

CSCI- B 551 ELEMENTS OF ARTIFICIAL INTELLIGENCE

Assignment 1: Search

Supreeth Prakash(skeragod@umail.iu.edu)

Shivani Gupta(shivgupt@umail.iu.edu)

Geetanjali Bhagwani(gcbhagwa@umail.iu.edu)

September 25, 2016

1.

(1) Which search algorithm seems to work best for each routing options?

DFS seems to work best for segments, distance routing option.

DFS and A* algorithm both work best for time routing option.

(2) Which algorithm is fastest in terms of the amount of computation time required by your program, and by how much, according to your experiments? (To measure time accurately, you may want to temporarily include a loop in your program that runs the routing a few hundred or thousand times.)

DFS algorithm is fastest. (below are the screenshot displaying the time with the output)

(3) Which algorithm requires the least memory, and by how much, according to your experiments?

A* Algorithm requires the least memory.

(4) Which heuristic function did you use, how good is it, and how might you make it better?

Euclidean distance function is used.

```
return sqrt((float(graph[end][0]) - float(graph[node][0]))**2 + \
            (float(graph[end][1]) - float(graph[node][1]))**2)/25
```

(5) Supposing you start in Bloomington, which city should you travel to if you want to take the longest possible drive (in miles) that is still the shortest path to that city? (In other words, which city is furthest from Bloomington?)

Stagway, _Alaska is farthest city from Bloomington (4542 miles)

The graph dictionary for road-segment.txt file:

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
{'Eastman_Georgia': [('McRae_Georgia', 15, 45, 'US_23/341', 20.0, 1), ('Perry_Georgia', 41, 45, 'US_341',
54.666666666666664, 1), ('Tartersville_Georgia', 30, 45, 'US_23', 40.0, 1)], 'Paragould_Arkansas':
[('Corning_Arkansas', 29, 45, 'AR_135', 38.666666666666667, 1), ('Jonesboro_Arkansas', 21, 45, 'US_49', 28.0, 1),
('Kennett_Missouri', 32, 45, 'US_412', 42.666666666666667, 1), ('Piggott_Arkansas', 32, 45, 'US_49', 42.666666666666667,
1), ('Walnut_Ridge_Arkansas', 26, 45, 'US_412', 34.666666666666664, 1)], 'Jct_I-29_&ND_200_S_North_Dakota':
[('Fargo_North_Dakota', 34, 65, 'I-29', 31.384615384615387, 0), ('Halstad_Minnesota', 11, 45, 'MN/ND_200',
14.666666666666666, 1), ('Jct_I-29_&ND_200_N_North_Dakota', 11, 65, 'I-29', 10.153846153846155, 0)], 'Oyen_Alberta':
[('Alaska_Saskatchewan', 34, 50, 'AB_9', 40.800000000000004, 1), ('Dunmore_Alberta', 180, 50, 'AB_41', 216.0, 1),
('Hanna_Alberta', 100, 50, 'AB_9', 120.0, 1)], 'Pierreville_Quebec': [('Drummondville_Quebec', 30, 50, 'PQ_143', 36.0,
1), ('St-Gregoire_Quebec', 42, 50, 'PQ_132', 50.4, 1), ('Yamaska-Est_Quebec', 8, 50, 'PQ_132', 9.6, 1)],
'Payson_Arizona': [('Globe_Arizona', 82, 45, 'AZ_88_&AZ_188', 109.33333333333333, 1), ('Mesa_Arizona', 78, 52,
'AZ_87', 90.0, 1), ('Show_Low_Arizona', 97, 45, 'AZ_260', 129.33333333333331, 1), ('Winslow_Arizona', 92, 45, 'AZ_87',
122.66666666666666, 1)], 'Ayer's_Cliff_Quebec': [('Coaticook_Quebec', 19, 50, 'PQ_141', 22.8, 1), ('Jct_A-
55_&PQ_141_Quebec', 6, 55, 'PQ_141', 6.545454545454545, 0), ('Lennoxville_Quebec', 30, 50, 'PQ_143', 36.0, 1),
('Stanstead_Quebec', 18, 50, 'PQ_143', 21.599999999999998, 1)], 'Willow_Springs_Illinois': [('Bolingbrook_Illinois',
10, 55, 'I-55', 10.909090909090909, 0), ('Chicago_Illinois', 16, 40, 'I-55', 24.0, 1), ('Hillside_Illinois', 8, 55, 'Tri-
State_Tollway_I-294', 8.727272727272727, 0), ('Jct_I-80_&I-294_Illinois', 18, 55, 'Tri-State_Tollway_I-294',
19.636363636363637, 0)], 'Morehead_Kansas': [('Jct_US_160_&US_169_Kansas', 9, 45, 'US_169', 12.0, 1),
('Neodesha_Kansas', 13, 50, 'US_400', 15.600000000000001, 1), ('Parsons_Kansas', 16, 50, 'US_400', 19.2, 1),
('Thayer_Kansas', 13, 45, 'US_169', 17.333333333333332, 1)], 'Hamilton_Alabama': [('Jct_US_78_&AL_17_Alabama', 3, 45,
'AL_17', 4.0, 1), ('Jct_US_78_&AL_74_Alabama', 5, 40, 'AL_74', 7.5, 1), ('Jct_US_78_&US_43/278_Alabama', 5, 40,
'US_43/278', 7.5, 1), ('Natural_Bridge_Alabama', 24, 45, 'US_278', 32.0, 1), ('Spruce_Pine_Alabama', 24, 45, 'US_43',
32.0, 1)], 'Syracuse_Kansas': [('Granada_Colorado', 31, 45, 'US_50', 41.333333333333336, 1)].
```

```
starting = "Indianapolis,_Indiana"
destination = "Bloomington,_Indiana"
```

How to Run the Program

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 0 0

```
routing_option = segment
routing_algo = DFS
```

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.00600004196167
['Indianapolis,_Indiana', 'Jct_I-465_&IN_37_S,_Indiana', 'Martinsville,_Indiana', 'Bloomington,_Indiana']
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 0 1

```
routing_option = segment
routing_algo = BFS
```

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.0139999389648
['Indianapolis,_Indiana', 'Keystone_Crossing,_Indiana', 'Carmel,_Indiana', 'Tetersburg,_Indiana', 'Peru,_Indiana',
'Rochester,_Indiana', 'Warsaw,_Indiana', 'Wallen,_Indiana', 'Sedan,_Indiana', 'Napoleon,_Ohio', 'New_Haven,_Indiana',
'Van_Wert,_Ohio', 'Lima,_Ohio', 'Wapakoneta,_Ohio', 'Vandalia,_Ohio', 'Richmond,_Indiana', 'Spiceland,_Indiana',
'Rushville,_Indiana', 'Shelbyville,_Indiana', 'Franklin,_Indiana', 'Martinsville,_Indiana', 'Bloomington,_Indiana']
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 0 2

```
routing_option = segment
```

routing_algo = IDS

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.0329999923706
[['Indianapolis,_Indiana', 'Jct_I-74_&_I-465_E,_Indiana', 'Jct_I-465_&_US_52,_Indiana', 'Rushville,_Indiana',
'Shelbyville,_Indiana', 'Franklin,_Indiana', 'Martinsville,_Indiana', 'Bloomington,_Indiana']]
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 0 3

routing_option = segment
routing_algo = A* Algo

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.119999885559
[['Indianapolis,_Indiana', 'Jct_I-465_&_IN_37_S,_Indiana', 'Martinsville,_Indiana', 'Bloomington,_Indiana'], 51]
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 1 1

routing_option = distance
routing_algo = BFS

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
[['Indianapolis,_Indiana', 'Jct_I-465_&_IN_37_S,_Indiana', 'Martinsville,_Indiana', 'Bloomington,_Indiana'], 51]
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 2 0

routing_option = time
routing_algo = DFS

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.130000114441
[['Indianapolis,_Indiana', 'Jct_I-65_&_I-465_S,_Indiana', 'Jct_I-465_&_IN_37_S,_Indiana', 'Martinsville,_Indiana',
'Bloomington,_Indiana'], 63.13286713286713]
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 2 3

routing_option = time
routing_algo = A* Algo

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
0.144000053406
[['Indianapolis,_Indiana', 'Jct_I-65_&_I-465_S,_Indiana', 'Jct_I-465_&_IN_37_S,_Indiana', 'Martinsville,_Indiana',
'Bloomington,_Indiana'], 63.13286713286713]
```

Python2.7 Indianapolis,_Indiana Bloomington,_Indiana 1 3

```
routing_option = distance
routing_algo = A* Algo
```

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q1\map.py
56.5420000553
[['Bloomington, Indiana', 'Spencer, Indiana', 'Terre Haute, Indiana', 'Paris, Illinois', 'Mattoon, Illinois',
'Sullivan, Illinois', 'Decatur, Illinois', 'Harristown, Illinois', 'Grandview, Illinois', 'Southern View, Illinois',
'Jacksonville, Illinois', 'Oxville, Illinois', 'Jct_US_36 & US_54, Illinois', 'Barry, Illinois', 'Hull, Illinois',
'East Hannibal, Illinois', 'Hannibal, Missouri', 'Rensselaer, Missouri', 'Monroe City, Missouri', 'Macon, Missouri',
'Bucklin, Missouri', 'Laclede, Missouri', 'Chillicothe, Missouri', 'Hamilton, Missouri', 'Cameron, Missouri',
'Jct_US_36 & MO_6/31, Missouri', 'Easton, Missouri', 'Jct_I-29 & US_36, Missouri', 'St. Joseph, Missouri',
'Hiawatha, Kansas', 'Fairview, Kansas', 'Marysville, Kansas', 'Belleville, Kansas', 'Mankato, Nebraska',
'Lebanon, Kansas', 'Smith Center, Kansas', 'Phillipsburg, Kansas', 'Woodruff, Kansas'], 742]
```

2. Consider a variant of the 15-puzzle, but with the following important change: when the empty tile is on the edge of the board, the tile on the opposite side of the board can be slid into the opening.

Explanation

The program outputs a series of steps that the 0th tile has to take in order to solve the 16 tile puzzle.

The program first checks the permutation-inversion. If the value returned is 1, it will consider it as unsolvable. If not it will go to the next step of solving the problem.

Which heuristic functions did you try, and which works best?

We finally could come up with 4 heuristic functions,

1. City Block Distance - The Manhattan distance as it is usually called calculates the distance of each tile from where it was supposed to be. This heuristic function works really well for most of the boards. Next best state is calculated by $f(s) = h(s)$
2. Misplaced Tiles - This function calculates the sum of total misplaced tiles on the board. This particular heuristic generates a lot of states for complex boards and is not ideal for complex ones. Next best state is calculated by $f(s) = g(s) + h(s)$
3. Modified Manhattan - Since Professor has given us an option of moving tiles from the rightmost column to the first column and vice versa, also top most row to last row and vice versa. It will calculate the city block distance in the same manner. It produces similar or a few states compared to manhattan. Next best state is calculated by $f(s) = h(s)$

4. Misplaced Tiles in Rows/Columns - This heuristic function returns the misplaced tiles in columns and rows. This is a naive approach and unnecessarily generates a large number of states. Next best state is calculated by $f(s) = g(s) + h(s)$.

Function FindAdjoints finds all the adjoints of a 0 placed tile by moving it in all the possible directions.

A function named backTrack does the job of backtracking and finds the path required to get the board in it's required shape.

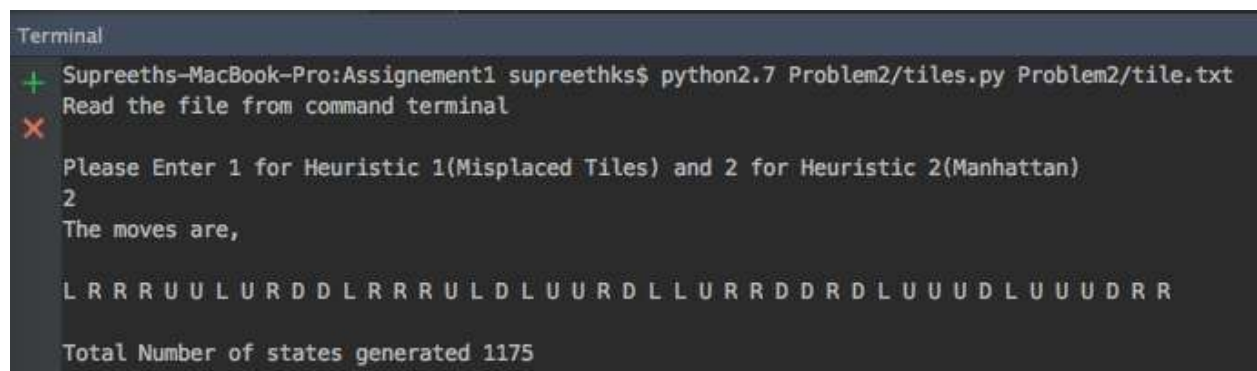
The Solution contains 7 python files, which are interconnected.

How to Run the code,

```
python2.7 Problem2/tiles.py Problem2/tile.txt
```

It will then ask for which heuristic function to use. Provide Manhattan for complex boards.

The console will look something similar to this.



```
Terminal
+ Supreeths-MacBook-Pro:Assignment1 supreethks$ python2.7 Problem2/tiles.py Problem2/tile.txt
Read the file from command terminal
X Please Enter 1 for Heuristic 1(Misplaced Tiles) and 2 for Heuristic 2(Manhattan)
2
The moves are,

L R R R U U L U R D D L R R R U L D L U U R D L L U R R D D R D L U U U D L U U U D R R

Total Number of states generated 1175
```

As we have said in the above explanation, for a very complex board, Manhattan / Manhattan Modified generates less amount of states and is indeed very fast. For somewhat easy looking boards the MisplacedTiles, Misplaced Tiles in Rows/Cols, Manhattan and Manhattan Modified works almost the similar way.

3. Kam and Lars are planning their upcoming wedding. They'd like to make their banquet as awkward as possible by ensuring that at each dinner table, no one knows anyone else at the table. Write a program that computes an assignment of people to tables using as few tables as possible

Explanation

The program output the tables of the guest on a wedding event such that none of guests familiar to each other.

The program first makes the dictionary of all the entries of the guest in the input list.

The dictionary contains the key values where each guest is the key having the list of the guest he knows. It is done by the creategraph() function.

The solve function first arrange the guest by checking that each key in the graph is not present in its list of value by taking the number of seats as $N=3$ by calling successor function.

The input to the program is

```
adavis steven sal joe jinnie zeming emma
jayceon jinnie zeming
jinnie zeming jayceon li
emma jinnie li
steven sal joe
sven steven joe li
```

How to run the program

Python2.7 Program3/ wedding.py Program3/myfriends.txt 3

The output is obtained

```
<terminated> C:\Users\gitanjali\workspace\Assignment1\Q3\wedding.py
{1: ['emma', 'sven', 'sal'], 2: ['li', 'joe', 'zeming'], 3: ['steven', 'jinnie'], 4: ['jayceon', 'adavis']}
```

References :

1. <http://codereview.stackexchange.com/questions/33473/solving-15-puzzle>
2. http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Clustering_Parameters/Manhattan_Distance_Metric.htm
3. Discussed the approach on a high level to have the solve function with Anup and team, Raghuveer and team.
4. Professor David Crandall's Lecture slides.
5. <https://docs.python.org/2/>