CSI344 Assignment 1
Semester 2 2020/2024
Pacman Game Using Uniform Cost Search
and 8 Puzzle Using Iterative Deepening
Submit by 11PM Monday 25th March 2024
Using the Assignment 1 (SUBMIT HERE) Link in Moodle

[100 marks]

Note: This assigment is meant to be done by each group without help from people external to the group. Any sign of involvement in copying or external assistance will result in zero being awarded for the lab. It is also your responsibility to ensure that no other person has access to your code.

Note: Your solution must follow the specifications given below precisely. Deviation will earn you a zero. Your program must run to earn any marks.

Note: Your program must use the combined functionality of the Iterative Deepening driver function and Depth First Search in order to return a result, not just one of them.

Download the search.zip file from Moodle. Enter your Student ID, Surname, Initials, Programme Code at the beginning of search.py and eightpuzzle.py. These are the only files you have to modify.

Once your code is complete, for Pacman (Part 1) run the relevant code using "python -l tinyMaze -p SearchAgent -a fn=ucs" from a Windows command prompt, once you are in the right directory. For eightpuzzle, run with "python eightpuzzle.py"

Welcome to Pacman (ai.berkeley.edu)

After downloading the code, unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line (Windows command prompt):

```
python pacman.py
```

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman's first step in mastering his domain.

The simplest agent in `searchAgents.py` is called the `GoWestAgent`, which always goes West (a trivial reflex agent). This agent can occasionally win:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

But, things get ugly for this agent when turning is required:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal.

Soon, your agent will solve not only `tinyMaze`, but any maze you want.

Note that `pacman.py` supports a number of options that can each be expressed in a long way (e.g., `--layout`) or a short way (e.g., `-l`). You can see the list of all options and their default values via:

```
python pacman.py -h
```

Running Pacmac Overview

In `searchAgents.py`, you'll find a fully implemented `SearchAgent`, which plans out a path through Pacman's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented -- that's your job. As you work through the following questions, you might find it useful to refer to the object glossary (the second to last tab in the navigation bar above).

First, test that the `SearchAgent` is working correctly by running:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

The command above tells the `SearchAgent` to use `tinyMazeSearch` as its search algorithm, which is implemented in `search.py`. Pacman should navigate the maze successfully.

Now it's time to write full-fledged generic search functions to help Pacman plan routes! You are to implement the recursive Depth First Search algorithm from the uninformed search slides. Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state.

Important note: All of your search functions need to return a list of *actions* that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).

Important note: **DO NOT use** the `Stack`, `Queue` and `PriorityQueue` data structures provided to you in `util.py`!

Implement the Uniform Cost Search algorihm in the UCS function in `search.py`. To make your algorithm *complete*, write the graph search version. You have to keep track of the visited states using a python set.

Your code should quickly find a solution for:

```
python pacman.py -l tinyMaze -p SearchAgent

python pacman.py -l mediumMaze -p SearchAgent

python pacman.py -l bigMaze -z .5 -p SearchAgent
```

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration).

Run the relevant code using “python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs” from a Windows command prompt, once you are in the right directory. Also use mediumMaze.

-l is for layout: the possible options are tinyMaze, mediumMaze and bigMaze.

-p allows agents to be selected but should remain fixed as SearchAgent for this lab.

-a allows functions to be specified, which are used as search algorithms. Enter fn=id for this lab. Another option is bfs.

Part 1. Uniform Cost Search for Pacman Implementation Overview

Study the Iterative Heuristic Search Algorithm in the slides carefully.

Modification of search.py

You should implement the **uniform cost search** algorithm in search.py.

You should implement **UCS** search using the heuristic search algorithm in the slides.

Note that UCS keeps track of costs. Assume each action has the same cost (1). But do not forget to add a heurstic cost for a state, as you implemented in the state class.

Note: In order to sort a list mylist, you can use the sorted(mylist) function. This function can accept a second parameter to identify an element of a tuple to use in sorting. Also do not forget to keep track of expanded nodes. Use a set variable named closed to store all nodes which have been expanded. Only consider nodes which are not members of this set.

In a comment at the bottom of your search.py file, answer the following questions.

i. What is the advantage does using Depth First Search for an algorithm like Iterative Deepening?
ii. Explain one disadvantage of Iterative Deepening as compared to Breadth First Search.

Part 2. Iterative Deepening Algortihm for Eightpuzzle Implementation Overview

Study the Iterative Deepening Algorithm in the slides carefully.

def IDA(problem): needs to be defined so that it calls **DFSB()**, which you should also define. The recursive version should take in a state, the problem, and a set which is used to keep track of visited states. **Make sure you use the recursive and depth first search algorithms in the slides.**
For **IDA()**:

- You states should have the structure **(state,path)**, where path is a list of actions, which should be returned in state is the goal.
- You first of all have to get the start state, and package it with an empty action list (since we do not need to make an moves to get to the start state) in a tuple. This is the start node.
- Initialise an empty set to store a set of nodes that have been expanded.
- A set can be initalized as: myset=set().
- Include all other required functionality in the driver function (IDA).

You need to define **DFSB(state,problem,depth,depthlimit,visited)**.

- Any empty list named x can be created thus: x=[]
- The problem object can be used to check for goals; it has a method for this. To invoke a method m of object o, use o.m().
- In order to see which methods a problem object has, you can inspect the code.
- A newstates list should be a list of tuples as follows: [(state1,[list of actions]), (state2, [list of

actions]), ...]

- the problem object also has a method to generate successors.
- Successors are tuples of (state,action,cost).
- To check if x is in a set s, us “if x in s:”
- To add x to a set s, use “s.add(x)”
- To create a tuple, use: newtuple=(element1,element2), for a two element tuple, for example.
- To add a new element to a list, use mylist=mylist+[element]
- To get the first element of a list, use mylist[0].
- To return a Null, use “return None”.
- To check if returned value is None, use “if myvalue is None”.
- To extract pat of a list, skipping the first three elements, use mylist[3:].
- The problem methods isGoalState() and getSuccessors() will come in handy.
- **Do not expand any node that has been expanded previously.**

The list of actions captures the actions which led to indicated state, such that once the goal is identified, this list has to be returned as a solution.

IDA() will keep calling DFSB() with increasing depth limits until it returns the solution. DFSB() will keep calling itself until it returns a solution or **None**. IDA() should get the actual parameters for the call to DFSB() in order, before calling.

Modification of eightpuzzle.py

Modify the following section.

```
if __name__ == '__main__':
```

After setting up the environment it should call the IDA search function to get a solution.

At the end, there should be information summarizing the results.

In a comment at the bottom of your eightpuzzle.py file, answer the following questions.

i. What is the advantage does using Depth First Search for an algorithm like Iterative Deepening?
ii. Explain one disadvantage of Iterative Deepening as compared to Breadth First Search.

Comment your code, write your Surname, Initials, ID number, and programme code in your search.py and eightpuzzle.py files. The group leader should zip the files and submit the archive.

Contribution Information to be Submitted by Each Student as a pdf using the Assignment 1 (SUBMIT HERE) link; submitting this information is mandatory. **Students will be penalized for failing to submit Contribution Information.**

- ID, Surname, Initials of student submitting information.
- ID, Surname, Initials of each other group member and details of his/her contribution to assignment i.e. how the group member contributed to each Part of the assignment.

and submit it using the Assignment 1 (SUBMIT HERE) link.

Table I. Group Membership

ID	Surname	Group Leader	Group No.
202203639	Kootsere	*	1
202107679	Pule		1
201800332	Johnson		1
202000969	Ponego		1
202105130	Mogotlhwane	*	2
202103018	Setilo		2
201602388	Letlole		2
202100060	Fakudze		2
202105176	Malonje	*	3
202102742	Mndolo		3
202100148	Nanduri		3
202108203	Kadzongwe		3
202102163	Williams	*	4
202107760	Titus		4
202005767	Ndungo		4
202004347	Tamajasi		4
202001186	Ndwapi	*	5
202103415	Onneng		5
202100623	Ntwaetsile		5
202002603	Mokgoabone		5
202104527	Gabanakgotla	*	6
202001073	George		6
202103874	Molefabangwe		6
200901004	Mogotsi		6
202104060	Muyobo	*	7
202003674	Setlhako		7
202106954	Monageng		7
201801616	Malgas		7
202003402	Sibanda	*	8
202100521	Molebatsi		8
202005452	Zilwa		8
202005555	Mholo		8

~~~~~~~~~~ END OF ASSIGNMENT 1 ~~~~~~~~~~
~~~~~~~~~~