HƯỚNG DẪN THỰC HÀNH 1

Môn: Trí Tuệ Nhân Tạo (503030)

I. Mục Tiêu

- Nhắc lại lý thuyết về không gian trạng thái (states space), thành phần của một bài toán AI và các chiến lược tìm kiếm.
- Trình bày các giải thuật tìm kiếm: Depth-First Search (DFS), Breadth-First Search (BFS), Uniform Cost Search, Depth-Limited Search, Iterative Deepening Search.
- Hiện thực các giải thuật tìm kiếm bằng ngôn ngữ lập trình Java.
- Giới thiệu công cụ lập trình Prolog.

II. Giải Thuật Depth-First Search (DFS)

1. Ý tưởng giải thuật

Xuất phát từ trạng thái hiện hành, ta chọn một trạng thái kế tiếp làm trạng thái hiện hành cho đến lúc trạng thái hiện hành là trạng thái đích.

Trong trạng thái hiện hành đang xét, nếu ta khổng thể biến đổi thành trạng thái kế tiếp thì ta sẽ quay lui (back-tracking) lại trạng thái trước đó để chọn đường khác. Nếu trạng thái trước này mà cũng không thể biến đổi được nữa thì ta quay lui lại trạng thái trước nữa và cứ thế. Khi ta trở về trạng thái ban đầu và vẫn không tìm được đường đi thì giải thuật DFS dừng.

2. Mã giải (Pseudocode)

Trong mục này, ta xét mã giả của giải thuật DFS theo hai cách hiện thực:

- Dùng ngăn xếp (LIFO queue) để lưu các trạng thái trước đó.
- Dùng đệ quy để hiện kỹ thuật quay lui (back-tracking).

```
function DEPTH-FIRST-SEARCH (problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a stack with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP( frontier)    /* chooses the shallowest node in frontier */
     add node.STATE to explored
```

```
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
            frontier ← INSERT(child, frontier)
```

Figure 1. Dùng stack

```
function DEPTH-FIRST-SEARCH (graph G, Vertex v) returns a solution, or failure
  visited[v] ← TRUE
  for all w adjaccent to v do
    if not visited[w] then
      DEPTH-FIRST-SEARCH (G, w)
```

Figure 2. Dùng đệ quy

III. Giải thuật Breadth-First Search (BFS)

1. Ý tưởng giải thuật

Tìm kiếm theo chiều rộng mang hình ảnh của vết dầu loang. Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, ứng với mỗi trạng thái T_k trong tập S, ta xây dựng một tập S_k bao gồm các trạng thái kế tiếp của T_k rồi lần lượt bổ sung các S_k vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả S_k.

2. Mã giả

```
function BREATH-FIRST-SEARCH (graph G, Vertex v) returns a solution, or failure
  visited[v] ← TRUE
  Q.enQueue(v)
  while not Q.IsEmpty() do
    v ← Q.deQueue()
    for all w adjaccent to v do
      if visited[w] = FALSE then
        visited[w] ← TRUE
        Q.enQueue(w)
```

Figure 3. Mã giả BFS dùng Queue (Ver. 1)

```
function BREATH-FIRST-SEARCH (problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP( frontier)    /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Figure 4. Mã giả BFS dùng Queue (Ver. 2)

3. So sánh DFS và BFS

	DFS	BFS
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ
Trường hợp tốt nhất	Phương án chọn hướng đi *tuyệt đối* chính xác. Lời giải được xác định một cách trực tiếp.	Vét cạn toàn bộ

IV. Uniform-Cost Search

1. Ý tưởng giải thuật

Xét trường hợp chi phí giữa hai trạng thái là bằng nhau thì giải thuật BFS là tối ưu *(optimal)* vì nó luôn mở rộng đến những trạng thái nông nhất *(shallowest)*. Tuy nhiên, trong trường hợp chi phí giữa hai trạng thái là khác nhau thì BFS không đảm bảo tìm ra lời giải có chi phí nhỏ nhất và do vậy không tối ưu. Để giải quyết được vấn đề này, UCS chọn trạng thái có chi phí nhỏ nhất thay vì trạng thái nông nhất *(shallowest)*.

Thuật toán điển hình của UCS là *Dijsktra*.

2. Mã giả

```
function UNIFORM-COST-SEARCH (problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP( frontier)    /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is not in explored or frontier with higher PATH-COST then
           replace that frontier node with child
```

Figure 5. Mã giả UCS

V. Depth-Limited Search

1. Ý tưởng giải thuật

Giải thuật tìm kiếm theo chiều sâu (DFS) có ưu điểm là nó có thể sinh ra lời giải nhanh chóng mà không tốn kém bộ nhớ của máy tính. Tuy nhiên nếu không gian trạng thái của bài toán là vô hạn thì rất có thể nó không tìm được lời giải của bài toán khi hướng tìm kiếm không chứa trạng thái đích (goal state). Để khắc phục nhược điểm này, chúng ta có thể đặt giới hạn độ sâu trong giải thuật: nếu độ sâu của trạng thái đang xét vượt quá ngưỡng nào đó thì chúng ta không bổ sung các nút kề với trạng thái này nữa mà chuyển sang hướng tìm kiếm khác.

2. Mã giả

```
function DEPTH-LIMITED-SEARCH (problem, maxDepth) returns a solution, or failure
  Stack ← make-queue(make-node(initial-state[problem]));
  father(initial-state[problem]) ← empty;
  depth(initial-state[problem]) ← 0;
  loop do
     if Stack is empty then return failure;
     node ← pop(Stack) ;
     if test(node, Goal[problem]) then return path(node, father);
     if (depth(node) < maxDepth)
        expand-nodes ← adjacent-nodes(node, Operators[problem]);
        push(Stack, expand-nodes);
        for each expand-node in expand-nodes
           father(ex-node) ← node;
  end
```

Figure 6. Mã giả DLS

VI. Iterative Deepening Search

1. Ý tưởng giải thuật

Giải thuật tìm kiếm với chiều sâu có giới hạn (Depth-Limited Search) phụ thuộc vào giới hạn độ sâu lựa chọn ban đầu. Nếu biết trước trạng thái đích sẽ xuất hiện trong phạm vi độ sâu nào đó của cây tìm kiếm thì chúng ta đặt giới hạn độ sâu đó cho giải thuật. Tuy nhiên nếu chọn độ sâu tối đa không phù hợp, giải thuật DLS sẽ không tìm được lời giải của bài toán. Chúng ta có thể gọi thực hiện giải thuật tìm kiếm lời giải ở độ sâu khác nhau, từ bé đến lớn.

2. Mã giả

```
function ITERATIVE-DEEPENING-SEARCH (problem) returns a solution, or failure
  for depth = 0 to ∞ do
     result ← DEPTH-LIMITED-SEARCH (problem, depth)
     if result succeeds then return result
  end
  return failure
```

Figure 7. Mã giả IDS

VII. Bài Tập

1. Dựa vào ý tưởng giải thuật hoặc mã giả của các chiến lược tìm kiếm Depth-First Search (DFS), Breadth-First Search (BFS). Anh (chị) hãy cài đặt hai giải thuật này bằng ngôn ngữ lập trình Java theo yêu cầu sau:

a. Chương trình cho phép đọc vào một file TXT (`input.txt`) có nội dung là một đồ thị trạng thái với cấu trúc như sau.

```
8
0   1   0   1   0   0   1   0
1   0   0   0   1   1   0   0
0   0   0   0   0   1   0   1
1   0   0   0   0   1   0   0
0   1   0   0   0   0   1   0
0   1   1   1   0   0   0   0
1   0   0   0   1   0   0   0
0   0   1   0   0   0   0   0
```

Nội dung file `input.txt`, trong đó:

- Hàng đầu tiên: số đỉnh của đồ thị.
- Các hàng còn lại lập thành ma trận kề của đồ thị.

b. Chương trình xuất ra file TXT (`output.txt`) chứa nội dung là kết quả của hai chiến lược tìm kiếm DFS, BFS.

2. Anh (chị) hãy hiện thực giải thuật thuộc chiến lược *Uniform-Cost Search*. Sau đó, thực hiện đọc vào một đồ thị trạng thái và trả về kết quả là đường đi ngắn nhất từ trạng thái khởi đầu đến trạng thái đích.

--- HẾT ---