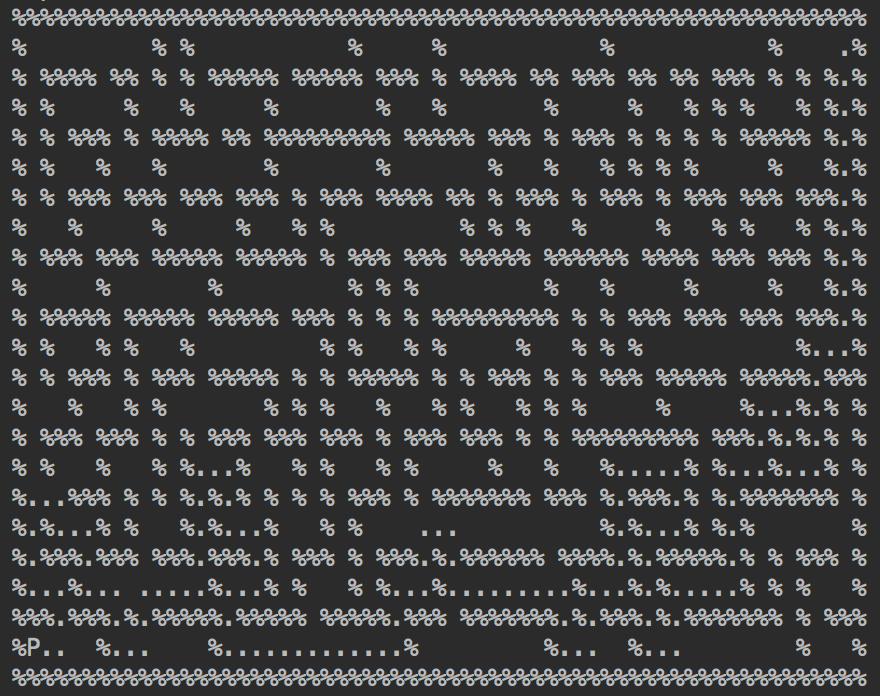
**Assignment 1 – Report**

Group members: Caiwei He(caiweih2), Chengrui Zhu, Kelong Wu

**Part 1.1**

1. DFS

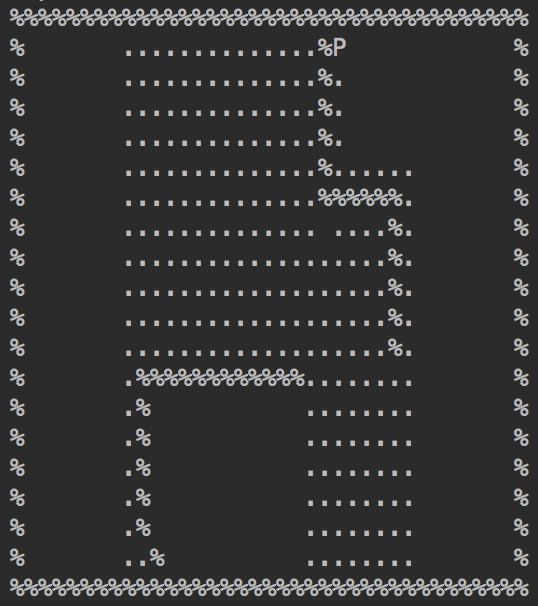
Medium maze:



Big maze:

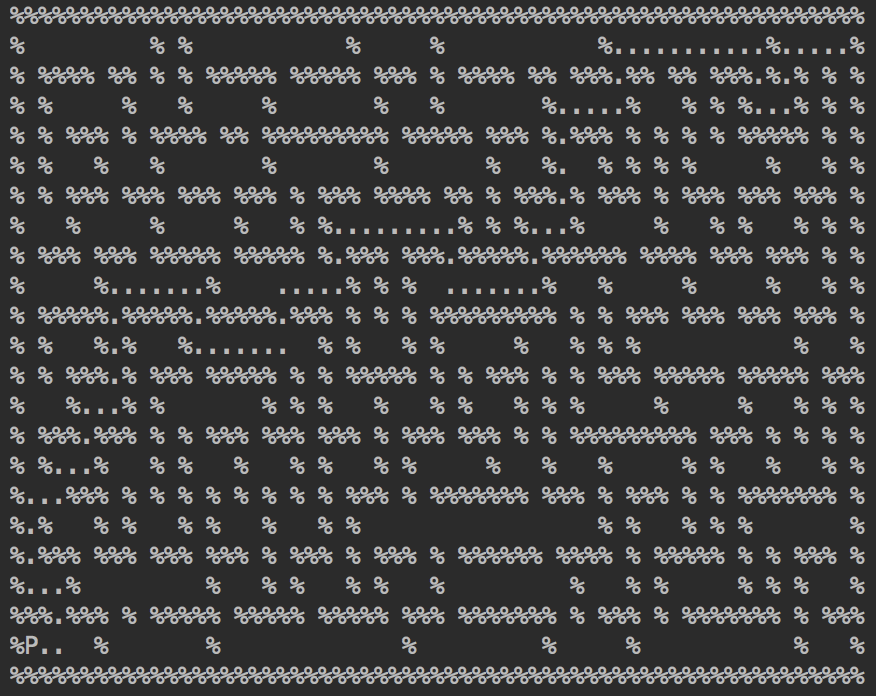


Open maze:



1. BFS

Medium maze:



Big maze:

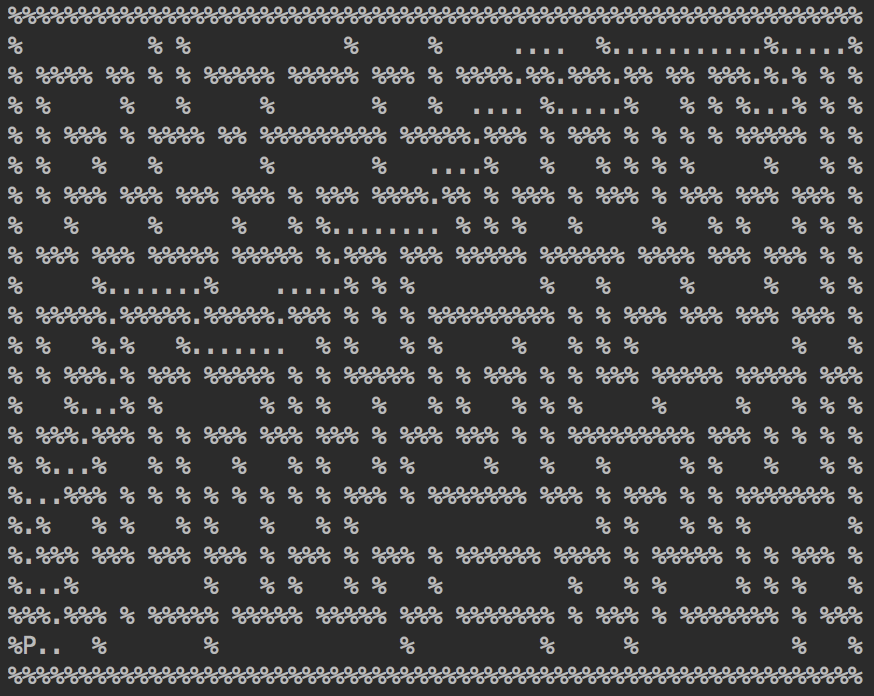


Open maze:



1. Greedy BFS

Medium maze:



Big maze:

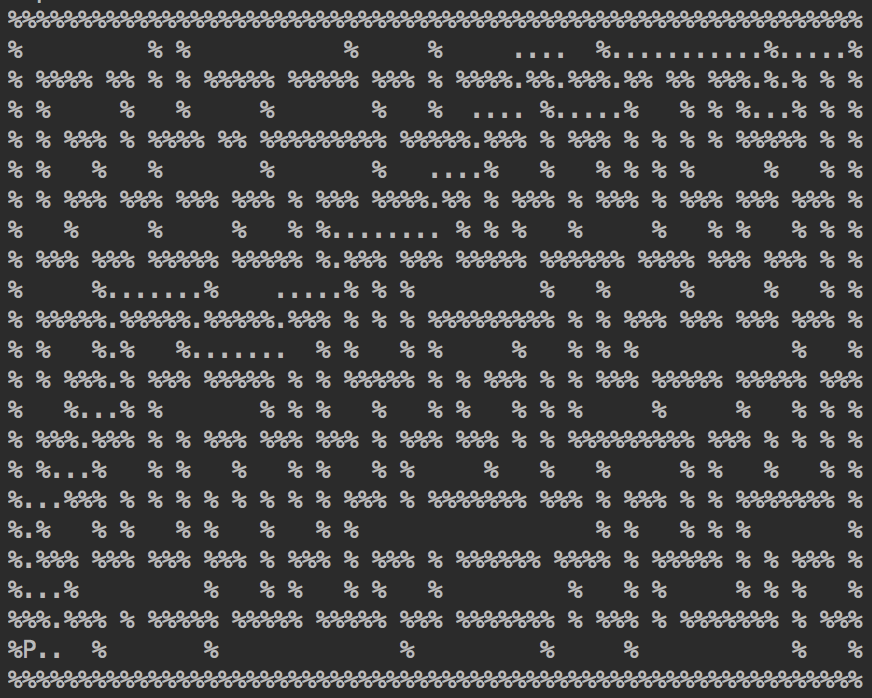


Open maze:



1. A\*

Medium maze:



Big maze:



Open maze:



Path cost & Expanded nodes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (cost/expanded) | DFS | BFS | Greedy BFS | A\* |
| Medium | 137/186 | 94/614 | 94/102 | 94/334 |
| Big | 239/255 | 148/1258 | 234/290 | 148/1112 |
| Open | 258/404 | 45/538 | 57/154 | 45/237 |

Analysis:

In all, A\* performs the best, then BFS, Greedy, and DFS performs the poorest. A\* always focuses on the best solution with regard to the heuristic function which is the Manhattan distance to the goal and the current cost, while greedy only focuses on the heuristic function. BFS searches through each layer while DFS sticks with one path and another if the previous one doesn’t work.

The order to find children of each node is down, right, up, and left. Changing the order only makes a big difference in DFS, because DFS sticks with one leaf first, and others if previous leaf does not return the result.

**Part 2 Sokoban**

**Introduction**

Our team solved the Sokoban problem with both an uninformed search – DFS, and informed search – A\* with a non-trivial heuristic function to guide the search. This section of the report will first include a brief explanation of the heuristic function implemented. It also presents an overview of the results for both algorithms followed by a short analysis of the result.

**Heuristic Function**

The heuristic consists of two part – the minimum matching of boxes and goals, and the minimum cost for the person to reach first box. For the first part, we assign each box a unique goal position. For each box-goal pair, we computed the cost of moving the box to goal by calculating distance between them. In our case, we simply use Manhattan distance. Hence, it’s now an optimization problem to minimize the sum of cost of matching all box-goal pairs.

In detail, we simple enumerated all possible matchings, computed the sum of the cost, and finally choose the one with minimum value. This method is feasible simply because the number of boxes is very small (less than 10) as it requires factorial time complexity.

For efficiency, the heuristic function is calculated online with a simple hash table as a cache. That is, we compute the heuristic function only when the state actually is being explored in the tree. For all situation we cached the computed result to avoid unnecessary computation. Each time we first check if the state is in the cache (a python dictionary) , and compute only when it’s necessary (not in the cache). This part intends to guide the person to push the right box to right place. For the second part, we simply compute the cost for the person to reach the closest box. The cost for person to box is calculated using Manhattan distance as well. This part intends to guide the person during the path from one box to another.

Now, we would like to show this heuristic function is admissible, that is, to show that our prediction is always optimistic. For the first part, we assume the cost of moving one box to its goal is simply the Manhattan distance. This is optimistic since there can never be a shorter path than it. In fact, we will be possibly blocked by walls and other boxes as well as cost for a man moving to the correct side of boxes. The sum of all individuals moves are also optimistic because we simply go through all the cases by brutal force and choose the one with smallest cost. There’s no way to be more optimistic than our agent. Hence, the first part of heuristic is indeed admissible. It’s okay to add the second part, because the man has to walk to the box first to start his work, and we choose the one closest to him so that there can be no better solution. By applying simple math, we can say that adding two optimistic parts can guarantee an admissible of the heuristic function as a whole.

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost (moves) | Node expanded | Cost of time (s) |
| BFS - input1 | 8 | 36 | 0.00329 |
| A\* - input1 | 8 | 24 | 0.00378 |
| BFS – input2 | 144 | 44545 | 8.731 |
| A\* - input2 | 144 | 44497 | 10.120 |
| BFS – input3 | 34 | **468493** | **134.444** |
| A\* - input3 | 34 | **49582** | **284.311** |
| BFS – input4 | 72 | 565102 | 491.806 |
| A\* - input4 | 72 | 540070 | 490.593 |

This table shows the result. The first column indicates the method with the corresponding input files’ number.

**Analysis**

For the cost of moves of the person, both A\*, BFS searches generate the same result, which hopefully means the A\* is able to generate the correct optimal solution. For the number of nodes expanded in the search tree, the A\* always beats BFS. The interesting happens in the 3rd map, where the nodes expanded by A\* is almost 10 times smaller than BFS. However, A\* has 2 times computation cost. This indicates that with more boxes the assignment of boxes to the correct goal position is crucial, while it’s sad that the computation cost grows too fast using our naïve brutal force matching algorithm. For the cost of time, the A\* usually performs worse. For small number of boxes – less than 7, it’s almost the same. However, after the number of boxes reaches 7, the computation time of heuristic function dominates the cost of time.

In conclusion, A\* reaches the optimal solution with fewer nodes expanded (or space complexity), while the computation time complexity grows when there are too many boxes.