* 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

* Node Class

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

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

* 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*** *== ‘D’:*

*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*

* Path Generation

path = []

cost = NOb.g

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

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

Reverse the path list

Print the path and cost