CMPE480 – PROJECT1

2018400291

ABDULLAH YILDIZ

I developed the project with Python language.

HOW TO RUN

To run the program in Linux, write following to terminal

```
python proj1 [Filename] [Algorithm_Choice]
```

where Algorithm_Choice is one of the [dfs, bfs, ucs, gs, as]

To run on Windows, either open the .py file in IDE or write the same code for Linux in the terminal

EXPLANATIONS

My algorithm prioritize directions in order L, U, R, D .

I read input from terminal with argv keyword via sys library, open the directory given in input and read its inputs: agent coordinates, obstacles, goal coordinates. Then I create a map having obstacles as 'h' characters. Whole map including agent, obstacles and goal can be printed with printmap(agent:Agent) function. This helped visualizing the map when debugging.

Orientation::ENUM

Shows the state of the agent as

Orientation.SINGLE , agent occupies a square

Orientation.VERTICAL , agent occupies adjacent two vertical squares

Orientation.HORIZONTAL, agent occupies adjacent two horizontal squares

Direction:: ENUM

Represents the move direction agent makes

LEFT
UP
RIGHT
DOWN

Agent::Class

Agents represent the moving object, members:

orientation=Orientation.SINGLE, Orientation.VERTICAL, Orientation.HORIZONTAL
self.xvalue=1 or 2 member list of integers
self.yvalue= 1 or 2 member list of integers
self.cost= the cost agent produced to achieve its place
self.path= the path agent followed to achieve this place
self.depth=the depth agent branched to achieve this place

printagentinfo(self) = prints the agent' members line by line

moveagent(agent:Agent, direction:Direction)

Takes parameters an agent and a direction. With respect to the state of the agent, changes the orientation of the agent and its coordinates.

Sorts the x and y value of the agent to make sure it is in ascending order thus not being copied to the TRAVERSED list more than once.

Returns agent

isoktomove(agent:Agent, direction:Direction)

Takes an agent and a Direction and decides that if it is legal to move to the direction.

Legal move is a move which doesn't make agent conflict with 'h' and doesn't let agent cross the borders of the map.

calculate_cost(agent:Agent, direction:Direction)

Takes an agent and a Direction as parameters and returns an integer.

If the size shrinks cost is 3, else 1

doesAgentTouchGoal(agent:Agent)

Takes agent as parameter and decides if any part of agent is on top of goal.

isGoalAchieved(agent:Agent)

Returns True if agent's orientation is Single and it is on top of goal.

greedy_estimate(agent:Agent)

Pseudo code is as follows

initializes totalcost=0

while agent doesn't touch goal:

    get a direction toward goal

    move agent in the direction

    calculate the cost in the direction

    increase the totalcost by cost

end while

return totalcost

get_direction(agent:Agent)

Takes agent as parameter and returns a direction according to its state prioritizing L, U, R, D

If agent.orientation == Orientation.single:

Try to choose directions in order L, U, R, D

Else if agent.orientation == Orientation.VERTICAL

Try to go Left if possible, Right otherwise (cost==1) ***preferred***

Try to go Up if possible, Down otherwise (cost==3)

Else if agent.orientation == Orientation.HORIZONTAL

Try to go Up if possible, Down otherwise (cost==1) ***preferred***

Try to go Left if possible, Right otherwise (cost==3)

DEPTH-FIRST SEARCH ALGORITHM

STACK : holds the main stack for depth-first search

TRAVERSED: a list to keep track of expanded nodes

In this algorithm, we used a directions list [Down,Right,Up,Left] to be able to push the agents to STACK in reverse order thus making possible to pop in the desired order Left, Up, Right, Down.

Pseudo-code is as follows:

begin

STACK=TRAVERSED=[]

max_depth=0

While there is an element in the STACK:

Pop element from the end of list

If element is traversed, continue

Else add to TRAVERSED list

For each direction:

If it is legal to move

NEW_AGENT=Move agent

NEW_AGENT's path, cost and depth is updated

Update max_depth

If (Goal) return

If(NEW_AGENT is not traversed)

Add to STACK

End

Function returns

- *the last agent to reach the goal*
- *TRAVERSED list*
- *max_depth*

Output:

52 35 32 32

RRDLLULURDLLURDRRDLLLURRDRURDRRD

1)agent.cost

2) length of TRAVERSED

3 max_depth

4) agent.depth

5) agent.path

Apart from the instructor's solution, my solution chooses to go RIGHT instead of DOWN after "RRDLLULURDLLURD" is read. (underlined)

My sol'n : RRDLLULURDLLURDRRDLLLURRDRURDRRD

Instr sol'n: RRDLLULURDLLURDDLURRDRRRRDLLULDRRDLURD

Other output values are different obviously.

BREADTH-FIRST SEARCH ALGORITHM

Breadth first search very similar to depth-first search .

Differs from DFS **in the order of directions** that they are pushed onto the STACK

and the popping element.

Directions=[L, U, R, D]

Pop(0) is used in BFS to be able to expand nodes level by level

Function returns

- *the last agent to reach the goal*
- *TRAVERSED list*
- *max_depth*

Output:

11 35 7 7

RRDRRRD

1)agent.cost

2) length of TRAVERSED

3 max_depth

4) agent.depth

5) agent.path

UNIFORM-COST SEARCH ALGORTIHM

This algorithm pushes the agent pairs to the STACK just like the DFS and BFS.

Each move agent makes, new_agent is created. New_agent is added to STACK if not added before. At each addition to the STACK, it is sorted with respect to the agent.cost to prioritize low-cost agents.

STACK=sorted(STACK,key=lambda x : x.cost)

Low-cost agents are popped from the list by *pop(0)* command since they are the first element in the list.

Function returns

- *the last agent to reach the goal*
- *TRAVERSED list*
- *max_depth*

Output:

10 32 8 8

DRRRRRRD

1)agent.cost

2) length of TRAVERSED

3 max_depth

4) agent.depth

5) agent.path

GREEDY SEARCH ALGORITHM

This algorithm uses the similar technique in Uniform-Cost Search but a new pair *(greedy_cost, agent)* is pushed to the STACK.

Heuristics:

Lowest greedy_cost will be preferred.

greedy_cost is the estimate cost calculated in greedy_estimate function

greedy_estimate calculates the estimate cost to reach to the goal by moving agent toward the goal without checking the borders. It prefers the lowest-cost moves. (stays away from shrinking size)

pseudo-code:

begin

cost=0

while agent does not touch goal:

get a direction according to agent's orientation (get_direction(agent))

cost is increased by cost of moving agent in direction

agent moves

return cost

end

Function returns

- *the last agent to reach the goal*
- *TRAVERSED list*
- *max_depth*

Output:

10 41 17 8

DRRRRRRD

1)agent.cost

2) length of TRAVERSED

3 max_depth

4) agent.depth

5) agent.path

ASTAR ALGORITHM

This algorithm uses the similar technique in Greedy Search but the cost to sort the STACK is calculated as

Heuristics

A* cost= cost + greedy_cost

Uses the technique in greedy search cost finding function in which agent decides to move in a specific direction regardless of checking if it is a legal move or not.

Moves toward the goal until it touches the goal.

greedy_cost is accumulated along the way from the agent's initial position to the goal position.

Cost is the cost to reach to the current point from the very beginning of the search.

Function returns

- *the last agent to reach the goal*
- *TRAVERSED list*
- *max_depth*

Output:

10 10 8 8

DRRRRRRD

1)agent.cost

2) length of TRAVERSED

3 max_depth

4) agent.depth

5) agent.path