Alex Shung
UIN: 653384880

State Representation:

- A state was uniquely defined by the coordinate in the maze and a list of the end goals remaining.
- A state also holds information about which State (parent) lead to this state and the path cost required to get here
- The path to a state can be obtained by continuously following the parent until there is none (which would be the starting position)
- The path cost required is used to help sort the States without having to traverse the path every time to get the path cost

Transition Model:

- For every State in the priority queue, attempt to explore the coordinate up, down, left, and right from the current coordinate
- If that coordinate is a wall, do not push it onto the queue
- If that coordinate has already been visited with this set of end goals, do not push it onto the queue
- If that coordinate is an end goal / '.', remove it from the end goals list and push the State onto the queue with those points and end goals
- Otherwise push that coordinate as a State back onto the queue

Goal Test

- The goal test was if there were any points remaining in the end goals remaining list of the state. This was checked every time before pushing the state back onto the queue

General Idea:

I used a priority queue to store the frontier nodes for all searches except for depth first search, where a list was used as a stack. I created a point which was a tuple representing coordinates and State as described above. The map was read in, and then converted to a 2d array for easy mutation of specific points. The transition model was moving up, down, left, or right. The goal state was when a state was reached which had no end points remaining. This implied that every end goal had been visited at least once. There was a priority queue used as each state was added based on the combination of the heuristic, to be explained later, and the step cost required to reach that state. In the event of a tie for priority, the step cost was used as a tie breaker.

Heuristic:

- For every permutation of end goals, found the optimal path assuming each one as the starting node and that there are no walls
    - For example, given the end goals [1, 2, and 3], I would find the optimal path from 1 which covers both 2 and 3 using Dijkstra's. I would then store this in a map from [1] -> path cost of traveling the end goals without walls [manhattan distance]. I would do the same for 2 and 3, adding those end points as keys to the map. I then store this map from end goals to optimal path cost in another map from the set of end goals [1,2,3] -> newly created map.
- Then for every state, take the minimum distance of the sum of the point to every remaining end + the optimal path cost stored for that set of ends for that end created above
- Use this value as the priority of the state in the priority queue. If the priority of two states match, take the state which has a lower step cost

Admissible Argument:

- The shortest path possible from any given State (point + remaining end goals) to a Goal state (no remaining end goals) will require it to go through at least one goal state. By calculating the optimal path between all goal states with the relaxed wall condition, which is what Dijkstra's gives, the optimal path from that goal state to the rest of the goal states will never be less than this proposed optimal path.
- Similarly, by taking the manhattan distance from the current point to each end goal, and adding that to the optimal path stored for that end goal, and then taking the minimum value for this sum, the optimal path will always be greater than or equal to this value from any given state.
- Since the optimal path will always be greater than or equal to this proposed heuristic, then the heuristic is admissible.
- This heuristic leads to an optimal solution because given how A* search works, if a better solution existed then it would have shown up earlier than any other solution as the optimal path taken will always have a lower or equal value than the heuristic presented here.

Implementation Description:

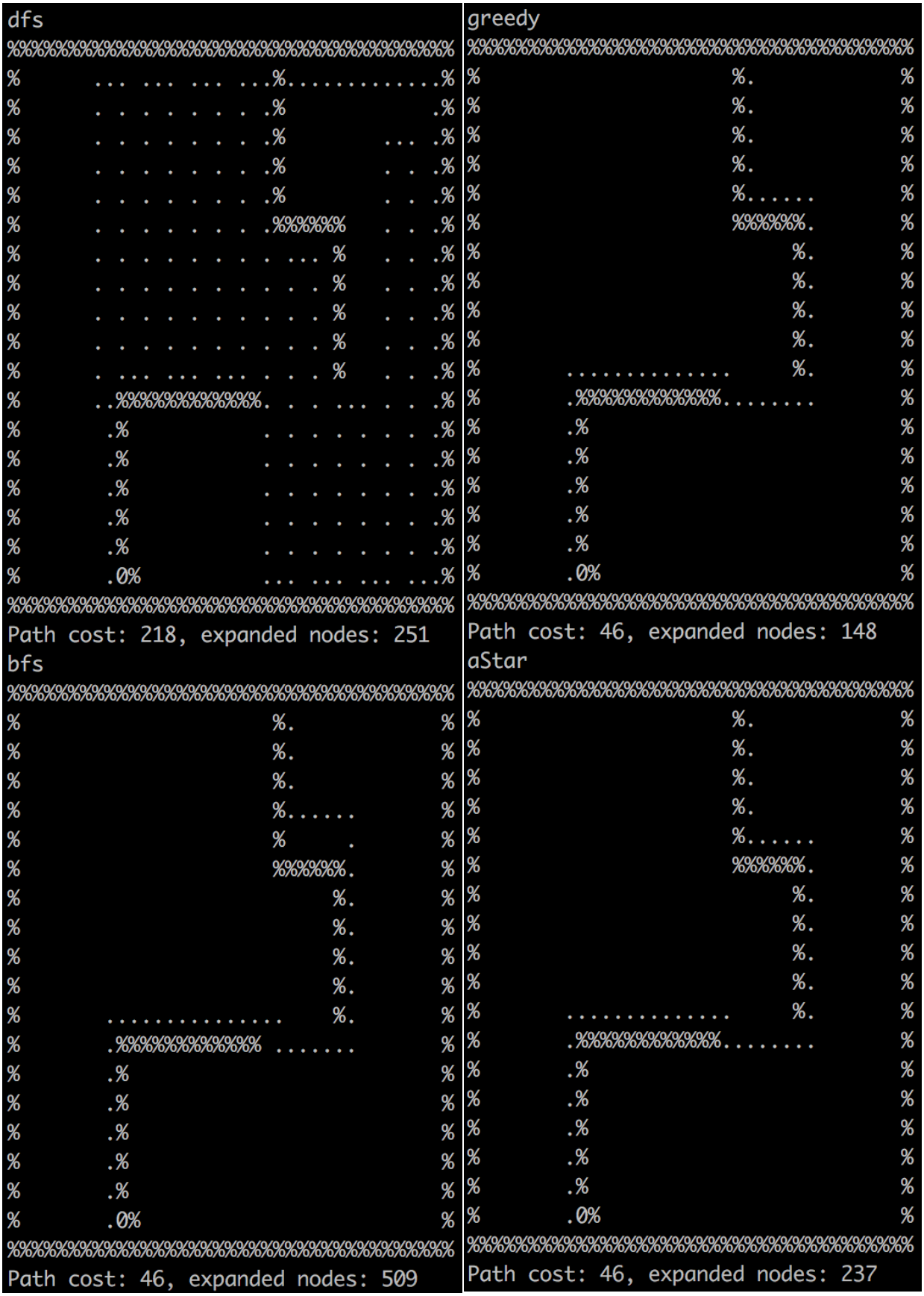
The top level search algorithm was essentially as follows:

1. Get the start and end positions as tuples
2. Create an initial state with that point, no parent, and 0 for the rest of the values
3. Preprocess the end points by finding the shortest path between all of them using Dijkstra's Algorithm with the end points representing nodes, and the manhattan distance between them representing the edges.
    a. Return a map containing each end -> shortest path cost

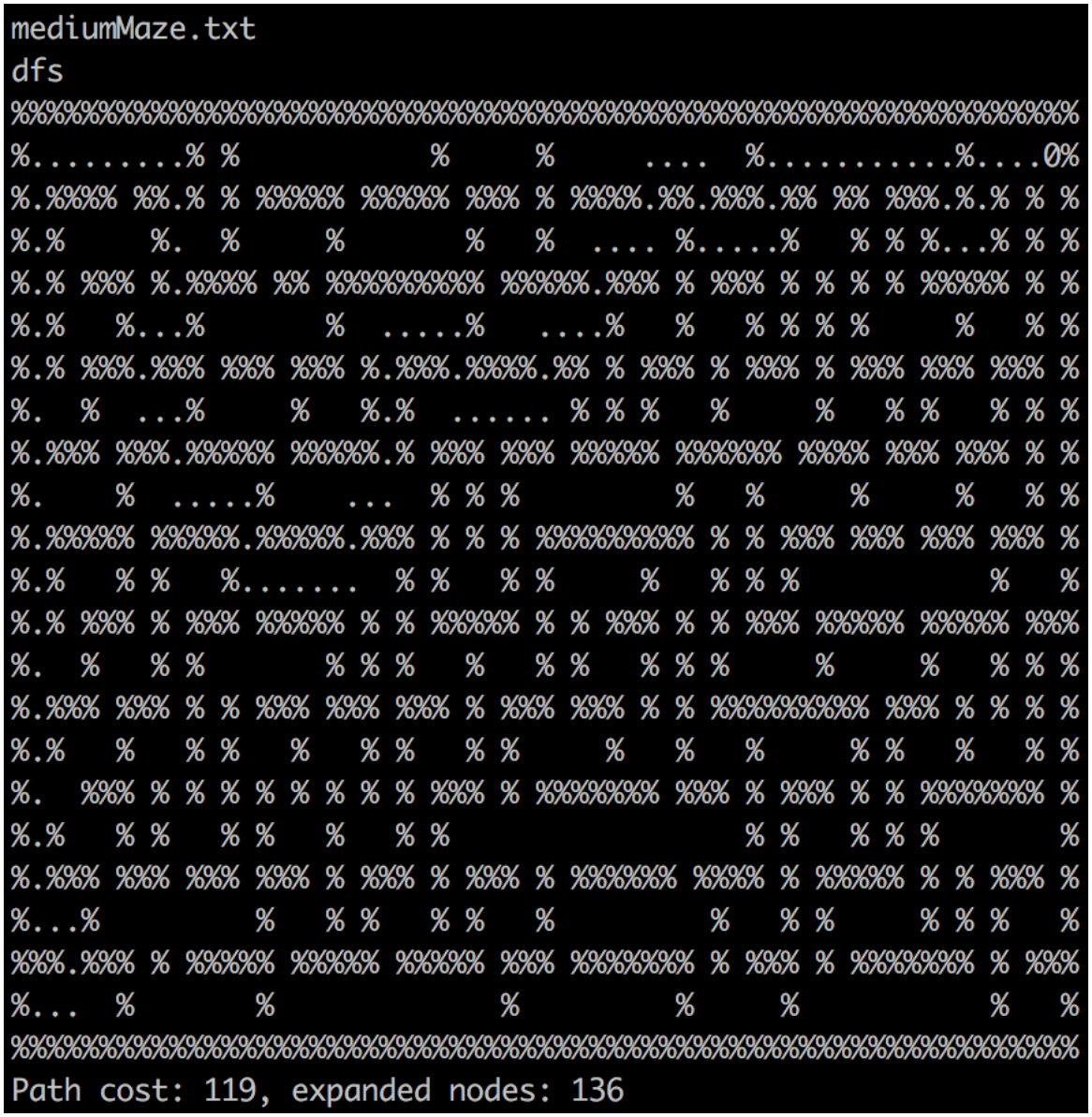
4. From the returned map, store that in a map with the list of end points as the key to the returned map
5. If it isn't dfs, have the frontierNodes be a Priority Queue. Otherwise a list
    a. The Priority Queue is ordered by the heuristic, and if the heuristic matches then it sorts by the number of explored nodes that were used to reach that point
6. Add the initial state to the frontierNodes
7. While there are still states in the frontierNodes
    a. Pop the next element (from the back for dfs)
    b. For each direction (up, down, left right):
        i. See if that direction has been visited before
            1. Repeated is defined as the coordinate and the associated end goals existing in the visited set or if that point is a wall
        ii. If it has, skip it
        iii. Otherwise continue
        iv. If that point exists in the goals of the current state:
            1. If that goal is the last goal:
                a. Return the State currently being evaluated (nextState)
            2. Otherwise add a state at that point with that goal removed from the goals
            3. Set the parent of this state to be the currentState
            4. Build another map as in 3, and store it with this set of ends in the map from 4
            5. Set the heuristic of the state to be the min of the sum from the point to each goal + the shortest path stored in the map from above.
            6. Add the state to frontier nodes and to the set of visited nodes
        v. Otherwise add that point with the current state's goals to frontier nodes and the set of visited nodes with the heuristic defined as 8.b.iv.3
        vi. Set the parent of this node to be the currentState
8. From the state returned in 7.b.iv.1.a, trace the path back through the parents

1.1 Solutions:

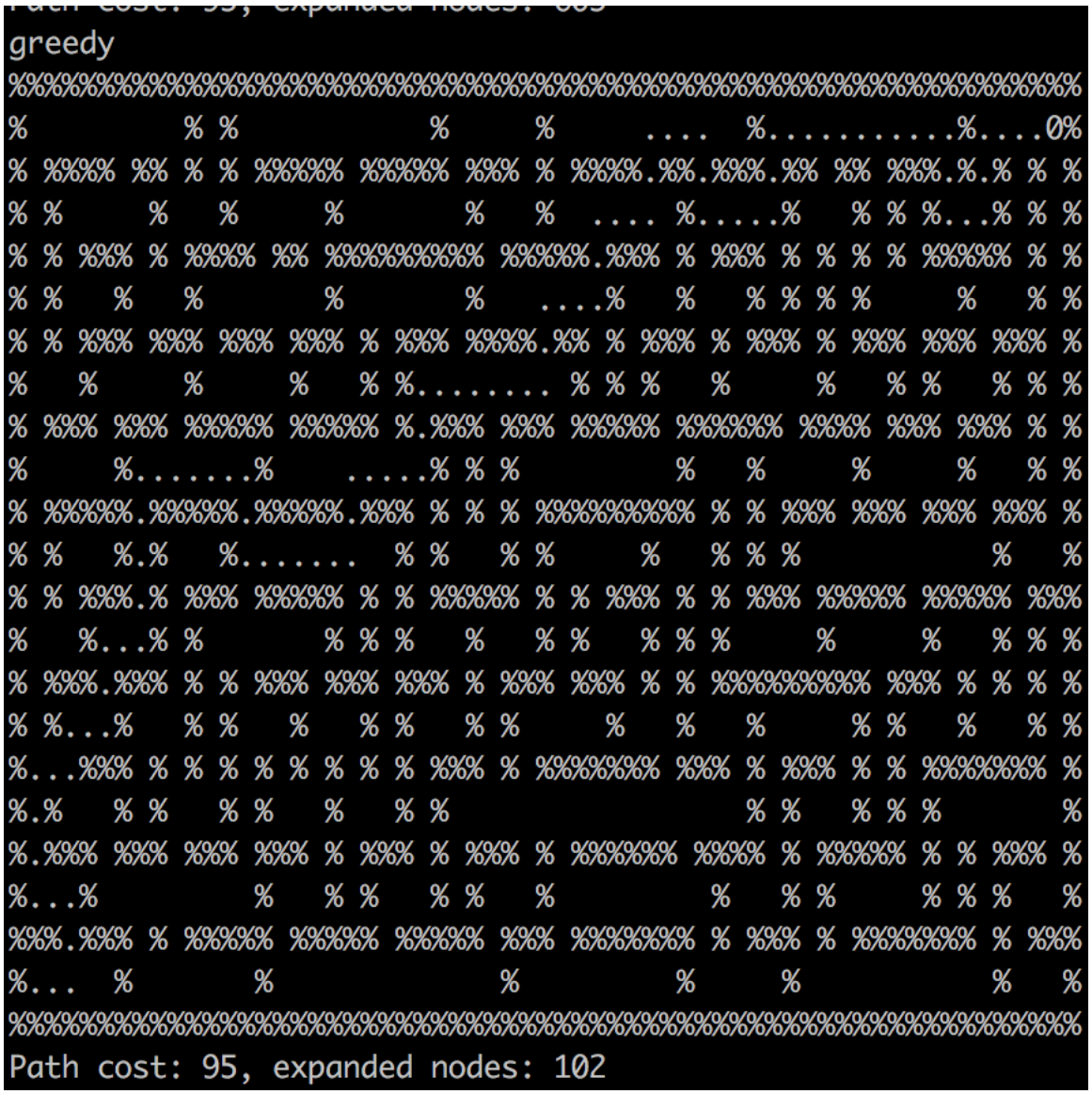
Open Maze:

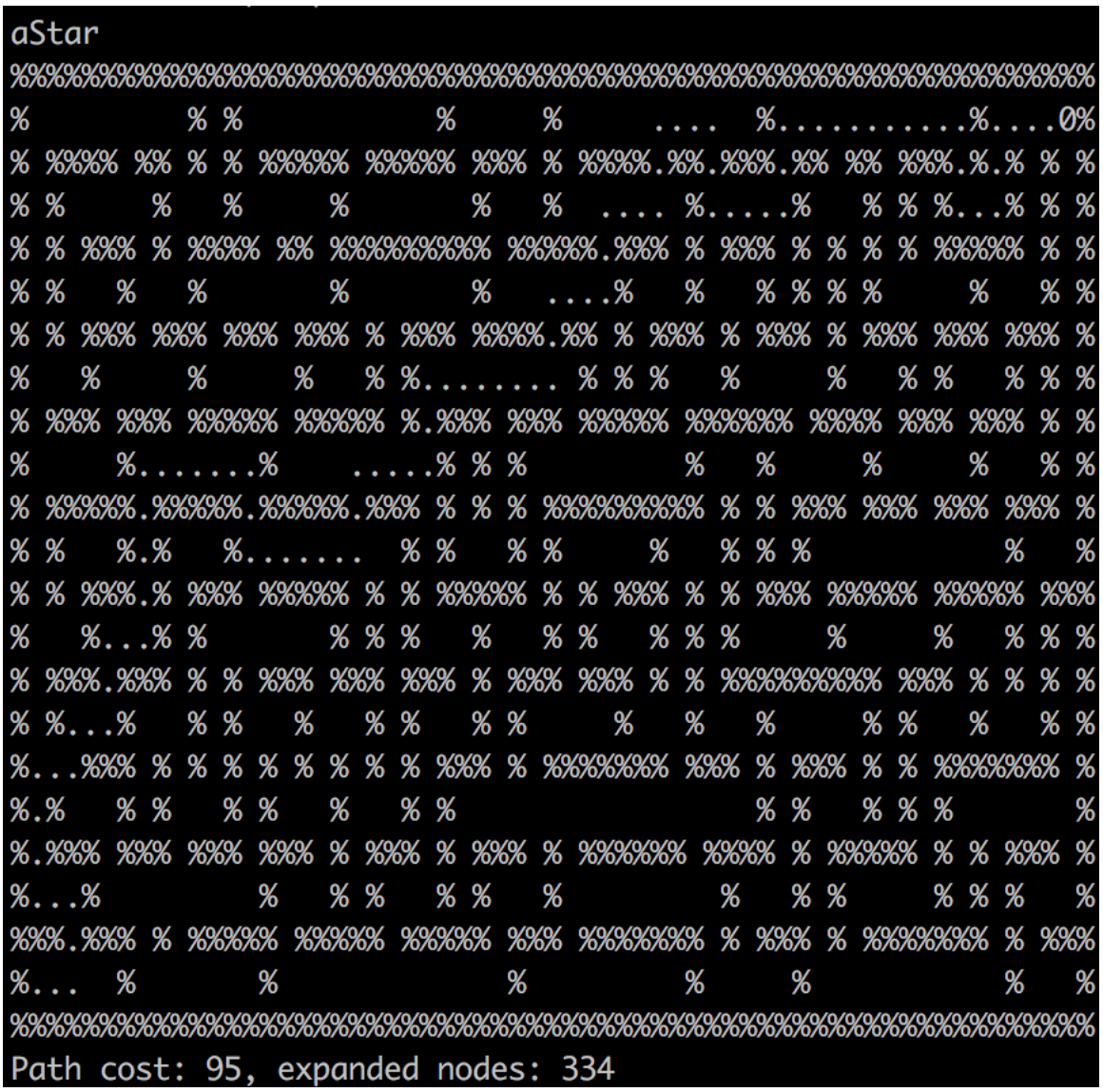

dfs
Path cost: 218, expanded nodes: 251
greedy
Path cost: 46, expanded nodes: 148
bfs
Path cost: 46, expanded nodes: 509
aStar
Path cost: 46, expanded nodes: 237

Medium Maze:



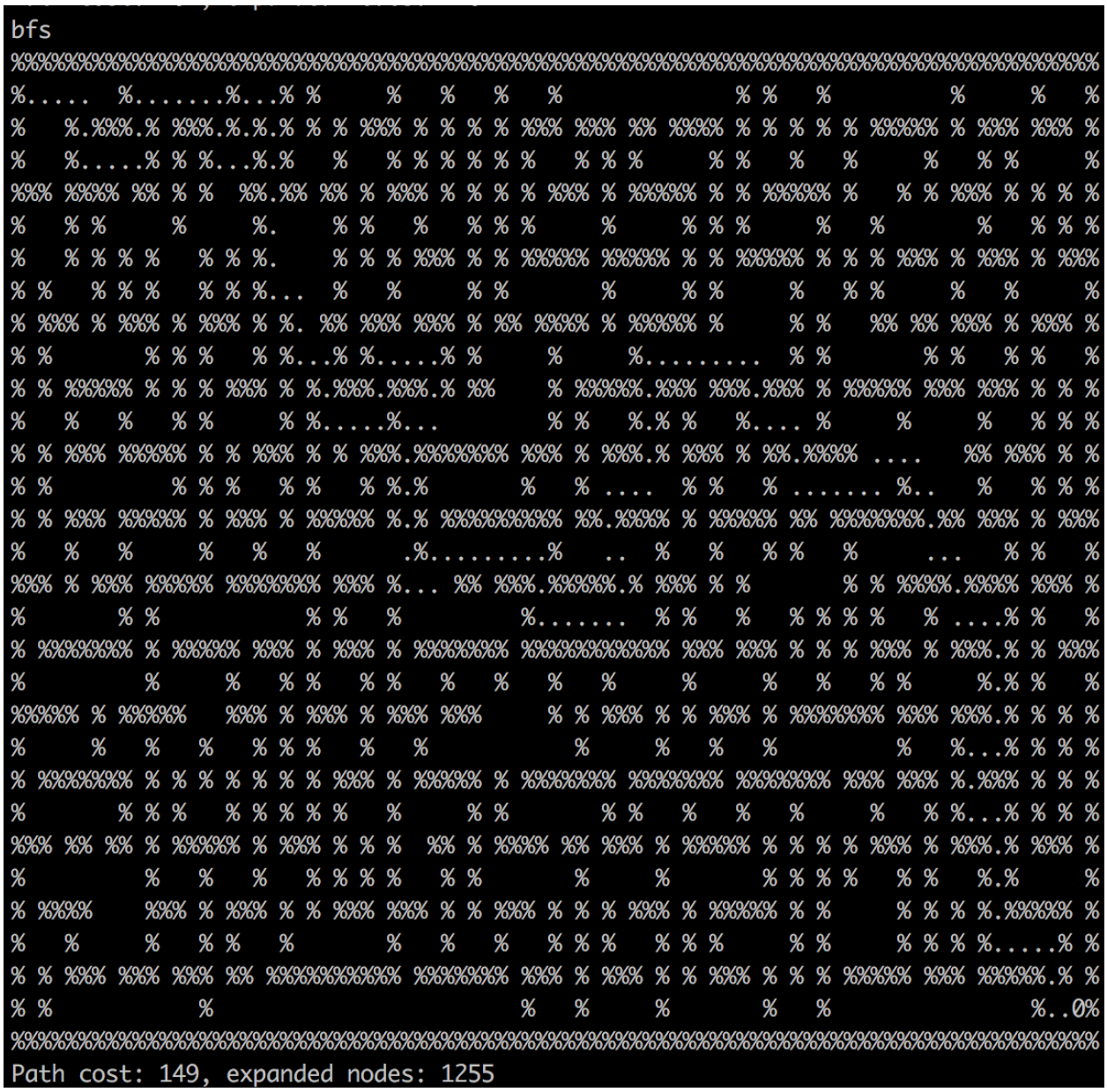

bfs
Path cost: 95, expanded nodes: 605


greedy
Path cost: 95, expanded nodes: 102

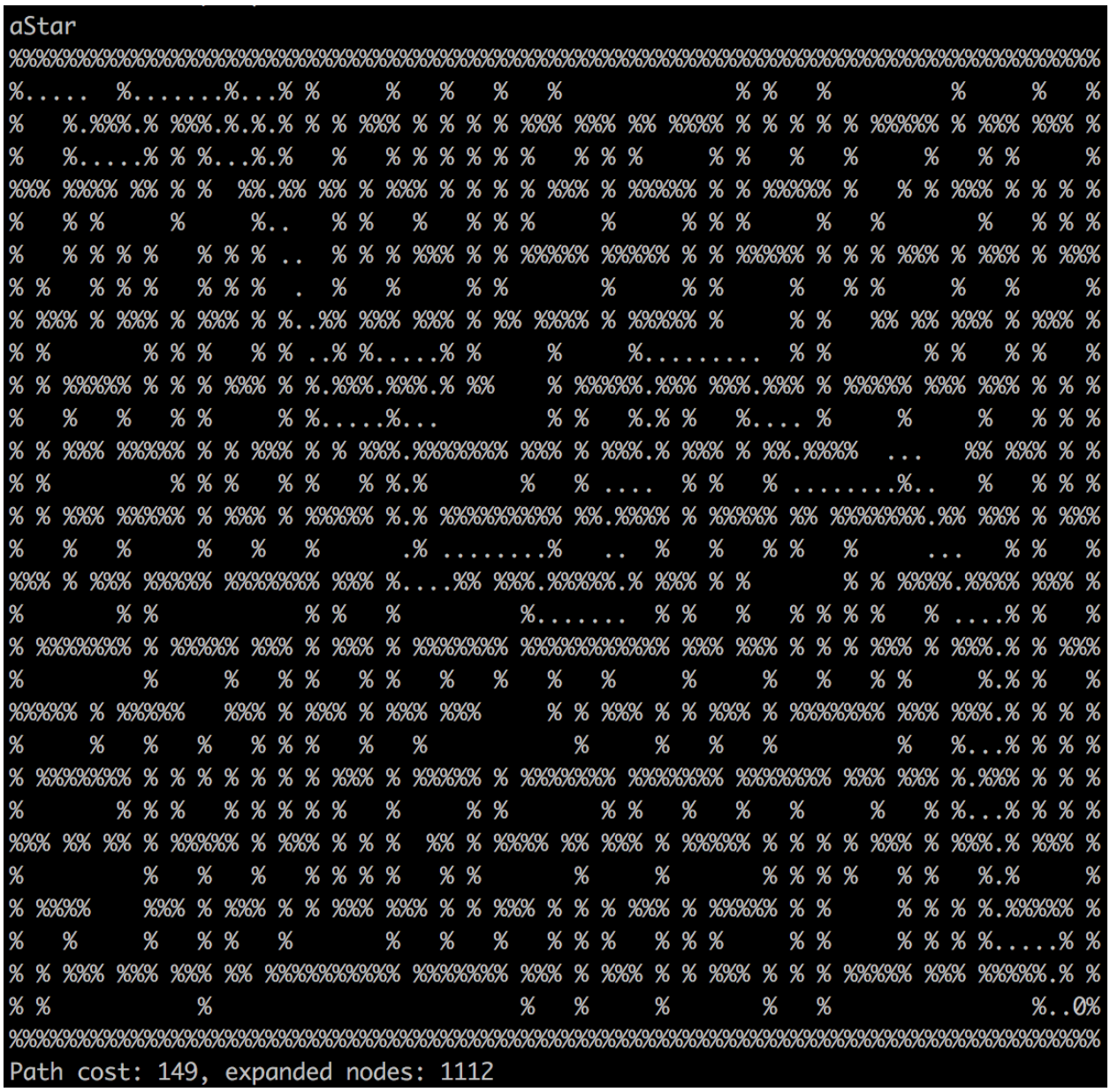


Big Maze:

```
ASHONG-01:MP1_2017 alex.shang$ python3 MP1.py
bigMaze.txt
dfs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.....  %.......%...% %...  %   %...%   %.............% %...%.........%.....%   %
%   %.%%%.% %%%.%.%.% %.%.%%% % %.%.% %%%.%%% %% %%%%.% %.%.%.% %%%%%.%.%%%.%%% %
%   %.....% % %...%.% ..%...% % %.%.% %...% % %     %.%...%...%     %...% %...  %
%%% %%%% %% % %  %%.%%.%% %.%%% %.%.% %.%%% % %%%%% %.%.%%%%% %   % % %%% % %.% %
%   % %     %     %.... % %...%  .%.% %.....%     % %.%.....%   %       %   %.% %
%   % % % %   % % %     % % %.%%%.%.% %%%%%.%%%%% % %.%%%%%.% % % %%% % %%% %.%%%
% %   % % %   % % %     %   %.....%.% ......%     % %.....%.  % %     %   %  ...%
% %%% % %%% % %%% % %  %% %%% %%% %.%%.%%%% % %%%%% %    .%.%   %% %% %%% % %%%.%
% %       % % %   % %   % %     % %.... %     %        ...%.%       % %   % %...%
% % %%%%% % % % %%% % % %%% %%% % %%    % %%%%% %%% %%%.%%%.% %%%%% %%% %%% %.% %
%   %   %   % %     % %     %           % %   % % %   %.....%     %     %   %.% %
% % %%% %%%%% % % %%% % % %%% %%%%%%% %%% % %%% % %%% % %% %%%%        %% %%%.% %
% %         % % %   % %   % % %       %   %       % %   %         %     %   %.% %
% % %%% %%%%% % %%% % %%%%% % % %%%%%%%%% %% %%%% % %%%%% %% %%%%%%% %% %%% %.%%%
%   %   %     %   %   %       %         %       %   %   % %   %           % %...%
%%% % %%% %%%%% %%%%%%% %%% %    %% %%% %%%%% % %%% % %       % % %%%% %%%% %%%.%
%       % %           % %   %         %         % %   %   % % % %   %     % %...%
% %%%%%%% % %%%%% %%% % %%% % %%%%%%% %%%%%%%%%%% %%% %%% % % % %%% % %%% % %.%%%
%         %     %   % %   % %   %   %   %   %     %     %   %   % %     % % %...%
%%%%% % %%%%%   %%% % %%% % %%% %%%     % % %%% % % %%% % %%%%%%% %%% %%% % % %.%
%     %   %   %   % % %   %   %           %     %   %   %         %   %   % % %.%
% %%%%%%% % % % % % % % %%% % %%%%% % %%%%%%% %%%%%%% %%%%%%% %%% %%% % %%% % %.%
%       % % %   % % % % %   %     % %       % %   %   %   %     %   % %   % % %.%
%%% %% %% % %%%%% % %%% % % %  %% % %%%% %% %%% % %%%%% % % % % %%% % %%% % %%%.%
%         %   %   %   % % % %   % %       %     %       % % % %   % %   % %    .%
% %%%%    %%% % %%% % % %%% %%% % % %%% % % % %%% % %%%%% % %     % % % % %%%%%.%
%   %     %   % %   %       %   %   %   % % %   % % %     % %     % % % %     %.%
% % %%% %%% %%% %% %%%%%%%%%% %%%%%%% %%% % %%% % % %%% % % % %%%%% %%% %%%%% %.%
% %           %                       %   %     %       %   %               %  0%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Path cost: 207, expanded nodes: 226
```

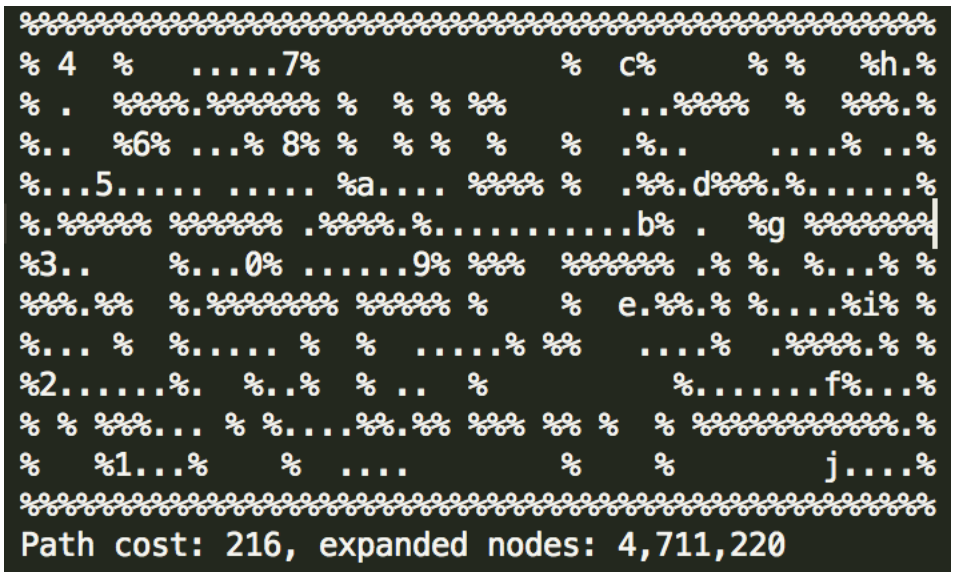

bfs
Path cost: 149, expanded nodes: 1255


greedy
Path cost: 235, expanded nodes: 292


aStar
Path cost: 149, expanded nodes: 1112

Solutions 1.2:

```
tinySearch.txt
aStar
%%%%%%%%%
%   %    %
% %0% %% %
% %...1%2%
% 6%.%...%
%...5..4.%
%.%%%% %.%
%..7   %3%
%%%%%%%%%
Path cost: 31, expanded nodes: 2548
smallSearch.txt
aStar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       .......0%......4..5...%
%   %%%%%.%%%%%.%..% %.%%   .%
%   %b % .    %   .%..% %. %  6%
%c.... % ....1...%3.   %. %%%%%
%%%%%.%%%% %%%..%%%%%%.....7%
%d..........   ...2    %.%%%  %
%%.%%%%%%%%.%%%%%%%%%%.%     %
%..      %  ...    ...    .% %%%%
%.       %%%%%.....%9.   .     %
%e% %%%    % .%   %%.%%.%%%%%%
%          % a%      ........8%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Path cost: 144, expanded nodes: 3622459
```



This one was saved on sublime in an attempt to never have to re-run it.