# How can educational institutions help tackle the issue of urbanisation. An interview with Mr Hammad Noor AHMED from Superior University in Lahore | by ASEFEdu (Editor) | ASEFEdu (Blog) | Medium

**Name: Tahir Fareed**

**Roll No: SU92-BSAIM-F24-036**

**Section: BSAI (3A)**

**Subject: AI Lab**

**Task : 07**

# A\* (A-Star) Algorithm - Simple Documentation

## What is A\* Algorithm?

The A\* (A-Star) algorithm is used to find the shortest path between two points. It is very popular in games, maps, and artificial intelligence. It helps a computer or robot decide the best and shortest route from start to goal.

## How A\* Works (In Simple Words)

1. Start from the starting point.  
2. Check all possible next steps (neighbors).  
3. For each step, calculate a score using this formula:  
 f(n) = g(n) + h(n)  
 - g(n): distance from the start point.  
 - h(n): estimated distance to the goal (heuristic).  
4. Choose the step with the smallest score (f).  
5. Repeat this process until you reach the goal.  
6. The path you followed is the shortest one.

## Python Code Example of A\* Algorithm

from queue import PriorityQueue  
  
def a\_star(graph, start, goal, h):  
 open\_list = PriorityQueue()  
 open\_list.put((0, start))  
 came\_from = {}  
 g\_score = {node: float('inf') for node in graph}  
 g\_score[start] = 0  
  
 while not open\_list.empty():  
 \_, current = open\_list.get()  
  
 if current == goal:  
 path = []  
 while current in came\_from:  
 path.append(current)  
 current = came\_from[current]  
 path.append(start)  
 return path[::-1]  
  
 for neighbor, cost in graph[current].items():  
 tentative\_g = g\_score[current] + cost  
 if tentative\_g < g\_score[neighbor]:  
 came\_from[neighbor] = current  
 g\_score[neighbor] = tentative\_g  
 f\_score = tentative\_g + h[neighbor]  
 open\_list.put((f\_score, neighbor))  
  
 return None

## Code Explanation (Easy Way)

1. We use a Priority Queue to always pick the node with the smallest f(n) value.

2. 'g\_score' keeps track of the cost from the start to each node.

3. 'h' is a dictionary with estimated distances (heuristic) from each node to the goal.

4. For every node, we calculate new possible paths and pick the one with the lowest cost.

5. Once we reach the goal, we go backwards to reconstruct the shortest path.

## Example

Suppose we have nodes: A, B, C, D.  
Distances between them:  
A → B = 1, A → C = 3, B → D = 1, C → D = 1  
Heuristic values (h): B = 2, C = 1, D = 0  
A\* will find that the best path is: A → B → D.

## Conclusion

A\* is a very smart algorithm because it finds the shortest path using both actual distance and an estimate. It is fast, efficient, and commonly used in navigation and games**.**

**Output Screenshot :**

