CS 440/ECE 448 Spring 2018
Assignment 2: Smart Manufacturing, Two-Player Games

Name: Haowen Jiang, Xiang Xiang, Haojia Yu
NetID: haowenj2, xiangx2, hyu59

Part 1: Smart Manufacturing

1.1 Planning Using A Search*

For the five different types of widgets:

- Widget 1 = Components AEDCA
- Widget 2 = Components BEACD
- Widget 3 = Components BABCE
- Widget 4 = Components DADBD
- Widget 5 = Components BECBD

The factory sequence with the smallest number of stops is: (Three results using different starting points)

1. ['D', 'B', 'A', 'E', 'D', 'C', 'A', 'B', 'C', 'D', 'E']
2. ['B', 'A', 'E', 'D', 'C', 'A', 'D', 'B', 'C', 'D', 'E']
3. ['A', 'B', 'E', 'D', 'C', 'A', 'D', 'B', 'C', 'D', 'E']

But the sequence starting with D have the least expanding which is 121

The factory sequence with the smallest number of miles traveled is:
['D', 'E', 'B', 'E', 'C', 'A', 'E', 'D', 'E', 'B', 'E', 'C', 'A', 'E', 'D']
The distance traveled is 5594

State representation:
In every state I stored

name

The name of current location (factory) of the truck
Exp. 'A', 'B', 'C', 'D' **or** 'E'

remainelements

All the elements still need installed (not installed yet) in every Widget
Exp. [['E', 'D', 'C', 'A']
['E', 'A', 'C', 'D']
['A', 'B', 'C', 'E'] After going through 'D', 'A', 'B'
['D', 'B', 'D']
['E', 'C', 'B', 'D']]

remaintype

All the type of elements still need installed (not installed yet) in all Widgets
Exp. ['B', 'C', 'D', 'E'] After going through 'D', 'B', 'A', 'E', 'D', 'C', 'A'

remainNum

The number of each type of elements need installed (not installed yet) in all Widgets
Exp. [3, 3, 4, 4, 4] After going through 'D', 'A', 'B'

path

The Path the truck has gone through to be current state.
Exp. ['D', 'B', 'A', 'E', 'D', 'C', 'A'] After going through factory 'D', 'B', 'A', 'E', 'D', 'C', 'A'

G

G: **Step**: The number of steps in the path the truck has already go through
Miles: The number of miles in the path the truck has already go through

H

H: **Step**: The number of types (kinds) of elements left (need installed and not installed yet)
Miles: The minimum summary of a set of minimum (weight) spanning trees. These trees are ones using from elements need installed to all the elements

F

F: G + H

Actions:

- **Going from one factory to another factory**
- **Change the name of the state**
- **Add the G by distance (if Step, add 1; if mile, add the mile distance from the last location)**
- **Predict the heuristic value**
- **Remove the first element in every widget if it's the same as the element of current factory**
- **Update the number of each type of every element**
- **Update the kinds of elements left**
- **Add the current factory to the path**
- **Go to the next state in the priority queue**

Heuristic:

- Steps:
  The number of types (kinds) of elements left (need installed and not installed yet)
- Miles:
  First Attempt:
  1. We get the types (kinds) of elements left and generate a minimum (weight) spanning trees.
  2. Get the summary of edges' length in minimum (weight) spanning trees and set it as a least value.
  3. Get all possible combination of types (kinds) of elements which have finished installation.
  4. For each time, add the possible combination to the elements which have not finished installation and generate a minimum (weight) spanning trees.
  5. Get the minimum value of summary of edges' length in all the minimum (weight) spanning trees. This is the heuristic value.

  Second Attempt:
  1. We get the types (kinds) of elements left and generate a minimum (weight) spanning trees.
  2. Get the summary of edges' length in minimum (weight) spanning trees and set it as a least value.

Implement:

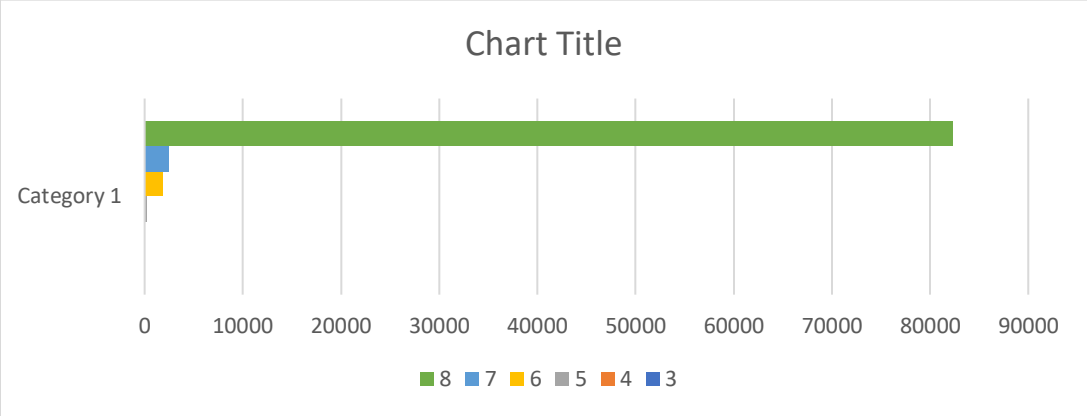
In the main function we get all the elements which are the first one of all the widgets and make them the possible start factory. Then we run a function named **step** considering all possible start factories. The function took the argument of the Widgets list and a start factory.

In the Step function, we generate a state object which represent the truck's starting at the "start factory", and calculate all the attribute in the state object including the name of current location (factory) of the truck, all the elements still need installed (not installed yet) in every Widget, all the types of elements which has not finished all the installation in all Widgets, number of each type of elements need installed (not installed yet) in all Widgets, the path, G, H, and the F. Then we add it to the priority queue which is sorted by the F value. Then we just implement the A* search to let it go through all the possible "Next State". When the current state doesn't have a next state, this is the final state. We can get the final result including the path it goes and the distance traveled.

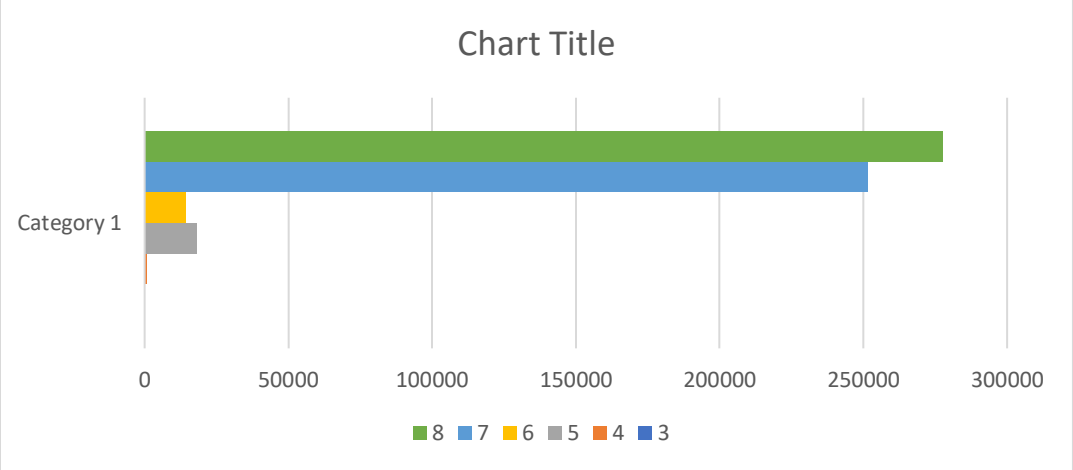
1.3 Extra Credit

Question 1.
nodes expanded
Step:
3: 14

4: 37
5: 258
6: 1859
7: 2510
8: 82351



Miles:
3:118
4:882
5:18397
6: 14367
7: 251535
8: 277594



Question 2.
Step:
3: 542
4: 1707
5: 3316
6: 106931
7: 277594
8: 402512

Miles:
3: 32961
4: 25040
5: 422281
6: 349219
7: 3419204
8: 2481598

The number of nodes expanded in uniform cost search is much larger than the A*.

Part 2: Game of Gomoku (Five-in-a-Row)

The purpose of this section is to implement several agents to play the game of Gomoku. Part 2.1 is reflex agent and the part2 includes minimax and alpha-beta.

Part 2.1 Reflex Agent

This part implements reflex agent which is defined in the assignment requirement.

Part 2.2 Minimax and Alpha-beta Agents

This part implements minimax agent and alpha-beta agent.

In the minimax function, for simplicity, I implement three for loop to represent search depth since only three is needed. Meanwhile, three lists are needed to save corresponding level of value. Find the maximum value in the third list and then record all the maximum values to the second one. Then get the minimum value in the second list and save them to the first one. The player will choose the next step according to the maximum value in the first list.

Alpha-beta function is similar to the minimax, but it will prunning some branches during process. For the very first, the player should search to depth three since no comparison value is initialed. For the continue branches, if the second level score is less than the comparison value, skip this branch and jump to the next since no matter what value next is, the player will not choose this branch.

To the evaluation function, we first find the number of winning blocks of player himself/herself and opponent winning blocks. Then count the self-stone in every block and set the self-score and opponent score like below.
Self-score:
count == 5: +10000000
count == 4: + 100000
count == 3: + 1000
count == 2: + 100
count == 1: + 10
count == 0: + 1
Opponent-score:
count == 4: - 1000000
count == 3: - 10000
Sum up all the winning blocks score as the evaluation score. We ignore opponent count equals to 5 since player will quit this branch when it finds the opponent can build a five-chain stone. In addition, we ignore opponent count equal to 2, 1 and 0 since the ratio of risk it does not matter.

1. alpha-beta vs. minimax

2. minimax vs. alpha-beta

3. alpha-beta vs. reflex

. . . .

K . . . . . .

. j . h . . .

. I k a . J .

c E b e d g G

B F . H i . .

A C D f . l .

4. reflex vs. alpha-beta

. . . .

f d . . . . .

E . B . D . .

e i . A . . .

c h F G C I H

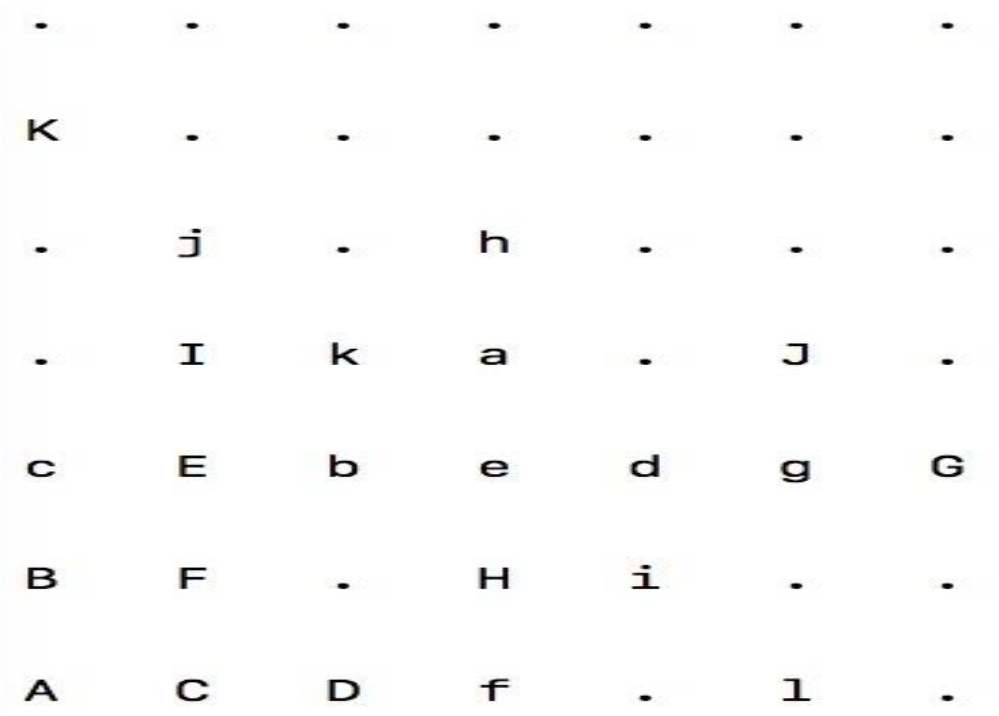
b g . . . . .

a . . . . . .

5. reflex vs. minimax


. . . .
f d . . . . .
E . B . D . .
e i . A . . .
c h F G C I H
b g . . . . .
a . . . . . .
winner is player2
[Finished in 308.1s]

6. minimax vs. reflex



The node expands of alpha-beta vs. minimax and minimax vs. alpha-beta shows below.

	alpha-beta vs. minimax	
1	110544	103776
2	97290	91080
3	85140	79464
4	72406	68880
5	63180	59280
6	53206	50616
7	28000	42840
8	36630	17440
9	31899	29760
10	25259	24360
11	17685	19656
12	16150	15600
13	12627	12144
14	6951	9240
15	7980	6840
16	5814	4896
17	4080	3360
18	2730	2184
19	550	1320
20	990	720
21	399	336
22	210	120
23	60	24
24	6	0
25	0	None

	minimax vs. alpha-beta	
1	110544	98210
2	97290	89892
3	85140	78624
4	74046	66960
5	63960	59280
6	54834	49500
7	46620	38420
8	39270	17440
9	32736	26280
10	26970	22680
11	21924	15730
12	17550	13560
13	13800	10032
14	10626	7980
15	7980	5976
16	5814	2832
17	4080	2604
18	2730	1716
19	1716	1320
20	990	472
21	504	336
22	210	120
23	60	24
24	6	0
25	0	None

Alpha-beta agents is realized based on minimax agent. Thus, each player step of these two agents are the same. The different between these two are the number of expanded nodes. In other words, the running time of alpha-beta is faster than minimax since alpha-beta pruning some braches compare to the other one.

Extra Credit:

2.4.2 Linear Regression.

In this part, we designed a linear regression model, which is used to train the evaluation function for rating the board position.

Firstly, we generated a series of board with random initialization with only integers 0, 1, 2 being assigned for each of the grid in the board. Then we calculated player 1 and 2 (indicated by) winningblocks amount and categorized them into several types. These types are divided by how many integers of each player are in their own winningblocks (5,4, 3, 2, 1). While for this training, we stand in one of the player's (player 1) view point to acquire the evaluation function. The evaluation function is actually acquired by a weight vector multiply the features vectors.

The feature vector is derived from the number of winning blocks of having 5, 4, 3, 2, 1, 0 player 1 own integers and 4, 3 of opponent integers: [count5, count4, count3, count2, count1, count0, oppo4, oppo3]. Our only goal is to derive the weight vector.

Notice that in the weight vector, we not only contain the weights for each category, but also added the bias term, the weight vector is at dimension of 8+1=9.

For other information about the evaluation function, please check part 2.2.

And here is our weight vector output.

```
Trained Model is:
 [[ 9.99982051e+06]
 [ 9.98496554e+04]
 [ 8.44143982e+02]
 [-5.52270023e+01]
 [-1.45180926e+02]
 [-1.54685054e+02]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [ 3.96084674e+04]]
```

The last row is the bias term.

The random initialization method has limits of correctly reflex the realistic gomoku board position, in the future better initialization might perform better.