1. Graph Data Structure

Keep a dictionary of the adjacent list as, ***adjacency_list***:

Nodename as key, value as list of tuples

Keep another dictionary as ***H***:

Nodename as key, heuristic value as value

2. Node Class

Write a python class named ***Node*** with the following attributes:

- ***nodename*** **: String**
- ***parent*** **: Node**
- ***g*** **: float**
- ***h*** **: float**
- ***f*** **: float**

Add a constructor that takes four parameters(***nodename***, ***parent***, ***g***, ***h***) to initialize the attributes

3. A* Search & Solution Finding

Create an empty list name ***priority_queue***

Create a Node object, ***NOb*** of the "S" node with (***nodename***: 'S', ***parent***: None, ***g***: 0, ***h***: H['S'])

and

Insert the node in ***priority_queue***

Now inside a while loop:

while ***priority_queue*** is not empty:

Find out the Node object in ***priority_queue*** with the minimum value of f

Extract it from the ***priority_queue*** and store it in ***NOb***

If ***NOb.nodename*** == *'G':*

break

For every neighbor of ***NOb.nodename*** from ***adjacency_list***

Insert a new node in ***priority_queue*** with (***nodename***: neighbor_name, ***parent***: ***NOb***, ***g***: ***NOb.g*** + edge_cost, ***h***: H[***NOb.nodename***])

Set ***NOb*** = *None*

4. Path Generation

path = []

cost = NOb.g

while ***NOb.parent*** is not None:

path.insert(***NOb.nodename***)

NOb = ***NOb.parent***

Reverse the path list

Print the path and cost