****

**Ahsanullah University of Science and Technology**

*Department of Computer Science & Engineering*

|  |  |
| --- | --- |
| Course No. | CSE 4108 |
| Course Name | Artificial Intelligence Lab |
| Assignment No. | 05 |

**Submitted To:**

Md. Siam Ansary Tonmoy Hossain

Department of CSE, AUST Department of CSE, AUST

**Submitted By:**

|  |  |
| --- | --- |
| Name | Tahiya Ahmed Chowdhury |
| ID No. | 17.02.04.048 |
| Session | Fall – 2020 |
| Section | A (A2) |

**Date of Submission:** September 19,2021

**Best first search code:**

**import** copy

heuristic **=** **{**'S'**:** 10**,** 'A'**:** 9**,** 'B'**:** 7**,** 'C'**:** 8**,** 'D'**:** 8**,** 'H'**:** 6**,** 'L'**:** 6**,** 'F'**:** 6**,** 'G'**:** 3**,** 'I'**:** 4**,** 'J'**:** 4**,** 'K'**:** 3**,** 'E'**:** 0**}**

traversal **=** **{**

'S'**:** **[**'A'**,** 'B'**,** 'C'**],**

'A'**:** **[**'B'**,** 'D'**],**

'B'**:** **{**'H'**,** 'D'**},**

'C'**:** **[**'L'**],**

'D'**:** **[**'F'**],**

'H'**:** **[**'F'**,** 'G'**],**

'L'**:** **[**'I'**,** 'J'**],**

'G'**:** **[**'E'**],**

'I'**:** **[**'K'**],**

'J'**:** **[**'K'**],**

'K'**:** **[**'E'**]**

**}**

**def** Best\_First\_Search**(**start**,** end**):**

path **=** **[]**

Q **=** **[]**

priorityQueue **=** **[[[**start**],** heuristic**[**start**]]]**

visited **=** **[]**

**while** priorityQueue **!=** **[]:**

Q **=** priorityQueue**.**copy**()**

path**.**append**(**priorityQueue**.**pop**(**0**))**

node **=** path**[-**1**][**0**][-**1**]**

visited**.**append**(**node**)**

**if** node **==** end**:**

finalPath **=** copy**.**deepcopy**(**path**[-**1**])**

**print(**"Final Path"**,** finalPath**[**0**:**1**])**

**return** "Found"

**for** neighbor **in** traversal**[**node**]:**

**if** neighbor **not** **in** visited**:**

newPath **=** copy**.**deepcopy**(**path**[-**1**])**

newPath**[**0**].**append**(**neighbor**)**

newPath**[**1**]** **=** heuristic**[**neighbor**]**

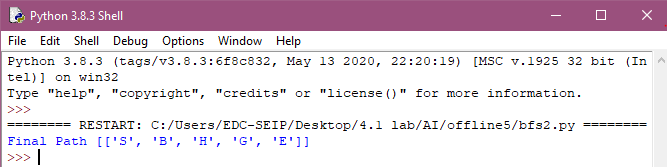
priorityQueue**.**append**(**newPath**)**

priorityQueue**.**sort**(**key**=lambda** x**:** x**[**1**])**

**print(**"Visited"**,** visited**)**

Best\_First\_Search**(**'S'**,** 'E'**)**

**Best first search output:**

****

**A star search code:**

**class** **Graph:**

**def** \_\_init\_\_**(**self**,** graph**):**

self**.**graph **=** graph

**def** get\_neighbors**(**self**,** v**):**

**return** self**.**graph**[**v**]**

**def** h**(**self**,** n**):**

Heuristic **=** **{**'S'**:** 10**,** 'A'**:** 9**,** 'B'**:** 7**,** 'C'**:** 8**,** 'D'**:** 8**,** 'H'**:** 6**,** 'L'**:** 6**,** 'F'**:** 6**,** 'G'**:** 3**,** 'I'**:** 4**,** 'J'**:** 4**,** 'K'**:** 3**,** 'E'**:** 0**}**

**return** Heuristic**[**n**]**

**def** A\_star\_algorithm**(**self**,** start\_node**,** end\_node**):**

open\_list **=** **set([**start\_node**])**

closed\_list **=** **set([])**

li **=** **[]**

g **=** **{}**

g**[**start\_node**]** **=** 0

parents **=** **{}**

parents**[**start\_node**]** **=** start\_node

**while** **len(**open\_list**)** **>** 0**:**

n **=** **None**

**for** v **in** open\_list**:**

**if** n **==** **None** **or** g**[**v**]** **+** self**.**h**(**v**)** **<** g**[**n**]** **+** self**.**h**(**n**):**

n **=** v

**if** n **==** **None:**

**print(**'Path does not exist!'**)**

**return** **None**

**if** n **==** end\_node**:**

li**.**append**(**n**)**

**print(**"Visited: "**,** li**)**

reconst\_path **=** **[]**

**while** parents**[**n**]** **!=** n**:**

reconst\_path**.**append**(**n**)**

n **=** parents**[**n**]**

reconst\_path**.**append**(**start\_node**)**

reconst\_path**.**reverse**()**

**print(**'Final Path: {}'**.format(**reconst\_path**))**

**return** reconst\_path

**for** **(**m**,** weight**)** **in** self**.**get\_neighbors**(**n**):**

**if** m **not** **in** open\_list **and** m **not** **in** closed\_list**:**

open\_list**.**add**(**m**)**

parents**[**m**]** **=** n

g**[**m**]** **=** g**[**n**]** **+** weight

**else:**

**if** g**[**m**]** **>** g**[**n**]** **+** weight**:**

g**[**m**]** **=** g**[**n**]** **+** weight

parents**[**m**]** **=** n

**if** m **in** closed\_list**:**

closed\_list**.**remove**(**m**)**

open\_list**.**add**(**m**)**

li**.**append**(**n**)**

**print(**"Visited: "**,** li**)**

open\_list**.**remove**(**n**)**

closed\_list**.**add**(**n**)**

**print(**'Path does not exist!'**)**

**return** **None**

graph **=** **{**

'S'**:** **[(**'A'**,** 7**),** **(**'B'**,** 2**),** **(**'C'**,** 3**)],**

'A'**:** **[(**'B'**,** 3**),** **(**'D'**,** 4**)],**

'B'**:** **[(**'D'**,** 4**),** **(**'H'**,** 1**)],**

'C'**:** **[(**'L'**,** 2**)],**

'D'**:** **[(**'F'**,** 5**)],**

'H'**:** **[(**'F'**,** 3**),** **(**'G'**,** 2**)],**

'L'**:** **[(**'I'**,** 4**),** **(**'J'**,** 4**)],**

'F'**:** **[],**

'G'**:** **[(**'E'**,** 2**)],**

'I'**:** **[(**'K'**,** 4**)],**

'J'**:** **[(**'K'**,** 4**)],**

'K'**:** **[(**'E'**,** 5**)],**

'E'**:** **[]**

**}**

graph1 **=** Graph**(**graph**)**

graph1**.**A\_star\_algorithm**(**'S'**,** 'E'**)**

**while** parents**[**n**]** **!=** n**:**

reconst\_path**.**append**(**n**)**

n **=** parents**[**n**]**

reconst\_path**.**append**(**start\_node**)**

reconst\_path**.**reverse**()**

**print(**'Final Path: {}'**.format(**reconst\_path**))**

**return** reconst\_path

**for** **(**m**,** weight**)** **in** self**.**get\_neighbors**(**n**):**

**if** m **not** **in** open\_list **and** m **not** **in** closed\_list**:**

open\_list**.**add**(**m**)**

parents**[**m**]** **=** n

g**[**m**]** **=** g**[**n**]** **+** weight

**else:**

**if** g**[**m**]** **>** g**[**n**]** **+** weight**:**

g**[**m**]** **=** g**[**n**]** **+** weight

parents**[**m**]** **=** n

**if** m **in** closed\_list**:**

closed\_list**.**remove**(**m**)**

open\_list**.**add**(**m**)**

li**.**append**(**n**)**

**print(**"Visited: "**,** li**)**

open\_list**.**remove**(**n**)**

closed\_list**.**add**(**n**)**

**print(**'Path does not exist!'**)**

**return** **None**

graph **=** **{**

'S'**:** **[(**'A'**,** 7**),** **(**'B'**,** 2**),** **(**'C'**,** 3**)],**

'A'**:** **[(**'B'**,** 3**),** **(**'D'**,** 4**)],**

'B'**:** **[(**'D'**,** 4**),** **(**'H'**,** 1**)],**

'C'**:** **[(**'L'**,** 2**)],**

'D'**:** **[(**'F'**,** 5**)],**

'H'**:** **[(**'F'**,** 3**),** **(**'G'**,** 2**)],**

'L'**:** **[(**'I'**,** 4**),** **(**'J'**,** 4**)],**

'F'**:** **[],**

'G'**:** **[(**'E'**,** 2**)],**

'I'**:** **[(**'K'**,** 4**)],**

'J'**:** **[(**'K'**,** 4**)],**

'K'**:** **[(**'E'**,** 5**)],**

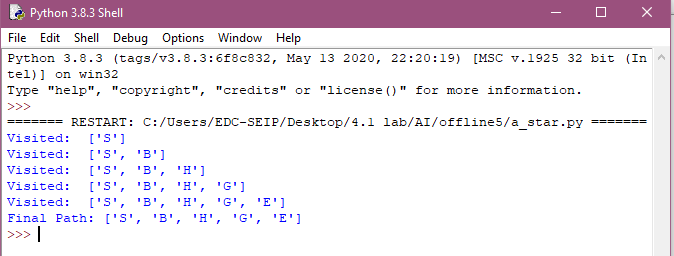
'E'**:** **[]**

**}**

graph1 **=** Graph**(**graph**)**

graph1**.**A\_star\_algorithm**(**'S'**,** 'E'**)**

**A star search output:**

****