Seth Horowitz
Collaborators: Pauline Nguyen and Pablo Volpe

CSE 101 Homework 1

Question 1.

This algorithm works is essentially a DFS where each hexagonal space with no obstacle is a node, and each connection between two hexagonal spaces is an edge. In the DFS each new node explored must be adjacent to both the current and the previous nodes. If a path so that the two hexagons for the end position can be reached consecutively then the movement is possible.

```
findpath(startA, startB, endA, endB):
  current = startA
  prev = startB
  if (current, prev) == (endA, endB) or
      (prev, current) == (endA, endB):
     return true

  for nodes adjacent to current:
    if node is also adjacent to prev:
      if findpath(node, current, endA, endB) returns true:
        return true

  return false
```

Set start1 and start2 to initial positions of table and end1 and end2 to final positions of table if either $findpath(start1, start2, end1, end2)$ or $findpath(start2, start1, end1, end2)$ return true, there exists a path to move the table.

Question 2.

```
testDag(node):
  mark node as visited
  for all connected nodes:
    if node is visited:
      return false
    else:
      if testDag(connectedNode) == false:
        return false
  mark node as unvisited
  return true

run testDag on all source nodes
nodes are unvisited by default
```

Question 3.

Start by setting the value ”Largest” to zero. For each source node, set the source node to current and recursively explore each node connected to the current node. Each time node is explored, if the distance from the current source is greater than Largest, update Largest. Once all paths have been explored, return Largest.

Question 4.

```
cheapestDest(startcity): #finds cheapest destination reachable from startcity
  mark startcity as visited
  lowest = startcity.outcost
  for each city connected to current:
    if city marked unvisited:
      temp = cheapestDest(city)
      if temp < lowest:
        lowest = temp
  return lowest

define priority queue "starts" sorted by incost values
for each n in cities:
  priorityqueue.add(n)
lowest = infinity
for each city in starts:
  if city is unvisited:
    temp = city.incost + cheapestDest(city)
    if temp < lowest:
      lowest = temp
```

After running the for loop, the value of lowest will be the value of the cheapest possible trip.

Question 5.

(a) This graph is impossible because it contains the interweaving $B_{pre} < C_{pre} < B_{post} < C_{post}$.

(b)

$$\begin{tikzcd}
E & A \arrow[r] \arrow[dr] & B \arrow[r] & C \\
 & & D &
\end{tikzcd}$$

(c) This graph is impossible. There are non existent pre and post numbers, and even if there was a vertex that filled the missing pre and post numbers it would contain interweaving, $C_{pre} < 6 < C_{post} < 9$.

Question 6 (Extra credit, 1 point)**.**

If n is the number of questions on the homework, time spent is bound by $Homework(n) = O(nlogn)$.