**AI Assignment 2A**

**Neeti Kurulkar**

**A\* Search Algorithm**

**Code:**

**class Graph:**

**def \_\_init\_\_(self, adj\_list, h\_dist):**

**self.*adj\_list* = adj\_list**

**self.*h\_dist* = h\_dist**

**def find\_neighbours(self, v):**

**"""Find the neighbours of a node"""**

**return self.*adj\_list*.*get*(v, [])**

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

**"""Return the heuristic distance from current node to goal node"""**

**return self.*h\_dist*[n]**

**def a\_star(self, start, goal):**

**"""Return the path from start node to goal node (if exists)"""**

**open\_set = set(start)**

**closed\_set = set()**

**g, parents = {}, {}**

**g[start] = 0**

**parents[start] = start**

**while open\_set:**

**print(f"\nOpen Set: {open\_set}")**

**print(f"Closed Set: {closed\_set}")**

**# Select node with the smallest f = g + h**

**current = min(open\_set, key=lambda v: g[v] + self.*h*(v))**

**print(f"Processing Node: {current}, f = g + h = {g[current]} + {self.*h*(current)} = {g[current] + self.*h*(current)}")**

**if current == goal:**

**path = []**

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

**path.*append*(current)**

**current = parents[current]**

**path.*append*(start)**

**path.*reverse*()**

**print(f"\nPath Found: {path}")**

**return path**

**open\_set.*remove*(current)**

**closed\_set.*add*(current)**

**for neighbour, weight in self.*find\_neighbours*(current):**

**if neighbour in closed\_set:**

**continue**

**tentative\_g = g[current] + weight**

**if neighbour not in open\_set or tentative\_g < g.*get*(neighbour, float('inf')):**

**parents[neighbour] = current**

**g[neighbour] = tentative\_g**

**open\_set.*add*(neighbour)**

**if open\_set:**

**next\_node = min(open\_set, key=lambda v: g[v] + self.*h*(v))**

**print(f"Next Selected Node: {next\_node}, f = g + h = {g[next\_node]} + {self.*h*(next\_node)} = {g[next\_node] + self.*h*(next\_node)}")**

**print("\nPath Does not exist!")**

**return None**

**# Adjacency list representation of the graph**

**adj\_list = {**

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

**'A': [('D', 6), ('C', 12)],**

**'B': [('C', 10), ('E', 7)],**

**'C': [('G', 5)],**

**'D': [('G', 16)],**

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

**'G': []**

**}**

**# Heuristic distances from each node to the goal**

**h\_dist = {**

**'S': 14,**

**'A': 12,**

**'B': 11,**

**'C': 4,**

**'D': 11,**

**'E': 6,**

**'G': 0**

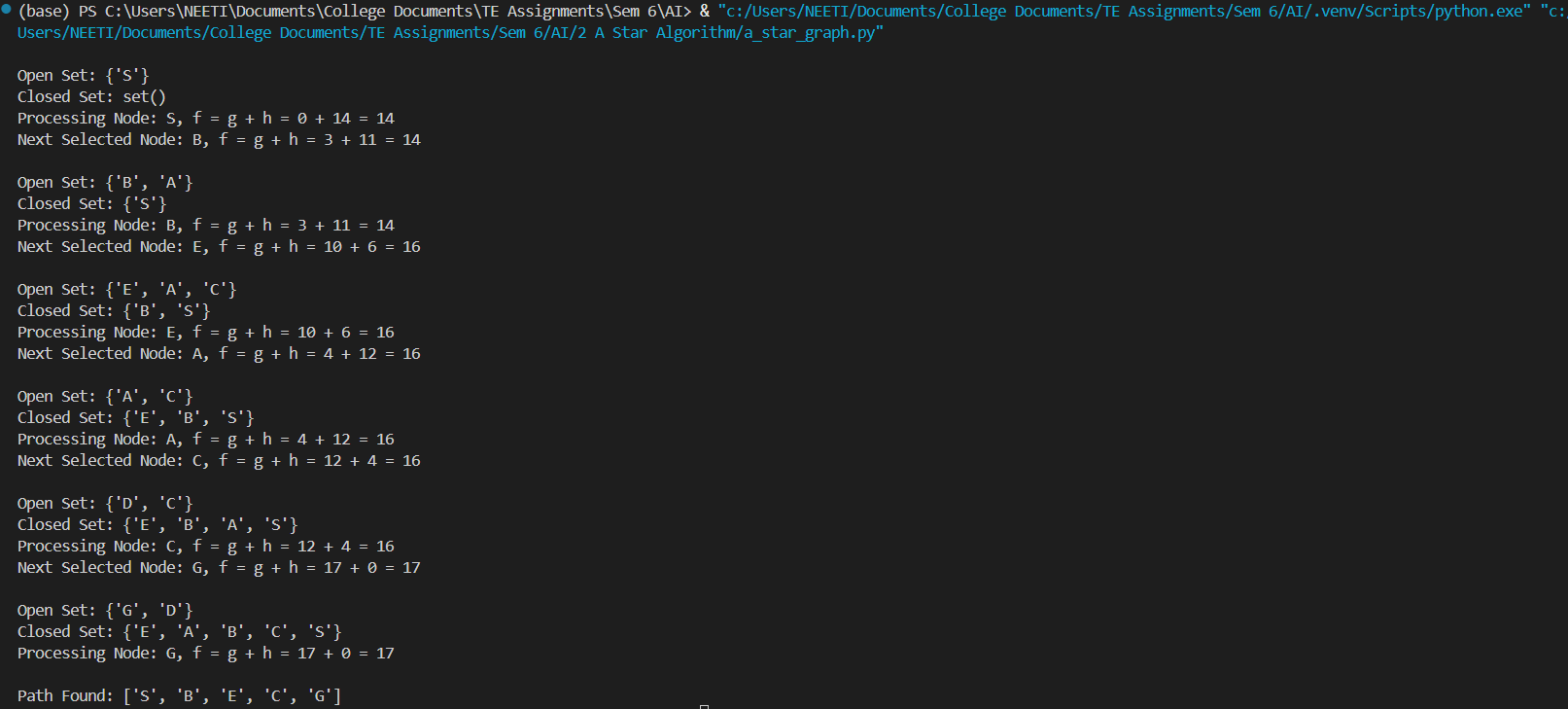
**}**

**# Create the graph and execute A\* search**

**g = Graph(adj\_list, h\_dist)**

**g.*a\_star*('S', 'G')**

**Output:**



**‘A’ ---> ‘E’**

A black screen with blue and green text

AI-generated content may be incorrect.