Equivalence writeup

(1,0,1) and (2,0,2) are distinct points in R^2 but are equal in a projective finite field Z_m with m = 3 by the third property. Letting p = (1,0,1) and q = (2,0,2), the cross product is computed as

$$p \times q = [0 * 2 - 1 * 0, 1 * 2 - 1 * 2, 1 * 0 - 0 * 2] = [0,0,0]$$

By substituting in each coordinate for the given formula.

(1,0,0) and (0,1,0) are distinct points in both R^2 and the projective finite field Z_m with m = 3 by the third property. Letting p = (1,0,0) and q = (0,1,0), the cross product is computed as

$$p \times q = [0 * 0 - 0 * 1, 0 * 0 - 1 * 0, 1 * 1 - 0 * 0] = [0,0,1]$$

To double check and see if p and q are on the line, we need to calculate the multiplication of each coordinate in [0,0,1] with each coordinate in (1,0,0) and (0,1,0). If the result is 0 for one of the points, then the point is on the line.

$$0 * 1 + 0 * 0 + 1 * 0 = 0$$

$$0 * 0 + 0 * 1 + 1 * 0 = 0$$

Each equation is true, thus each point is on the line.

Generating all points recipe

Recipe name: generate_all_points

Inputs:

- mod, the modulus equal to m for Z_m, the finite field in which the projective geometry is taking place

Outputs:

- points_list, a sequence of all the points generated by the function

Steps:

1. Create points_list as an empty sequence.
2. Create a Boolean variable is_in_list that will represent whether or not a newly generated point is already in points_list. Initialize it to False.
3. For each x_coord in 0, 1, 2,…, m-1, do
    a. For each y_coord in 0, 1, 2,…, m-1, do
        i. For each z_coord in 0, 1, 2,…, m-1, do
            1. Check to see if the new point would be (0,0,0). If it is, then set the new point to (0,0,1). Since the next iteration would be this anyways and thus will be removed, this skips (0,0,0) without using bad programming techniques like using break or too much nested code.
            2. Create a tuple new_point that is assigned the value (x_coord, y_coord, z_coord)
            3. Set is_in_list to false.
            4. For each point in points_list, do
                a. Check to see if new_point is equal to point using the function equivalent with the arguments (new_point, point, mod)
                b. If it is, set is_in_list equal to True
            5. If is_in_list is false, then create a new sequence to replace points_list that is essentially points_list but with new_point at the end of the sequence.
4. Return points_list

Output:

- points_list, a sequence of all the points generated by the function

<u>Create cards recipe</u>

Recipe name: create_cards

Inputs:

- points, a sequence of unique points, each represented by a tuple of 3 integers
- lines, a sequence of unique lines, each represented by a tuple of 3 integers
- mod, an integer representing the prime modulus

Outputs:

- cards_list, a sequence of sequences where each subsequence represents a card in the game

Steps:

1. Create a new sequence, cards_list, that will store every generated card
2. for each line in lines, do
    a. initialize a new sequence to an empty sequence called card
    b. for each point_idx in the indices of points, do
        i. Use the function incident with the arguments (points[point_idx] which is the element of points that is at the index point_idx, line, mod). If it returns true, then
            1. Add the point_idx to card
    c. Append card to the end of cards_list
3. Return cards_list

Outputs:

- cards_list, a sequence of sequences where each subsequence represents a card in the game

Discussion

It is not possible to create a valid deck of 40 “Spot it!” cards because there is no space Z_m where m is prime (and thus a valid projective geometrical space) that would produce 40 spot it cards. When m = 5, 31 cards are generated. When m = 7, 57 cards are generated. Since there’s no prime number between 5 and 7, and the number of generated cards is proportional to the number of spot it cards, 40 isn’t possible.