# Department of Computing

# CS370: Artificial Intelligence

# Class: BSCS-6AB

# Lab 3: Solving Problem by Searching

# Date: 07-02-2019

# Time: 10am-1pm & 2pm-5pm

# Instructor: Dr. Imran Malik

# Lab 3: Solving Problems by Searching

**Introduction**

This lab describes one kind of goal-based agent called a problem-solving agent. Problem-solving agents start by describing the goal, formulating the problem and then using searching techniques to find sequence of actions.

**Objectives**

In this lab you will implement problem-solving agent. Specifically you will formulate the specifics of your search problem and implement it. You will then use uninformed search strategies to solve your search problem and evaluate different strategies by measuring the cost of your solution and the time it takes to find the solution.

**Pre-requisites:**

Basic know how of OOP Concepts in Python

Basic know how of Python

**Tools/Software Requirement**

Python 2.7.9

Simpleai (http://simpleai.readthedocs.org/en/latest/)

**Description**

1. First you have to install python 2.7.9 download it from following link.

<https://www.python.org/downloads/release/python-279/>

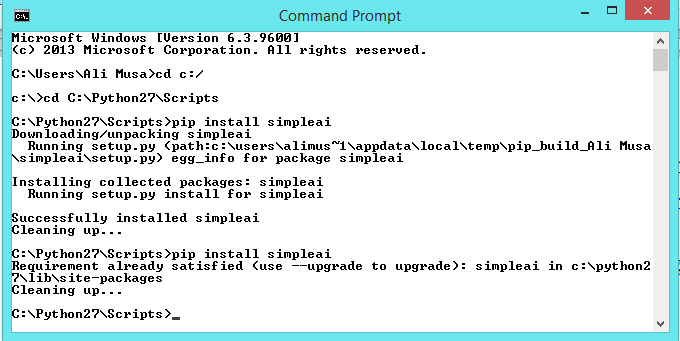
1. Open your command prompt and go in to following path.

C:\Python27\Scripts

1. Simpleai is a Python implementation of many of the artificial intelligence algorithms described in your text book: Artificial Intelligence: A modern Approach.

Install simpleai using the following command:

pip install simpleai



import sys

sys.path.append(“\path\_where\_simpleai\_installed”)

from simpleai.search import SearchProblem, breadth\_first, depth\_first, limited\_depth\_first, iterative\_limited\_depth\_first, uniform\_cost

To solve problem using searching you will need to define the specifics of your particular search problem, which includes defining the following:

1.Formulate Goal:

2.Formulate Problem:

1. Initial state
2. Actions
3. Transition model
4. Goal test
5. cost

3.Use searching algorithm to find solution:

See lab03.py to see how these things are defined for the case of 8-puzzle problem. Also see the documentation <http://simpleai.readthedocs.org/en/latest/search_problems.html>

Lab3.py

'''

8 puzzle problem, a smaller version of the fifteen puzzle:

http://en.wikipedia.org/wiki/Fifteen\_puzzle

States are defined as string representations of the pieces on the puzzle.

Actions denote what piece will be moved to the empty space.

States must allways be inmutable. We will use strings, but internally most of

the time we will convert those strings to lists, which are easier to handle.

For example, the state (string):

'1-2-3

4-5-6

7-8-e'

will become (in lists):

[['1', '2', '3'],

['4', '5', '6'],

['7', '8', 'e']]

'''

from timeit import default\_timer as timer

from simpleai.search import \*

'''

SearchProblem, breadth\_first, depth\_first, limited\_depth\_first, iterative\_limited\_depth\_first, uniform\_cost'''

GOAL = '''1-2-3

4-5-6

7-8-e'''

INITIAL = '''4-1-2

7-e-3

8-5-6'''

def list\_to\_string(list\_):

return '\n'.join(['-'.join(row) for row in list\_])

def string\_to\_list(string\_):

return [row.split('-') for row in string\_.split('\n')]

def find\_location(rows, element\_to\_find):

'''Find the location of a piece in the puzzle.

Returns a tuple: row, column'''

for ir, row in enumerate(rows):

for ic, element in enumerate(row):

if element == element\_to\_find:

return ir, ic

# we create a cache for the goal position of each piece, so we don't have to

# recalculate them every time

goal\_positions = {}

rows\_goal = string\_to\_list(GOAL)

for number in '12345678e':

goal\_positions[number] = find\_location(rows\_goal, number)

class EigthPuzzleProblem(SearchProblem):

def actions(self, state):

'''Returns a list of the pieces we can move to the empty space.'''

rows = string\_to\_list(state)

row\_e, col\_e = find\_location(rows, 'e')

actions = []

if row\_e > 0:

actions.append(rows[row\_e - 1][col\_e])

if row\_e < 2:

actions.append(rows[row\_e + 1][col\_e])

if col\_e > 0:

actions.append(rows[row\_e][col\_e - 1])

if col\_e < 2:

actions.append(rows[row\_e][col\_e + 1])

return actions

def result(self, state, action):

'''Return the resulting state after moving a piece to the empty space.

(the "action" parameter contains the piece to move)

'''

rows = string\_to\_list(state)

row\_e, col\_e = find\_location(rows, 'e')

row\_n, col\_n = find\_location(rows, action)

rows[row\_e][col\_e], rows[row\_n][col\_n] = rows[row\_n][col\_n], rows[row\_e][col\_e]

return list\_to\_string(rows)

def is\_goal(self, state):

'''Returns true if a state is the goal state.'''

return state == GOAL

def cost(self, state1, action, state2):

'''Uniform cost

'''

return 1

start = timer()

result = breadth\_first(EigthPuzzleProblem(INITIAL), graph\_search=True)

end = timer()

# Time

print "Time: " + str(end - start)

# cost of solution

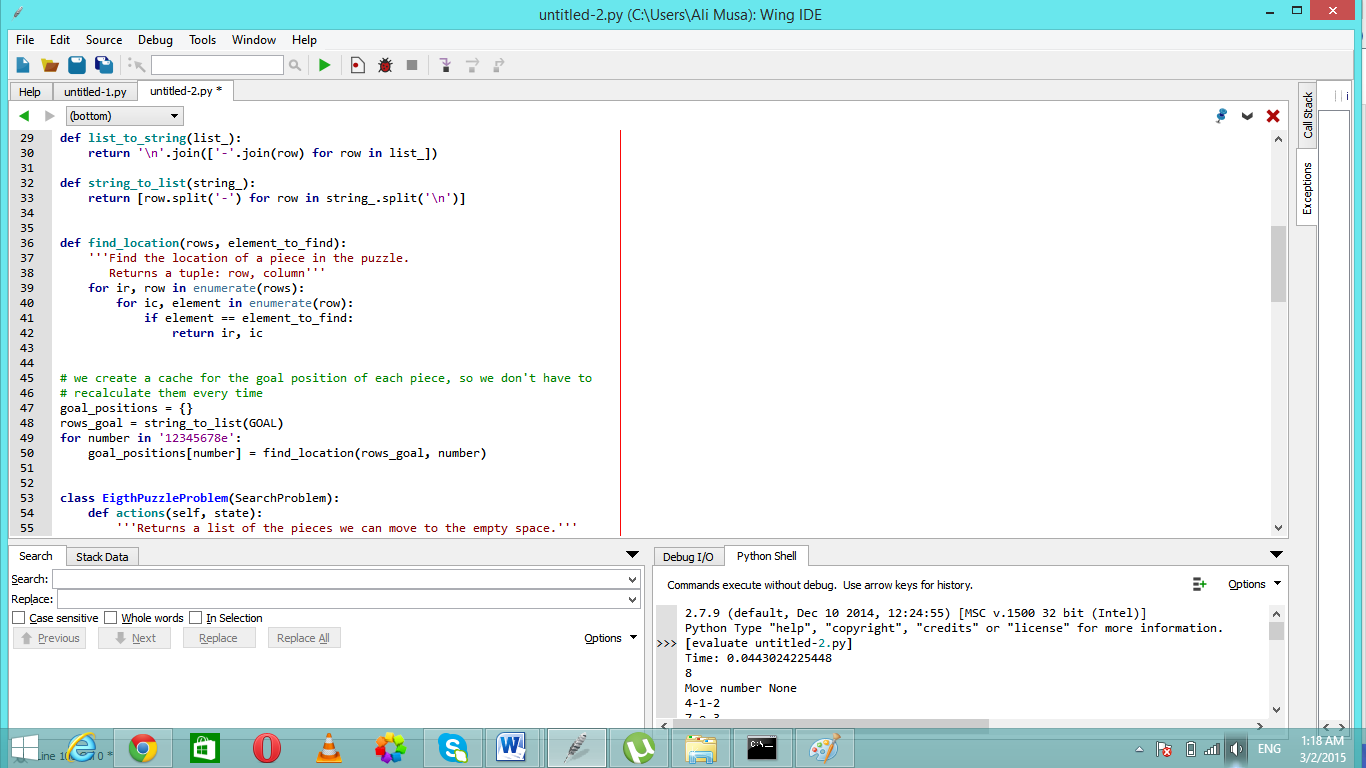
print result.cost

# Solution

for action, state in result.path():

print 'Move number', action

print state



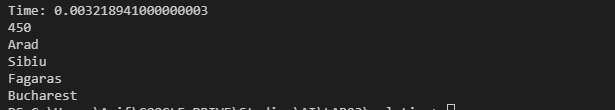
**Lab Task 1**

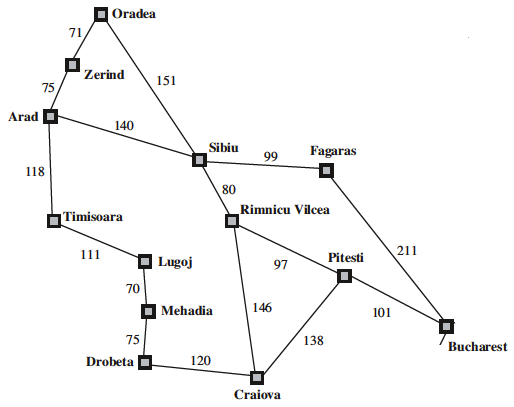
Solve the 8-puzzle problem using Breadth first, uniform cost, limited depth first, iterative limited depth first (These algorithms are implemented in the simpleai library). Evaluate these algorithms by filling out the following table.

|  |  |  |
| --- | --- | --- |
| Algorithms / Performance | Cost | Time |
| Breadth First | 8 | 0.164756141889 |
| Uniform Cost | 8 | 0.034845555 |
| Limited Depth First (L=30) | 28 | 0.6275625469999999 |
| Limited Depth First (L=50) | 38 | 0.5147101780000001 |
| Limited Depth First (L=100) | 100 | 3.9826210559999997 |
| Iterative Limited Depth First | 8 | 0.017054783000000004 |

**Lab Task 2**

|  |  |  |
| --- | --- | --- |
| Algorithms / Performance | Cost | Time |
| Breadth First | 450:  Arad  Sibiu  Fagaras  Bucharest | 0.002259715999999995 |
| Uniform Cost | 418:  Arad  Sibiu  Rimnicu Vilcea  Pitesti  Bucharest | 0.004273104000000014 |
| Depth First | 733:  Arad  Timisoara  Lugoj  Mehadia  Drobeta  Craiova  Pitesti  Bucharest | 0.002138261999999988 |
| Limited Depth First (L=5) | 418  Arad  Sibiu  Rimnicu Vilcea  Pitesti  Bucharest | 0.002869548999999999 |
| Limited Depth First (L=10) | 733  Arad  Timisoara  Lugoj  Mehadia  Drobeta  Craiova  Pitesti  Bucharest | 0.00216434900000001 |
| Iterative Limited Depth First | 450  Arad  Sibiu  Fagaras  Bucharest | 0.018824835999999998 |





**Deliverables:** Upload a single Word file with the code in text boxes. The code should be properly documented.

**Time** to complete lab task: 3rd March 2015, 2355hrs