**Lab Report: Help the Rat Escape the Underground Pipes**

**Problem Statement:**

Terry the rat is stuck inside an underground pipe system. Each junction in the network is connected by pipes, and each pipe has a certain travel cost (difficulty to crawl). Terry needs to reach the cheese (target junction) from his starting junction. We have to try out different strategies (DFS, BFS, Dijkstra, and A\*) and see how the rat would escape.

**How I Understood:**

Junctions = nodes

Pipes = edges with weights (costs)

Task = find a path from start to target using different algorithms.

**Plan / Approach / Solution:**

1. Represent the pipe network as a graph.

2. Apply strategies: - DFS: Go deep first, backtrack if stuck. - BFS: Explore nearby nodes first. - Dijkstra: Choose lowest cost paths. - A\*: Dijkstra + heuristic (distance to target).

3. Compare paths, costs, and junctions visited.

**Input:**

Number of junctions and pipes

Pipe connections + travel cost

Coordinates of junctions (for A\*)

Start and target junction

**Output:**

Path Terry takes

Total travel cost

Number of junctions visited

**Algorithms Used:**

DFS: stack/recursion, deep exploration

BFS: queue, level-wise exploration

Dijkstra: priority queue, minimum cost

A\*: priority queue + heuristic, faster optimal path

**Reason for Approach:**

DFS/BFS: simple exploration, may not be cost-efficient

Dijkstra: guaranteed minimum cost path

A\*: faster than Dijkstra using heuristic guidance

**Conclusion:**

DFS: may waste time, not cost-optimal

BFS: fewest hops, not minimum cost

Dijkstra: finds lowest-cost path

A\*: same lowest-cost path as Dijkstra but faster

**Summary:**

The experiment showed how the same problem can be solved differently. DFS/BFS are simple but less efficient, Dijkstra is reliable, and A\* is most efficient.

**Python Code:**

import heapq  
from collections import deque, defaultdict  
import math  
  
class PipeNetwork:  
 def \_\_init\_\_(self, n):  
 self.n = n  
 self.graph = defaultdict(list)  
 self.coords = {}  
  
 def add\_pipe(self, u, v, cost):  
 self.graph[u].append((v, cost))  
 self.graph[v].append((u, cost))  
  
 def set\_coord(self, node, x, y):  
 self.coords[node] = (x, y)  
  
 def dfs(self, start, target):  
 stack = [(start, [start], 0)]  
 visited = set()  
 while stack:  
 node, path, cost = stack.pop()  
 if node == target:  
 return path, cost, len(visited)  
 if node not in visited:  
 visited.add(node)  
 for neighbor, c in self.graph[node]:  
 stack.append((neighbor, path + [neighbor], cost + c))  
 return None, float("inf"), len(visited)  
  
 def bfs(self, start, target):  
 queue = deque([(start, [start], 0)])  
 visited = set()  
 while queue:  
 node, path, cost = queue.popleft()  
 if node == target:  
 return path, cost, len(visited)  
 if node not in visited:  
 visited.add(node)  
 for neighbor, c in self.graph[node]:  
 queue.append((neighbor, path + [neighbor], cost + c))  
 return None, float("inf"), len(visited)  
  
 def dijkstra(self, start, target):  
 pq = [(0, start, [start])]  
 visited = set()  
 while pq:  
 cost, node, path = heapq.heappop(pq)  
 if node == target:  
 return path, cost, len(visited)  
 if node not in visited:  
 visited.add(node)  
 for neighbor, c in self.graph[node]:  
 heapq.heappush(pq, (cost + c, neighbor, path + [neighbor]))  
 return None, float("inf"), len(visited)  
  
 def astar(self, start, target):  
 def heuristic(a, b):  
 (x1, y1), (x2, y2) = self.coords[a], self.coords[b]  
 return math.sqrt((x1 - x2)\*\*2 + (y1 - y2)\*\*2)  
  
 pq = [(heuristic(start, target), 0, start, [start])]  
 visited = set()  
 while pq:  
 est, cost, node, path = heapq.heappop(pq)  
 if node == target:  
 return path, cost, len(visited)  
 if node not in visited:  
 visited.add(node)  
 for neighbor, c in self.graph[node]:  
 new\_cost = cost + c  
 heapq.heappush(pq, (new\_cost + heuristic(neighbor, target), new\_cost, neighbor, path + [neighbor]))  
 return None, float("inf"), len(visited)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 n, m = 5, 6  
 network = PipeNetwork(n)  
  
 coords = {1: (0,0), 2: (1,2), 3: (2,1), 4: (3,3), 5: (4,0)}  
 for node, (x,y) in coords.items():  
 network.set\_coord(node, x, y)  
  
 edges = [(1,2,2), (1,3,4), (2,3,1), (2,4,7), (3,5,3), (4,5,1)]  
 for u,v,c in edges:  
 network.add\_pipe(u, v, c)  
  
 start, target = 1, 5  
 print("DFS:", network.dfs(start, target))  
 print("BFS:", network.bfs(start, target))  
 print("Dijkstra:", network.dijkstra(start, target))  
 print("A\*:", network.astar(start, target))

**Sample Output:**

DFS: ([1, 3, 5], 7, 2)

BFS: ([1, 3, 5], 7, 4)

Dijkstra: ([1, 2, 3, 5], 6, 3)

A\*: ([1, 2, 3, 5], 6, 3)

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of code

AI-generated content may be incorrect.A computer screen shot of a computer code

AI-generated content may be incorrect.

**OUTPUT**:

A black background with white text and red numbers

AI-generated content may be incorrect.