Offline 01: A Star Tree Search

Implement A* search using python for finding the optimal path from startnode to goalnode in a **directed** graph. The x, y coordinates of each vertex/node will be given. ***Use the euclidean distance between any node to goal as the heuristic***. Define a function **heuristic** to calculate the heuristic of any node.
Here is a template code for taking inputs. A detailed pseudocode and a concise one are given in the next pages.

Sample Input	Sample output
V n_1 x_1 y_1 … E n_1 n_2 cost … Start node Goal node	(The optimal path from start state to goal state)
6 S 6 0 A 6 0 B 1 0 C 2 0 D 1 0 G 0 0 9 S A 1 S C 2 S D 4 A B 2 B A 2 B G 1 C S 2 C G 4 D G 4 S G	Solution path S–C–G Solution cost 6
3 S 1 2 A 2 2 G 4 5 3 S A 1 A G 1 S G 10 S G	Solution path S – A – G Solution cost 2

Instructions:

- Read the questions very carefully and answer all parts of the question.
- **The input will be given in input.txt file** and will be in the same folder as your code.
- Your code must be implemented for the given sample input format. Your output should also match the sample outputs. Your code will be tested on other inputs not given in the sample input
- You will get -100% for adopting any unfair means.
- **Your marks will fully depend on your viva and understanding.**
  - **Total 20 marks**
    - Output = 4 marks
    - Input + Heuristic = 6 marks
    - Astar search = 10 marks
- **Submit the .ipynb file**

Concise pseudocode

Initialize a **min priority queue** $minQ$ **ordered based on $f = (g+h)$**

$g \leftarrow 0$

$h \leftarrow$ euclidean_distance($startnode$, $goalnode$)

$f \leftarrow g+h$

create $startstate$($startnode$, g, f, parent=None)

insert $startstate$ to $minQ$

While $minQ$ **not** empty:

    $currstate \leftarrow$ extract_min($minQ$)

    If $currstate.node$ is the $goalnode$ **then**

        <u>print solution path and cost</u>

        return

    End if

    For each node M adjacent to node currstate.node **do**

        g $\leftarrow$ currstate.g + edge_cost(currstate.node, M)

        h $\leftarrow$ euclidean_distance(M, goalnode)

        f $\leftarrow$ g+h

        create $newstate$(M, g, f, **parent=*currstate***)

        insert $newstate$ to $minQ$

    End for

End while

Detailed Pseudocode

1. Keep a dictionary **coords** for keeping the x, y coordinates of each node where node id is the key. E.g. $\text{coords}[\text{'S'}]=(1,2)$, it stores the x, y coordinates of node S. **[done]**

2. Keep another dictionary of lists **adjlist** for keeping the adjacent nodes of node 'S', 'A', etc where node id is the key. E.g. $\text{adjlist}[\text{'S'}] = [(\text{'A'}, 1), (\text{'G'}, 10)]$, it will store the adjacent node and edge costs of S. **[done]**

3. The heuristic of any node is the euclidean distance from that node to the goal node. Define a function **heuristic** to calculate the heuristic of any node using its coords.

4. Make a class that maintains the state of each node while performing A*-search. A node state should store the **node id**, the immediate previous node state (**parent**), actual cost so far (**g**) and total estimated cost(**f** = g+h). [**Remember, there can be different state class objects with the same node id**]

5. Initialize a min priority queue **minQ** with a node state containing starting node **S**.

   Here, node_id = S, prev_node_state = NULL, actual_cost_so_far = 0, total_estimated_cost = 0 + 7 (*h value of S which is the euclidean distance between S(1,2) and G(4,5): sqrt(9+9)=3*1.414=4.242*)

6. **while minQ** not empty:

7.   extract the nodestate **N** from **minQ** that has the minimum total_estimated_cost

8.   **if N** is the destination (goal node), **then**

9.   Print the whole path by backtracking, print the cost

10.   **return**

11.   **for** each node **M** adjacent to node **N do**

12.   create a new node state **N_prime**.

13.   Here,   node_id = **M**,

14.   prev_node_state = **N**,

15.   actual_cost_so_far = **N**.actual_cost_so_far + edge_cost,

16.   total_estimated_cost = actual_cost_so_far + **heuristic**(**M**)

17.   push **N_prime** to **minQ**

Class Lecture:

Do not forget to bring a calculator in the Final exam.

A star search:

Actual_cost_so_far **g**

Heuristic_cost (estimated cost from current node to goal) **h**

total_estimated_cost **f** = g+h

Practice problem

1. Draw the **A* search** tree for finding the optimal path from B to E in the following **directed** graph. The x, y coordinates of each node are given. **Use the Euclidean distance between any node to goal as the heuristic**. Clearly show the value of g and h in each node in the search tree and finally show the found path with total path cost. **[5]**
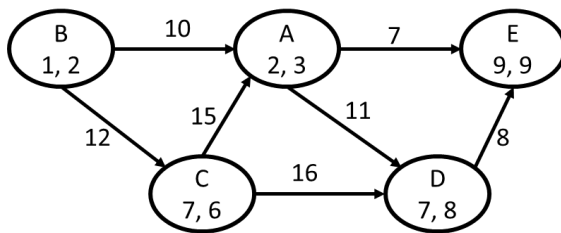
First, let's calculate the heuristic value for each node.

```
h(A) = dist(A, E) = sqrt( (2-9)^2+(3-9)^2) = 9.2
h(B) = dist(B, E) = sqrt( (1-9)^2+(2-9)^2) = 10.6
h(C) = dist(C, E) = sqrt( (7-9)^2+(6-9)^2) = 3.6
h(D) = dist(D, E) = sqrt( (7-9)^2+(8-9)^2) = 2.2
h(E) = dist(E, E) = 0
```
Now, let's draw the search tree.