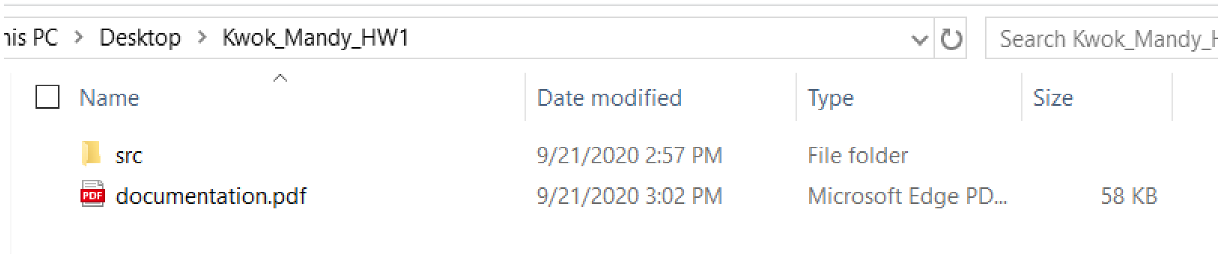
Mandy Kwok 9/21/20
111458972 CSE537 Artificial Intelligence

Assignment# 1

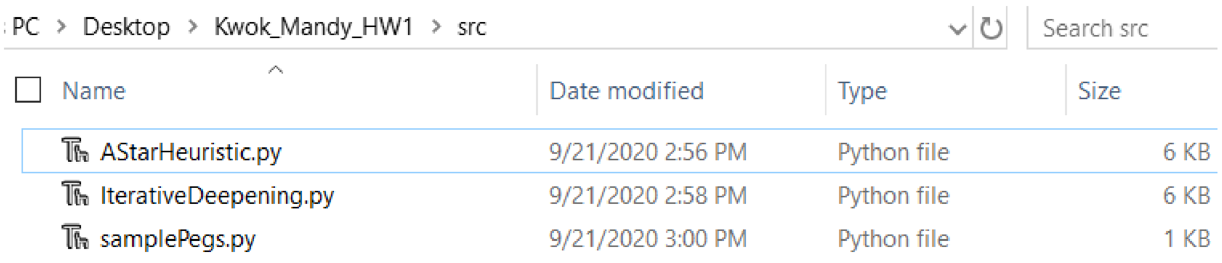
File Structure



ɪis PC > Desktop > Kwok_Mandy_HW1

Search Kwok_Mandy_I

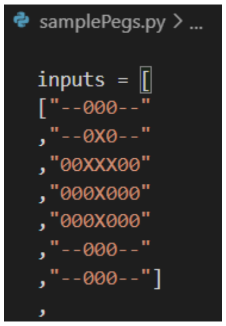
Name	Date modified	Type	Size
src	9/21/2020 2:57 PM	File folder	
documentation.pdf	9/21/2020 3:02 PM	Microsoft Edge PD...	58 KB



PC > Desktop > Kwok_Mandy_HW1 > src

Search src

Name	Date modified	Type	Size
AStarHeuristic.py	9/21/2020 2:56 PM	Python file	6 KB
IterativeDeepening.py	9/21/2020 2:58 PM	Python file	6 KB
samplePegs.py	9/21/2020 3:00 PM	Python file	1 KB



samplePegs.py > ...

```
inputs = [
["--000--"
,"--0X0--"
,"00XXX00"
,"000X000"
,"000X000"
,"--000--"
,"--000--"]
,
```

src folder contains all python script files.

samplePegs.py contains all the test sets represented in python arrays with symbols {X,0,-}

IterativeDeepening.py contains code that takes input from samplePegs.py and display required outputs on screen using Iterative Deeping Search algorithm.

AStarHeuristic.py contains code that takes input from samplePegs.py and display required outputs on screen using A* algorithm with two heuristics in choice.

Compilation

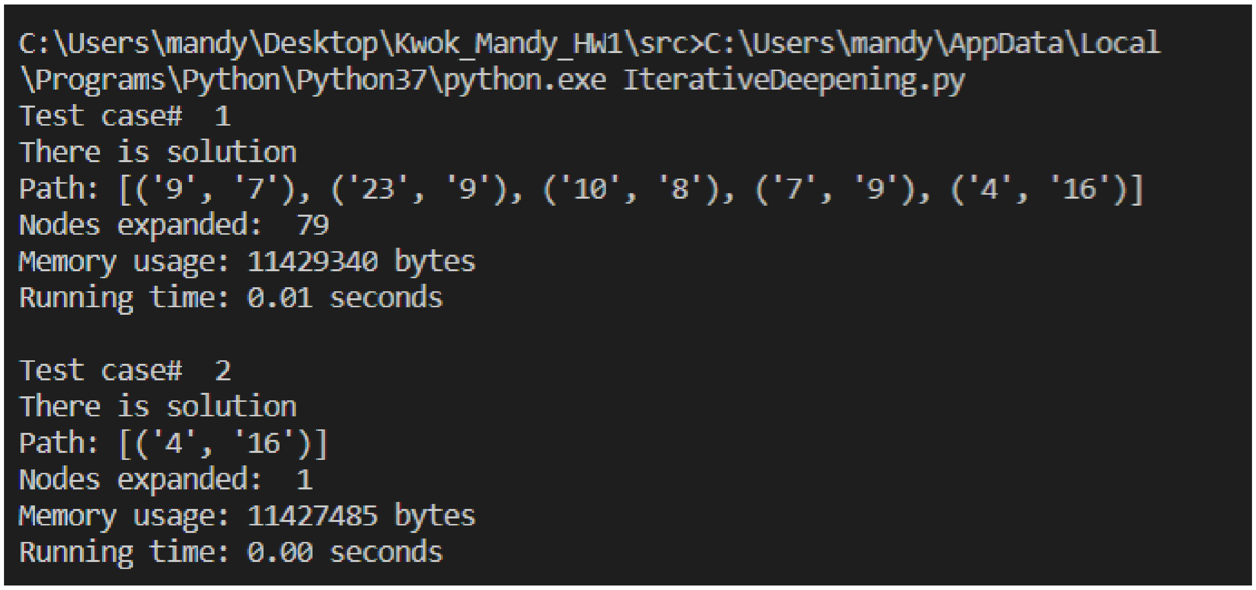
After unzipping the file, navigate into directory Kwok_Mandy_HW1/src

Open terminal in this directory

Install file dependencies if applicable (pip install guppy3, pip install numpy, etc.)

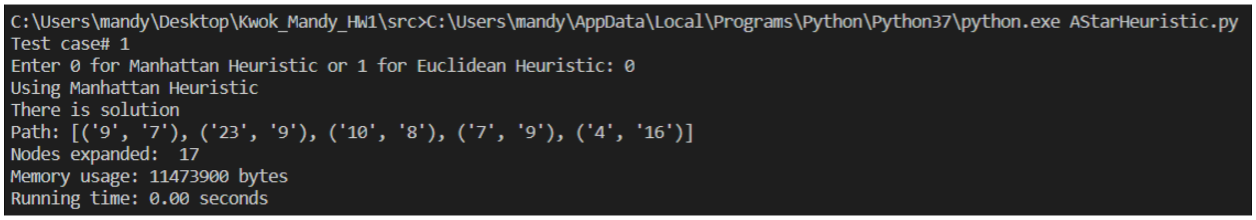
Run python with either one of IterativeDeepening.py or AStarHeuristic.py as argument

Sample output:

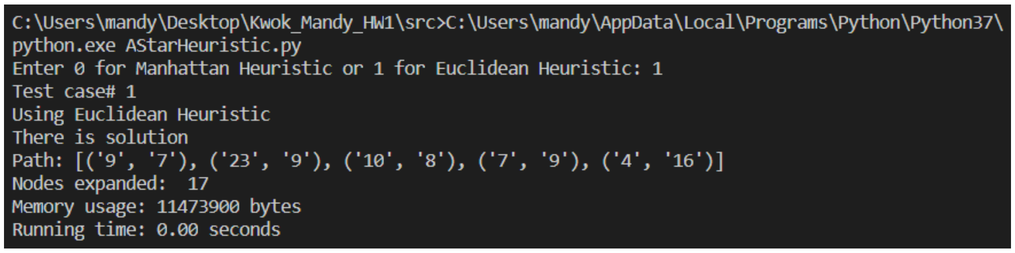


```
C:\Users\mandy\Desktop\Kwok_Mandy_HW1\src>C:\Users\mandy\AppData\Local
\Programs\Python\Python37\python.exe IterativeDeepening.py
Test case#  1
There is solution
Path: [('9', '7'), ('23', '9'), ('10', '8'), ('7', '9'), ('4', '16')]
Nodes expanded:  79
Memory usage: 11429340 bytes
Running time: 0.01 seconds

Test case#  2
There is solution
Path: [('4', '16')]
Nodes expanded:  1
Memory usage: 11427485 bytes
Running time: 0.00 seconds
```



```
C:\Users\mandy\Desktop\Kwok_Mandy_HW1\src>C:\Users\mandy\AppData\Local\Programs\Python\Python37\python.exe AStarHeuristic.py
Test case# 1
Enter 0 for Manhattan Heuristic or 1 for Euclidean Heuristic: 0
Using Manhattan Heuristic
There is solution
Path: [('9', '7'), ('23', '9'), ('10', '8'), ('7', '9'), ('4', '16')]
Nodes expanded:  17
Memory usage: 11473900 bytes
Running time: 0.00 seconds
```



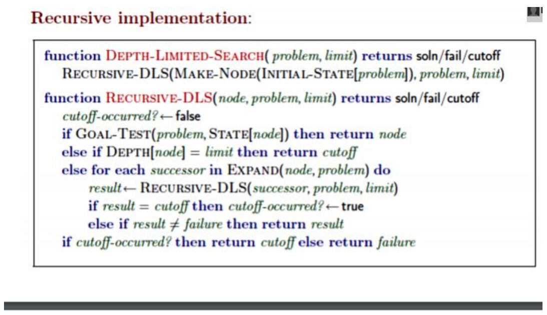
```
C:\Users\mandy\Desktop\Kwok_Mandy_HW1\src>C:\Users\mandy\AppData\Local\Programs\Python\Python37\
python.exe AStarHeuristic.py
Enter 0 for Manhattan Heuristic or 1 for Euclidean Heuristic: 1
Test case# 1
Using Euclidean Heuristic
There is solution
Path: [('9', '7'), ('23', '9'), ('10', '8'), ('7', '9'), ('4', '16')]
Nodes expanded:  17
Memory usage: 11473900 bytes
Running time: 0.00 seconds
```

For AStarHeuristic.py, please input 0 or 1 to decide the heuristic method use by A* search.

For prune method, please input 0(No) or 1(Yes) to indicate whether to run with prune

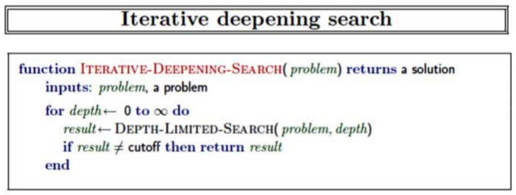
Trace of execution

IterativeDeepening.py



Recursive implementation:

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```



Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

I implemented the Iterative Deepening Search algorithm by following the recursive approach. The variable named **problem** would be a pegs board represented in Python array with the symbol{X,0,-} and index [0-32]. The variable named **limit** would be the number of filled peg minus 1. For example, the peg on homework document has 6 filled positions and we need exactly 5 moves to reach goal, thus (6-1)= 5. The IDS algorithm will return three value: a list of moves(path), count of expanded nodes, and boolean solutionFound. I keep track of the path by getting the output path from recursive call of successors and return the path appended with the output path. And I check goal state when there is only one filled peg at the center. To do this, I also keep track of the successor game state by passing it as argument.

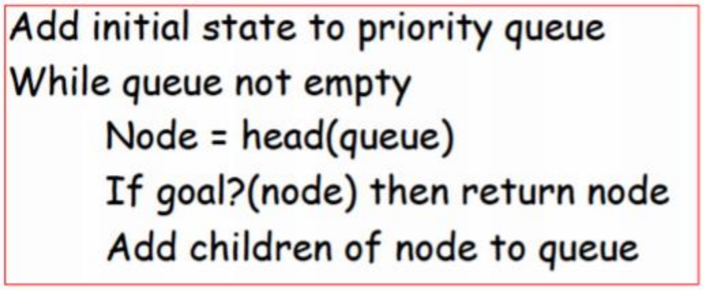
If output = [], 0, False , the test case has no solution found

If output = [], 0, True , the test case is already in goal state

If output = [(a,b),(c,d)], x, True, the test case has solution with path (a,b),(c,d) and x nodes expanded

From the output of the ITD algorithm, ITD usually result in larger number of nodes expanded but uses less memory storage. Space usage is linear because other nodes exceed the depth limit will not be expanded.

AStarHeuristic.py



```
Add initial state to priority queue
While queue not empty
    Node = head(queue)
    If goal?(node) then return node
    Add children of node to queue
```

I implemented the A*Search algorithm by using BFS with priority queue (similar to UCS but using heuristic values for priority) and two heuristic method choices. First add the start state to the priority queue. While not all possible states are visited, pop from the priority queue and expand this node to get its successors. For each successor, push to priority queue with the value of path length + heuristic value. If any of the successor is a goal state, return the tracked path and other required values. Else, repeat this process(pop, expand, and check goal).

Manhattan Heuristic

This heuristic function receives a move input (Ex. (9,7))

Since 7 is the final position after the move, calculate its coordinate in the 2d array of pegs.

Calculate the center position of the game board.

Return their distance as calculated by the formula abs(y1 – y0) + abs(x1 – x0)

Position farther away from center will return a greater heuristic value, less prioritize

Less Node Heuristic

This heuristic function receives a pegs board as argument.

Less nodes on the board can imply closer to goal state, therefore the less node on board the less heuristic value return(more prioritize).

Count the number of filled position and return.

Euclidean Heuristic

Similar as Manhattan Heuristic but use a more accurate distance formula. Return ((y1-y0)**2 + (x1-x0)**2)**0.5

From the output of the ITD algorithm, ITD usually result in lower number of nodes expanded but uses more memory storage. An effective heuristic method will give more accurate estimate of cost from certain node to goal, thus better prediction that leads to goal with less iteration.

Performance: LessNode>Manhattan=Euclidean

Mandy Kwok 9/21/20
111458972 CSE537 Artificial Intelligence

Prune method

Set up a dictionary that holds a game state as index and a list [path, next state] as value. This dictionary will keep track of all visited game state. If an encountered game state is in the dictionary or visited, take the value directly without having to expand the node and increment the expand count.

For IDS, assign values in the dictionary before any return statement that contains the path and next game state. After initializing the dictionary, check if this game state is in the dictionary, only if not we run the recursive DLS to get the result.

For A*, assign values in the dictionary before any return statement that contains the path and next game state. Check if the game state is in dictionary. If so, get the successor directly from dictionary without calling the get Successor method to expand the node.