Project Report
Search

Members:
1. Nguyễn Ngọc Minh – 17021300 – 10
2. Nguyễn Nam – 17021304 – 10
3. Phạm Công Nam – 17021306 – 10
(Grade là điểm các thành viên trong nhóm tự đánh giá lẫn nhau dựa trên mức độ đóng góp, theo thang điểm 0-10)

Question 1: Depth First Search

- Idea: Duyệt từng state được thăm, duyệt đi xa nhất theo từng nhanh, khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo.
  Pseudo code:

```
DFS(Problem, Stack S = {start state with action_list }, visited_list V)
          while S is not empty
             current_state, action_list := pop S
             Push V, current_state
             if current_state is goal
                return action_list
             for each successor s of current_state
                if s is not in V and s is not in S
                   new_action_list = action_list + direction of s
                   Push S, s
```

- Result: 3/3 tests

Question 2: Breadth First Search

- Idea: Duyệt tất các state kề với state hiện tại rồi sau đó duyệt từng đỉnh.
  - B1: Cứ mỗi state được thăm, ta sẽ duyệt tất cả các successor kề với state đó (đẩy tất cả các successor của state vào hàng đợi, state đã duyệt thì đẩy vào danh sách state đã thăm).
  - B2: Thăm đến từng state có trong hàng đợi (mỗi lần thăm phải đẩy state đã thăm ra ngoài) và lặp lại bước B1 và dừng lại khi đạt tới đích (goal, trả về danh sách action list).
- Pseudo code:

```
BFS(Problem, Queue Q = {start state with action_list}, visited_list V)
          while Q is not empty
             current_state, action_list := pop Q
             if current_state is goal
                return action_list
             for each successor s of current_state
                if s is not in V and s is not in Q
                   new_action_list = action_list + direction of s
                   Push Q, s
                   Push V, current_state
```

- Result: 3/3 tests

Question 3: Uniform-cost Search

- Idea:
  - Duyệt state có độ ưu tiên cao nhất để đi, độ ưu tiên ở đây tương ứng với chi phí đường đi, càng nhỏ thì càng tốt (Sử dụng hàm đợi ưu tiên để kiểm tra)
  - Nếu state đã thăm mà có độ ưu tiên mới cao hơn độ ưu tiên cũ thì cập nhật lại giá trị ưu tiên của state đó với độ ưu tiên mới.
- Pseudo code:

```
UCS(Problem, Frontier F = {start state with path, 0}, visited_list V)
      // Frontier F is a priority queue(state, priority point)
         while F is not empty
            current_state, action_list := pop F
            Push V, current_state
            if current_state is goal
               return action_list
             for each successor and path cost of s of current_state
               if s is not in V and s is not in heap tree of F
                  new_action_list = action_list + direction of s
                  Add S with new_action_list
               else if s is not in V and s is not in heap tree of F
                  new_action_list = action_list + direction of s
                  if newCost of s < currentPriority of s
                     Update Q, s with newCost
```

- Result: 3/3 tests

Question 4: A* Search

- Idea:
  - Tạo 2 list (stack) open list lưu trữ các node chưa được chọn làm node trên đường ngắn nhất đến goal và close list chứa các node đã được chọn để tạo đường ngắn nhất đến goal
  - Xét các con (successor) của node có giá trị f nhỏ nhất chưa được chọn và cập nhật giá trị f,g,h của các con.
  - Khi tìm được goal, dựa vào cấu trúc của node, ta có thể lần ngược lại từ goal đến start để tìm ra đường đi đến goal.
- Pseudo code:
- Data structure of a node includes (parent node, action, position, cost, f=0, g=0, h=0)

```
A*(Problem, heuristic)
      //Initiate 2 empty lists(stacks) open_list and close_list
      //Add start node of pacman to open_list
      while open_list not empty
            //Find node with minimum f value in open_list
            //then remove from open_list and add to close_list
            //call above node as current_node
            if current_node is goal
                  //Traverse back from current_node back to start node
                  return list of action
            for each successor of current_node
                  if successor not in close_list
                      g = current_node.g + path cost
                      h = heuristic(successor, goal)
                      f = g+h
                      if successor not in open_list
                           //Add successor to open_list
```

- Result: 3/3 tests

Question 5: Finding All the Corners

- Idea:

- Ở câu hỏi này không cần dùng các vị trí của ghosts, food,… mà chỉ quan tâm đến vị trí của Pacman và tọa độ của 4 góc. Do đó, ta định nghĩa state riêng cho bài toán này là một tuple chứa tọa độ của pacman và tọa độ của 4 góc.
- State = ((current_x, current_y), (corner_1, corner_2, corner_3, corner_4)))
- Khi pacman đến được 1 góc, ta loại bỏ góc đó ra khỏi state, do đó, goal state là khi không còn tọa độ 4 góc trong state nữa.
- Goal state ⇔ length(state[1]) = 0
- Tại mỗi bước, ta xét bước tiếp theo theo các hướng có thể đi được (không có wall), nếu bước tiếp theo là 1 trong 4 góc, thì ta sẽ loại góc đó ra khỏi state, trả lại vị trí của tọa độ của bước đó và tọa độ các góc còn lại.

- Pseudo code:

```
State = (current_position, (corner_1, corner_2, corner_3, corner4))
Goal state ⇔ length(state[1]) = 0
Successors = []
For every action
      // Calculate the next position
      next_x = current_x +dx
      next_y = current_y + dy
      
      // Check to see if the next position is wall
      If walls[next_x][next_y] == false:
            // If the next position is 1 of the corners, then remove it
            If next_position in remaining_corners:
                  remaning_corners.remove(next_position)
            // Return new state
            new_state = (next_position, remaining_corners)
            Successors.append((new_state, action, action_cost))
Return successors
```

- Result: 3/3 tests

Question 6: Corners Problem: Heuristic

- Idea:
  - Từ vị trí ban đầu của pacman, tìm ra góc nào là gần nhất, rồi tiếp tục tìm góc gần nhất tính từ góc đó (coi góc vừa tìm được là vị trí của pacman), cứ vậy đến khi ta tìm hết 4 góc. Tổng khoảng cách từ vị trí ban đầu của pacman đến lần lượt 4 góc là heuristic
  - Ta không xét đến tường bởi nó không ảnh hưởng đến giá trị heuristic, khoảng cách từ pacman đến góc được tính bằng khoảng cách manhanttan
  - Goal state do đó là khi pacman đến được góc xa nhất (góc cuối cùng được tìm đến)
- Pseudo code:

```
//heuristic
Total_cost = 0
While len(remaining_corners) != 0:
      Distance_to_corners = []
      For corner in remaining_corners:
            distance = manhanttanDistance(current_position, corner)
            Distance_to_corners.add(distance)
            Min_distance  = min(distance_to_corners)
            Total_cost += min_distance
            Current_position = closest_corner
            Remaining_corners.remove(closest_corner)
Return total_cost
```

- Result: 3/3 tests

Question 7: Eating All The Dots

- Idea:
  - Sử dụng trực tiếp function mazeDistance (hàm này sử dụng BFS để tìm khoảng cách giữa 2 điểm)

- Tuy niên, ta lựa chọn mazeDistance có giá trị lớn nhất giữa pacman và food (viên thức ăn của pacman) làm heuristic.

- Pseudo code:

```
heuristic = 0
for each food
    distance = mazeDistance(pacman position, food position)
    if distance > heuristic
        heuristic = distance
```

- Result: 5/4 tests

Question 8: Suboptimal Search

- Idea: Sử dụng BFS, tuy nhiên do ta kiểm tra node đang xét chỉ cần thuộc danh sách các food là goal nên không thể chắc chắn về việc food đó là food gần nhất.
- Pseudo code:

```
Suboptimal Search(Problem, Queue Q = {start state with action_list},
visited_list V)
        while Q is not empty
           current_state, action_list := pop Q
           if current_state is in food list
              return action_list
           for each successor s of current_state
              if s is not in V and s is not in Q
                 new_action_list = action_list + direction of s
                 Push Q, s
                 Push V, current_state
```

- Result: 3/3 tests