**Graphs 11**

**HTH (State Graphs)**

Consider this game: start with three coins arranged in a row, Heads-Tails-Heads (HTH). The goal is to change the arrangement into THT within the rules, namely, 1) you may flip the middle coin (from H to T or vice versa) whenever you want to. 2) You may flip the end coins only if the other two coins are the same as each other (both H or both T). 3) You may not change the coins in any other way, for example by shuffling them around.

Play the game here: starting with HTH, flip coins according to the rules to get THT.

H T H

H H H

H H T

H T T

T T T

T H T

T T H

T H H

What does this problem have to do with graphs? Many problems, like this one, can be rephrased a search problem on a graph. If you can build the appropriate graph, then searching graph will find the solution. The general technique is to generate the different possible *states*, or configurations, of the world that are relevant to the problem.

In this example, the initial state of the world is HTH. The goal state is THT. In your drawing above you have generated several, or maybe all, possible states in this world. After generating the states, we ask what operations (or transitions) are available that change one state into another. If a transition allows one state to turn into another, then we draw an edge (of course!) between the two states.

Using our familiar terms for graphs, each vertex contains the information for each state, as well as an edge-list of vertices to reach the next state.

If the states are few enough, exhaustively generate all the possible vertices. Then process the edges of the source vertex. Connect the vertices that can be connected. When the graph has been built, then finding the shortest path to the goal is simply a matter of running the breadth-first search.

What does the new Vertex class need to do?

Write an API for the Vertex class:

class Vertex  
{

**Assignment**

Open HeadsTailsDriver. Write the new Vertex class. Complete the class methods makeGraph and findBreadth.

**Sample Run**

Enter the initial state, three H and/or T: HTH  
The state graph has been made.  
Enter the final state, three H and/or T: THT  
The shortest path from HTH to THT is:   
HTH  
HHH  
HHT  
HTT  
TTT  
THT  
  
Enter the final state, three H and/or T: HHH  
The shortest path from HTH to HHH is:   
HTH  
HHH  
  
Enter the final state, three H and/or T: -1