# SEA400- Assignment 2

| Total Mark: | 10 marks |
| --- | --- |
| Submission file(s): | * Asg2.docx * search.py with your solution |

Please work in **groups** to complete this lab. This lab is worth 7% of the total course grade and will be evaluated through your written submission, as well as the lab demo. During the lab demo, group members are *randomly* selected to explain the submitted solution. Group members absent during the lab demo will lose the demo mark.

Please submit the submission file(s) through Blackboard. Only one person must submit for the group and only the last submission will be marked.

Please cite any resources you used.

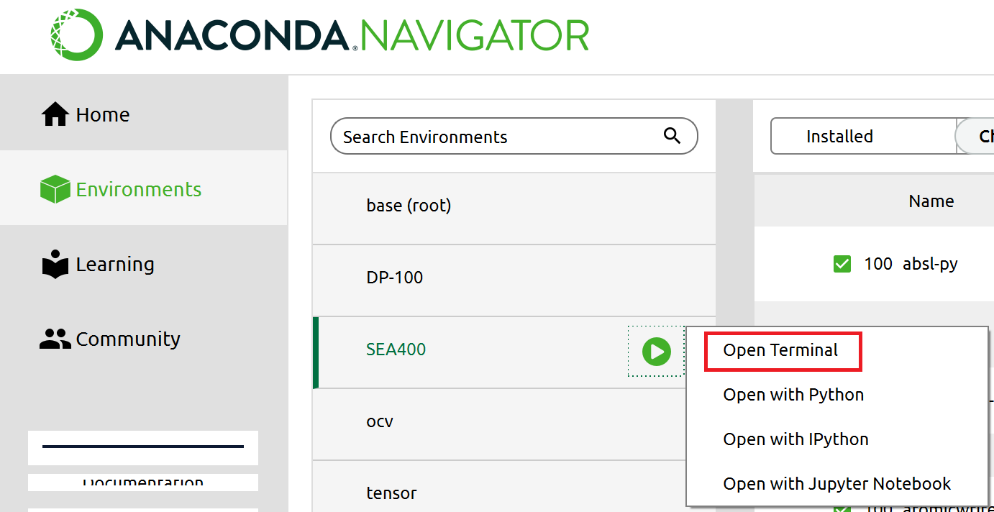
## **Part I: Search Algorithms**

1. In this assignment, you will be implementing uninformed and heuristic search algorithms. You will be using code from

[Project 1 | CS 188 Spring 2023 (berkeley.edu)](https://inst.eecs.berkeley.edu/~cs188/sp23/projects/proj1/)

Download and unzip <https://inst.eecs.berkeley.edu/~cs188/sp23/assets/projects/search.zip>

1. In Anaconda Navigator, go to Environments tab and click on the triangle next to SEA400. Choose “Open Terminal”.



Change directory to the unzipped folder, for example (this may be different from your path):

cd C:\SEA400\search

Keep this terminal open and run code here. For example, you can play a game of Pacman by running:

python pacman.py

1. Open Anaconda Navigator, activate SEA400 environment and launch VS Code (or your editor). Open above unzipped folder.
2. Follow instructions to solve and test questions 1 to 4 from above link (Project 1).

For each question,

* + write the algorithm (pseudo-code) you implemented, for example:



* + Explain how you implemented each step of the algorithm.

**depth first search ():**

current = <- getstartstate()

nodes <- push current node to stack

expanded <- a list of expanded nodes

parents <- a dictionary of nodes, this is used to store the moves to a goal state

A screen shot of a computer

Description automatically generated

while nodes is not empty:

current <- retrieve item at the top of the stack, or the last in.



if current is the goal:

return the path taken to get there

A screen shot of a computer

Description automatically generated

if the current node is not in visited

add it to the list of visited nodes



if successor not in expanded

append the parents and action

push to the stack

return an empty list if nothing is found

A screen shot of a computer screen

Description automatically generated

**breadth first search ():**

current = <- getstartstate()

nodes <- push current node to queue

expanded <- a list of expanded nodes

parents <- a dictionary of nodes, this is used to store the moves to a goal state

A screen shot of a computer

Description automatically generated

while nodes is not empty:

current <- retrieve item at the top of the queue, or the first in.



if current is the goal:

return the path taken to get there

A screen shot of a computer

Description automatically generated

if successor not in expanded

append the parents and action

add to the expanded nodes list

push to the queue

return an empty list if nothing is found

A screen shot of a computer

Description automatically generated

**uniform cost search ():**

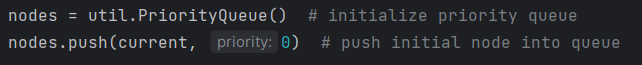
start state <- getstartstate()

nodes <- priorityQueue()

expanded <- a list of expanded nodes

parents <- a dictionary of nodes, this is used to store the moves to a goal state





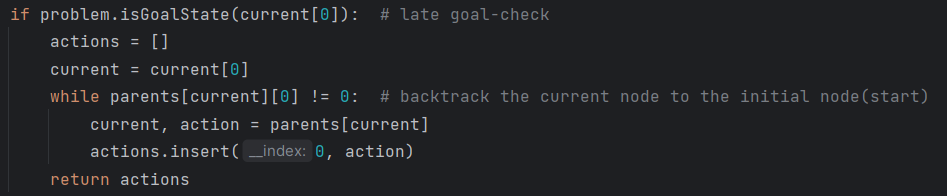
while nodes is not empty:

current <- retrieve highest priority element from the queue (the one with lowest cost)



if current is the goal:

return the path taken to get there



else

calculate all the successor’s movement cost using only the actual cost



if successor not in expanded or if the successor has a lower cost than the expanded node

append the parents and action

add to the expanded nodes list

push to the queue

return an empty list if nothing is found

A computer screen shot of a black screen

Description automatically generated

**astar search ():**

start state <- getstartstate()

nodes <- priorityQueue()

expanded <- a list of expanded nodes

parents <- a dictionary of nodes, this is used to store the moves to a goal state

A screen shot of a computer

Description automatically generated

while nodes is not empty:

current <- retrieve highest priority element from the queue (the one with lowest cost)



if current is the goal:

return the path taken to get there

A screen shot of a computer

Description automatically generated

else

calculate a movement cost combining the heuristic and actual cost

if successor not in expanded or if the successor has a lower cost than the expanded node

append the parents and action

add to the expanded nodes list

push to the queue

return an empty list if nothing is found

A screen shot of a computer code

Description automatically generated

1. If you want to use the debugger in VS code, see:

[Debugging in Visual Studio Code](Debugging%20in%20Visual%20Studio%20Code) <https://code.visualstudio.com/Docs/editor/debugging>

For example, to run the debugger with *autograder.py* test samples for question 1, change *launch.json* to include

"args": ["-q", "q1"],

(Have autograder.py open and press F5)

## **Part II: Run Tests**

Paste screenshots of the results of the following commands here:

python autograder.py -q q1

Result:

A screenshot of a computer

Description automatically generated

python autograder.py -q q2

Result:

A screenshot of a computer

Description automatically generated

python autograder.py -q q3

Result:

A screenshot of a computer

Description automatically generatedA computer screen shot of a black screen

Description automatically generated

python autograder.py -q q4

Result:

A screenshot of a computer

Description automatically generated

python submission\_autograder.py

Result:

A screenshot of a computer

Description automatically generated

## **Part III: Group work**

1. Complete this declaration by adding your names:

We, Chet, Kasra, Nahaeli, Atem-Ako, declare that the attached assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this assignment, manually or electronically, from any other source including web sites, unless specified as references. We have not distributed our work to other students.

1. Specify what each member has done towards the completion of this work:

|  |  |  |
| --- | --- | --- |
|  | Name | Task(s) |
| 1 | Sthapanavichet Long | All parts |
| 2 | Atem-Ako Eyong Atem | All parts |
| 3 | Nahaeli Brunder | All parts |
| 4 | Kasra Bina | All parts |