# A\* Pathfinding Algorithm – Code Explanation

This document explains the working of the provided A\* (A-star) Pathfinding Algorithm implemented in Python. The A\* algorithm is one of the most popular pathfinding algorithms used in Artificial Intelligence (AI) to find the shortest route between two points in a weighted graph.

## Python Code

from collections import deque  
  
class PathFinder:  
 def \_\_init\_\_(self, graph):  
 self.graph = graph  
  
 def heuristic(self, node):  
 heuristics = {  
 'A': 1,  
 'B': 1,  
 'C': 1,  
 'D': 1  
 }  
 return heuristics[node]  
  
 def a\_star(self, start, goal):  
 open\_nodes = set([start])  
 closed\_nodes = set([])  
  
 g\_cost = {start: 0}  
 parent = {start: start}  
  
 while open\_nodes:  
 current = None  
  
 for node in open\_nodes:  
 if current is None or g\_cost[node] + self.heuristic(node) < g\_cost[current] + self.heuristic(current):  
 current = node  
  
 if current is None:  
 print("No route could be found!")  
 return None  
  
 if current == goal:  
 route = []  
 while parent[current] != current:  
 route.append(current)  
 current = parent[current]  
 route.append(start)  
 route.reverse()  
 mapping = {'A': 'X', 'B': 'Y', 'C': 'Z', 'D': 'W'}  
 converted\_route = [mapping.get(node, node) for node in route]  
  
 print(f" Best route discovered: {converted\_route}")  
 return converted\_route  
  
 for (neighbor, cost) in self.graph[current]:  
 if neighbor not in open\_nodes and neighbor not in closed\_nodes:  
 open\_nodes.add(neighbor)  
 parent[neighbor] = current  
 g\_cost[neighbor] = g\_cost[current] + cost  
 else:  
 if g\_cost[neighbor] > g\_cost[current] + cost:  
 g\_cost[neighbor] = g\_cost[current] + cost  
 parent[neighbor] = current  
  
 if neighbor in closed\_nodes:  
 closed\_nodes.remove(neighbor)  
 open\_nodes.add(neighbor)  
  
 open\_nodes.remove(current)  
 closed\_nodes.add(current)  
  
 print("Route does not exist!")  
 return None  
  
graph\_data = {  
 'A': [('B', 1), ('C', 3), ('D', 7)],  
 'B': [('D', 5)],  
 'C': [('D', 12)]  
}  
  
finder = PathFinder(graph\_data)  
finder.a\_star('A', 'D')

## Explanation of the Code

1. \*\*Importing Libraries:\*\* The code imports `deque` from the `collections` module, though it is not directly used here.  
  
2. \*\*Class Definition (`PathFinder`):\*\* This class encapsulates the A\* search algorithm and takes a `graph` as input when initialized.  
  
3. \*\*Heuristic Function:\*\*  
 The `heuristic()` function provides an estimated cost from a given node to the goal. In this example, all nodes are assigned the same heuristic value (1). In a real-world scenario, this could be based on distance, coordinates, or other estimates.  
  
4. \*\*A\* Function (`a\_star`):\*\*  
 This is the main function that finds the best path from the starting node to the goal node using both the actual cost (`g\_cost`) and heuristic estimates.  
  
 - `open\_nodes`: Set of nodes that need to be evaluated.  
 - `closed\_nodes`: Set of nodes already evaluated.  
 - `g\_cost`: Dictionary that stores the cost to reach each node from the start.  
 - `parent`: Keeps track of each node’s predecessor for path reconstruction.  
  
5. \*\*Selecting the Best Node:\*\*  
 The algorithm selects the node with the lowest total cost (`g\_cost + heuristic`) from the open list.  
  
6. \*\*Goal Check:\*\*  
 If the goal node is reached, the algorithm reconstructs the path by tracing back through the `parent` dictionary.  
  
7. \*\*Mapping Conversion:\*\*  
 The nodes A, B, C, and D are mapped to X, Y, Z, and W for more readable output.  
  
8. \*\*Exploring Neighbors:\*\*  
 For each neighboring node, the algorithm updates its cost and parent if a better path is found. Nodes move from `open\_nodes` to `closed\_nodes` after being processed.  
  
9. \*\*Final Output:\*\*  
 Once the goal is reached, it prints the best route found, e.g., ['X', 'Y', 'W'].

## Output

When executed, the code prints the optimal path discovered by the A\* algorithm:  
  
Output:  
 Best route discovered: ['X', 'Y', 'W']  
  
This means that the shortest path from node 'A' to 'D' in the given weighted graph is through nodes 'B' (mapped as Y) and then directly to 'D' (mapped as W).