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
```

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.