Thuật toán Deep Limited Search

Giải thích:

Depth-Limited Search (DLS) là một biến thể của **Depth-First Search (DFS)** với giới hạn độ sâu (depth limit). Thay vì tiếp tục đệ quy vô hạn trong DFS, DLS dừng khi đạt đến một giới hạn độ sâu được định trước. Điều này giúp ngăn chặn việc duyệt mãi mãi trong đồ thị có chu trình hoặc không thể tìm thấy mục tiêu trong không gian tìm kiếm vô hạn.

- **Cơ chế hoạt động:**
    1. Bắt đầu từ nút gốc.
    2. Thực hiện tìm kiếm theo chiều sâu đến các nút con.
    3. Dừng lại khi đạt đến độ sâu giới hạn hoặc tìm thấy nút mục tiêu.
    4. Nếu không tìm thấy mục tiêu và tất cả các nút đã được duyệt ở mức giới hạn, báo rằng tìm kiếm thất bại.
- **Ưu điểm:**
    - Hiệu quả về mặt bộ nhớ (giống DFS).
    - Giới hạn độ sâu giúp tránh vấn đề duyệt vô tận.
- **Nhược điểm:**
    - Có thể không tìm thấy giải pháp nếu giải pháp nằm ngoài độ sâu giới hạn.
    - Không tối ưu và không đảm bảo tìm được đường đi ngắn nhất.

**Mã giả**:

```
DLS(node, goal, depth_limit):
  if node == goal:
    return Solution
  elif depth_limit == 0:
    return Cutoff
  else:
```

```
    cutoff_occurred = False
    for each child in successors(node):
      result = DLS(child, goal, depth_limit - 1)
      if result == Cutoff:
        cutoff_occurred = True
      elif result != Failure:
        return result
    if cutoff_occurred:
      return Cutoff
    else:
      return Failure
```

Triển khai Python:

```
import json

graph = {
  'Arad': [('Zerind', 75), ('Timisoara', 118), ('Sibiu', 140)],
  'Zerind': [('Arad', 75), ('Oradea', 71)],
  'Oradea': [('Zerind', 71), ('Sibiu', 151)],
  'Timisoara': [('Arad', 118), ('Lugoj', 111)],
  'Lugoj': [('Timisoara', 111), ('Mehadia', 70)],
  'Mehadia': [('Lugoj', 70), ('Dobreta', 75)],
  'Dobreta': [('Mehadia', 75), ('Craiova', 120)],
  'Craiova': [('Dobreta', 120), ('Rimnicu Vilcea', 146), ('Pitesti', 138)],
  'Sibiu': [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu Vilcea', 80)],
  'Fagaras': [('Sibiu', 99), ('Bucharest', 211)],
```

```
    'Rimnicu Vilcea': [('Sibiu', 80), ('Craiova', 146), ('Pitesti', 97)],
    'Pitesti': [('Rimnicu Vilcea', 97), ('Craiova', 138), ('Bucharest', 101)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101), ('Giurgiu', 90), ('Urziceni', 85)],
    'Giurgiu': [('Bucharest', 90)],
    'Urziceni': [('Bucharest', 85), ('Vaslui', 142), ('Hirsova', 98)],
    'Hirsova': [('Urziceni', 98), ('Eforie', 86)],
    'Eforie': [('Hirsova', 86)],
    'Vaslui': [('Urziceni', 142), ('Iasi', 92)],
    'Iasi': [('Vaslui', 92), ('Neamt', 87)],
    'Neamt': [('Iasi', 87)]
}

def depth_limited_search(node, goal, limit, path, current_cost):

    path.append(node)
    if (node == goal):
        return path, current_cost
    if (limit == 0):
        return False, 0

    shortest_path = []
    min_cost = 10000
    for successor, cost in graph.get(node):
        if (successor not in path):

            result, child_cost = depth_limited_search(successor, goal, limit-1, path.copy(), cost)
```

```
        if result == False:
            shortest_path = False
            continue
        else:
            if (current_cost + child_cost < min_cost):
                min_cost = current_cost + child_cost
                shortest_path = result
    return shortest_path, min_cost

# Searching with deep limited search
root = "Arad"
goal = "Bucharest"
depth_limit = 6
path = []
result, cost = depth_limited_search(root, goal, depth_limit, path, 0)
print("Path shorted is: ", result)
print("Min cost is: ", cost)
```

**Ví dụ đầu ra:**

```
Path shorted is:  ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Min cost is:  418
```