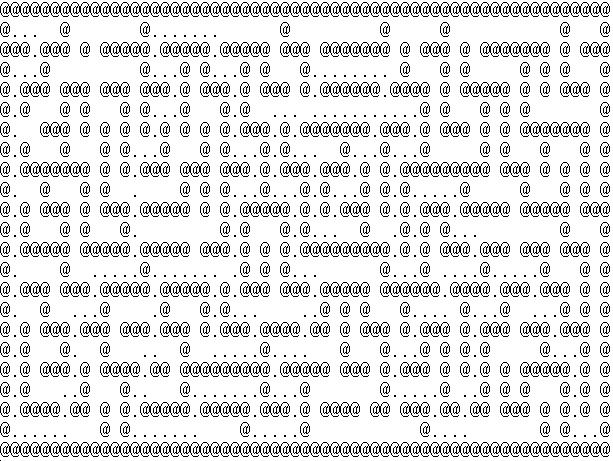
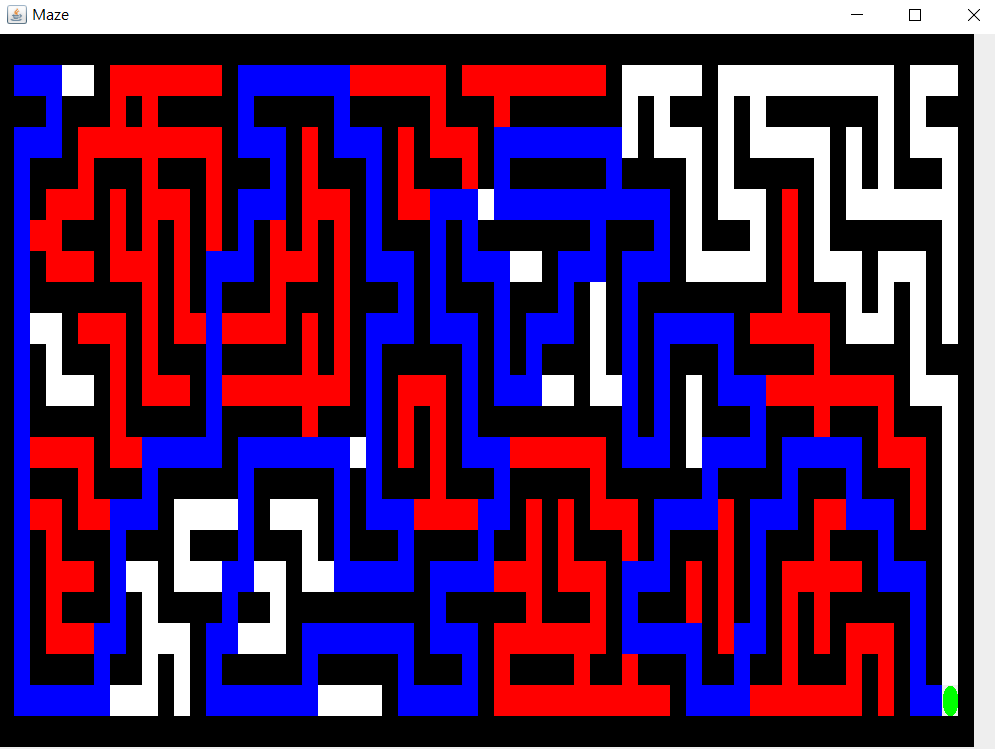
**Single point**

**One: mediumMaze**

1. **depth first search**

(1)





(2)

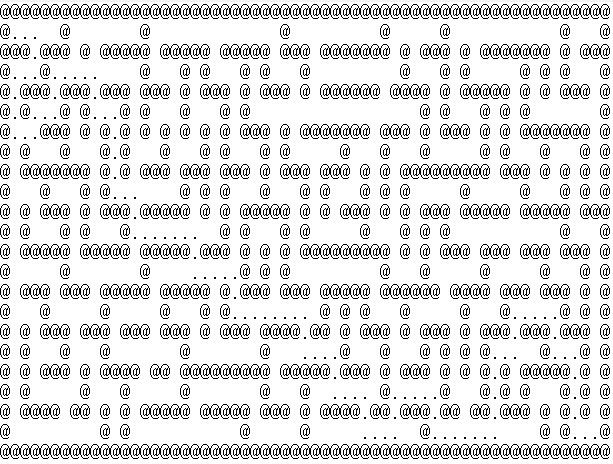
Total past cost:283

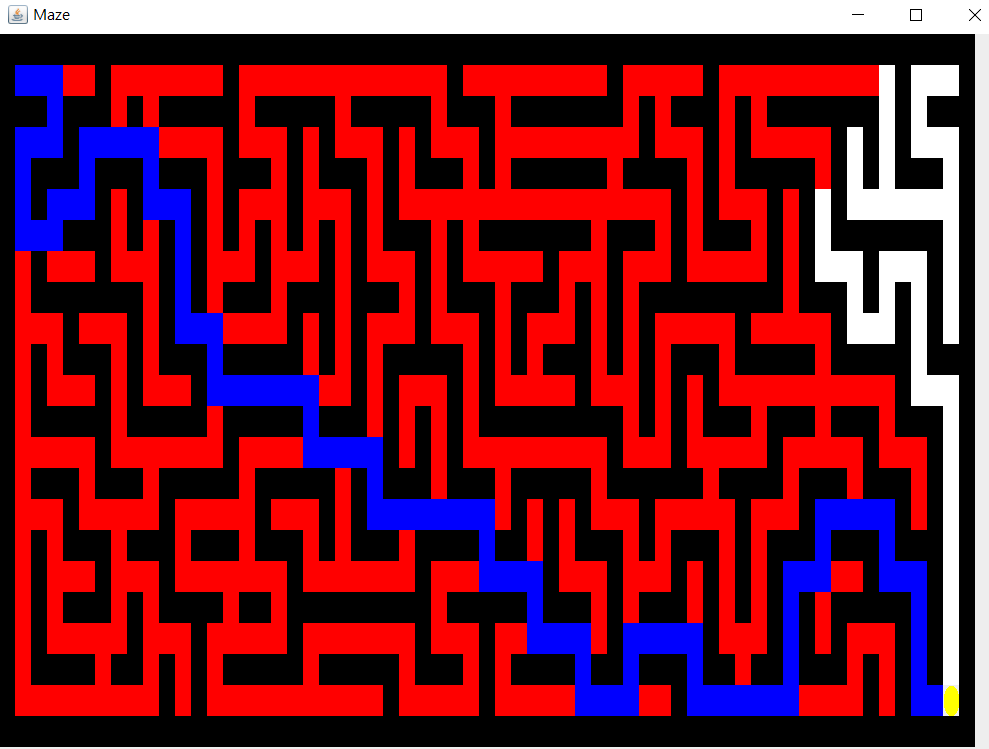
(3)

Node expanded:528

1. **breadthFirstSearch**

(1)





(2)

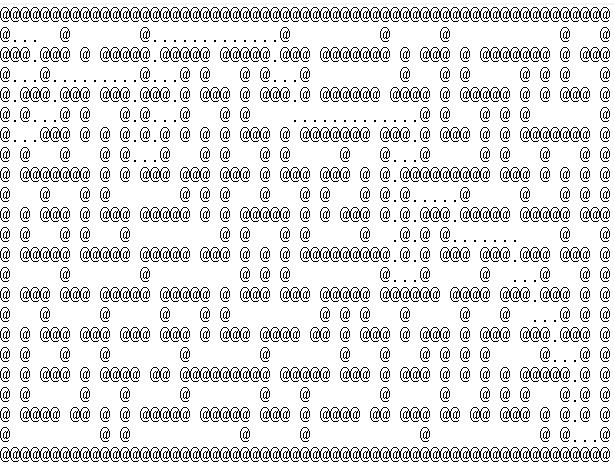
Total path cost:105

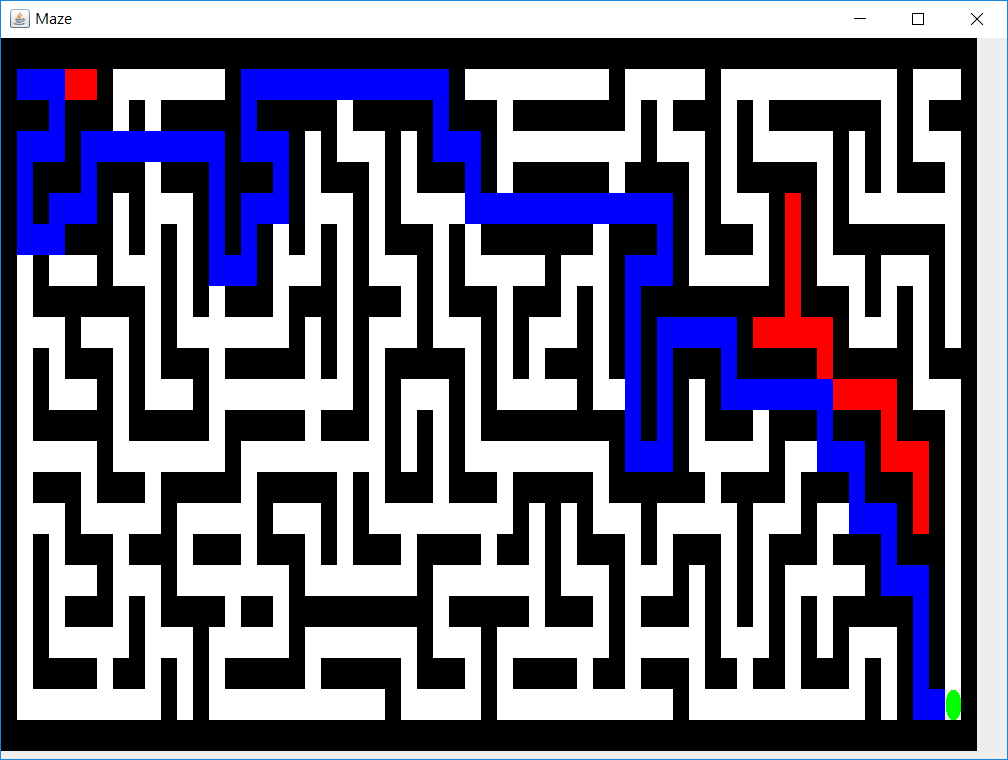
(3)

Total expanded node:629

1. **best first search**

(1)





(2)

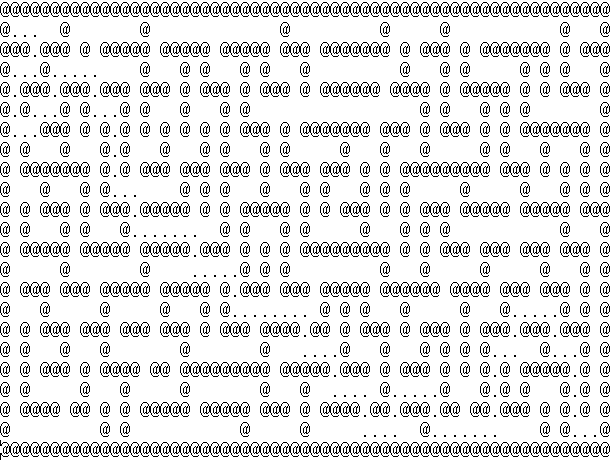
Total path cost:117

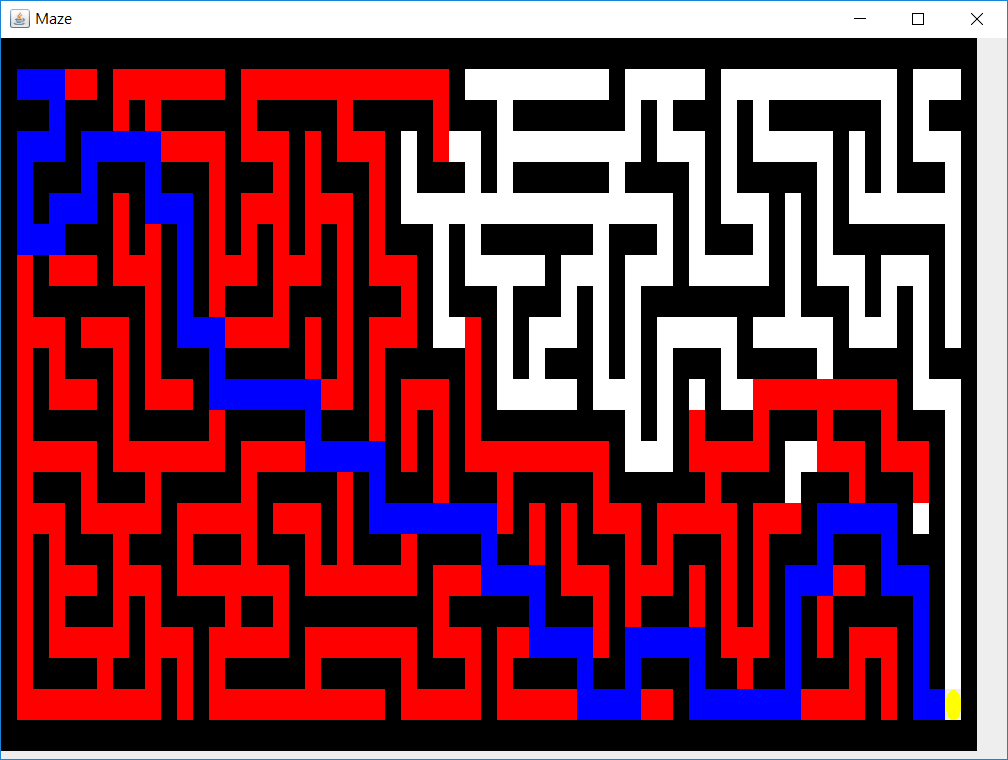
(3)

Total expanded node:138

1. **A\* search**

(1)





(2)

Total path cost:105

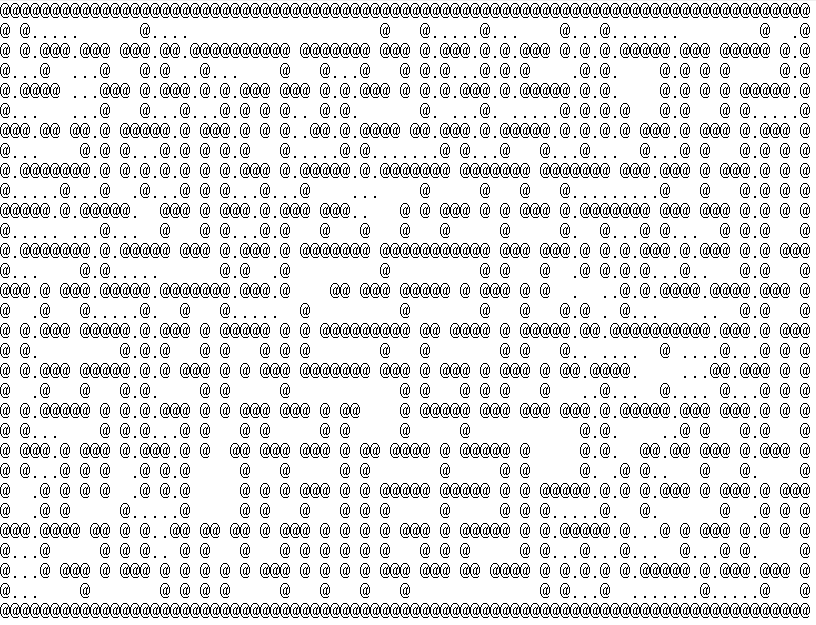
(3)

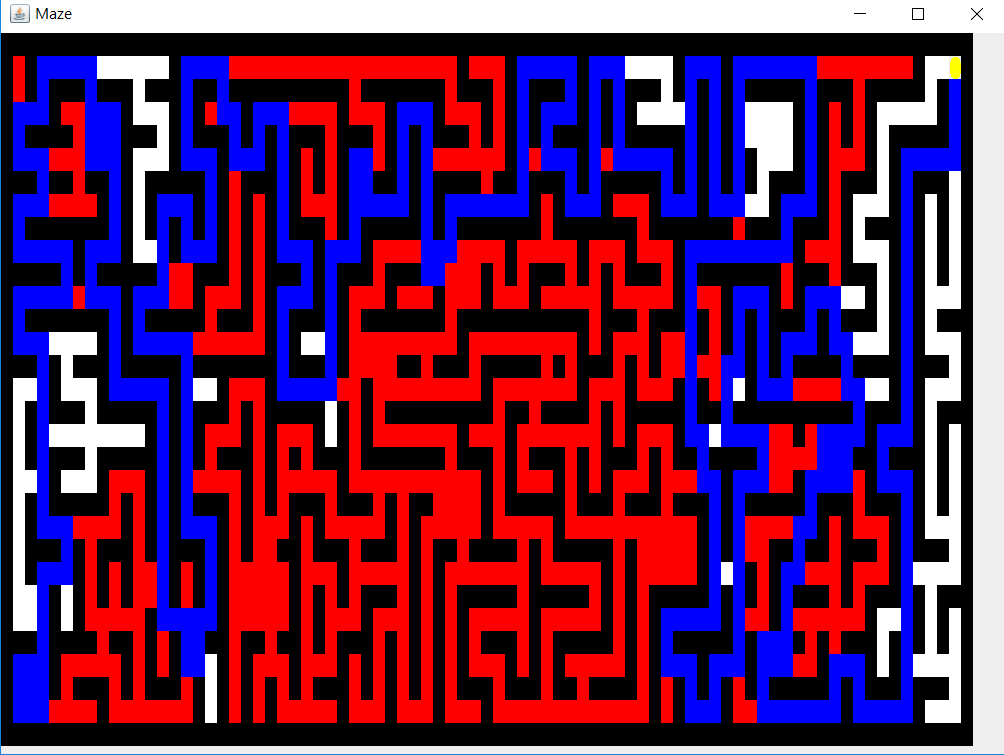
Total expanded node:473

**Two:bigMaze**

1. **depth first search**

(1)





(2)

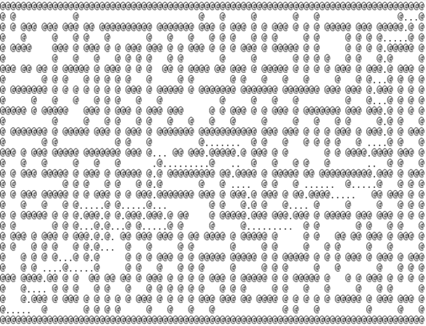
Total past cost:469

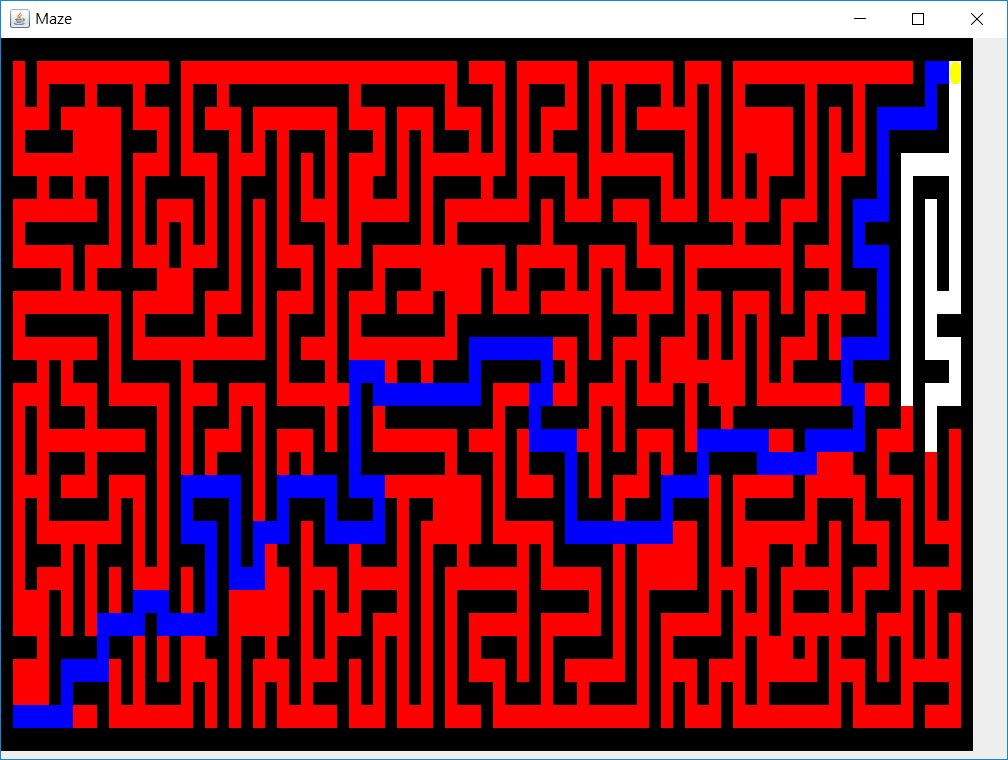
(3)

Total expanded node:1124

1. **breadth first search**

(1)





(2)

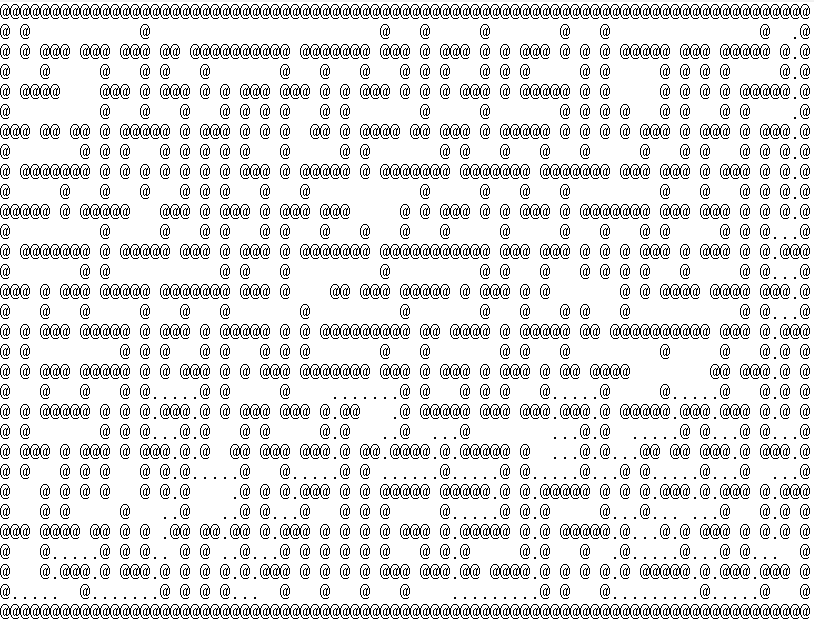
Total path cost:157

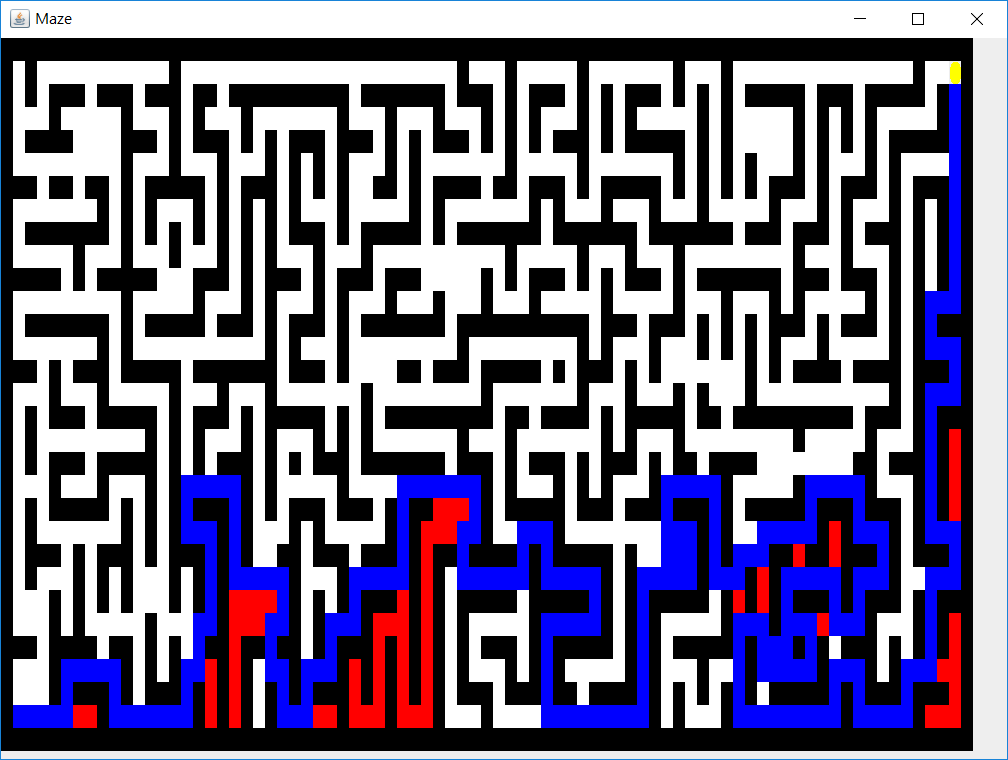
(3)

Total expanded node:1258

1. **best first search**

(1)





(2)

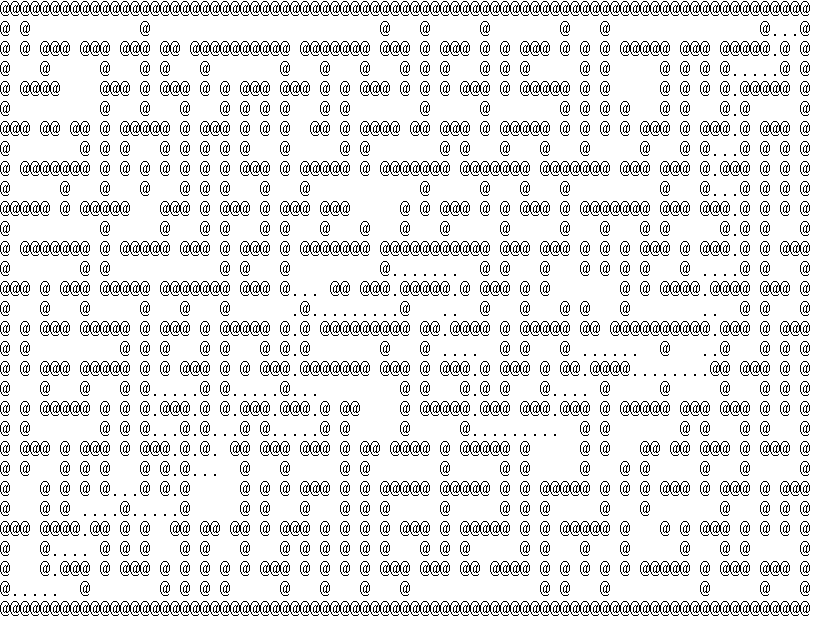
Total path cost:255

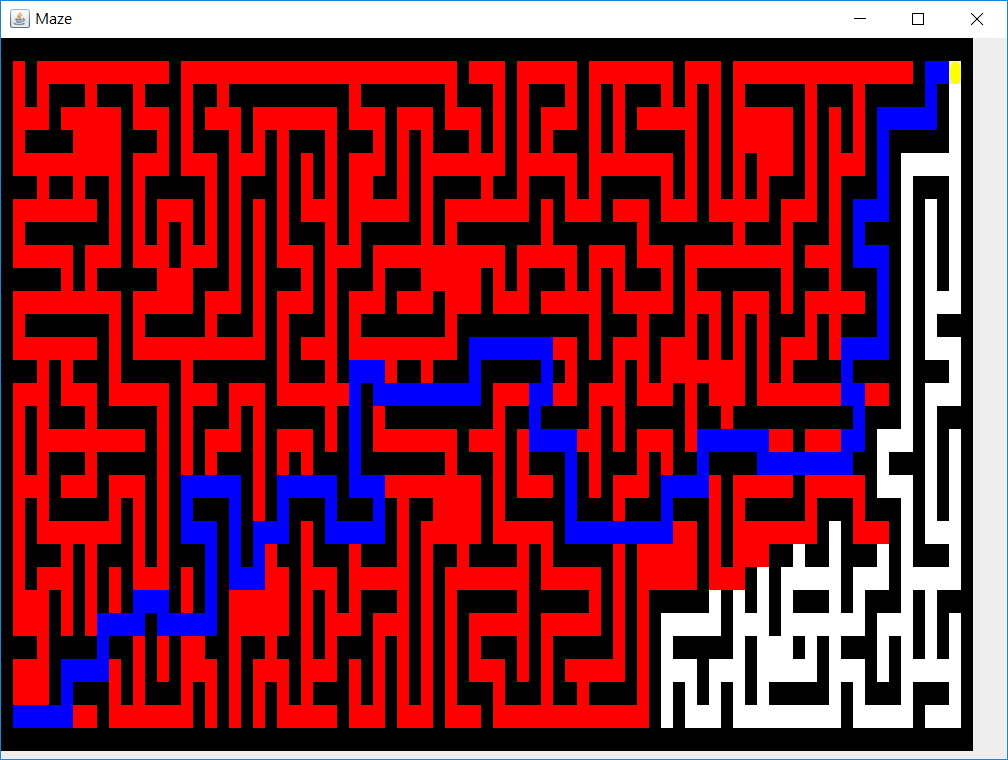
(3)

Total expanded node:322

**d.A\* search**

(1)





(2)

Total path cost:157

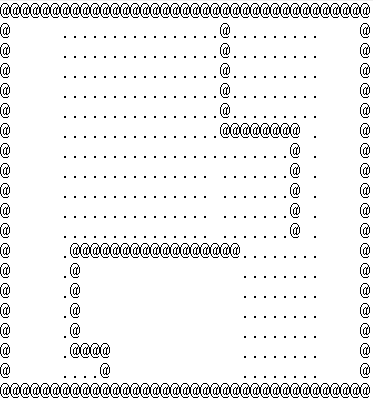
(3)

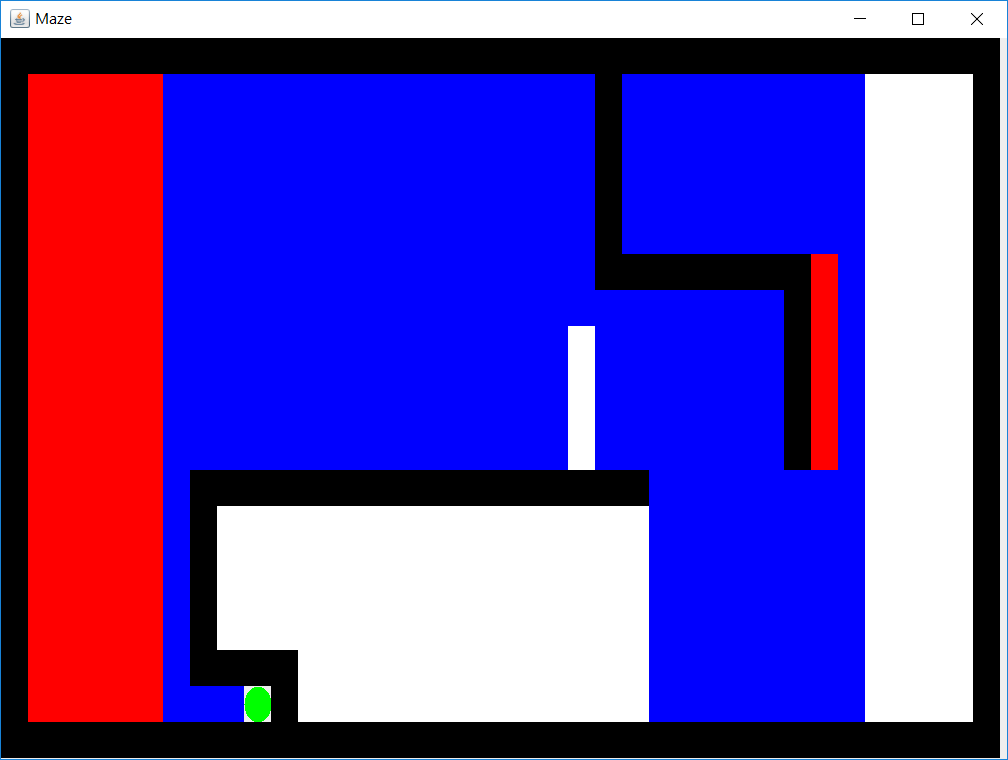
Total expanded node:1129

**Three:openMaze**

1. **depth first search**

(1)





(2)

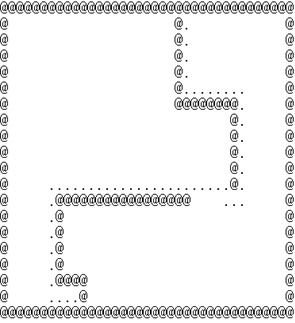
Total past cost:324

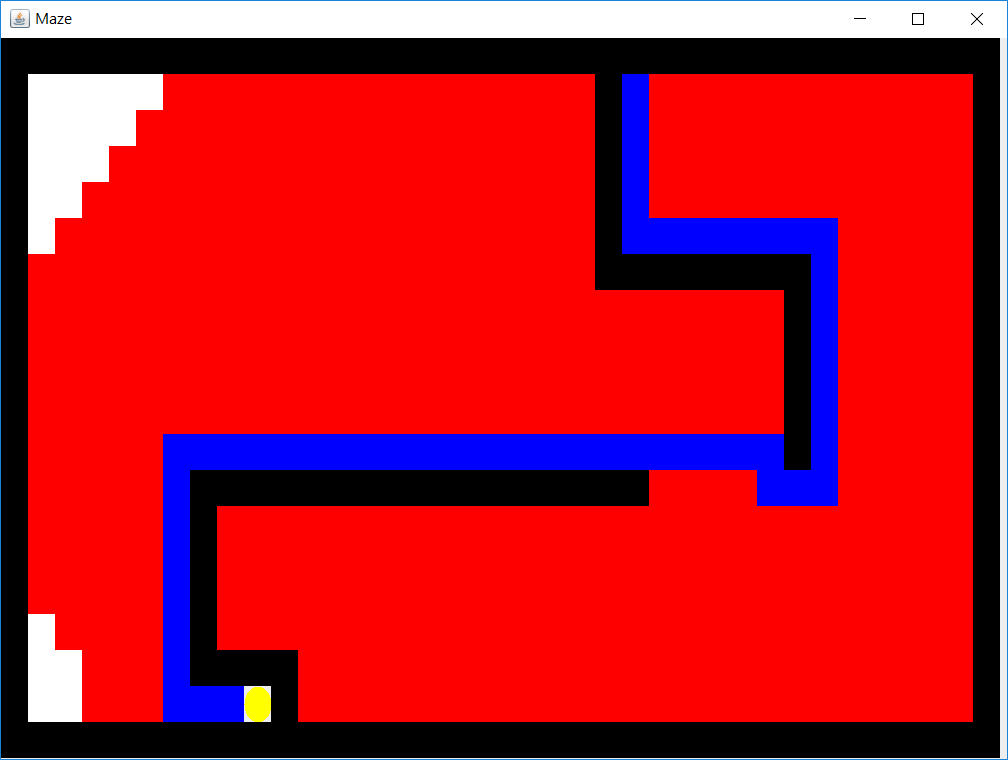
(3)

Total expanded node:419

1. **breadFirstSearch**

(1)





(2)

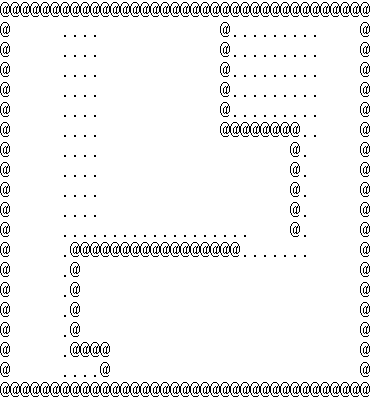
Total path cost:54

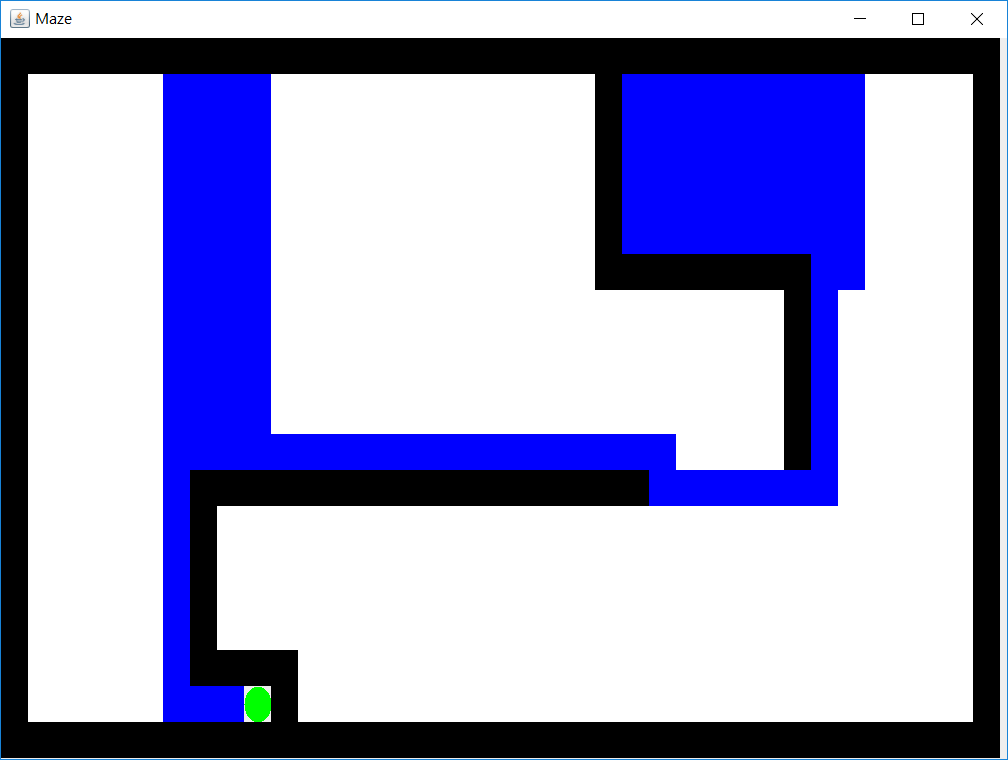
(3)

Total expanded node:565

1. **best first search**

(1)





(2)

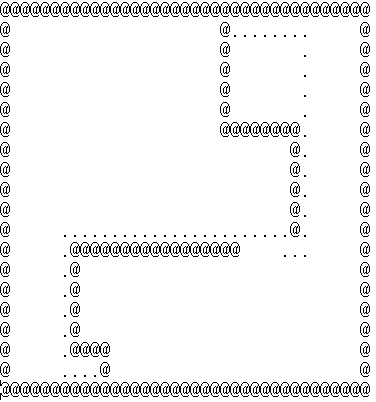
Total path cost:128

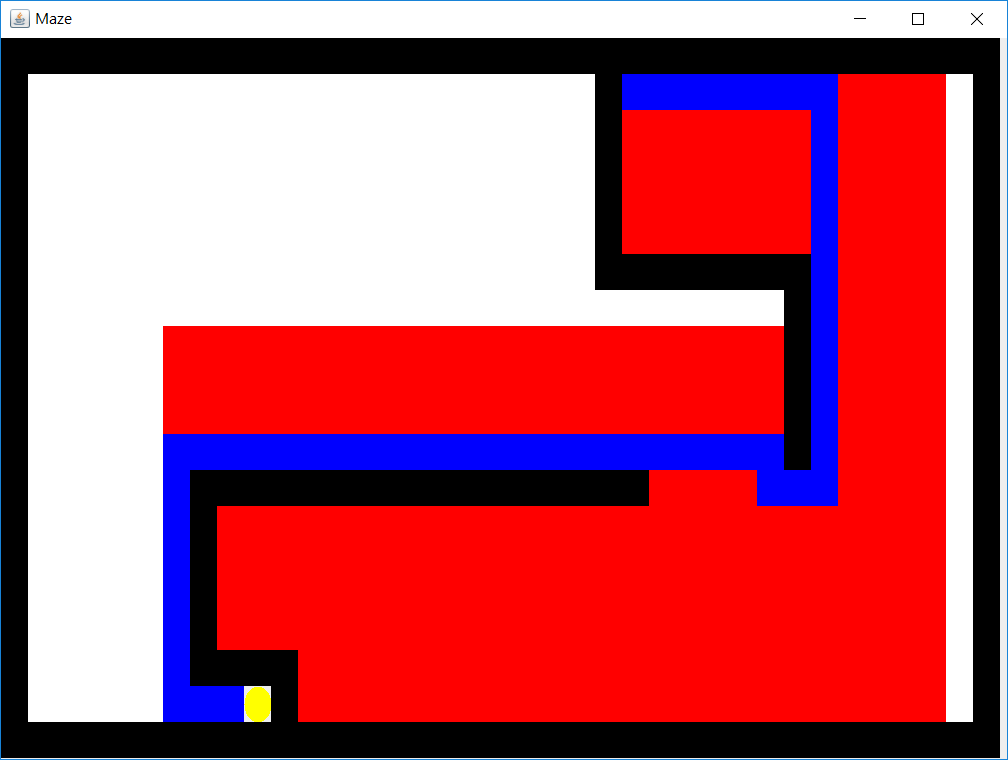
(3)

Total expanded node:127

1. **A\* search**

(1)





(2)

Total path cost:54

(3)

Total expanded node:358

**Implementation**

(a)DFS

For the depth first search, we use stack as our data structure, and we keep trying each solution until it hits the end and has no place to go. We then pop the elements from the stack to return to the intersection and start trying a new path. This often leads to the worst path cost, since no optimal solution is guaranteed. The solution is simply the first one we find.

(b)BFS

For the breadth first search, we are guaranteed the best solution, since we simply expand every node in each level. The data structure we use is Queue, since every node in each level is going to be expanded until we found the solution. So the space complexity is much higher. However, we can get the path with minimum cost.

(c)Best first search

This is greedy search, it’s similar to DFS, however, we add the heuristic function to make the solution more rational, rather than simply try each path without thinking. However, this kind of search can’t guarantee to find the best path cost, nonetheless, it’s usually much better than DFS.

(d)A\* search

This is the best method, it’s similar to BFS, however, we use the heuristic function to restrict the node to be expanded. So the total expanded node can be less, the heuristic function we choose is Manhattan distance, which is admissible. Since the grid is of Manhattan form, so no heuristic function can be greater than the actual cost.

———————————————————————————————————————————————

**1-2:Multiple points**

1. Multiple points

Heuristic:

In this problem, we calculated the costMatrix(the distance between each food pallet and another food pallet calculated by aStarSearch in section 1-1) and the pathMatrix(an arraylist of arraylist of queue,which stores the path from each food pallet to another food pallet acquired with aStarSearch).

With the costMatrix and pathMatrix defined, we then calculate the evaluation function of each node as the sum of the travelled distance (path cost) and target distance(heuristic function). The travelled distance is sum of the distance of the path already taken and the current path cost from the food pallet to another food pallet defined in cost matrix(which is acquired by the aStarSearch in section 1-1). While the target distance is the sum of the minimum distance from remaining food pallets to other remaining food pallets and negative number of eaten food. For example, there are three food(A,B,C), and A has been eaten, then the heuristic will be min(A->B,C->B) + min(A->C,B->C) - 1. With the heuristic function defined above, we can guarantee that it is admissible, since we use the minimum value to get to remaining food pallets and it must be equal or less than true cost. In addition, we minus the number of eaten food to decrease the value of deeper nodes(eat more food) rendering that nodes almost completed to finish faster.

(1)

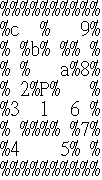
Tiny\_search.txt:

Solution cost:36

Expanded nodes:188

Total time:1 second

Results:



(2)

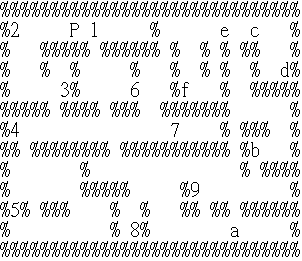
Small\_search.txt

Solution cost:124

Expanded nodes:84837

Total time:6 seconds

Result:



(3)

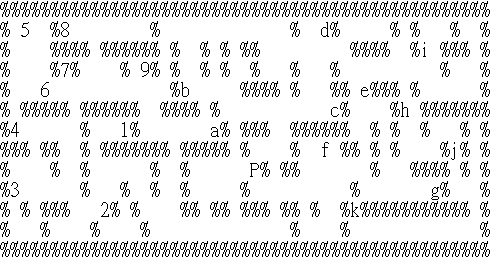
Medium\_search.txt

Solution cost:198

Expanded nodes:2753830

Total time:3068 seconds

Result:



b.Suboptimal

implementation:

We use 3 parameters for the heuristic function:

1.Whether or not we can eat the food pallet in next step(top priority)

2.Whether the state is visited(second top priority)

3.distance to the closest food pallet

In this case, we can eat all the food pallets with a total of 318 steps. The animation is given in the same file folder.

———————————————————————————————————————————————

**2-1:Basic Sokoban**

For each step in the solving procedure, we define a data structure that records current position, the positions of all the boxes, the current Sokoban array state(what’s on each position, ex:human, box, storage point, wall, space, human on storage point, box on storage point), the path already taken, the direction already chosen(so as to avoid repeat visiting, thus we can be guaranteed to get the final answer), and the heuristic.

**heuristic:**

For this heuristic, we calculate the summation of the distances from the boxesNotOnTheStorage to the closest storagePosition in order to move the boxes closer to the storage point and we minus the number of the box points already on storage so as to make the boxes to be pushed on the storage point more easily. Since we use the closest distance between the box and the storage, the heuristic function is admissible.

**heuristic:**

For the heuristic, we first calculate the summation of the distances from the boxesNotOnTheStorage to the closest storagePosition in order to move the boxes closer to the storage point. What’s more, to speed up the process, we would like to make the agent closer and closer to the box(so as to move them and change the state). So we add another heuristic that calculates the average distance from agent to all the boxes. Hence, we can be convinced that the agent will keep moving closer to the boxes and make the movement.

The heuristic is admissible because we calculate the distance from the boxStoragePoints to the closest storagePosition. Thus, no other total distance between boxes and storages found can be less then this.

For the distance between human agents and the boxes, we calculate their distance with Manhattan distance, which is also guaranteed to be admissible.

According to above, the heuristic is admissible.

**input1.txt**

a.

heuristic:

path length:33

answer:(up:0,right:1,down:2,left:3)

1, 2, 1, 2, 1, 2, 2, 3, 3, 1, 1, 0, 0, 3, 2, 3, 2, 1, 0, 0, 0, 3, 2, 2, 0, 0, 0, 0, 1, 2, 2, 2, 2

node expanded:1369

running time:5.3 second

b.

heuristic:

path length:34

answer:(up:0,right:1,down:2,left:3)

1, 2, 1, 2, 1, 2, 2, 3, 3, 1, 1, 0, 0, 3, 2, 3, 2, 1, 0, 0, 0, 3, 0, 0, 1, 2, 2, 2, 2, 0, 0, 3, 2, 2

node expanded:1119

running time:1.2 second

**input2.txt:**

a.

heuristic:

path length:51

answer: (up:0,right:1,down:2,left:3)

0, 1, 2, 1, 1, 2, 2, 1, 2, 2, 3, 3, 0, 3, 3, 2, 3, 0, 1, 1, 1, 2, 1, 1, 0, 0, 3, 0, 0, 3, 3, 2, 2, 0, 0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 3, 3, 1, 1, 0, 3, 3

node expanded:2977

running time:12.2 second

b.

heuristic:

path length:55

answer: (up:0,right:1,down:2,left:3)

0, 1, 2, 2, 0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 0, 3, 3, 2, 3, 0, 1, 1, 1, 2, 3, 1, 1, 1, 0, 0, 3, 0, 0, 3, 3, 2, 2, 0, 0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 0, 3, 2, 3, 1, 0, 3

node expanded:2024

running time:3.3 second

**input3.txt:**

a.

heuristic:

path length:34

answer: (up:0,right:1,down:2,left:3)

1, 0, 1, 1, 2, 2, 2, 2, 3, 2, 1, 0, 0, 0, 0, 3, 3, 3, 1, 2, 1, 2, 1, 2, 2, 3, 3, 2, 3, 3, 0, 1, 3, 0

node expanded:24442

running time: 371.3 second

b.

heuristic:

path length:34

answer: (up:0,right:1,down:2,left:3)

1, 0, 1, 1, 2, 2, 2, 2, 3, 2, 1, 0, 0, 0, 0, 3, 3, 3, 1, 2, 1, 2, 1, 2, 2, 3, 3, 2, 3, 3, 0, 0, 2, 1

node expanded:14795

running time: 107.2 second

**input4.txt:**

a.

heuristic:

path length:144

answer: (up:0,right:1,down:2,left:3)

2, 2, 0, 0, 1, 1, 2, 2, 3, 3, 1, 1, 2, 2, 3, 3, 0, 3, 3, 0, 0, 1, 2, 1, 2, 2, 1, 1, 0, 0, 3, 3, 1, 1, 0, 0, 3, 3, 2, 3, 3, 2, 2, 2, 1, 1, 0, 0, 3, 1, 1, 1, 0, 0, 3, 3, 2, 3, 2, 1, 2, 2, 3, 0, 1, 0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 0, 0, 1, 3, 2, 2, 1, 1, 0, 0, 0, 2, 3, 3, 2, 3, 3, 2, 1, 1, 0, 0, 0, 3, 2, 1, 2, 2, 3, 3, 0, 1, 0, 1, 1, 1, 2, 2, 3, 3, 0, 3, 0, 1, 1, 3, 2, 2, 1, 1, 0, 0, 3, 3, 3, 2, 3, 2, 1, 1, 1, 3, 0, 0, 0, 3, 3, 2

node expanded:63631

running time: 1882.3 second

b.

heuristic:

path length:146

answer: (up:0,right:1,down:2,left:3)

2, 2, 0, 0, 1, 1, 2, 2, 3, 3, 1, 1, 2, 2, 3, 3, 0, 3, 3, 0, 0, 1, 2, 1, 2, 2, 1, 1, 0, 0, 3, 3, 1, 1, 0, 0, 3, 3, 2, 2, 0, 3, 3, 2, 2, 2, 1, 1, 0, 0, 3, 1, 1, 1, 0, 0, 3, 3, 2, 3, 2, 1, 2, 2, 3, 0, 1, 0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 0, 0, 1, 3, 2, 2, 1, 1, 0, 0, 0, 2, 3, 3, 2, 3, 3, 2, 1, 1, 0, 0, 0, 3, 2, 1, 2, 2, 3, 3, 0, 1, 0, 1, 1, 1, 2, 2, 3, 3, 0, 3, 0, 1, 1, 3, 2, 2, 1, 1, 0, 0, 3, 3, 3, 2, 3, 2, 1, 1, 1, 3, 0, 0, 3, 0, 3, 2

node expanded:29112

running time: 362.3 second

**Sokoban extra:**

**Heuristic for Sokoban\_extra1:**

In Sokoban\_extra 1 problem, since the state is rather complicate, so we define some pattern as deadlock, and never expand them so as to reduce a lot of computation. We also introduce the concept of tunnel, that is, the agent will keep pushing the box if the 2 opposite sides of the box are walls. In this case, we can further reduce the computation requirement.

**Heuristic for Sokoban\_extra2:**

We add the constraints on the boxex so that it won’t be able to move to form a cluster of 4(a 2\*2 square, which forms a deadlock). We also prevent the box from going too close together(by reducing the heuristic if the box are not side by side). And for the the outmost column or row, if there are no storage points there, we won’t allow the box to be moved to that column or row, since this will certainly lead to deadlock(the box at the outermost column or row can not be moved inside).

**Sokoban\_extra1.txt**

path length:501

answer: (up:0,right:1,down:2,left:3)

2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 3, 0, 3, 3, 3, 2, 2, 1, 1, 0, 2, 3, 3, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 3, 3, 2, 2, 2, 0, 1, 2, 2, 3, 2, 3, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 0, 0, 0, 3, 0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 2, 1, 2, 2, 3, 2, 3, 3, 1, 1, 0, 1, 0, 0, 3, 0, 3, 0, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 3, 3, 0, 1, 2, 1, 0, 0, 0, 0, 3, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 3, 2, 3, 3, 1, 1, 0, 0, 3, 0, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 3, 2, 1, 2, 3, 3, 1, 1, 0, 0, 3, 0, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0, 0, 1, 1, 1, 0, 2, 3, 3, 3, 0, 0, 1, 2, 3, 2, 1, 3, 2, 2, 2, 2, 1, 2, 2, 3, 2, 3, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 0, 0, 3, 0, 1, 2, 3, 2, 2, 1, 2, 3, 0, 3, 3, 3, 2, 2, 1, 1, 0, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 2, 1, 2, 2, 3, 3, 3, 0, 3, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 0, 1, 1, 2, 3, 3, 1, 1, 0, 1, 0, 0, 3, 2, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 2, 2, 3, 2, 3, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 0, 2, 3, 0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 2, 3, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 3, 0, 1, 3, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 3, 0, 3, 3, 0, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 3, 1, 2, 3, 2, 3, 3, 0, 0, 1, 1, 1, 2, 1, 0, 3, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 3, 2, 0, 3, 3, 0, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 0, 3, 3, 3, 2, 2, 1, 1, 0, 2, 3, 3, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 3, 3, 3, 0, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 1

node expanded:24163

running time: 231.2 second

**Sokoban\_extra2.txt**

path length:202

answer: (up:0,right:1,down:2,left:3)

1, 1, 1, 0, 3, 1, 1, 2, 1, 1, 0, 0, 3, 0, 3, 2, 2, 1, 0, 0, 3, 3, 3, 2, 1, 3, 3, 2, 2, 0, 0, 0, 3, 2, 3, 2, 1, 0, 1, 2, 1, 0, 1, 0, 1, 1, 2, 2, 3, 3, 1, 1, 1, 2, 2, 2, 3, 0, 0, 0, 2, 2, 2, 3, 3, 0, 2, 3, 3, 3, 0, 0, 0, 3, 0, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 2, 1, 1, 0, 0, 0, 3, 3, 2, 0, 3, 0, 0, 1, 2, 2, 1, 1, 0, 3, 3, 2, 1, 1, 2, 2, 2, 3, 0, 0, 0, 1, 0, 3, 2, 2, 2, 2, 3, 3, 0, 1, 2, 1, 1, 0, 0, 3, 0, 3, 2, 0, 3, 2, 1, 1, 0, 0, 0, 3, 2, 2, 1, 0, 0, 3, 3, 3, 2, 1, 2, 1, 1, 1, 2, 2, 2, 3, 0, 0, 1, 0, 3, 2, 2, 2, 3, 3, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0, 0, 3, 3, 2, 2, 3, 0, 0, 3, 3, 2, 2, 2, 2, 2, 1, 1, 0, 0

node expanded:15973

running time: 38.2 second