CS4750/7750 HW #2 (20 points)

Consider a 2-D 20-room vacuum-cleaner world as follows:

- The world is a 2-D grid with 4x5 = 20 rooms, as shown below. The agent knows the environment and dirt distribution. This is a fully observable problem.
- The agent can choose to move left (Left), right (Right), up (Up), down (Down), or suck up the dirt (Suck). Clean rooms stay clean. The agent cannot go outside the environment, i.e., the actions to bring the agent outside the environment are not allowed.
- Goal: clean up all dirt in the environment, i.e., make all rooms clean.
- Action cost:
  a) 1.0 for Left
  b) 0.9 for Right
  c) 0.8 for Up
  d) 0.7 for Down
  e) 0.6 for Suck

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	…		
(3,1)	…			
(4,1)				(4,5)

Instance #1: Initial agent location: (2,2). Dirty squares: (1,2), (2,4), (3,5).
Instance #2: Initial agent location: (3,2). Dirty squares: (1,2), (2,1), (2,4), (3,3).

In this programming assignment, you are asked to implement the following 3 algorithms to solve the two problem instances:

a) uniform cost **tree** search,
b) uniform cost **graph** search,
c) iterative deepening **tree** search.

Follow the Tree-Search and Graph-Search pseudocode shown below (the same as in the lecture slides). Breaking ties of search nodes based on the coordinate of the agent location as follows: first consider the row numbers, where smaller numbers have higher priority; then consider the column numbers, where smaller numbers have higher priority.

You may use any programming language for this assignment.

```
function Tree-Search (problem, fringe) returns a solution, or failure
    fringe = Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node = Remove-Front(fringe)
        if Goal-Test(problem, State(node)) then return node
        fringe = InsertAll (Expand(node, problem), fringe)
    end

function Graph-Search (problem, fringe) returns a solution, or failure
    closed = an empty set
    fringe = Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node = Remove-Front(fringe)
        if Goal-Test(problem, State[node]) then return node
        if State[node] is not in closed then
            add State[node] to closed
            fringe = InsertAll (Expand(node, problem), fringe)
    end

    function Expand (node, problem) returns a set of nodes
        successors ← the empty set
        for each action, result in Successor-Fn(problem, State[node]) do
            s ← a new Node
            Parent-Node[s] ← node; Action[s] ← action; State[s] ← result
            Path-Cost[s] ← Path-Cost[node] + Step-Cost(node, action, s)
            Depth[s] ← Depth[node] + 1
            add s to successors
        return successors
```

Each group's submission should consist of two files:

1) A pdf file for the report, including names of team members, a description of your implementation, the programming language and hardware used, and the running result of each algorithm on the two problem instances containing the following:
    a. The states of the first 5 search nodes in the order they would be expanded.
    b. The total number of nodes expanded, the total number of nodes generated, and the CPU execution time in seconds. If your program cannot finish in one hour, you may stop your program after running for one hour and report these three numbers.
    c. The solution found (i.e., the sequence of moves), the number of moves, and

the cost of the solution.

2) A zip file containing your code with appropriate comments. You may use code found on the Internet but need to give credits to the sources.

==

Some Q&A.

Q: I know that Uniform cost chooses the lowest cost option at each step. So, if I'm correct, the graph will travel down until it no longer can, then up, right, and left ignoring already visited nodes until it no longer can. How is the graph search different from the tree search?

A: The nodes in the fringe are sorted based on their path costs g(n), which is the sum of step costs of all steps reaching a node from the root node.

In uniform cost tree search, the suck-suck node has a cost of 0.6+0.6 = 1.2, higher than the other nodes on the first level of the search tree. So, the other nodes will be expanded first.

In uniform cost graph search, the suck node contains the same state as the root node does, so the suck node won't be expanded.

The state is not just the location of the agent, but also the dirt distribution of the rooms. For example, the initial state is [V (2,2), d (1,2), d (2,4), d (3,5)]. The states of the 5 successor nodes are:

1. [V (2,1), d (1,2), d (2,4), d (3,5)] with g = 1.0
2. [V (2,3), d (1,2), d (2,4), d (3,5)] with g = 0.9
3. [V (1,2), d (1,2), d (2,4), d (3,5)] with g = 0.8
4. [V (3,2), d (1,2), d (2,4), d (3,5)] with g = 0.7
5. [V (2,2), d (1,2), d (2,4), d (3,5)] with g = 0.6

The (5) node will be expanded next to generate all its successor nodes due to lowest g in the uniform cost tree search. However, in the uniform cost graph search, because the (5) node has the same state as the root node, the (5) node will be dropped, and the (4) node will be expanded next.

Q: Since Uniform Cost Tree search can repeat states, will it not continuously repeat the suck action, since it is lowest cost?

A: In this assignment, because the step costs of all actions are positive, the repeat states will have larger path cost g(n), which will rank them lower in the fringe. For example, the first-level node of a single "suck" action has g=0.6; the second-level node of two "suck" actions has g=1.2; the third-level node of 3 “suck” actions has g=1.8; and so on. The search will not continuously repeat the suck action.

However, if the step cost of an action is negative, for example, if we give -0.1 to suck action, then, when the suck action is repeated, the repeated node will have lower g cost and rank high in the fringe. In this case, the search will keep repeating this action.

Q: What is the difference between expanded and generated nodes?

A: The Expand function expends one node and generates its child nodes. In this assignment, when the root node is expanded, 5 child nodes are generated.