Written Problems:

- Find a plane graph that is self-dual (you can do this by trial-and-error or by looking at Wikipedia, but let us know what method you use). A graph G is *self-dual* if it is isomorphic to its dual. Give the bijection between the vertices of the two graphs that demonstrates that they are isomorphic.
- Below is the rotation system of a plane graph. Draw all possible embeddings in the plane (i.e., each face should be an outer face in exactly one of the embeddings).

```
0: 4,1,3
1: 0,2,2
2: 1,1
3: 0,4,3,3
4: 3,0
```

- Below is the rotation for a graph embedded on a surface. What surface is it embedded on? Draw a nice diagram of the graph as embedded on this surface.

```
a: b,d,g
b: d,c,a
c: b,e,d
d: a,c,b
e: h,c,f
f: h,g,e
g: a,h,f
h: f,e,g
```

- Find an example of a biconnected bipartite non-Hamiltonian graph. Noting that any Hamiltonian bipartite graph must be *balanced* in the sense that the cardinalities of the partite sets must be equal, it is not hard to find an example graph that is unbalanced. Can you find one that is balanced? Explain why your graph has the required properties.
- One more problem to be added later.

Programming Problem:

Background:

This assignment is based on a problem from a programming contest. It has been modified somewhat, partially so that we can use JUnit testing. You are encouraged to try to solve it as you would encounter it in an actual contest. That is, try to solve it in an hour or two, without using the internet, but using only paper materials.

the Problem:

Two intrepid astronauts, Fred and Wilma, have volunteered to undertake travel from Earth to beyond the galaxy in order to verify whether certain planets are inhabitable or not. Fortunately, some wormholes have been discovered which effectively reduces the travel distance between some planets to zero. For all other planets distance is simply the Euclidean distance between them. Given the location of Earth, planets, and wormholes, your task is to determine the shortest travel distance between any pairs of planets. Input to our main:

- The first line of input is a single integer, T ($1 \leq T \leq 10$), the number of test cases.
- Each test case consists of planets, wormholes, and a set of distance queries.
- The planets list for a test case starts with a single integer, p ($1 < p < 60$), the number of planets. Following this are p lines, where each line contains a planet name along with the planet's integer coordinates, i.e. `name x y z` ($0 < x, y, z < 2 \cdot 10^6$). The names of the planets will consist only of ASCII letters and numbers, and will always start with an ASCII letter. Planet names are case-sensitive (Earth and earth are distinct planets). The length of a planet name will never be greater than 50 characters. All coordinates are given in parsecs.
- The wormholes list for a test case starts with a single integer, w ($0 < w < 40$), the number of wormholes, followed by the list of w wormholes. Each wormhole consists of two planet names separated by a space. The first planet name marks the entrance of wormhole, and the second planet name marks the exit from the wormhole. The planets that mark wormholes will be chosen from the list of planets given in the preceding section. *Note: you can't enter a wormhole at its exit.*
- The queries list for a test case starts with a single integer, q ($1 < q < 20$), the number of queries. Each query consists of two planet names separated by a space. Both planets will have been listed in the planet list.

Output for the main:

For each test case, output a line. "`Case` i:", the number of the ith test case. Then, for each query in that test case, output a line that states "`The distancee from planet1, to planet2 is` d `parsecs using` w `wormholes.`", where the planets are the names from the query, d is the shortest possible travel distance between the two planets, and w is the *least*number of wormholes used in any shortest route (because of ties, there could be several shortest routes, which could use different numbers of wormholes). Round d to the nearest integer. `Sample Input for main:`

```
3
4
Earth 0 0 0
Proxima 5 0 0
Barnards 5 5 0
Sirius 0 5 0
2
Earth Barnards
Barnards Sirius
6
Earth Proxima
Earth Barnards
Earth Sirius
Proxima Earth
Barnards Earth
Sirius Earth
3
z1 0 0 0
z2 10 10 10
z3 10 0 0
1
z1 z2
3
z2 z1
z1 z2
z1 z3
2
Mars 12345 98765 87654
```

```
Jupiter 45678 65432 11111
0
1
Mars Jupiter
```

Sample Output

```
Case 1:
The distance from Earth to Proxima is 5 parsecs using 0 wormholes.
The distance from Earth to Barnards is 0 parsecs using 1 wormholes.
The distance from Earth to Sirius is 0 parsecs using 2 wormholes.
The distance from Proxima to Earth is 5 parsecs using 0 wormholes.
The distance from Barnards to Earth is 5 parsecs using 1 wormholes.
The distance from Sirius to Earth is 5 parsecs using 0 wormholes.
Case 2:
The distance from z2 to z1 is 17 parsecs using 0 wormholes.
The distance from z1 to z2 is 0 parsecs using 1 wormholes.
The distance from z1 to z3 is 10 parsecs using 0 wormholes.
Case 3:
The distance from Mars to Jupiter is 89894 parsecs using 0 wormholes.
```

Program Structure:

You should write a class `Worm` whose outline is shown below. (In a programming contest, you would have to write everything from scratch.)

```
public class Worm {

   // PUT YOUR CLASS VARIABLE HERE

   public Worm( In in ) {
      // Create a new problem instance.
   }

   public double dist( String origP, String destP ) {
      // return the distance from origP to destP
   }

   public int   worms( String origP, String destP ) {
      // least number of wormholes in any shortest path from origP to destP
   }

   public String query( String origP, String destP ) {
      // Output the "The distance from ... wormholes." string.
   }

   public static void main(String[] args) {
      // You can test your program with something like this.
      In in = new In( args[0] );
      int T = in.readInt();
      for (int t=1; t<=T; t++) {
         System.out.println("Case " + t + ":") ;
         Worm w = new Worm( in );
         int Q = in.readInt() ;
         for (int i=0; i<Q; i++) {
            String p1s = in.readString() ;
            String p2s = in.readString() ;
            System.out.println( w.query( p1s, p2s ) ) ;
         }
      }
   }
}
```

In case it is not clear from the `main` above (which you should read carefully), the constructor takes as input some text in the following format.

```
4
Earth 0 0 0
Proxima 5 0 0
Barnards 5 5 0
Sirius 0 5 0
2
Earth Barnards
Barnards Sirius
```