Assignment 1: Search

Group Members: Liyi Sun,Haoran Wang
Section: R4
Credits: 4

Introduction

The report is divided into two parts. The first part addresses the Pacman searching problem, which includes single-goal searching and multi-goal searching. As for part 1.1, we implement four searching stratregy: depth-frist search, breadth-first search, greedy best-frist seach and A* search. And the heuristic used for greedy best-first and A* algorithm is Manhattan distance in this part. For part 1.2, we use A* search with specific heuristic function to find all dots in the maze.

We started by designing a funciton: $readData(filename)$ to read the maze into a matrix from files, which is hepful for us to represent the nodes in the form of coordinates with the purpose of using Manhattan distance in this case. Then the node is represented as tuple of (x, y) in the algorithms. So the path (aka the sequence of movement) could be expressed as a list of tuples of nodes. (For instance, $path = [(x_1, y_1), (x_2, y_2), (x_3, y_3)......(x_n, y_n)]$) In this part, $set(\,)\ of\ python$ is used to handle the repeated states detection. We implemented a for loop to control the four possible movements for the agent. Besides, we also keep track of the cost when making a movement and the number of nodes expanded in the algorithms to the goal state. The outcome of different algorithms are written into files using the $plotPath$ function. Details are addressed in the following parts.

// This should be a brief introduction to Part 2

Part 1 Pacman Search

1. Basic pathfinding

1.1 Breadth-Frist Search

The idea of breadth-first search (BFS) is searching the shallowest unexpanded nodes. So we implemented the First-In-First-Out strategy by using a *queue* for the nodes. Specifically speaking, *queue* is implemented via the module *collections.deque* of Python. Since *deque* object is flexible that it has two methods: $popleft(\,)$ and $pop(\,)$. The method $popleft(\,)$ could remove and return the leftmost element of the queue. The second one $pop(\,)$ could remove and return the rightmost element, which indicates it is a kind of implement of stack and we could use it to implement depth-frist search. So we first add the coordinate of the start node (x_{start}, y_{start}) into the queue. Then while the queue is not empty, we pop the leftmost element and check whether it is the goal state(i.e. whether the element is a dot or not). If it is ture, we return the path to this node. Then we check whether the node is a wall (i.e. the node is '%' if it is wall), if it is a wall, we continue the loop. The last check is whether this node has been visited, if yes, we just continue; if no, we add the tuple $(x_{current}, y_{current})$ into the visited set. Next, we need to get each of the four adjacent nodes of current node and append them into the *queue.* What is appended is a *tuple*, including the current node and its parents in this form:

$$[(current\ node),(currentnode,(parent))]$$

So we can get the entire path from the second tuple in the above expression when we finally reach the goal state. Besides, in the mean time, we also keep the record of the amount of expanded nodes. Once we get the returned path, we use $pathToGoalState(path)$ to get the correct order of path and the number of cost. In sum, BFS could always figure out the optimal path to the goal state but it would have a heavier space complexity since it would expand each node in the same layer of a tree. That's the implementation of breadth-first searching algorithm in part 1.

1.2 Depth-First Search

The idea of depth-first search (DFS) is searching the deepest unexpanded node first. So it actually could be handled as a *stack* since the Last-In-First-Out strategy. As mentioned above, we use $deque.pop(\,)$ to get the rightmost element in a stack, which is the last one entered. So the sequence of implementation is the same as BFS. The only tiny difference is the way to handle *queue* and *stack* via different methods of *deque* in this case. So the biggest difference between BFS and DFS is that BFS will expand every node that has the same length to the starting node, while DFS will expand one chosen direction until to the bottom of the branch then explore the other ones. One important thing is that the total cost of path and number of expanded nodes are decided on which direction order you choose. For example, the total cost of $direction : [left, right, down, up]$ in *openMaze* is 343, number of expaned nodes is 504. However, if we change it to: $direction : [down, up, left, right]$, the cost is 321 and number of expaneded nodes is 452. Therefore, DFS doesn't always guarantee the optiaml path to the goal. Since DFS could "forget" the path in one direction, so for each direction, they should use different visited set of expanded nodes. Other steps are the same as BFS.

1.3 Greedy Best-Frist Search

Greedy best-frist search (GBFS) is an informed search strategy. It employs a heuristic funciton to estimate the cost of reaching the goal from the current node, then decides which node to expand base on this value. To implement GBFS, we use the *prioprity queue* to sort the heuristic values of each node in the frontier. Specially speaking, we use the $heapq$ to implement the prioprity queue as its method $heapq.heappop(\,)$ could pop the smallest item from the prioprity queue. We construct a tuple in the following form:

$$(heuristic,(current\ node,\ parent)\,)$$

and we add the tuple into the prioprity queue at each possible movement. The mentioned $heapq.heappop(\,)$ method will sort the prioprity queue and return the node that has the smallest value of the first element, which is heuristic of the node. While the prioprity queue is not empty, we check the popped node as mentioned in the previous two algorithm. It includes whether it is the goal, whether it has been visited, and whether it is a wall. If it satisfies one of the last two condition, then we break the current loop and go to the next node. So GBFS is kind of similar to DFS, but it uses heuristic value to choose the next expanded node from the frontier instead of the invariant direction.

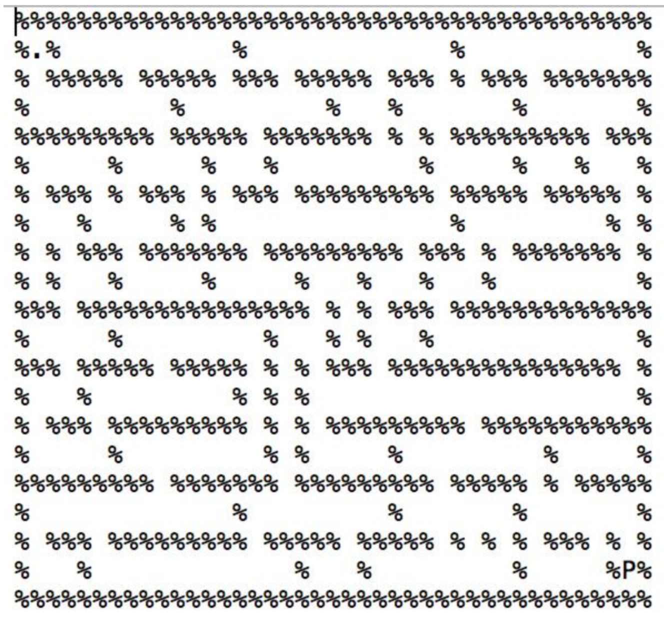
1.4 A* Search

Greedy best-first search may lead to an unended loop if we only rely on heuristic of manhattan distance between two nodes in the grid. So A* is an improvement based on the GBFS. We use the cost so fat to reach the current node plus manhattan distance to the goal as evaluation value instead of the only heuristic in GBFS. To implement this, we use the basic structure of GBFS as mentioned above. The first found path maybe not the optimal path found by A*. The sum of heuristic value may be lower for other possible path. So the condition to finish the while loop should be the priopity queue is empty. Once we find a path we assign it to the optimal path and compare the cost of it and new found path. In this way, A* could return the optimal path to the goal.

1.5 Outcome of Three Mazes

1.5.1 Overview

In this part, we would like to show the result of the four algorithms applied to solving the different three kind of mazes: medium maze, big maze and open maze. The maze is shown as follows:

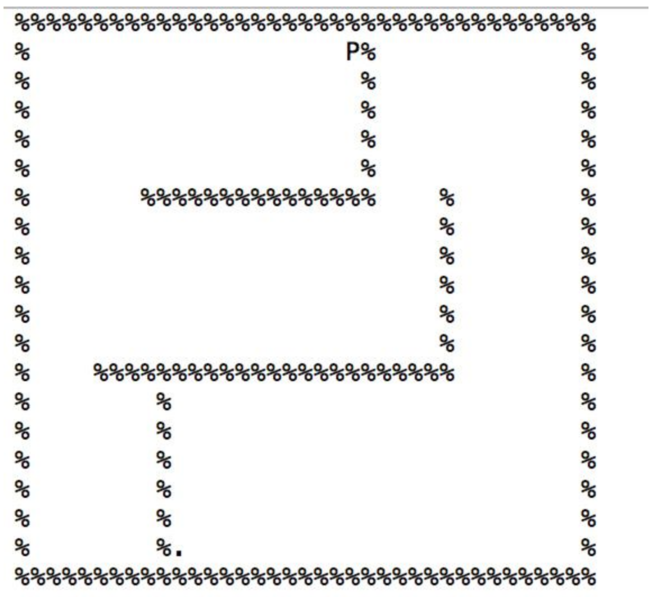


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.%           %             %           %
% %%%%% %%%%% %%% %%%%% %%% % %%% %%%%%%%
%         %         %   %       %       %
%%%%%%%%% %%%%% %%%%%%% % % %%%%%%%%% %%%
%     %     %   %         %     %   %   %
% %%% % %%% % %%% %%%%%%%%% %%%%% %%%%% %
%   %     % %               %         % %
% % %%% %%%%%%% %%%%%%%%% %%% % %%%%%%% %
% %   %     %     %   %   %   %         %
%%% %%%%%%%%%%%%%%% % % %%% %%%%%%%%%%%%%
%     %         %   % %   %             %
%%% %%%%% %%%%% % % %%% %%%%%%%%%%%%%%% %
%   %         % % %                     %
% %%% %%%%%%%%% % % %%%%%%%%% %%%%%%%%%%%
%     %         % %     %         %     %
%%%%%%%%% %%%%%%% %%%%%%%%% %%%%% % %%%%%
%             %         %       %       %
% %%% %%%%%%%%% %%%%% %%%%% % % % %%% % %
%   %             %   %         %     %P%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 1 Medium Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       %     %       % %     % %   % % %
% % %%% % %%% %%% %%%%% % %%%%% % %%% % %
% %   % %   %   %   %       % %         %
%%%%%%% %%% %%% % % % % % %%% % %%% % % %
%   %         %   % % % % %     % % % % %
% %%% % %%% %%%%%%% % %%% % %%%%% % %%% %
% %   %   %   % %   %   %   % %     %   %
% % %%% % %%%%% % %%%%% %%%%% %%% %%%%%%%
%   %   % %     % %     %   %       %   %
%%% %%%%%%%%% %%% % %%% %%% %%%%%%% %%% %
%     % %   %     % % % % %         %   %
%%%%% % % %%%%%%% % % %%% %%% %%% %%% % %
% %   %     %   %   %           % %   % %
% % %%% % %%% %%%%%%%%% %%%%%%% %%% %%%%%
%       %     %       % %     %   %     %
% %%%%%%%%% %%%%% %%%%% %%%%% % % %%% %%%
% %   % %         % % %   %   % %     % %
% % % % %%%%% %%%%% % %%% % % %%% % %%%%%
% % %       %   % %     % %       % % % %
% % %%% % %%% %%% %%%%% % % %%% % %%% % %
%   % % % %       %   % % %   % % %     %
%%% % %%%%%%%%% %%% %%% % %%% %%%%%%% %%%
%     % %   %           %   %           %
%%% % % % %%% %%% %%%%% % % %%%%%%%%%%% %
%   % %   %     %   %   % %       %     %
%%% %%% %%%%% % %%% % %%% % %%%%% %%% % %
%     %   %       % %   % % % % %     % %
% % % % %%%%% % %%%%%%%%%%% % % %%%%%%% %
% % % %     % %           %   %       % %
% %%% %%% % %%%%% % %%%%% %%% % % %%%%%%%
%   % %   %       % %   %   %   %     % %
% %%%%%%% %%%%% %%%%% % %%% %%% %%% % % %
%   %     %   % %     %       % %   %   %
% % %%%%%%%%%%% % % %%%%% % % % %%%%% %%%
% %   %       % % %     % % % %   %     %
% %%%%% % %%%%%%% %%% %%%%%%% %%% % %%% %
%   %   %     %   %   %     % %   % % % %
%%%%% % %%% %%%%% %%%%% %%% % %%% % % %%%
%.    % %                 % %   % %    P%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 2 Big Maze



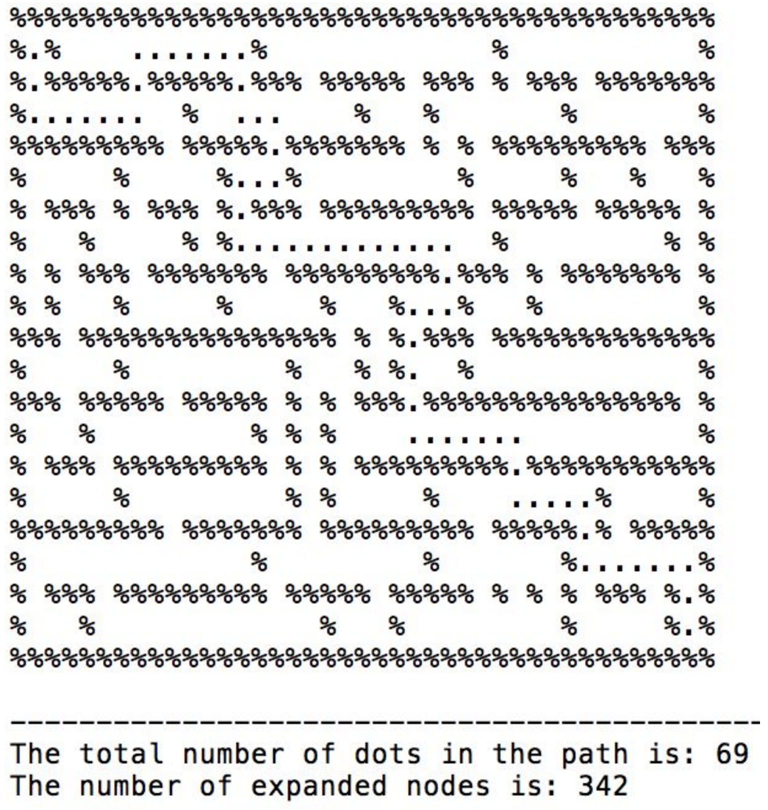
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    P%             %
%                     %             %
%                     %             %
%                     %             %
%                     %             %
%       %%%%%%%%%%%%%%%    %        %
%                          %        %
%                          %        %
%                          %        %
%                          %        %
%                          %        %
%    %%%%%%%%%%%%%%%%%%%%%%%        %
%        %                          %
%        %                          %
%        %                          %
%        %                          %
%        %                          %
%        %.                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 3 Open Maze

where P is the start node, ‘.’ is the goal.

1.5.2 Outcome of Medium Maze

Algorithm: BFS
Cost of the path: 69
Number of expanded nodes: 342

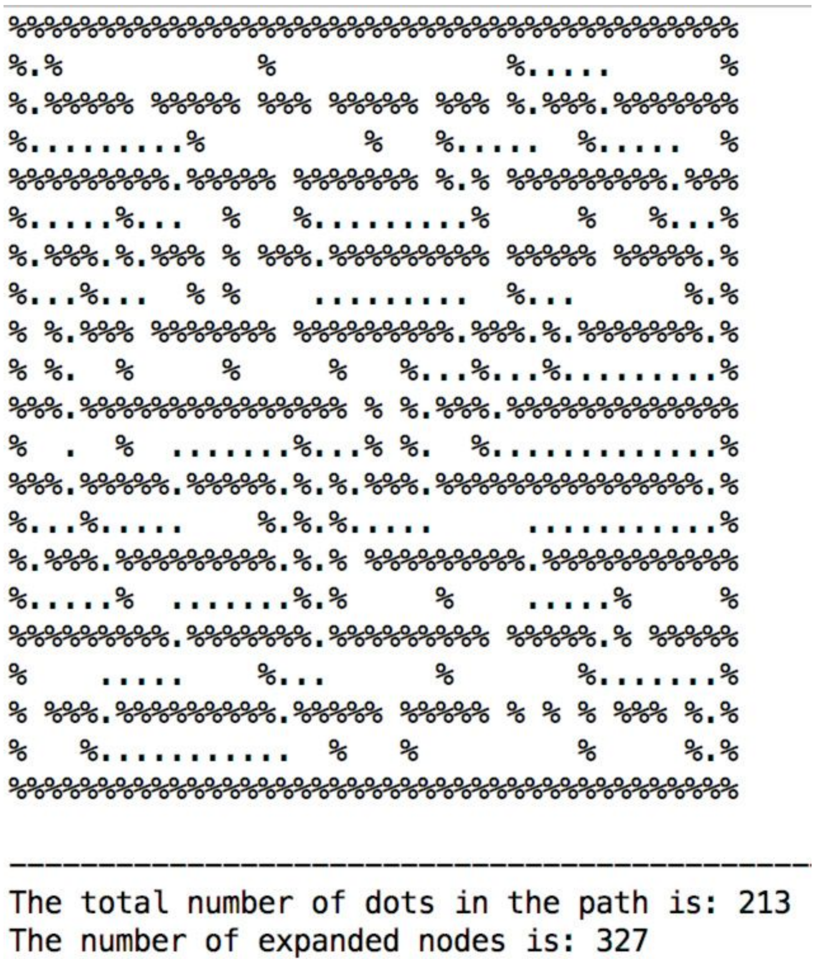


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.%    .......%             %           %
%.%%%%%.%%%%%.%%% %%%%% %%% % %%% %%%%%%%
%.......  %  ...    %   %       %       %
%%%%%%%%% %%%%%.%%%%%%% % % %%%%%%%%% %%%
%     %     %...%         %     %   %   %
% %%% % %%% %.%%% %%%%%%%%% %%%%% %%%%% %
%   %     % %.............  %         % %
% % %%% %%%%%%% %%%%%%%%%.%%% % %%%%%%% %
% %   %     %     %   %...%   %         %
%%% %%%%%%%%%%%%%%% % %.%%% %%%%%%%%%%%%%
%     %         %   % %.  %             %
%%% %%%%% %%%%% % % %%%.%%%%%%%%%%%%%%% %
%   %         % % %    .......          %
% %%% %%%%%%%%% % % %%%%%%%%%.%%%%%%%%%%%
%     %         % %     %    .....%     %
%%%%%%%%% %%%%%%% %%%%%%%%% %%%%%.% %%%%%
%             %         %       %.......%
% %%% %%%%%%%%% %%%%% %%%%% % % % %%% %.%
%   %             %   %         %     %.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

--------------------------------------------
The total number of dots in the path is: 69
The number of expanded nodes is: 342
```

Figure 4 BFS for Medium Maze

Algorithm: DFS
Cost of the path: 213
Number of expanded nodes: 327

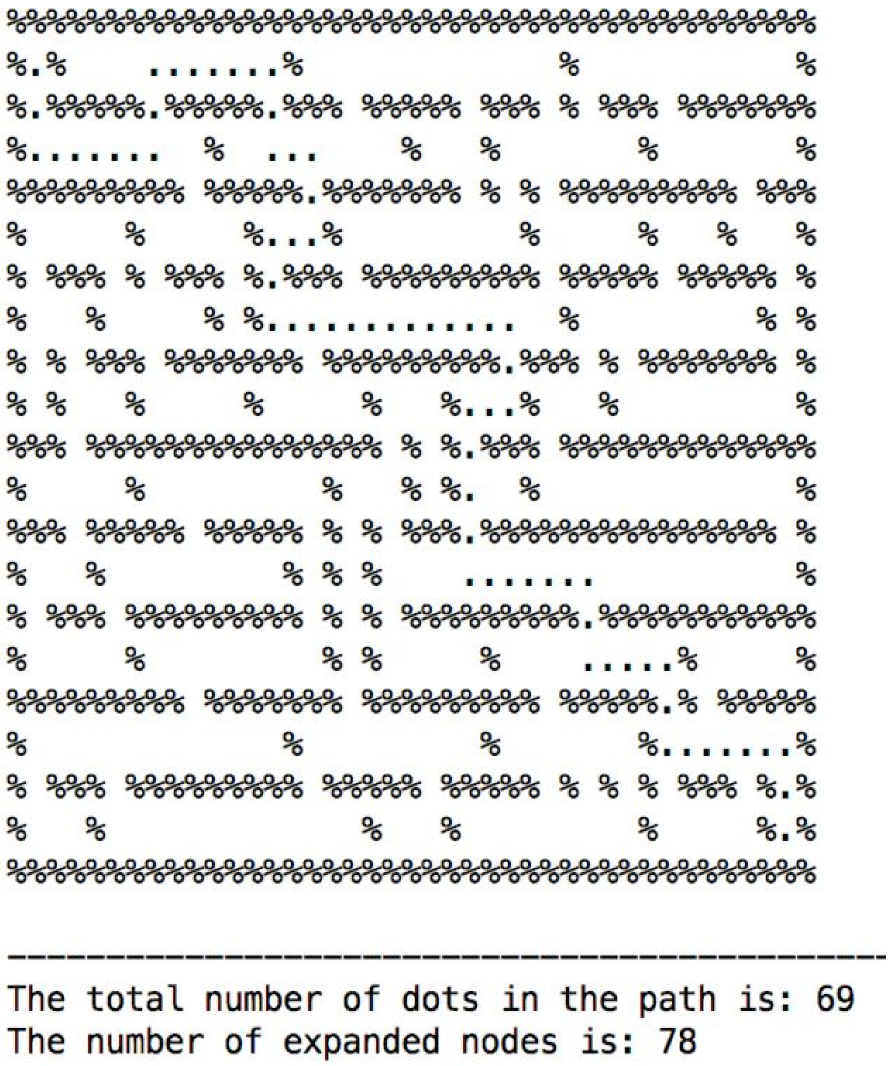


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.%           %             %.....      %
%.%%%%% %%%%% %%% %%%%% %%% %.%%%.%%%%%%%
%.........%         %   %.....  %.....  %
%%%%%%%%%.%%%%% %%%%%%% %.% %%%%%%%%%.%%%
%.....%...  %   %.........%     %   %...%
%.%%%.%.%%% % %%%.%%%%%%%%% %%%%% %%%%%.%
%...%...  % %    .........  %...      %.%
% %.%%% %%%%%%% %%%%%%%%%.%%%.%.%%%%%%%.%
% %.  %     %     %   %...%...%.........%
%%%.%%%%%%%%%%%%%%% % %.%%%.%%%%%%%%%%%%%
%  .  %  .......%...% %.  %.............%
%%%.%%%%%.%%%%%.%.%.%%%.%%%%%%%%%%%%%%%.%
%...%.....    %.%.%.....     ...........%
%.%%%.%%%%%%%%%.%.% %%%%%%%%%.%%%%%%%%%%%
%.....%  .......%.%     %    .....%     %
%%%%%%%%%.%%%%%%%.%%%%%%%%% %%%%%.% %%%%%
%    .....    %...      %       %.......%
% %%%.%%%%%%%%%.%%%%% %%%%% % % % %%% %.%
%   %...........  %   %         %     %.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

---------------------------------------------
The total number of dots in the path is: 213
The number of expanded nodes is: 327
```

Figure 5 DFS for Medium Maze

Algorithm: GBFS
Cost of the path: 69
Number of expanded nodes: 78

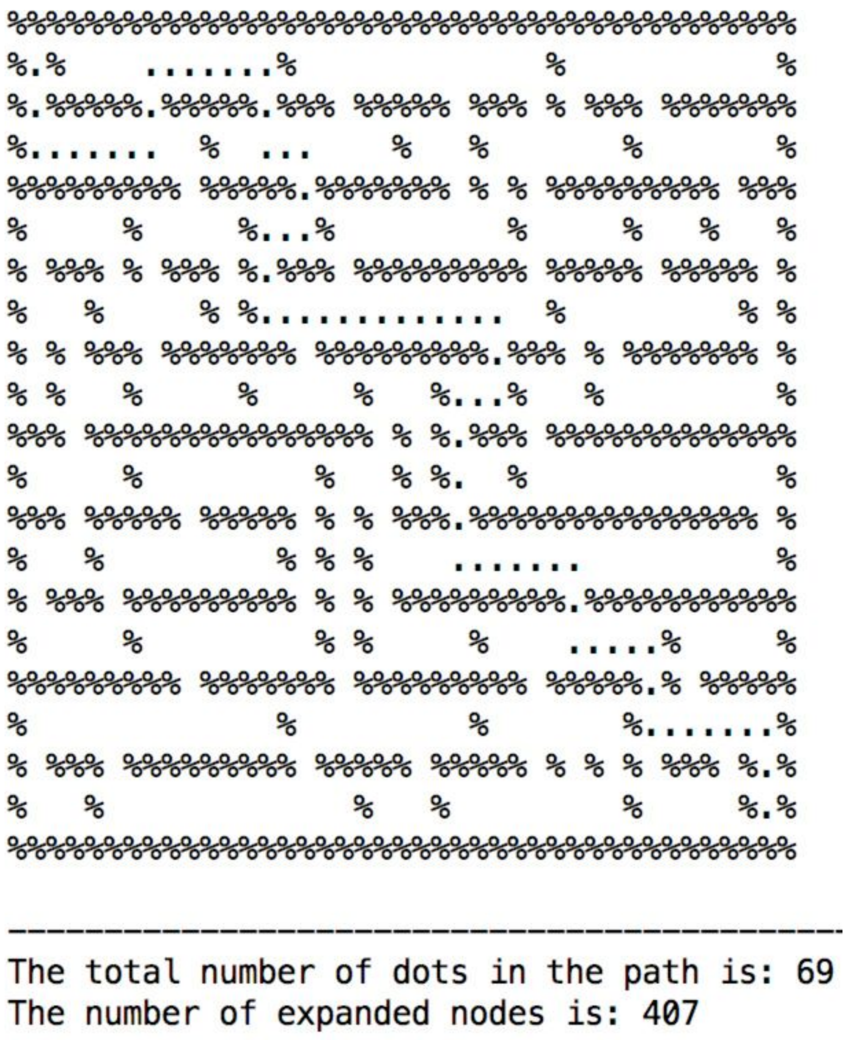


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.%    .......%             %           %
%.%%%%%.%%%%%.%%% %%%%% %%% % %%% %%%%%%%
%.......  %  ...    %   %       %       %
%%%%%%%%% %%%%%.%%%%%%% % % %%%%%%%%% %%%
%     %     %...%         %     %   %   %
% %%% % %%% %.%%% %%%%%%%%% %%%%% %%%%% %
%   %     % %.............  %         % %
% % %%% %%%%%%% %%%%%%%%%.%%% % %%%%%%% %
% %   %     %     %   %...%   %         %
%%% %%%%%%%%%%%%%%% % %.%%% %%%%%%%%%%%%%
%     %         %   % %.  %             %
%%% %%%%% %%%%% % % %%%.%%%%%%%%%%%%%%% %
%   %         % % %    .......          %
% %%% %%%%%%%%% % % %%%%%%%%%.%%%%%%%%%%%
%     %         % %     %    .....%     %
%%%%%%%%% %%%%%%% %%%%%%%%% %%%%%.% %%%%%
%             %         %       %.......%
% %%% %%%%%%%%% %%%%% %%%%% % % % %%% %.%
%   %             %   %         %     %.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


--------------------------------------------
The total number of dots in the path is: 69
The number of expanded nodes is: 78
```

Figure 6 GBFS for Medium Maze

Algorithm: A*
Cost of the path: 69
Number of expanded nodes: 407



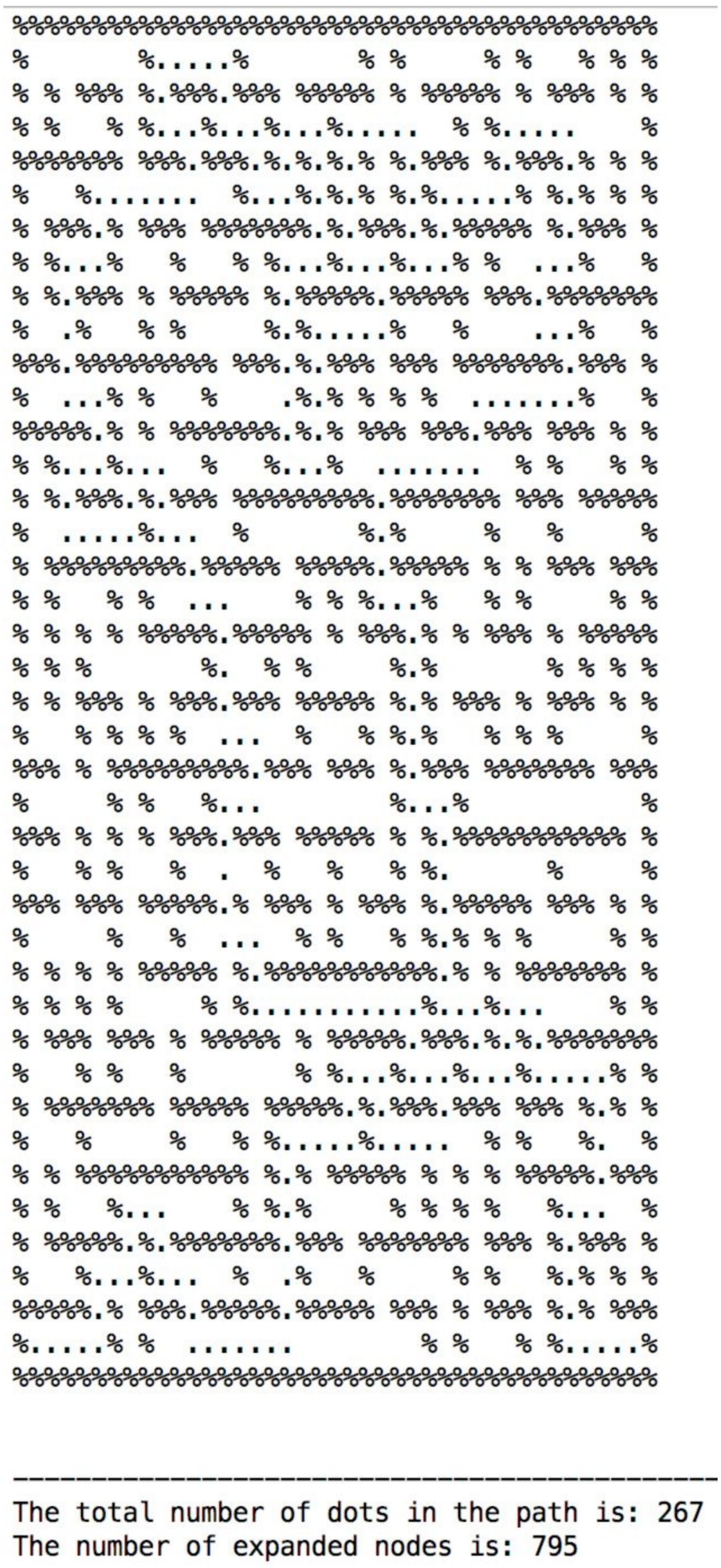
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.%    .......%             %           %
%.%%%%%.%%%%%.%%% %%%%% %%% % %%% %%%%%%%
%.......  %  ...    %   %       %       %
%%%%%%%%% %%%%%.%%%%%%% % % %%%%%%%%% %%%
%     %     %...%         %     %   %   %
% %%% % %%% %.%%% %%%%%%%%% %%%%% %%%%% %
%   %     % %.............  %         % %
% % %%% %%%%%%% %%%%%%%%%.%%% % %%%%%%% %
% %   %     %     %   %...%   %         %
%%% %%%%%%%%%%%%%%% % %.%%% %%%%%%%%%%%%%
%     %         %   % %.  %             %
%%% %%%%% %%%%% % % %%%.%%%%%%%%%%%%%%% %
%   %         % % %    .......          %
% %%% %%%%%%%%% % % %%%%%%%%%.%%%%%%%%%%%
%     %         % %     %    .....%     %
%%%%%%%%% %%%%%%% %%%%%%%%% %%%%%.% %%%%%
%             %         %       %.......%
% %%% %%%%%%%%% %%%%% %%%%% % % % %%% %.%
%   %             %   %         %     %.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-----------------------------------------
The total number of dots in the path is: 69
The number of expanded nodes is: 407
```

Figure 7 A* for Medium Maze

1.5.3 Outcome of Big Maze

Algorithm: BFS
Cost of the path: 267
Number of expanded nodes: 795

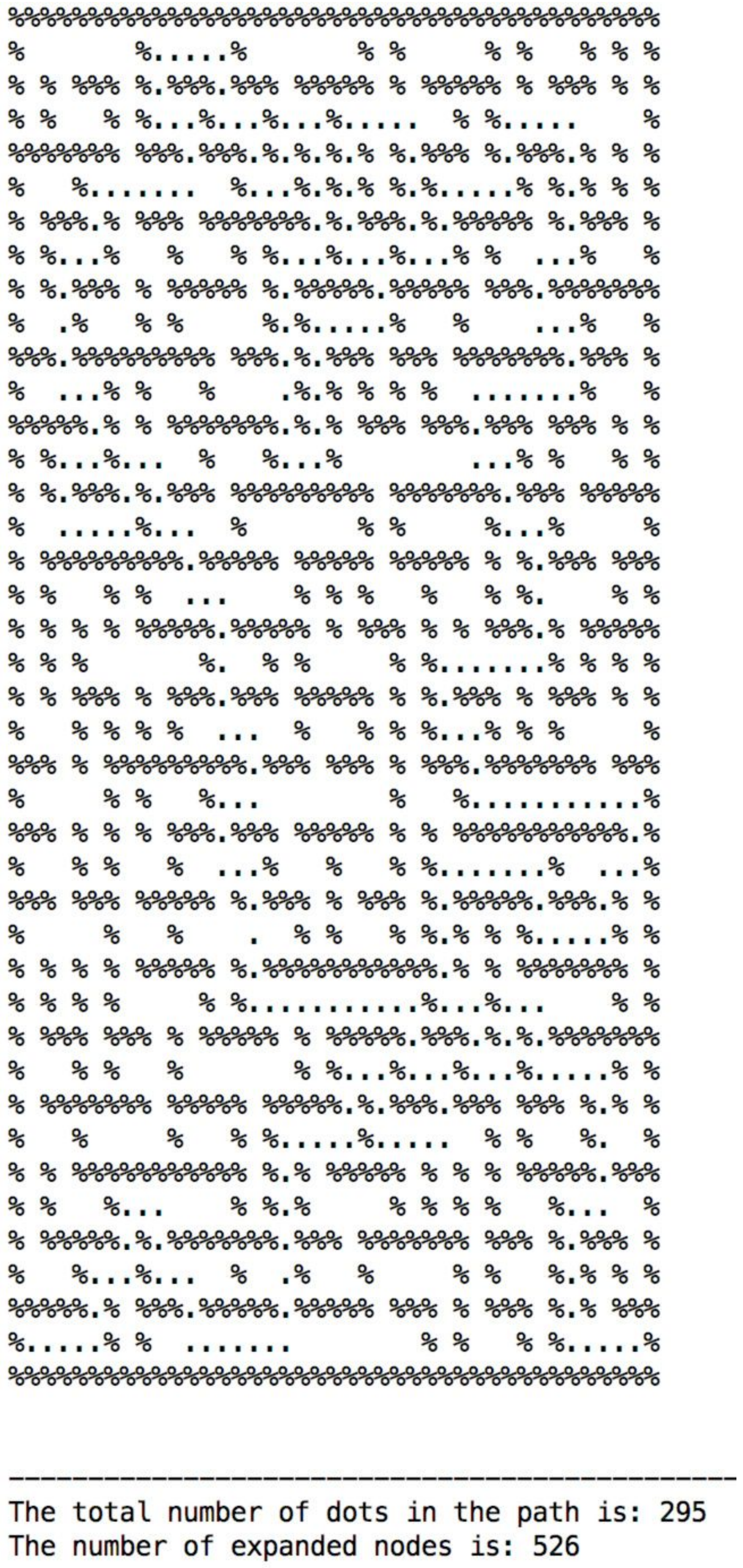


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       %.....%       % %     % %   % % %
% % %%% %.%%%.%%% %%%%% % %%%%% % %%% % %
% %   % %...%...%...%.....  % %.....    %
%%%%%%% %%%.%%%.%.%.%.% %.%%% %.%%%.% % %
%   %.......  %...%.%.% %.%.....% %.% % %
% %%%.% %%% %%%%%%%.%.%%%.%.%%%%% %.%%% %
% %...%   %   % %...%...%...% %  ...%   %
% %.%%% % %%%%% %.%%%%%.%%%%% %%%.%%%%%%%
%  .%   % %     %.%.....%   %    ...%   %
%%%.%%%%%%%%% %%%.%.%%% %%% %%%%%%%.%%% %
%  ...% %   %    .%.% % % %  .......%   %
%%%%%.% % %%%%%%%.%.% %%% %%%.%%% %%% % %
% %...%...  %   %...%  .......  % %   % %
% %.%%%.%.%%% %%%%%%%%%.%%%%%%% %%% %%%%%
%  .....%...  %       %.%     %   %     %
% %%%%%%%%%.%%%%% %%%%%.%%%%% % % %%% %%%
% %   % %  ...    % % %...%   % %     % %
% % % % %%%%%.%%%%% % %%%.% % %%% % %%%%%
% % %       %.  % %     %.%       % % % %
% % %%% % %%%.%%% %%%%% %.% %%% % %%% % %
%   % % % %  ...  %   % %.%   % % %     %
%%% % %%%%%%%%%.%%% %%% %.%%% %%%%%%% %%%
%     % %   %...        %...%           %
%%% % % % %%%.%%% %%%%% % %.%%%%%%%%%%% %
%   % %   %  .  %   %   % %.      %     %
%%% %%% %%%%%.% %%% % %%% %.%%%%% %%% % %
%     %   %  ...  % %   % %.% % %     % %
% % % % %%%%% %.%%%%%%%%%%%.% % %%%%%%% %
% % % %     % %...........%...%...    % %
% %%% %%% % %%%%% % %%%%%.%%%.%.%.%%%%%%%
%   % %   %       % %...%...%...%.....% %
% %%%%%%% %%%%% %%%%%.%.%%%.%%% %%% %.% %
%   %     %   % %.....%.....  % %   %.  %
% % %%%%%%%%%%% %.% %%%%% % % % %%%%%.%%%
% %   %...    % %.%     % % % %   %...  %
% %%%%%.%.%%%%%%%.%%% %%%%%%% %%% %.%%% %
%   %...%...  %  .%   %     % %   %.% % %
%%%%%.% %%%.%%%%%.%%%%% %%% % %%% %.% %%%
%.....% %  .......        % %   % %.....%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


---------------------------------------------
The total number of dots in the path is: 267
The number of expanded nodes is: 795
```

Figure 8 BFS for Big Maze

Algorithm: DFS
Cost of the path: 295
Number of expanded nodes: 526

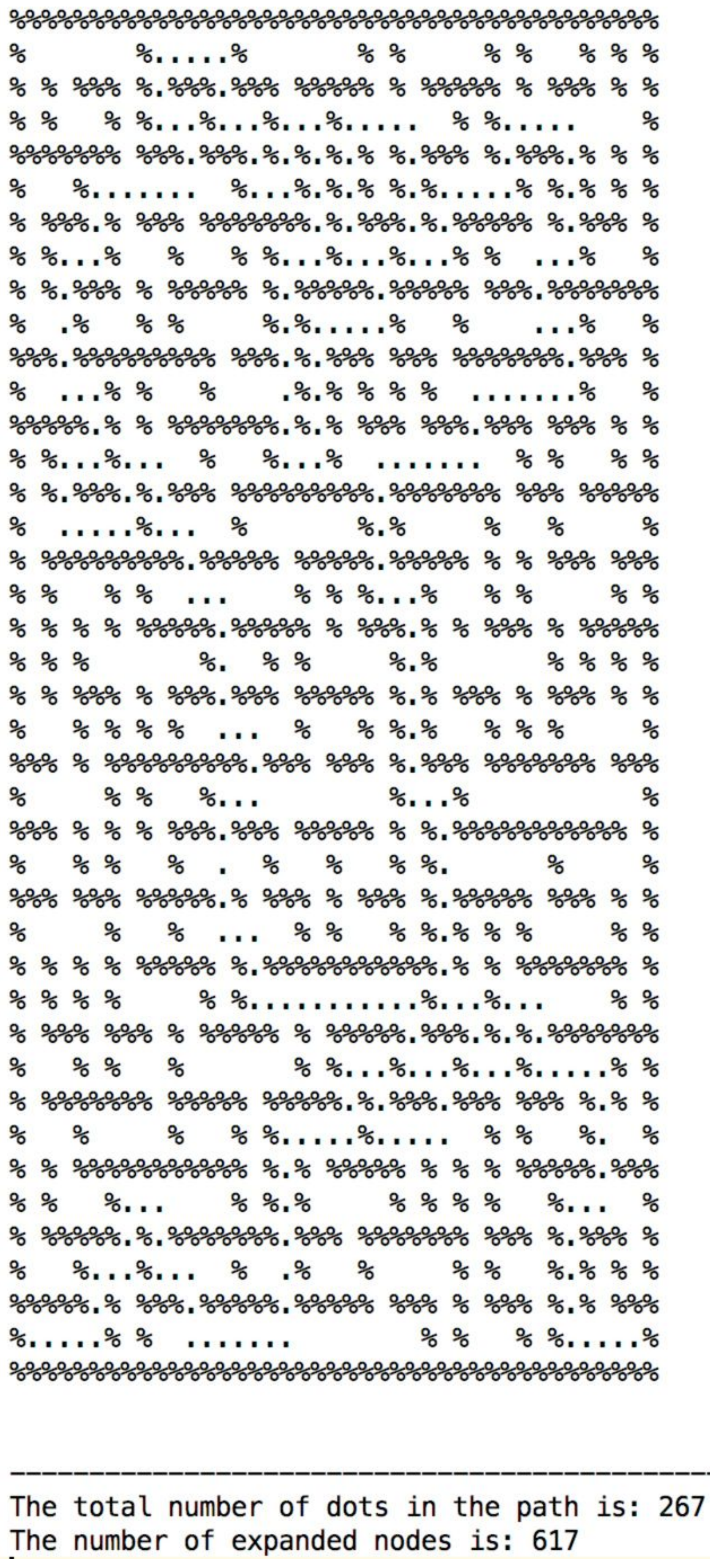


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       %.....%       % %     % %   % % %
% % %%% %.%%%.%%% %%%%% % %%%%% % %%% % %
% %   % %...%...%...%.....  % %.....    %
%%%%%%% %%%.%%%.%.%.%.% %.%%% %.%%%.% % %
%   %.......  %...%.%.% %.%.....% %.% % %
% %%%.% %%% %%%%%%%.%.%%%.%.%%%%% %.%%% %
% %...%   %   % %...%...%...% %  ...%   %
% %.%%% % %%%%% %.%%%%%.%%%%% %%%.%%%%%%%
%  .%   % %     %.%.....%   %    ...%   %
%%%.%%%%%%%%% %%%.%.%%% %%% %%%%%%%.%%% %
%  ...% %   %    .%.% % % %  .......%   %
%%%%%.% % %%%%%%%.%.% %%% %%%.%%% %%% % %
% %...%...  %   %...%        ...% %   % %
% %.%%%.%.%%% %%%%%%%%% %%%%%%%.%%% %%%%%
%  .....%...  %       % %     %...%     %
% %%%%%%%%%.%%%%% %%%%% %%%%% % %.%%% %%%
% %   % %  ...    % % %   %   % %.    % %
% % % % %%%%%.%%%%% % %%% % % %%%.% %%%%%
% % %       %.  % %     % %.......% % % %
% % %%% % %%%.%%% %%%%% % %.%%% % %%% % %
%   % % % %  ...  %   % % %...% % %     %
%%% % %%%%%%%%%.%%% %%% % %%%.%%%%%%% %%%
%     % %   %...        %   %...........%
%%% % % % %%%.%%% %%%%% % % %%%%%%%%%%%.%
%   % %   %  ...%   %   % %.......%  ...%
%%% %%% %%%%% %.%%% % %%% %.%%%%%.%%%.% %
%     %   %    .  % %   % %.% % %.....% %
% % % % %%%%% %.%%%%%%%%%%%.% % %%%%%%% %
% % % %     % %...........%...%...    % %
% %%% %%% % %%%%% % %%%%%.%%%.%.%.%%%%%%%
%   % %   %       % %...%...%...%.....% %
% %%%%%%% %%%%% %%%%%.%.%%%.%%% %%% %.% %
%   %     %   % %.....%.....  % %   %.  %
% % %%%%%%%%%%% %.% %%%%% % % % %%%%%.%%%
% %   %...    % %.%     % % % %   %...  %
% %%%%%.%.%%%%%%%.%%% %%%%%%% %%% %.%%% %
%   %...%...  %  .%   %     % %   %.% % %
%%%%%.% %%%.%%%%%.%%%%% %%% % %%% %.% %%%
%.....% %  .......        % %   % %.....%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


-----------------------------------------------
The total number of dots in the path is: 295
The number of expanded nodes is: 526
```

Figure 9 DFS for Big Maze

Algorithm: GBFS
Cost of the path: 267
Number of expanded nodes: 617

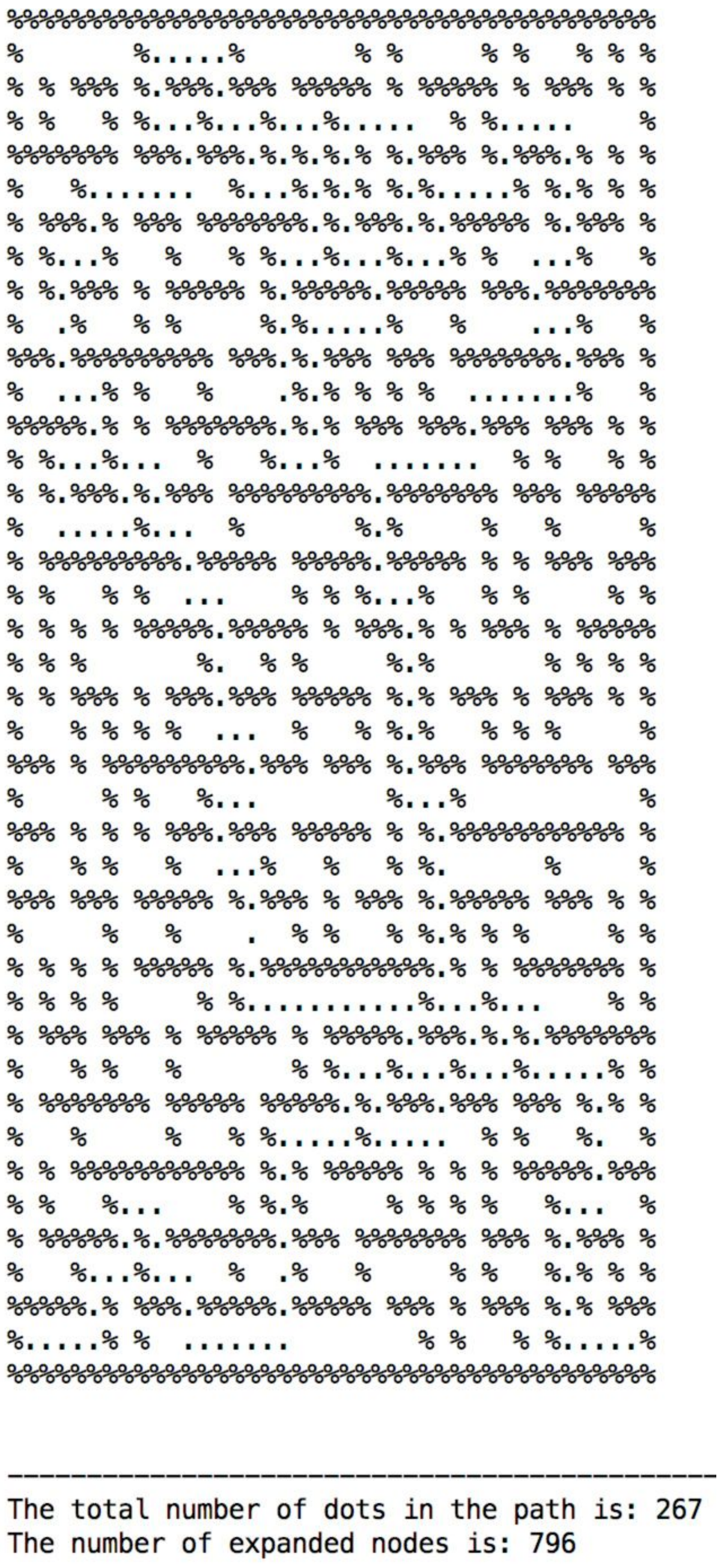


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       %.....%       % %     % %   % % %
% % %%% %.%%%.%%% %%%%% % %%%%% % %%% % %
% %   % %...%...%...%.....  % %.....    %
%%%%%%% %%%.%%%.%.%.%.% %.%%% %.%%%.% % %
%   %.......  %...%.%.% %.%.....% %.% % %
% %%%.% %%% %%%%%%%.%.%%%.%.%%%%% %.%%% %
% %...%   %   % %...%...%...% %  ...%   %
% %.%%% % %%%%% %.%%%%%.%%%%% %%%.%%%%%%%
%  .%   % %     %.%.....%   %    ...%   %
%%%.%%%%%%%%% %%%.%.%%% %%% %%%%%%%.%%% %
%  ...% %   %    .%.% % % %  .......%   %
%%%%%.% % %%%%%%%.%.% %%% %%%.%%% %%% % %
% %...%...  %   %...%  .......  % %   % %
% %.%%%.%.%%% %%%%%%%%%.%%%%%%% %%% %%%%%
%  .....%...  %       %.%     %   %     %
% %%%%%%%%%.%%%%% %%%%%.%%%%% % % %%% %%%
% %   % %  ...    % % %...%   % %     % %
% % % % %%%%%.%%%%% % %%%.% % %%% % %%%%%
% % %       %.  % %     %.%       % % % %
% % %%% % %%%.%%% %%%%% %.% %%% % %%% % %
%   % % % %  ...  %   % %.%   % % %     %
%%% % %%%%%%%%%.%%% %%% %.%%% %%%%%%% %%%
%     % %   %...        %...%           %
%%% % % % %%%.%%% %%%%% % %.%%%%%%%%%%% %
%   % %   %  .  %   %   % %.      %     %
%%% %%% %%%%%.% %%% % %%% %.%%%%% %%% % %
%     %   %  ...  % %   % %.% % %     % %
% % % % %%%%% %.%%%%%%%%%%%.% % %%%%%%% %
% % % %     % %...........%...%...    % %
% %%% %%% % %%%%% % %%%%%.%%%.%.%.%%%%%%%
%   % %   %       % %...%...%...%.....% %
% %%%%%%% %%%%% %%%%%.%.%%%.%%% %%% %.% %
%   %     %   % %.....%.....  % %   %.  %
% % %%%%%%%%%%% %.% %%%%% % % % %%%%%.%%%
% %   %...    % %.%     % % % %   %...  %
% %%%%%.%.%%%%%%%.%%% %%%%%%% %%% %.%%% %
%   %...%...  %  .%   %     % %   %.% % %
%%%%%.% %%%.%%%%%.%%%%% %%% % %%% %.% %%%
%.....% %  .......        % %   % %.....%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


----------------------------------------
The total number of dots in the path is: 267
The number of expanded nodes is: 617
```

Figure 10 GBFS for Big Maze

Algorithm: A*
Cost of the path: 267
Number of expanded nodes: 796



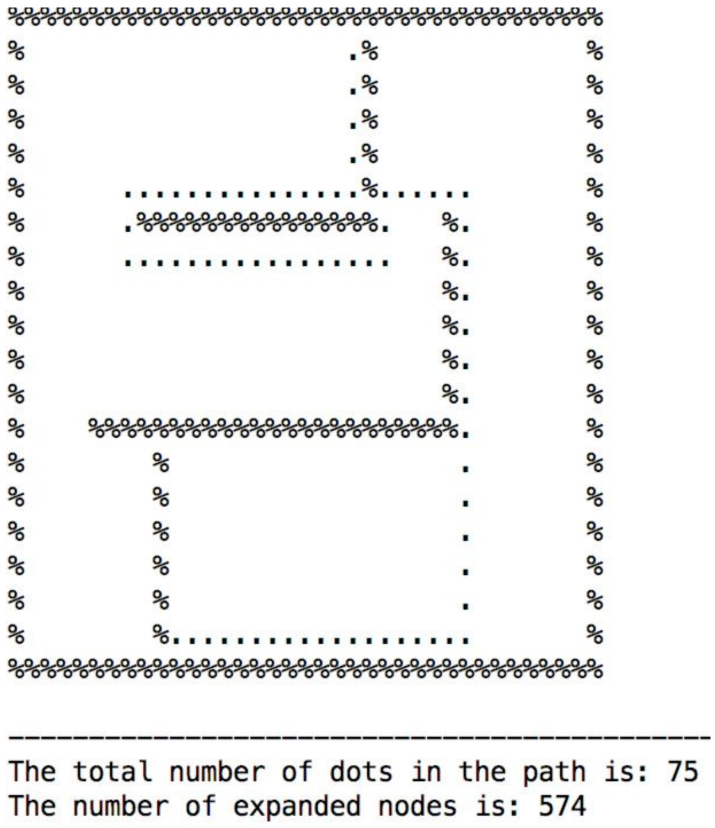
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       %.....%       % %     % %   % % %
% % %%% %.%%%.%%% %%%%% % %%%%% % %%% % %
% %   % %...%...%...%.....  % %.....    %
%%%%%%% %%%.%%%.%.%.%.% %.%%% %.%%%.% % %
%   %.......  %...%.%.% %.%.....% %.% % %
% %%%.% %%% %%%%%%%.%.%%%.%.%%%%% %.%%% %
% %...%   %   % %...%...%...% %  ...%   %
% %.%%% % %%%%% %.%%%%%.%%%%% %%%.%%%%%%%
%  .%   % %     %.%.....%   %    ...%   %
%%%.%%%%%%%%% %%%.%.%%% %%% %%%%%%%.%%% %
%  ...% %   %    .%.% % % %  .......%   %
%%%%%.% % %%%%%%%.%.% %%% %%%.%%% %%% % %
% %...%...  %   %...%  .......  % %   % %
% %.%%%.%.%%% %%%%%%%%%.%%%%%%% %%% %%%%%
%  .....%...  %       %.%     %   %     %
% %%%%%%%%%.%%%%% %%%%%.%%%%% % % %%% %%%
% %   % %  ...    % % %...%   % %     % %
% % % % %%%%%.%%%%% % %%%.% % %%% % %%%%%
% % %       %.  % %     %.%       % % % %
% % %%% % %%%.%%% %%%%% %.% %%% % %%% % %
%   % % % %  ...  %   % %.%   % % %     %
%%% % %%%%%%%%%.%%% %%% %.%%% %%%%%%% %%%
%     % %   %...        %...%           %
%%% % % % %%%.%%% %%%%% % %.%%%%%%%%%%% %
%   % %   %  ...%   %   % %.      %     %
%%% %%% %%%%% %.%%% % %%% %.%%%%% %%% % %
%     %   %    .  % %   % %.% % %     % %
% % % % %%%%% %.%%%%%%%%%%%.% % %%%%%%% %
% % % %     % %...........%...%...    % %
% %%% %%% % %%%%% % %%%%%.%%%.%.%.%%%%%%%
%   % %   %       % %...%...%...%.....% %
% %%%%%%% %%%%% %%%%%.%.%%%.%%% %%% %.% %
%   %     %   % %.....%.....  % %   %.  %
% % %%%%%%%%%%% %.% %%%%% % % % %%%%%.%%%
% %   %...    % %.%     % % % %   %...  %
% %%%%%.%.%%%%%%%.%%% %%%%%%% %%% %.%%% %
%   %...%...  %  .%   %     % %   %.% % %
%%%%%.% %%%.%%%%%.%%%%% %%% % %%% %.% %%%
%.....% %  .......        % %   % %.....%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


----------------------------------------
The total number of dots in the path is: 267
The number of expanded nodes is: 796
```

Figure 11 A* for Big Maze

1.5.4 Outcome of Open Maze

Algorithm: BFS
Cost of the path: 75
Number of expanded nodes: 574

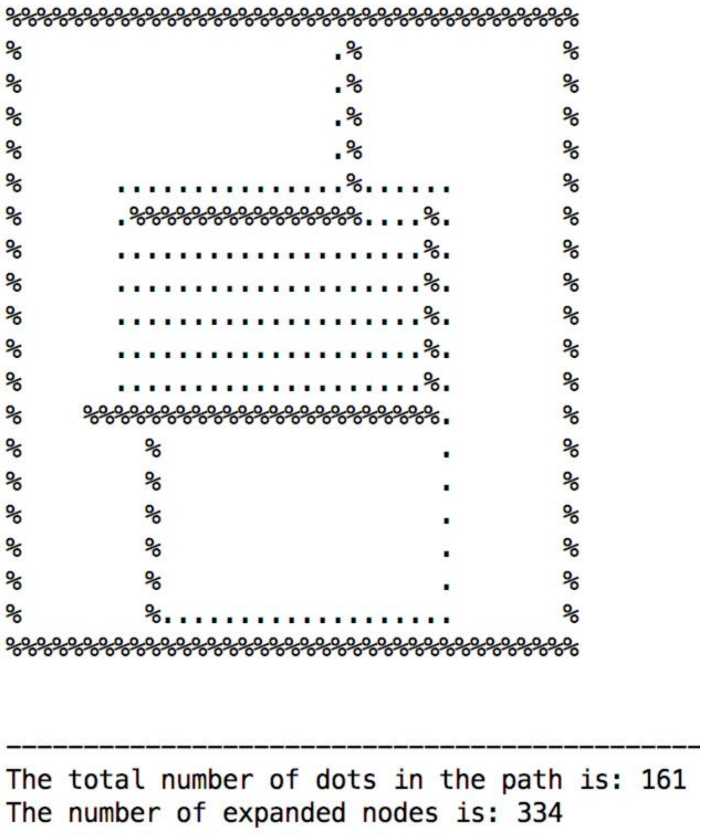


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    .%             %
%                    .%             %
%                    .%             %
%                    .%             %
%      ...............%......       %
%      .%%%%%%%%%%%%%%%.   %.       %
%      .................   %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%    %%%%%%%%%%%%%%%%%%%%%%%.       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %...................       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-------------------------------------------
The total number of dots in the path is: 75
The number of expanded nodes is: 574
```

Figure 12 BFS for Open Maze

Algorithm: DFS
Cost of the path: 161
Number of expanded nodes: 334

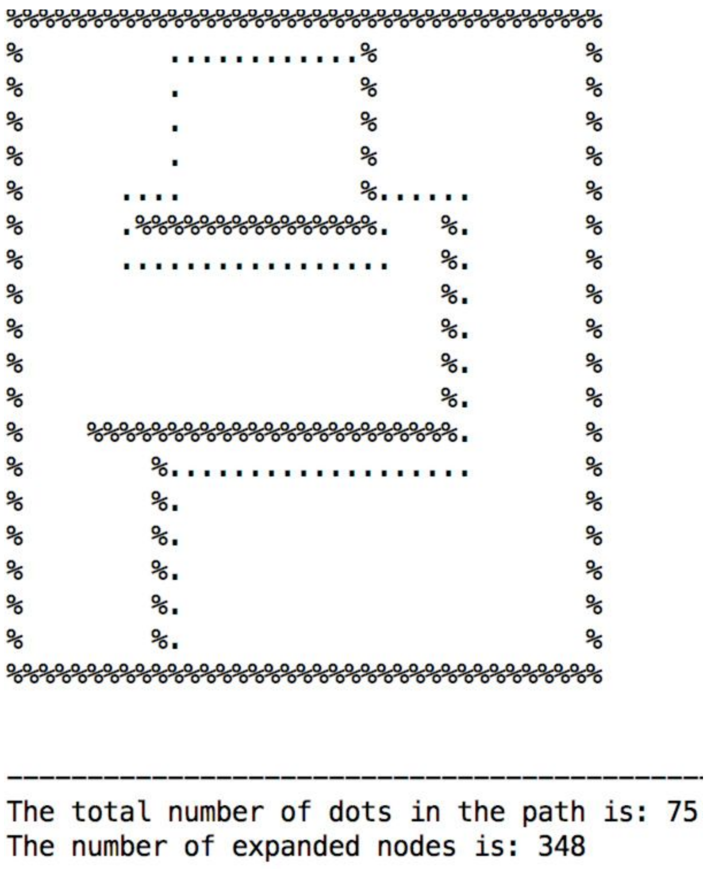


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    .%             %
%                    .%             %
%                    .%             %
%                    .%             %
%      ...............%......       %
%      .%%%%%%%%%%%%%%%....%.       %
%      ....................%.       %
%      ....................%.       %
%      ....................%.       %
%      ....................%.       %
%      ....................%.       %
%    %%%%%%%%%%%%%%%%%%%%%%%.       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %                  .       %
%        %...................       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


---------------------------------------------
The total number of dots in the path is: 161
The number of expanded nodes is: 334
```

Figure 13 DFS for Open Maze

Algorithm: GBFS
Cost of the path: 75
Number of expanded nodes: 348

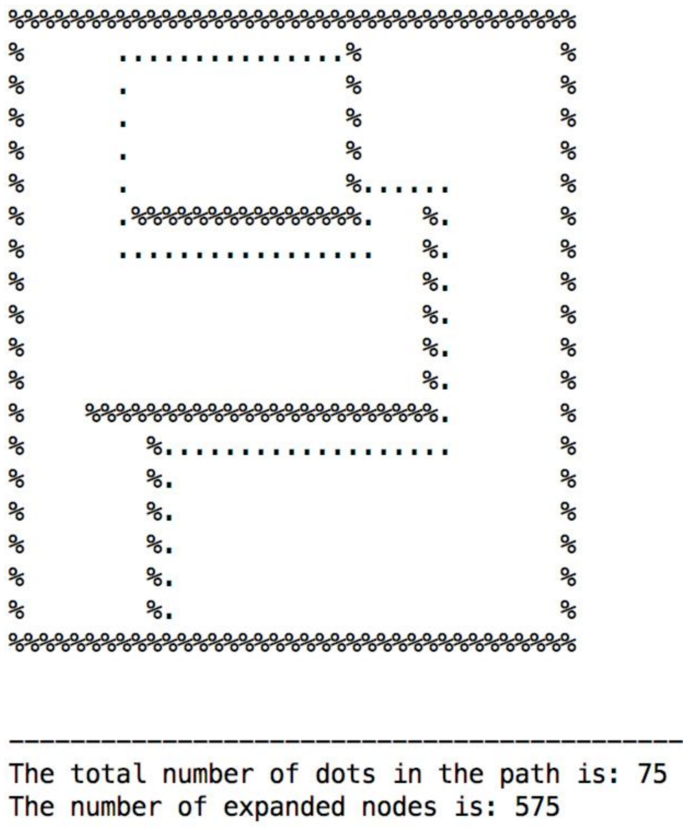


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         ............%             %
%         .           %             %
%         .           %             %
%         .           %             %
%      ....           %......       %
%      .%%%%%%%%%%%%%%%.   %.       %
%      .................   %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%    %%%%%%%%%%%%%%%%%%%%%%%.       %
%        %...................       %
%        %.                         %
%        %.                         %
%        %.                         %
%        %.                         %
%        %.                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


-------------------------------------------
The total number of dots in the path is: 75
The number of expanded nodes is: 348
```

Figure 14 GBFS for Open Maze

Algorithm: A*
Cost of the path: 75
Number of expanded nodes: 348



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      ...............%             %
%      .              %             %
%      .              %             %
%      .              %             %
%      .              %......       %
%      .%%%%%%%%%%%%%%%.   %.       %
%      .................   %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%                          %.       %
%    %%%%%%%%%%%%%%%%%%%%%%%.       %
%        %...................       %
%        %.                         %
%        %.                         %
%        %.                         %
%        %.                         %
%        %.                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-----------------------------------------
The total number of dots in the path is: 75
The number of expanded nodes is: 575
```

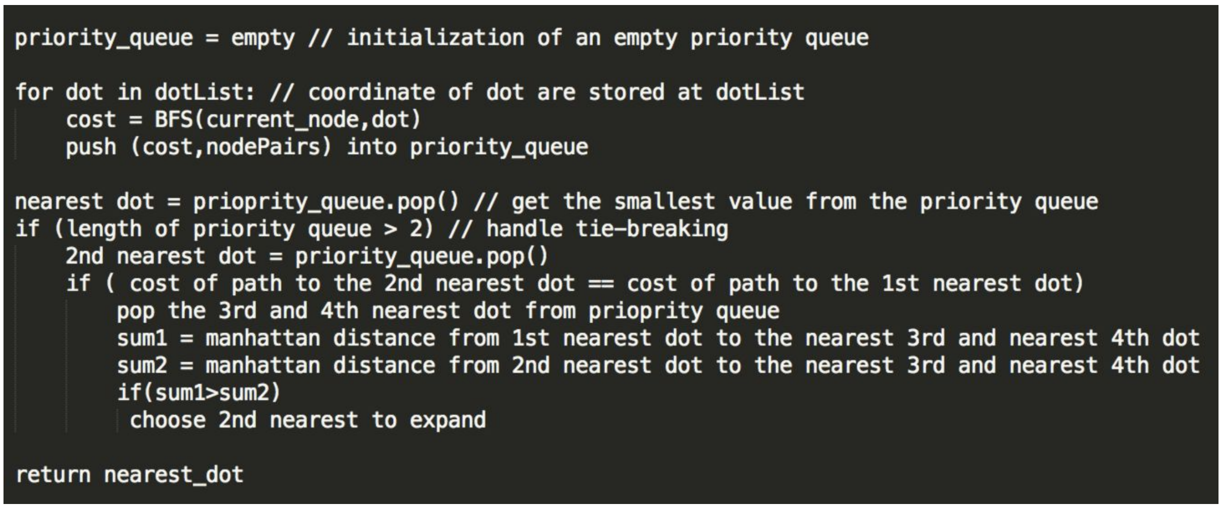
Figure 15 A* for Open Maze

2. Search with Multi-dots

2.1 Overview

To find a shortest path to eat all dots in the maze, we need to change our heuristic in part1. In this problem, our strategy is try to find out the nearest dot frist for the current node. Then we employ A* to find the optimal path to the nearest dot for current node. So it is an iterative process to empoly A* if we want to find a path to all dots in the maze.

First, we design a function: $findAllDots(filename)$ to find all dots in the maze and store them as a $tuple\ (x_i, y_i)$ in a list and return the list back. Then, we use BFS to find the actual cost of path to all dots in the list in order to select the smallest one. And a priority queue is implemented to find the smallest cost. The pesudocode is presented as follows:



```
priority_queue = empty // initialization of an empty priority queue

for dot in dotList: // coordinate of dot are stored at dotList
    cost = BFS(current_node,dot)
    push (cost,nodePairs) into priority_queue

nearest dot = priopritу_queue.pop() // get the smallest value from the priority queue
if (length of priority queue > 2) // handle tie-breaking
    2nd nearest dot = priority_queue.pop()
    if ( cost of path to the 2nd nearest dot == cost of path to the 1st nearest dot)
        pop the 3rd and 4th nearest dot from prioprity queue
        sum1 = manhattan distance from 1st nearest dot to the nearest 3rd and nearest 4th dot
        sum2 = manhattan distance from 2nd nearest dot to the nearest 3rd and nearest 4th dot
        if(sum1>sum2)
         choose 2nd nearest to expand

return nearest_dot
```

Figure 16 Pesudocode for Finding the Nearest Dot

Once we get the nearest dot, we set it as the goal in A* search algorithm and apparently the currrent node is the start node.(a tiny A* search process) As mentioned before, the first path found at A* maybe not the optimal path, we need to try all possible path to the goal and when the priority queue is empty then stop and return the path has lowest cost. So it would certainly expand lots of nodes in this process. Back to the goal of finding an optiaml path to eat all dots in the maze, once a tiny A* search is finished, we repeat this process and find the next nearest dot as the goal in the next tiny A* search. So if we do the search in this way, we will finally find all dots in the maze.

The mentioned proces is the detailed part of the search. Now if we look at the general process of the search, once we find a dot in the search, we will remove it from the dot list. So the goal state is that the number of dot list is zero. That's also the condition when we end up the while loop to call A* search on the next dot. So we actually use the actual cost of the path to the nearest cost as heuristic for the A* algorithm for search.

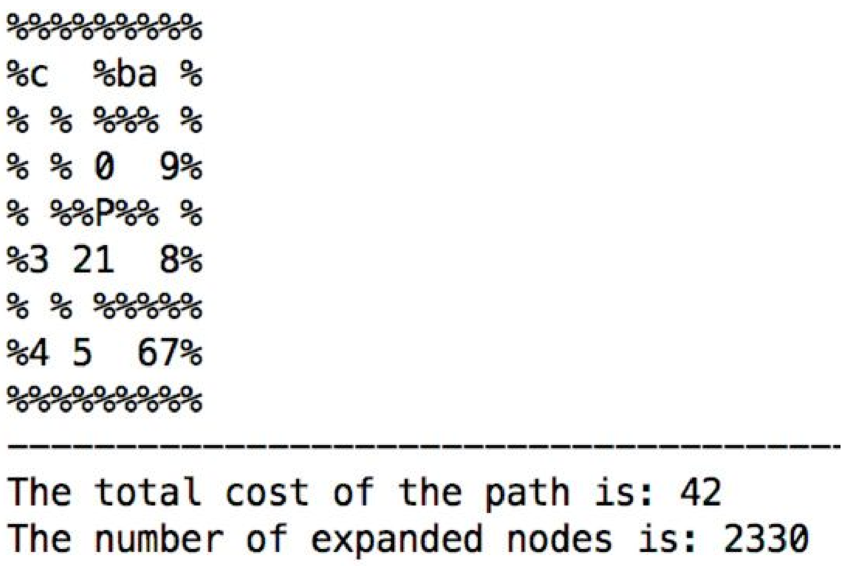
As for the admissibility of the heuristic for A*, we discuss it in several aspects. First, the definition of admissibility is that for every node n, the heuristic value should no larger than the true cost to reach the goal from node n. In our solution, we use the actual cost of the path to find the nearest dot, and it is obvious that manhattan distance between two nodes in grid is always no larger than the actual cost in the path between them. Second, we have lots of choose as we have multi-goals in the grid. How can we promise that the cost to a dot and its sequence is admissible? We think the right way to do that is try to find the nearest dot first and minimize the cost from one dot to another dot since our final goal is trying to find all dots in one path. This heuristic for the whole process should be less than the cost of connecting all dots to the current node in the grid. So if every move is admissible, the sum of them should be admissible.

(**Hint**: 1. The solution of path is labeled in the order: ‘0-9’,’a-z’ and ‘A-Z’;

2. The number of expanded nodes include the expanded node from the process of finding the nearest dot plus the expanded node from A* search.)

2.2 Tiny Search

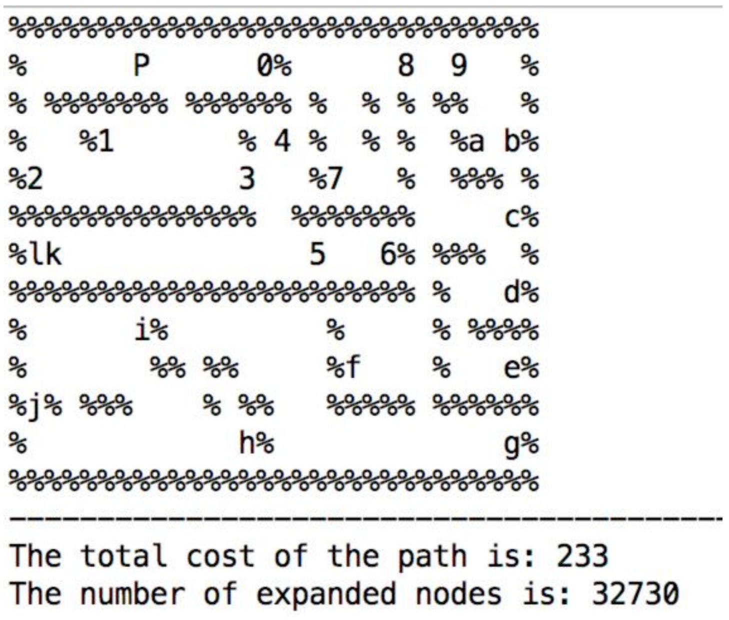
Algorithm: A*
Cost of the path: 42
Number of expanded nodes: 2330



```
%%%%%%%%%
%c  %ba %
% % %%% %
% % 0  9%
% %%P%% %
%3 21  8%
% % %%%%%
%4 5  67%
%%%%%%%%%
---------------------------------------
The total cost of the path is: 42
The number of expanded nodes is: 2330
```

Figure 17 Tiny Search

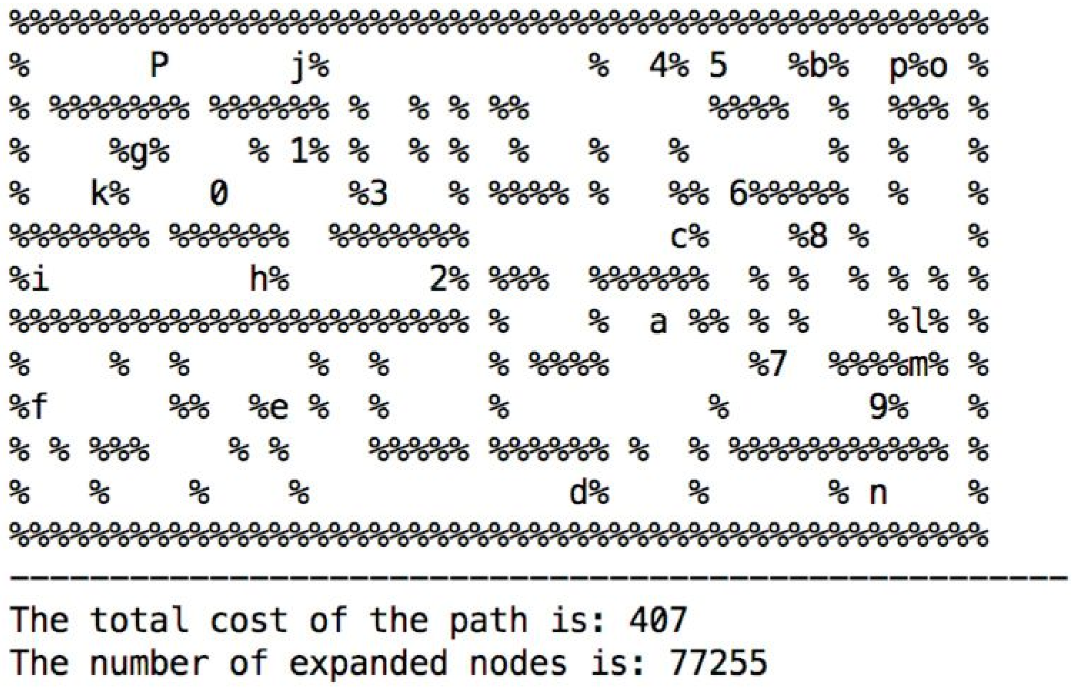
Algorithm: A*
Cost of the path: 233
Number of expanded nodes: 32730



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      P      0%      8  9   %
% %%%%%%% %%%%%% %  % % %%   %
%   %1       % 4 %  % %  %a b%
%2           3   %7   %  %%% %
%%%%%%%%%%%%%%  %%%%%%%     c%
%lk              5   6% %%%  %
%%%%%%%%%%%%%%%%%%%%%%% %   d%
%      i%         %     % %%%%
%       %% %%     %f    %   e%
%j% %%%    % %%   %%%%% %%%%%%
%            h%             g%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
------------------------------
The total cost of the path is: 233
The number of expanded nodes is: 32730
```

Figure 18 Small Search

Algorithm: A*
Cost of the path: 407
Number of expanded nodes: 77255



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      P      j%             %  4% 5   %b%  p%o %
% %%%%%%% %%%%%% %  % % %%         %%%%  %  %%% %
%    %g%    % 1% %  % %  %   %   %       %  %   %
%   k%    0      %3   % %%%% %   %% 6%%%%%  %   %
%%%%%%% %%%%%%  %%%%%%%          c%    %8 %     %
%i          h%       2% %%%  %%%%%%  % %  % % % %
%%%%%%%%%%%%%%%%%%%%%%% %    %  a %% % %    %l% %
%    %  %      %  %     % %%%%       %7  %%%%m% %
%f      %%  %e %  %     %          %       9%   %
% % %%%    % %    %%%%% %%%%%% %  % %%%%%%%%%%% %
%   %    %    %             d%    %      % n    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-----------------------------------------------------
The total cost of the path is: 407
The number of expanded nodes is: 77255
```

Figure 19 Medium Search

Part 2 Rubik's Cube

2.1 **Solving the Cube without Rotation Invariance**

2.1.1 Implementation

We design a $face$ object and a $cube$ Object in order to represent each element of the cube in the problem. The $cube$ object includes an array of face objects. In ech face object, we record the colors in that face, and the 8 adjacent small piece on 4 different adjacent faces. a funciton named as $getGraph(\,)$ which could return a 6x4 representation of the color matrix. (i.e. 6 faces with 4 colors in the face). In the cube without 3D invariance, the index for each face and the corresponding small pieces' indices are as manually defined as required in the instruction as the following:

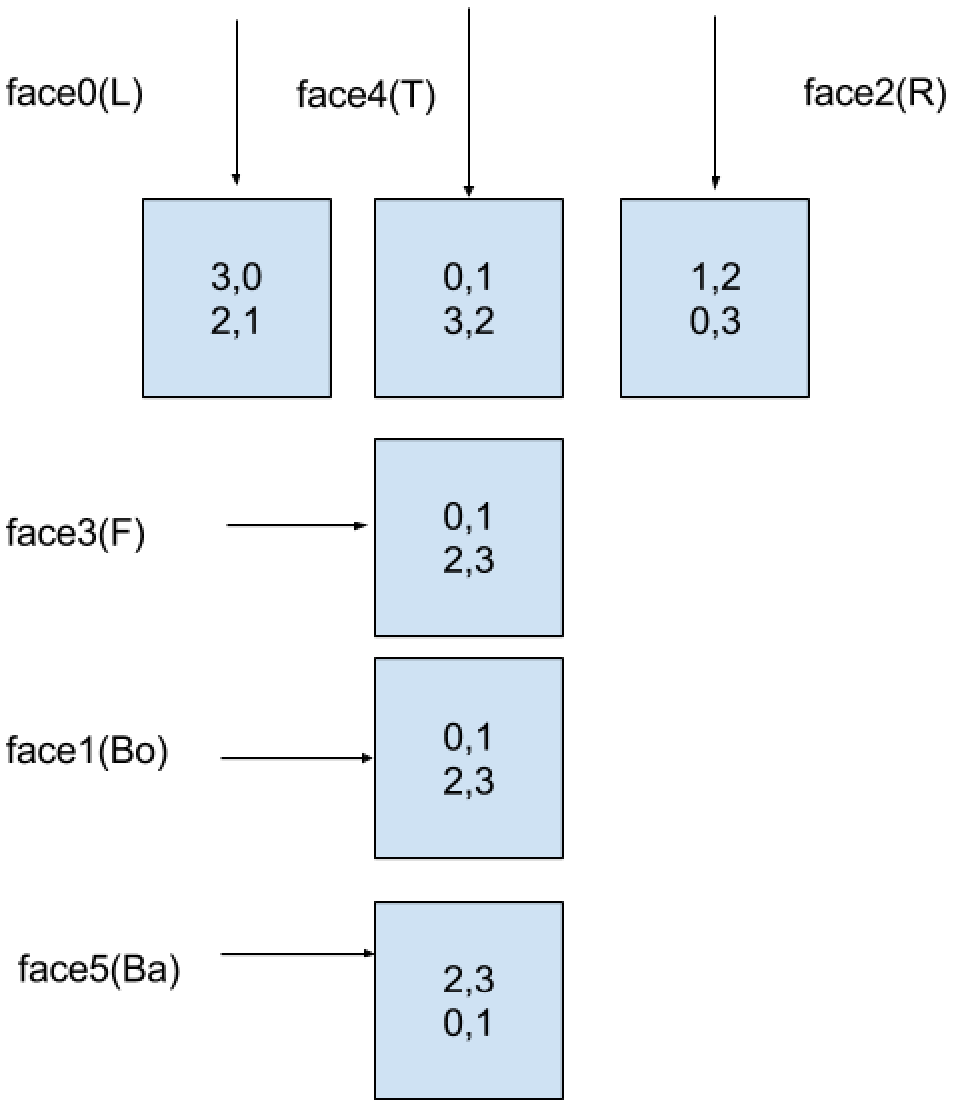
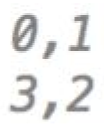


Figure 20 Representation of the Cube in 2-D



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

Figure 21 Color Index in the Face

For each face, say we have the indices in Figure 21.In the meanwhile, we record its 8 adjacent small facelet's color, and store then in an array. To do an rotate, we just rotate the array accordingly by position 2. For example, say the above face have adjacent color array C[0,1,2,3,4,5,6,7],.After rotating the above face clockwisely, the facelets becomes:

3,1
2,1

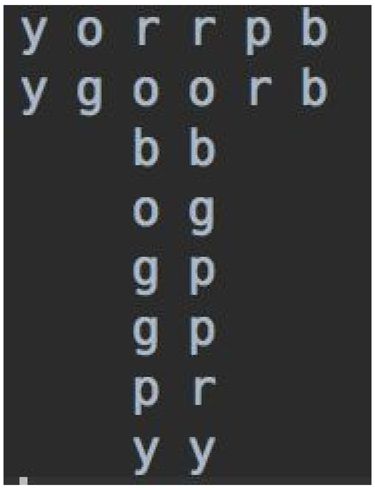
and the color array C becomes:[6,7,0,1,2,3,4,5,].
We can thus construct a cube according to the given input file and do rotations.

2.1.2 Heuristic and repeated state detection

The heuristic function in this is the number of different color pieces between the current 2D picture and the goal 2D picture.
In running the algorithm, we keep a list of visited state and a priority queue of the frontiers according to the sum of it's pathcost from the start point and it's heuristic value. Initially the visited state set is empty and the queue contains only the start state. At each step, take off a state from the queue, and calculate all the 12 possible moves and their corresponding rank, namely the sum of it's pathcost from the start point and it's heuristic value. If the possible nextstate is visited, we ignore it. If not, we check if it's already in the queue. If it is not in the queue, add it to the queue and if it is in the queue, we conpare the new estimated sum and the original sum in the queue. If the new sum is smaller, it means this is a better path then before so we update the state in the queue by assigning it a smaller rank and update its parent state.

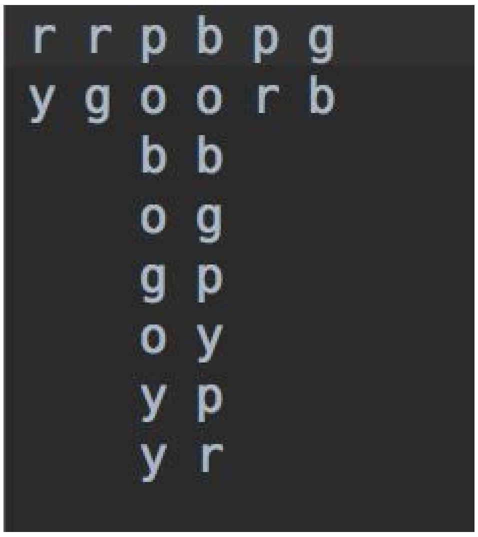
2.1.3 Result:

For Input1.1:

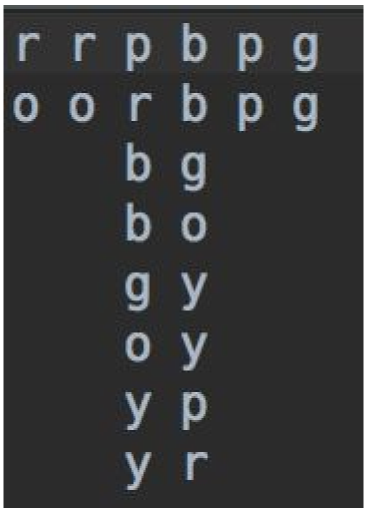


```
y o r r p b
y g o o r b
    b b
    o g
    g p
    g p
    p r
    y y
```

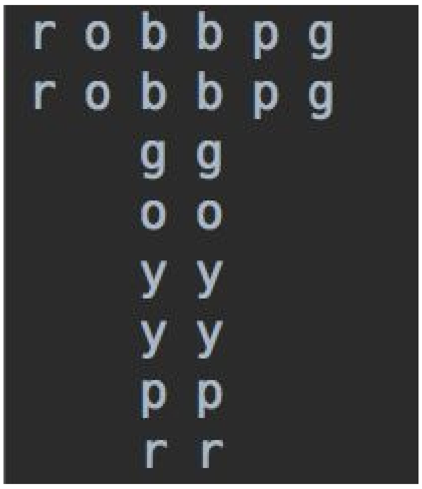
Step1: Clockwisely rotate Ba, we get:


```
r r p b p g
y g o o r b
    b b
    o g
    g p
    o y
    y p
    y r
```

Step2: CCW rotate Front, we get:

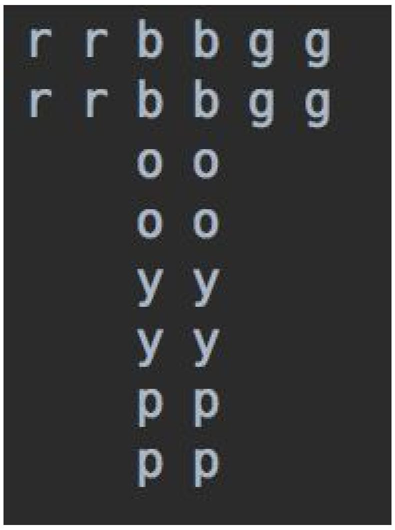

```
r r p b p g
o o r b p g
    b g
    b o
    g y
    o y
    y p
    y r
```

Step3: CCW rotate Left, we get:



```
r o b b p g
r o b b p g
    g g
    o o
    y y
    y y
    p p
    r r
```

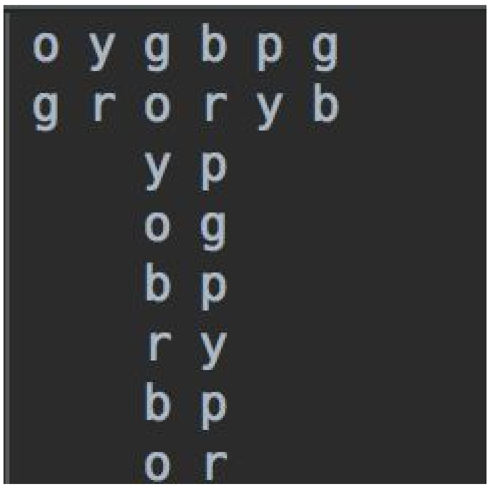
Step4: CCW rotate Top, we get:



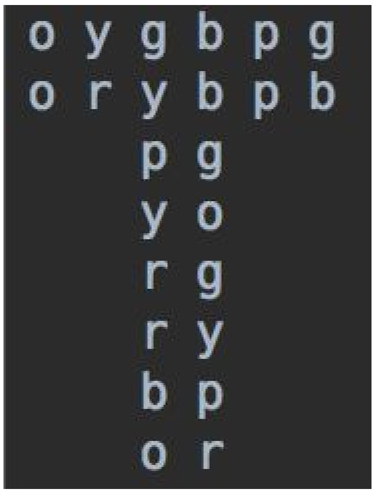
```
r r b b g g
r r b b g g
    o o
    o o
    y y
    y y
    p p
    p p
```

Which is also the goal state.
The whole process takes 4 steps,namely Ba, F’,L’,T’. It visits 9 states in total. The running time is approximately 0.8s in total.

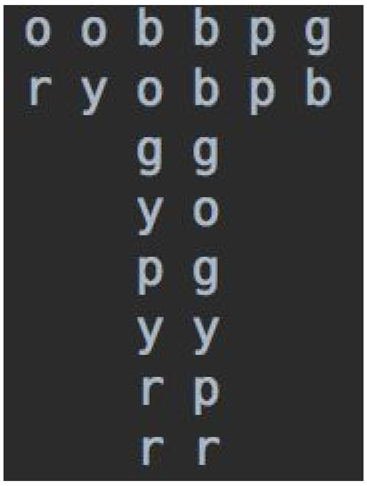
For Input1.2, the startstate looks like the following:


```
o y g b p g
g r o r y b
    y p
    o g
    b p
    r y
    b p
    o r
```

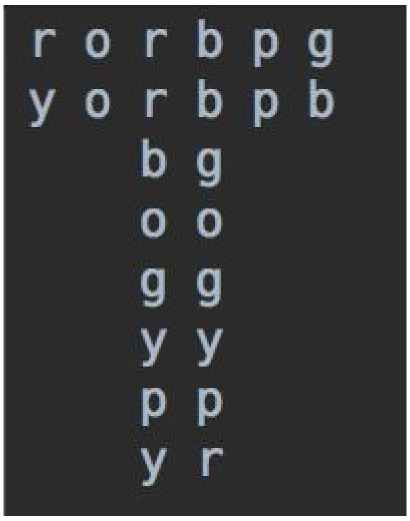
Step1: CCW Rotate F, we get:


```
o y g b p g
o r y b p b
    p g
    y o
    r g
    r y
    b p
    o r
```

Step2: CW rotate Left, we get:

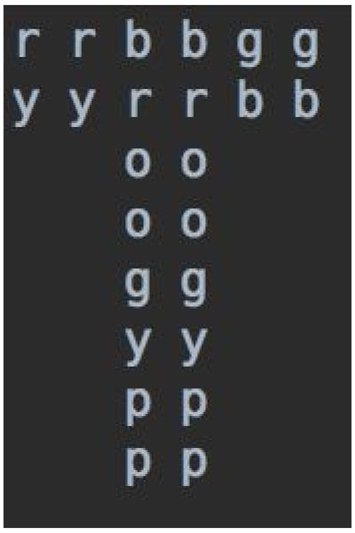

```
o o b b p g
r y o b p b
    g g
    y o
    p g
    y y
    r p
    r r
```

Step3: CW rotate Left, we get :



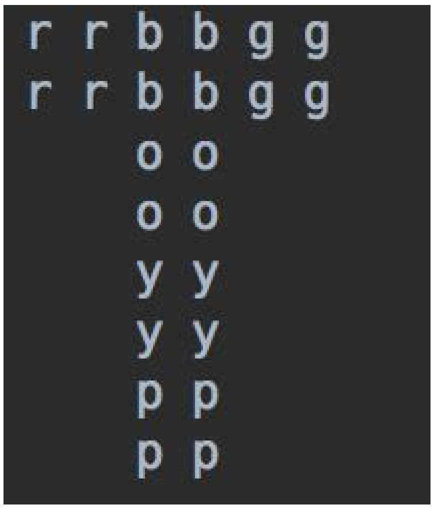
```
r o r b p g
y o r b p b
    b g
    o o
    g g
    y y
    p p
    y r
```

Step4: CCW Rotate Top, we get:



```
r r b b g g
y y r r b b
    o o
    o o
    g g
    y y
    p p
    p p
```

Step5: CCW Rotate Front,we get:

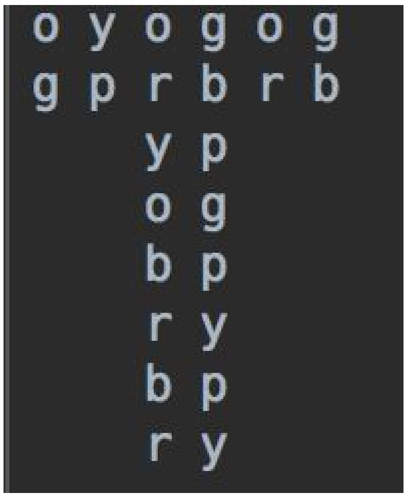


```
r r b b g g
r r b b g g
    o o
    o o
    y y
    y y
    p p
    p p
```

Which is also the goal state.

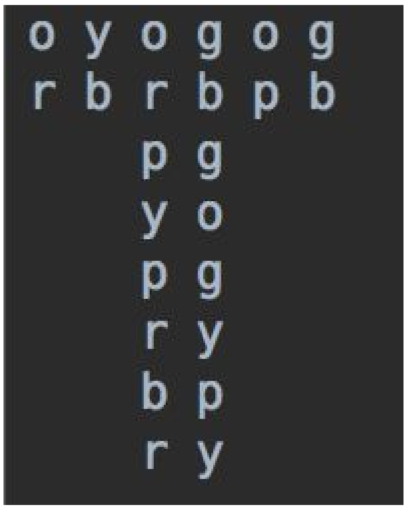
So for Input1.2, the whole process takes 5 steps. The sequence as indicated in the instruction is : F’, L, L,T’,F’. It visits 55 states in total.The running time is approximately 1.8 s in total.

For input1.3:
The Initial state is



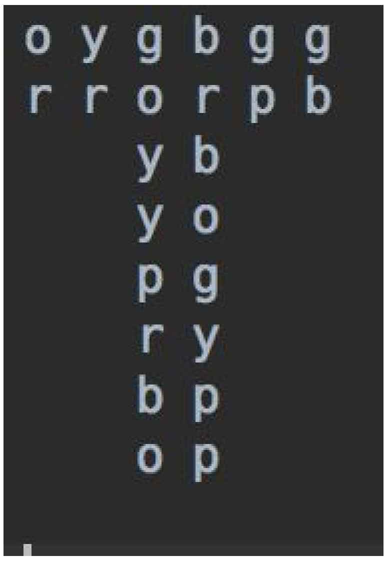
```
o y o g o g
g p r b r b
    y p
    o g
    b p
    r y
    b p
    r y
```

Step1: CCW Front:



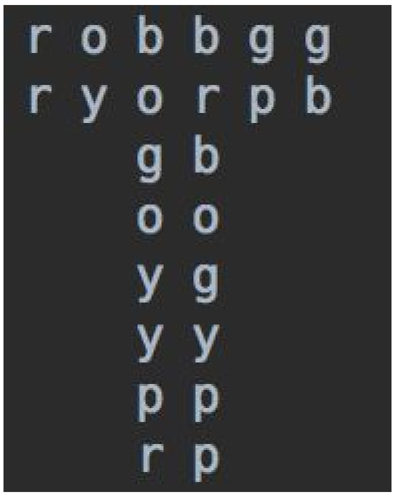
```
o y o g o g
r b r b p b
    p g
    y o
    p g
    r y
    b p
    r y
```

Step2: CCW Top:



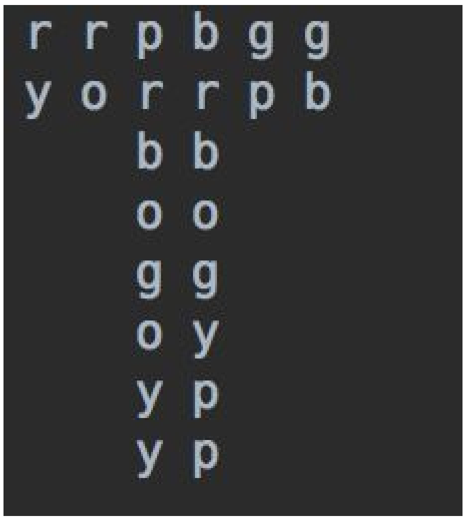
```
o y g b g g
r r o r p b
    y b
    y o
    p g
    r y
    b p
    o p
```

Step3: CW Left:



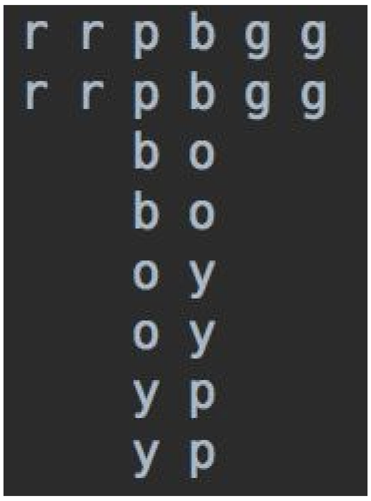
```
r o b b g g
r y o r p b
    g b
    o o
    y g
    y y
    p p
    r p
```

Step4:CW Left



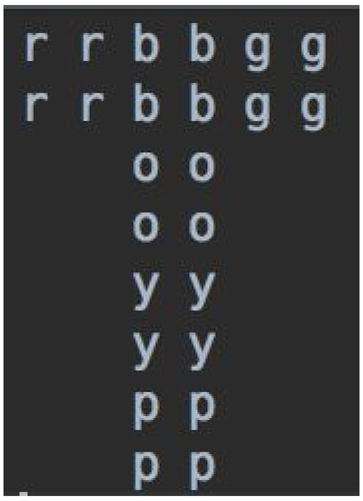
```
r r p b g g
y o r r p b
    b b
    o o
    g g
    o y
    y p
    y p
```

Step5: CCW Front:



```
r r p b g g
r r p b g g
    b o
    b o
    o y
    o y
    y p
    y p
```

Step6: CCW Left:



```
r r b b g g
r r b b g g
    o o
    o o
    y y
    y y
    p p
    p p
```

Which is also the goal state.

So for Input1.3, the whole process takes 6 steps. The sequence as indicated in the instruction is : F’, T’, L, L,F’,L’. It visits 84 states in total. The running time is aproximately 3.3 s.

2.2 **Adding Rotation Invariance**

2.2.1 Overview

Since we are going to add rotation invariance, the goal state will be totally 24 instead of 1. In order to calculate the heuristic of the current node, we change the representation of the heuristic and calculate the difference of the misplaced color facelet compared the final goal, which is each side should have the same color. The new heuristic function should compare the color of first element in the facelet with the remaining three in the same face. Then we need to get the number of differenced value in the face. For instance, if the colors in one face is shown as:

$$(r, b, r, b)$$

Then we compare the first element, which is ‘r’, with the rest of three elements: ‘b’,’r’,’b’. Obviously, the difference in this face is 2. So, in this way, we calculate the difference of each face and sum them together to get the total difference. And we use this difference as the heuristic for the next rotation. Compared with heuristic in part 2.1, the current heuristic is generated only via the state of six faces, the fixed goal state is not involved in the calculation although we could know that there is 24 different final goal states in the search.

2.2.2 Input 2.1(Same as Input 1.1)

For the Input 2.1(aka Input 1.1), we need four steps in order to reach the goal state. Which is

$$Ba\ (cw)\ F(ccw)L(ccw)T(ccw)$$

However, if we use the mentioned heuristic, we could get a better and fast result in only two steps, which is

$$Bo\ (ccw)\ F(ccw)\,.$$

The time for sloving Input 1.1 in part 2.1 is 0.8s, however, the time is 0.6s for the same input at part 2.2. The number of expaned nodes is 2 in second part. The number of expanded nodes is 8 in the first part.

Because first, the goal state is not fixed in the search, so heuristic could fall down quickly after one move compared with the movement in the part 2.1. Besides, the expaned node is less than the previous one as we pick up the state with smallest heuristic from the frontier to reach. One rotation will change 12 different facelet positions in the cube. So it is easier to reach the final state via only one rotation if the heuristic is small. Therefore, adding the rotation invariance is actually providing more possible ways to reach the final goal state since it is not fixed. And that’s another reason why the output of the Input 2.1 is better and fast than that of the Input 1.1 in the part 2.1.

As for A* algorithm, it is not too much difference included in the searcing process excepts the changed heuristic, it will search all possible rotation and return the optimal rotation option for the next rotation. The repeated state detection remains the same since there is no need to change it to cooperate with the heuristic or modified A* algorithm. We find that the time and expanded node wouldn’t case too much impact on the overall output. So we discuss less about it in this part. Another changed part is that the final goal state detection, as mentioned in the modification of the heuristic function, we changed the comparation of the current state and goal state, so in the final goal state detection after the executation of A* algorithm, we should also change it correspondingly. The detection should be whether the first element in a face is equal to the remaining three elements in the same face. When we apply this criteria into all of the six faces, the goal stae is reached if the result is TRUE. In this way, we could easily handle the multi-goal purpose in the searching problem for Rubik's Cube.

After the testing for Input 2.2 and Input 2.3, we find the running time and expanded nodes are acceptable. And the result shows that just a tiny modification in our previous code of part 2.1 could lead to the solution for 3-D situation when adding rotation invariance. And we think the modification of heuristic could avoid further modification in other implementation functions in our code. Moreover, we won't need to set up a cutoff on the number of nodes expanded in our code since it could always return an optimal solution from the start state to one of the 24 different goal states.

2.2.3 Input 2.2

Node expended 1334, Sequence: Ba',R',Bo,Bo,L',F'. Approximately 8min42s

2.2.2 Input 2.3

Node expended: 332
Time : 56s
sequence: Ba' Bo' R L F'

Contribution: Part1: Haoran
Part2.1:Liyi
Part2.2: Liyi and Haoran