### Features of the Recursion Algorithm

1. **Base Case**: This is the condition under which the recursion ends. It prevents the function from calling itself indefinitely.
2. **Call Stack**: Recursive calls are stored in a call stack, where each call to the function creates a new entry on the stack. This can lead to stack overflow if the recursion is too deep.
3. **Divide and Conquer**: Many recursive algorithms, like merge sort or quick sort, follow the divide and conquer approach where the problem is divided into smaller, similar sub-problems.

• How to implement one recursive function to calculate the factorial of the integer number n?

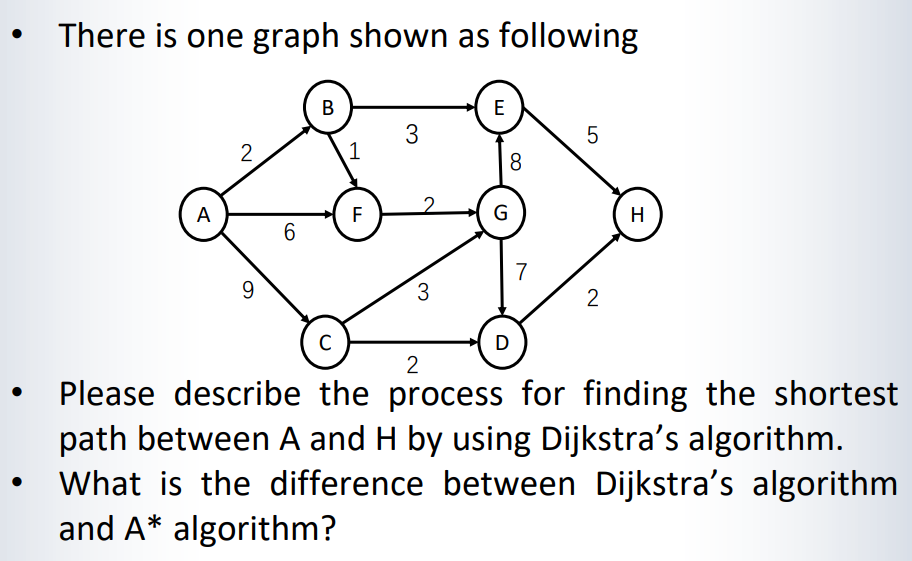
def factorial(n):

if n == 0:

return 1 # Base case: factorial of 0 is 1

else:

return n \* factorial(n - 1) # Recursive case



### Differences between Dijkstra's Algorithm and A\* Algorithm

1. **Heuristic Function**:
   * **Dijkstra’s Algorithm** does not use a heuristic function; it treats all edges equally, exploring them based on distance from the starting node.
   * A Algorithm\* uses a heuristic function to estimate the cost from the current node to the goal, prioritizing paths that are supposedly closer to the goal.
2. **Efficiency and Performance**:
   * **Dijkstra’s Algorithm** can be slower because it explores all possible paths to find the shortest path without considering the goal until it reaches it.
   * A Algorithm\* is generally faster because the heuristic function helps it to intelligently guess which paths are more likely to lead to the shortest path, reducing the number of paths it needs to explore.
3. **Use Cases**:
   * **Dijkstra’s Algorithm** is suitable when you need the shortest path to various nodes or when the destination is not known in advance.
   * A Algorithm\* is preferred in scenarios like pathfinding in games or GPS navigation where the goal is predefined, and you need the most efficient path to that specific target.

删除后pos=unused；