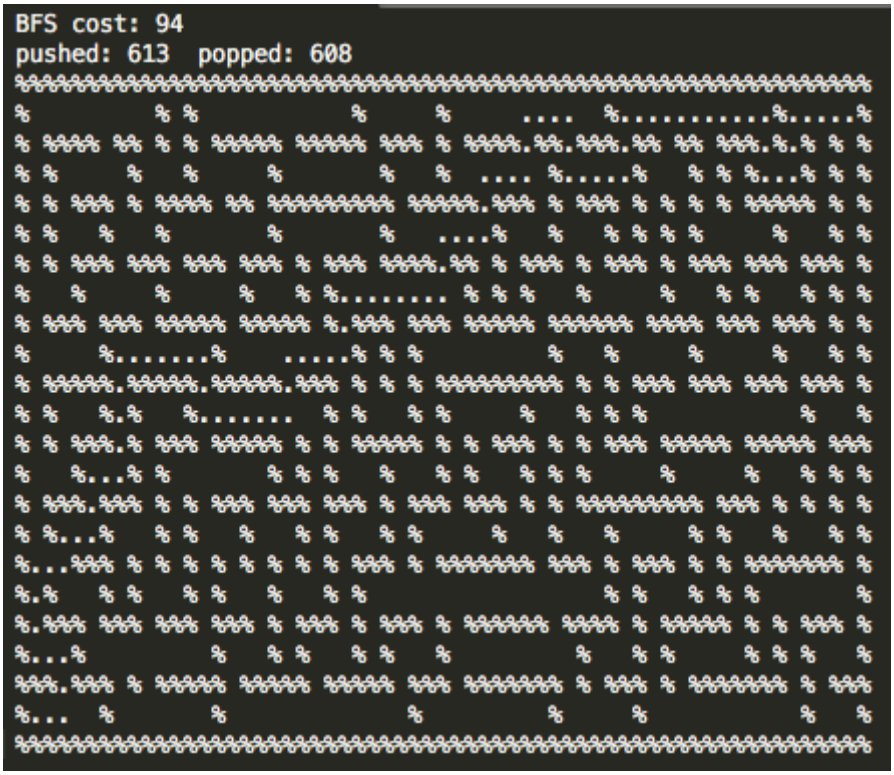

Figure 1: BFS

DFS cost: 116 and Expanded nodes: 164

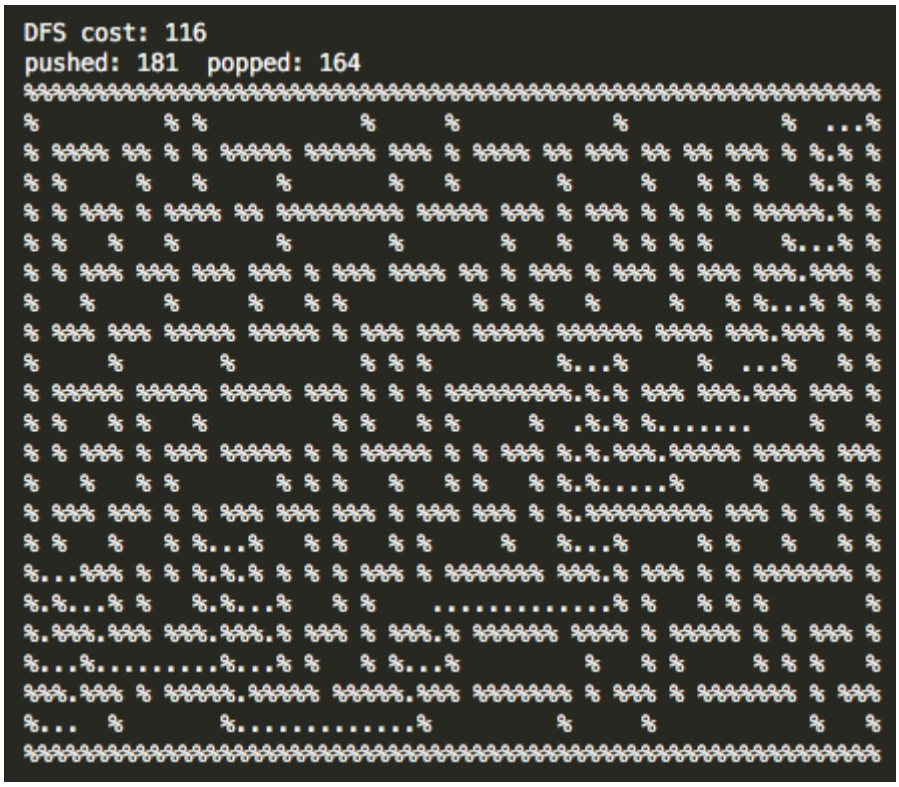

Figure 2: DFS

Greedy best first cost: 114 and Expanded nodes: 133

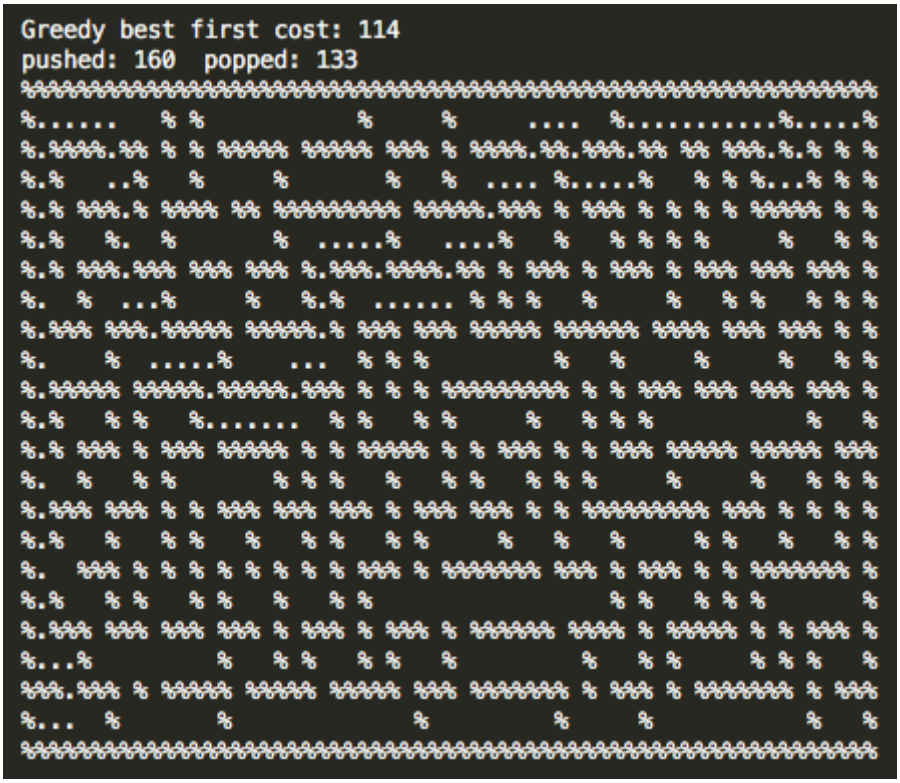

Figure 3: Greedy best first

A* cost: 94 and Expanded nodes: 301

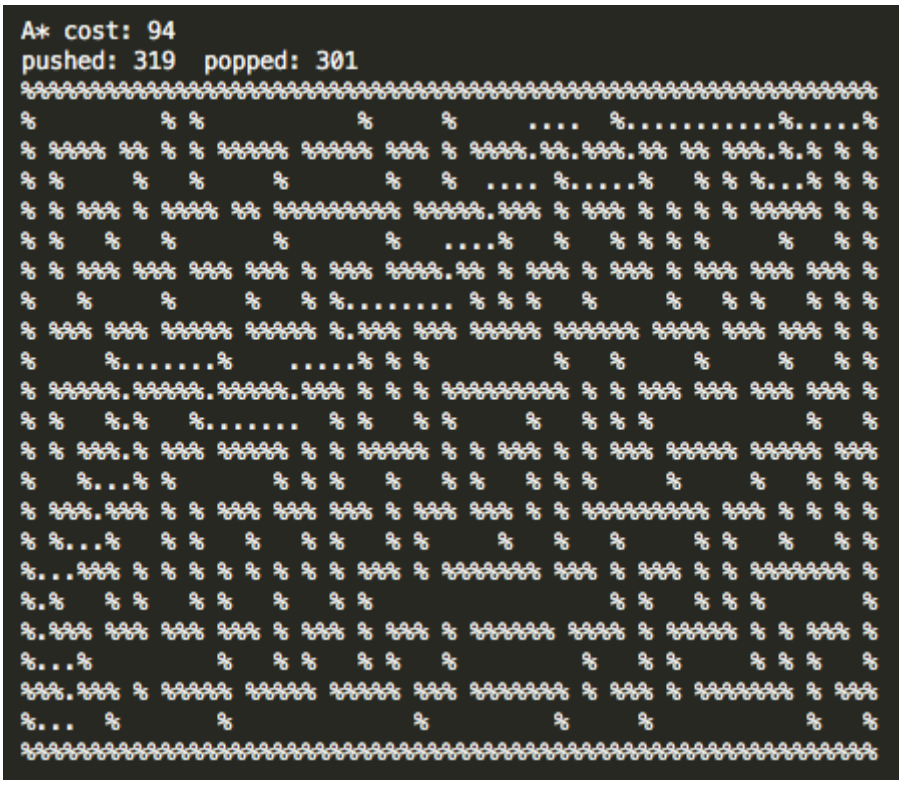

Figure 4: A*

1.1.2 Big Maze

BFS cost: 148 and Expanded nodes: 1255

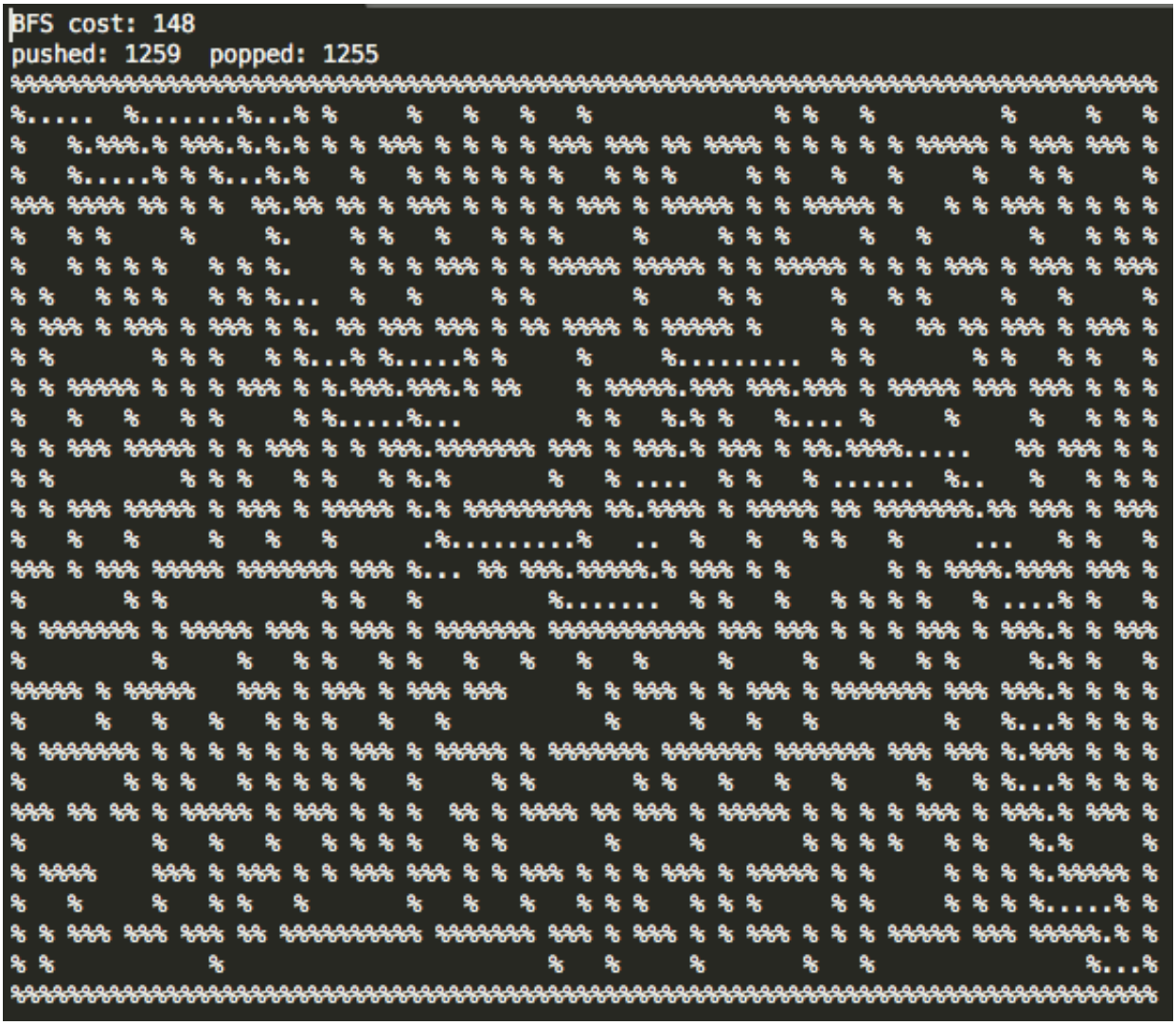

Figure 5: BFS

DFS cost: 442 and Expanded nodes: 830

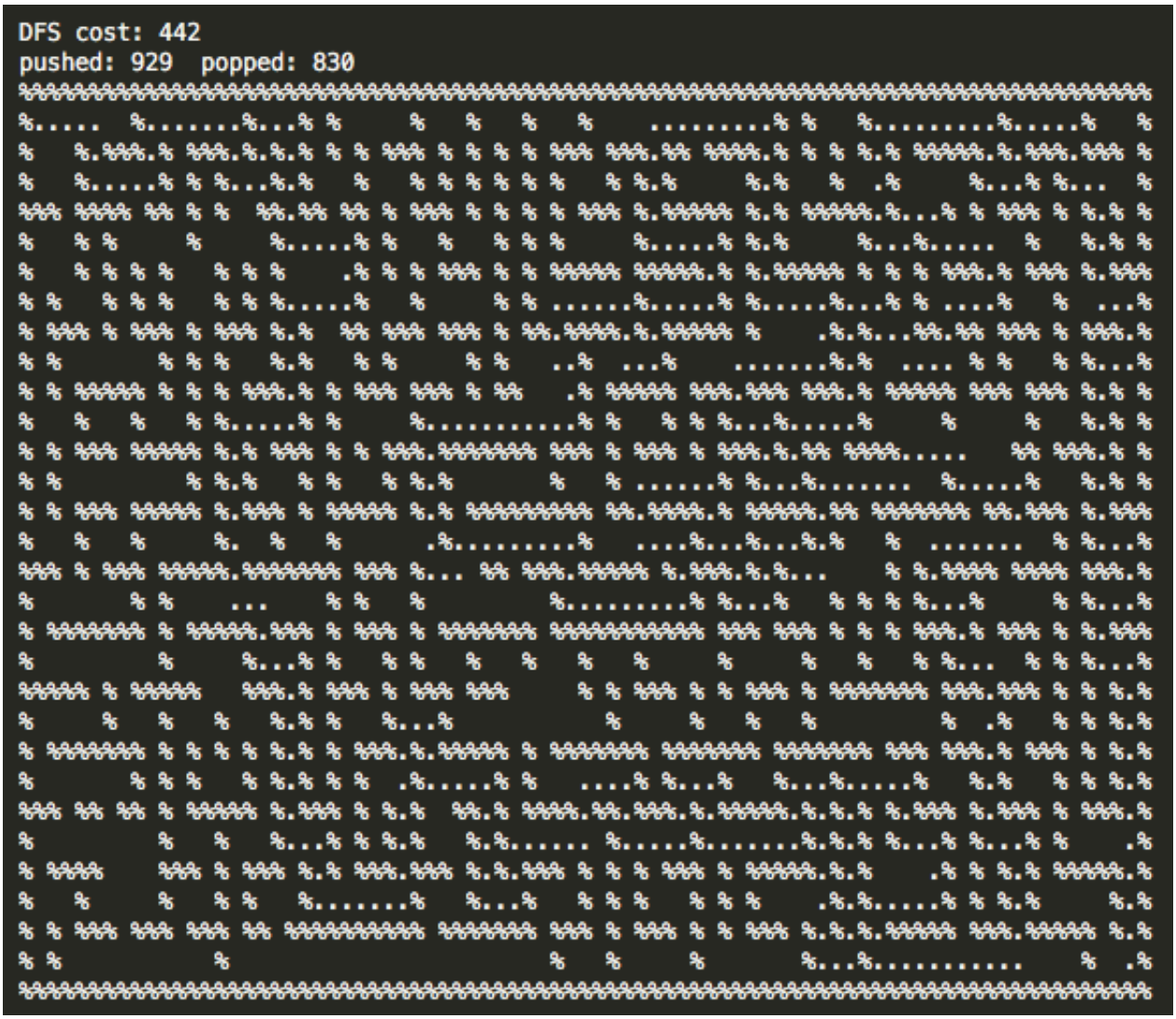

Figure 6: DFS

Greedy best first cost: 222 and Expanded nodes: 277

Figure 7: Greedy best first

A* cost: 148 and Expanded nodes: 1112

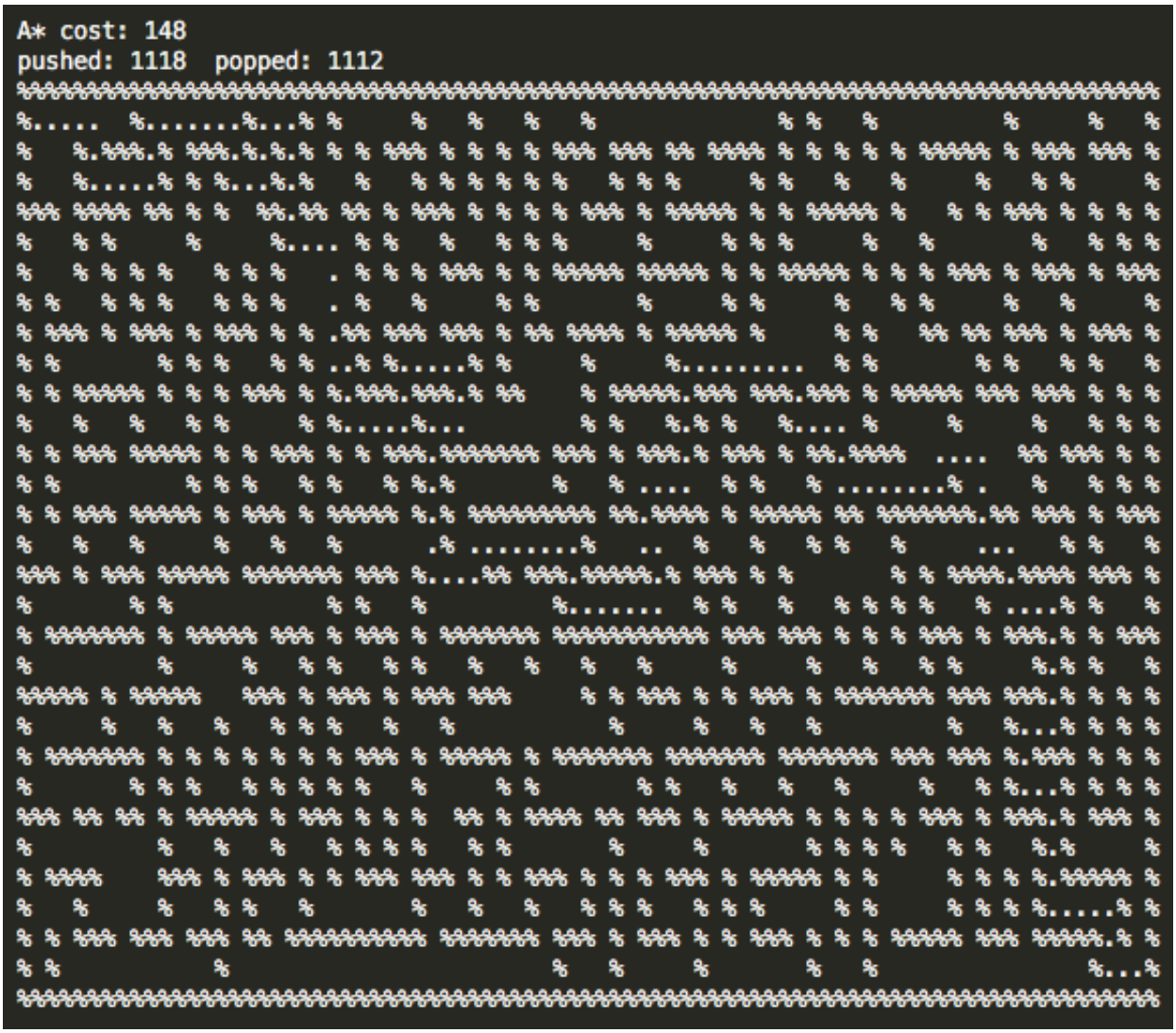

Figure 8: A*

1.1.3 Open Maze

BFS cost: 45 and Expanded nodes: 523

Figure 9: BFS

DFS cost: 125 and Expanded nodes: 273

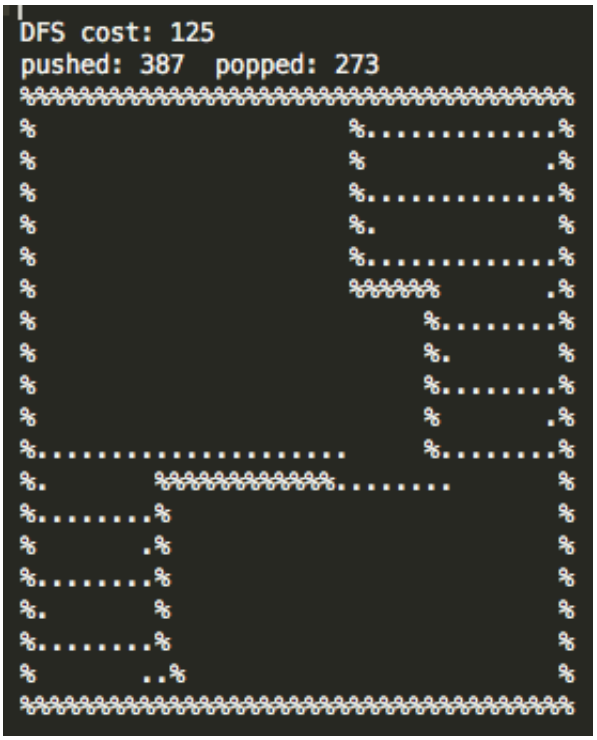

Figure 10: DFS

Greedy best first cost: 45 and Expanded nodes: 148

Figure 11: Greedy best first

A* cost: 45 and Expanded nodes: 236

Figure 12: A*

1.2 Multiple Pellet

A* for multiple dots. Observe that if we go from one dot to another, we take the shortest path and we only considered the order of the dots we visit. Therefore, we only consider the complete graph whose vertices are the dots and edges between vertices are the shortest distance between the dots. We use a bfs to compute the all pairs shortest path distance. We number the dots from 1 to the number of dots. The state is the number of our current dot and a bitmask where 1 means we visited that corresponding dot and 0 means we havent. There are at most 20 (21 including start) dots so we can use a simple array to keep track of the states because it isnt a lot of memory.

The heuristic we used was we go to the closest unvisited dot. This is admissible because it is obviously less than the actual distance required to visit the remaining dots from our state. The distance to one dot cannot be more than the distance to all the remaining dots.

Note because of our graph, the number of expanded nodes might be smaller than the correct number of expanded nodes. The observation above made the code run faster. The tiny maze ran in about a second, the small maze ran in about five seconds, and the medium maze ran in about thirty seconds. Note with our observation that the problem reduces to a modified TSP problem. We want to figure out the order of the dots we visit. Once we do, we take the shortest path there.

We used letters to denote our path. Lowercase a is where we start (the position of P in the maze).

1.2.1 Tiny maze

Distance cost: 36
Nodes expanded: 23417 (note this may be less than what you have because we preprocessed the graph with our observation above)

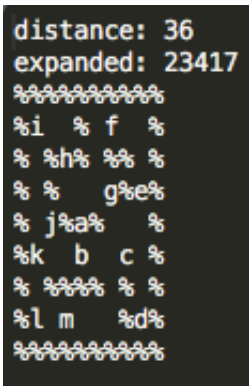

Figure 13: A* for tiny maze

1.2.2 Small maze

Distance cost: 143
Nodes expanded: 245328

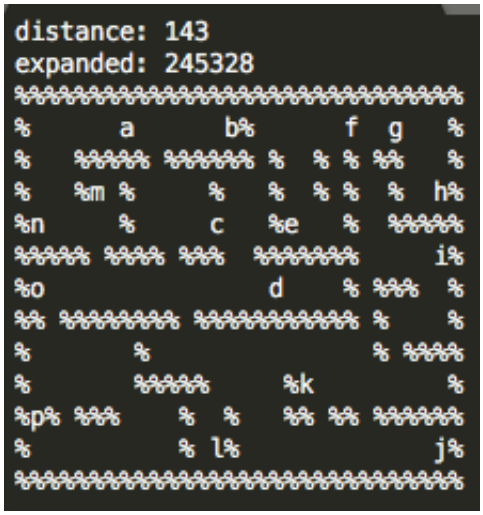

Figure 14: A* for small maze

1.2.3 Medium maze

Distance node: 207
Nodes expanded 10338033

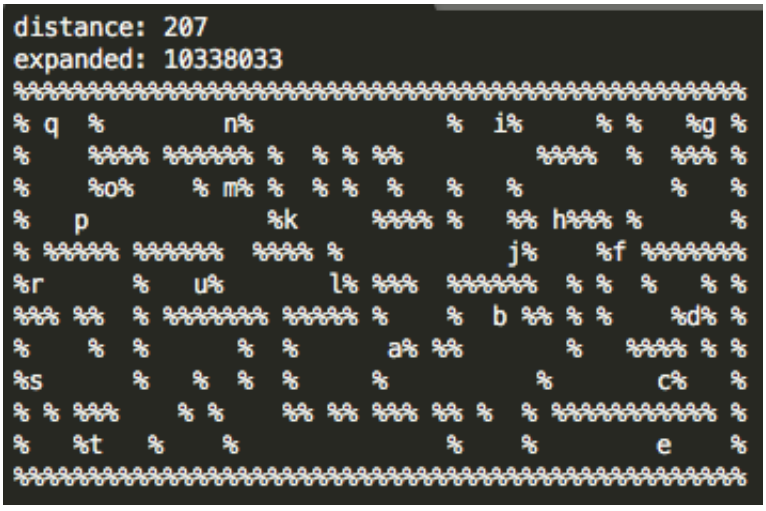

Figure 15: A* for medium maze

2 Sokoban

For the purpose of this problem our state representation is the location of the boxes and the agent. The heuristic we use for this problem is the sum total of the manhattan distance of all the boxes to their closest possible goals. In addition to using this heuristic, we also use deadlock recognition in order to expand fewer states. Observe that if we can push a box to a certain goal, we can also reverse the push action to a pull action and figure out all the possible locations from where a box can reach the specific goal location. As part of our calculation, we pre-compute all the locations from where it is not possible to push a box to one of the goal locations and whenever we add a state to the frontier we check if this leads to a deadlock position and if it does, then we dont push it to the frontier. Furthermore, we use a priority queue that can update the priority value for a given state if the new priority is lesser than the previous priority for the state, this leads to removing some of the unnecessary states that might have been added to the priority queue as the priority queue only consists of one entry for every state, particularly the one with the lowest priority value.

In addition to the A* (which is one of our approach) we also do a simple bfs (which is our other approach)

Admissibility of the heuristic: So the heuristic is admissible as the total distance for the agent to travel is more than or equal to pushing the box to the closest goal. Here, we do not take into account the distance agent has to travel to reach one of the boxes. Furthermore, all the boxes might not be pushed to their closest locations as two of the boxes might have the same goal location as their closest location to any of the goals in which case both cant be pushed to that goal. So it is definitely lesser than the total distance to be travelled.

Extra Credit: In addition to adding the solution below we have added all the animations to the animations folder. Please go and have a look at the animations for all the solutions.

2.1 Input 1

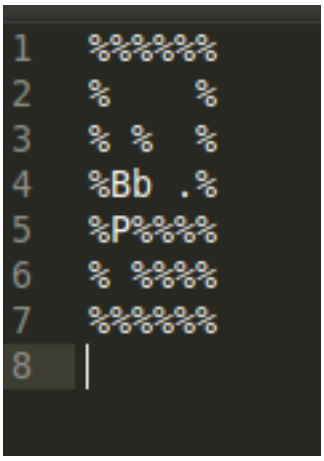

Figure 16: Input 1

Output

Approach 1: (BFS)

The number of moves needed to reach a goal configuration: 8
Nodes expanded: 35
Running time: 0.000244 seconds

Approach 2: (A* with the above heuristic)

The number of moves needed to reach a goal configuration: 8
Nodes expanded: 40
Running time: 0.000866 seconds

2.2 Input 2

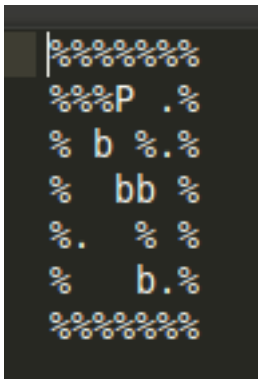

Figure 17: Input 2

Output

Approach 1: (BFS)

The number of moves needed to reach a goal configuration: 144
Nodes expanded: 44544
Running time: 0.219686 seconds

Approach 2: (A* with the above heuristic)

The number of moves needed to reach a goal configuration: 144
Nodes expanded: 761536
Running time: 33.042084 seconds

2.3 Input 3

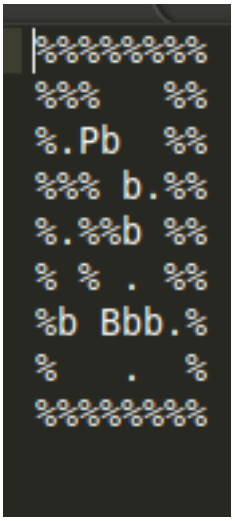

Figure 18: Input 3

Output

Approach 1: (BFS)

The number of moves needed to reach a goal configuration: 34
Nodes expanded: 396413
Running time: 2.925223 seconds

Approach 2: (A* with the above heuristic)

The number of moves needed to reach a goal configuration: 34
Nodes expanded: 35147
Running time: 2.473289 seconds

2.4 Input 4

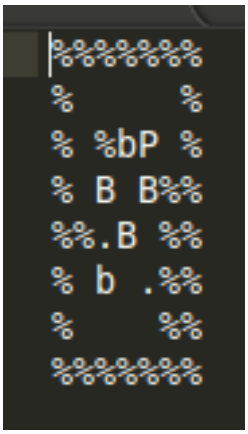

Figure 19: Input 4

Output

Approach 1-(BFS)

The number of moves needed to reach a goal configuration: 72
Nodes expanded: 564091
Running time: 3.518417 seconds

Approach 2: (A* with the above heuristic)

The number of moves needed to reach a goal configuration: 72
Nodes expanded: 86730
Running time: 4.243764 seconds

3 Statement of individual contribution

Mariam Vardishvili : Development of admissible functions for all of the problems. Involved in the code development process. Program for creating animation for sokoban using output from solution.

Sidhartha Satapathy : I wrote the code for both the techniques bfs and A* for part 2 Sokoban and wrote this part of the report. In addition to writing the code , I was also involved in the development of the state representation for all the parts of the problem. Furthermore, I was involved in the development of an admissible heuristic for all parts of the problem including 1.2 and 2. Finally, I figured out the deadlock recognition technique in order to expand fewer nodes for sokoban.

Shunping Xie : I wrote the code for parts 1.1 and 1.2. I also wrote part of the corresponding report here.