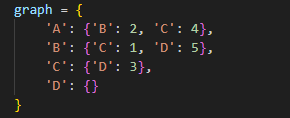
**Manual of code**

**Lab 07**

**Graph Representation**

The graph is represented as a dictionary where:

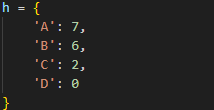
* Each key is a node (e.g., 'A', 'B', etc.).
* The value for each key is another dictionary representing the neighbors of that node and the cost to reach them.



* For example, 'A' is connected to 'B' with a cost of 2 and to 'C' with a cost of 4.
* 'D' has no outgoing connections, so it’s an empty dictionary.

**2. Heuristic Values**

The heuristic values (h) represent the estimated cost from each node to the goal node ('D'). These values are used to guide the A\* algorithm toward the goal.



* For example, the heuristic value for 'A' is 7, which means it’s estimated to cost 7 units to reach the goal ('D') from 'A'.
* The heuristic for the goal node ('D') is 0 because the cost to reach the goal from itself is zero.

***3. A Algorithm Implementation*\***

The A\_staric function implements the A\* algorithm. Here’s how it works:

Inputs

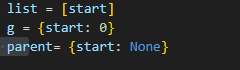
* start: The starting node (e.g., 'A').
* goal: The goal node (e.g., 'D').

**Variables**

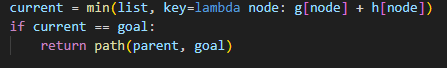
* list: A list of nodes to be explored (open list).
* g: A dictionary storing the actual cost from the start node to each node.
* parent: A dictionary storing the parent of each node to reconstruct the path later.

**Steps**

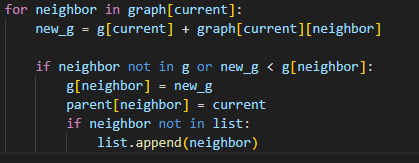
1. **Initialization:**
   * The list is initialized with the start node.
   * The g dictionary is initialized with the start node, and its cost is set to 0.
   * The parent dictionary is initialized with the start node, and its parent is set to None.

****

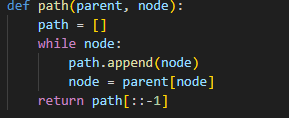
1. **Main Loop:**
   * The algorithm enters a loop that continues until the list is empty.
   * In each iteration, the node with the smallest f = g[node] + h[node] is selected as the current node. This is the node that is most promising to explore next.
   * If the current node is the goal, the algorithm reconstructs and returns the path using the path function.

****

1. **Exploring Neighbors:**
   * The current node is removed from the list.
   * For each neighbor of the current node, the algorithm calculates the new cost (new\_g) to reach that neighbor.
   * If the neighbor has not been visited or a cheaper path to it is found, the g and parent dictionaries are updated, and the neighbor is added to the list.

****

**4. Path Reconstruction**

The path function reconstructs the path from the start node to the goal node using the parent dictionary.

* It starts from the goal node and follows the parent pointers back to the start node.
* The path is reversed at the end to return it in the correct order (from start to goal).

**5. Example Execution**

Let’s walk through the example where the start node is 'A' and the goal node is 'D'.

**Step-by-Step Execution**

1. **Initialization:**
   * list = ['A']
   * g = {'A': 0}
   * parent = {'A': None}
2. **First Iteration:**
   * Current node: 'A' (since it’s the only node in the list).
   * Neighbors of 'A': 'B' and 'C'.
   * Update g and parent for 'B' and 'C'.
   * list = ['B', 'C']
3. **Second Iteration:**
   * Current node: 'C' (since g['C'] + h['C'] = 4 + 2 = 6 is smaller than g['B'] + h['B'] = 2 + 6 = 8).
   * Neighbors of 'C': 'D'.
   * Update g and parent for 'D'.
   * list = ['B', 'D']
4. **Third Iteration:**
   * Current node: 'B' (since g['B'] + h['B'] = 2 + 6 = 8 is smaller than g['D'] + h['D'] = 7 + 0 = 7).
   * Neighbors of 'B': 'C' and 'D'.
   * Update g and parent for 'D' (but no improvement is found).
   * list = ['D']
5. **Fourth Iteration:**
   * Current node: 'D' (goal node).
   * Path is reconstructed as ['A', 'C', 'D'].

**6. Output**

The output of the program is:



This is the shortest path from 'A' to 'D' based on the given graph and heuristic values.