Alphabet Soup
In this question we have two sets of letters, red and green and we are trying to match red letters with green letters, but not all of the red letters match with all of the green letters. Keeping all of this in mind, this is exactly the bipartite matching problem. (You can think of the red letters as students looking for internships and the green letters as the internship openings.) In this problem, instead of explicitly giving you which letters can be matched with which other ones, there's a simple rule: the difference in the letter values must be at least three. So, we create a network flow graph, source goes to each of the red letters with capacity 1, the red letters link to each green letter that isn't within 2 of its value with capacity 1 and then the green letters link to the sink, each with capacity 1. The max flow through this graph represents the maximum number of coffee mugs we can create.

Euclid
Take in the input and calculate the necessary vectors: AB, AC, DE and DF. From these vectors, use the cross product magnitude to get the areas of triangle DEF and the parallelogram defined by A, B and C. Divide these two areas to get a "scale factor" between the two figures. If ABC is "big" then this scale factor will be less than one. Multiply the scale factor by the vector AC. (To multiply a vector by a scalar, just multiply each component of the vector by that scalar.) This will be the vector AH. Since point A is known, just add vector AH to point A to obtain point H. From there, add vector AB (which is equal to vector HG) to point H to obtain point G.

Farmer John's Forest
It should be fairly clear that a convex hull is required to enclose all of the "outer most" trees with minimal fencing. However, the "extra margin of error seems quite problematic. One thing to notice is that in order for the tree to be exactly a distance of c away from the fence, at least a portion of the fence will be in the shape of a circle. Consider the case for one or two trees, drawn below:

In the first case, the fence is exactly a circle with radius c. In the second case, it appears as if both curved parts are semicircles with radius c. Now let's look at three trees but where I leave the pictures of the circles around the trees in.
If we carefully look at this picture, it appears as if each curved part is a "different" section of the circle and if we look at all the curved parts on the outside, together they form one circle. If you use some standard geometry, (sum of the external angles of a convex polygon equals $2\pi$ radians), then you see that this observation is true. Namely, the sum of the lengths of all the curved parts of the fence will just be a single circle of radius $c$. Thus the solution is as follows: (1) find the convex hull of the points, (2) determine the sum of the lengths of the edges on the convex hull, (3) add to the value from (2) the circumference of a circle with radius $c$, where $c$ is given in the input.

**Presidential Security**
The least cost connection between a set of nodes is simply the minimum spanning tree of the induced graph. Each room is a vertex. The edge weight between rooms is simply sum of the absolute values of the difference of floors and the absolute values of the difference of rooms. We can extract the floor by doing $/100$ and the room by doing $\%100$.

**Squirrel Territory**
If the radii of two circles are equal to $r$ units, then the circles intersect if and only if their centers are less than or equal to $2r$ units apart. So, the closer two centers are, the further restricted the radius is. Thus, we aim to find the closest pair of centers (tree locations) and divide this value by 2. Since the data is small, a double for loop through all pairs of tree locations is sufficient.

**Fujiyama Thursday**
This is probably the easiest problem in the set. Since the cars all leave at the same time, we want the students who eat the fastest to go in the slowest car and the students to eat the slowest to go in the fastest car. So, sort the driving times of the cars and the eating times of the students (two separate sorts). For implementation, reverse the array storing the student eating times, so the slowest student is in index 0. From there, just place the first four students (slowest) in the fastest car, keeping note of when the last student will finish eating.