# groupaction

November 10, 2019

## 1 Exercise 1

```
In [37]: # Exercise 1: Dihedral group acting on vertices of regular polygon

In [2]: # Let D4 act naturally  on the vertices {1,2,3,4} of a square

In [3]: D4 = DihedralGroup(4)

In [4]: # How many orbits are there?

In [5]: D4.orbits()

Out[5]: [[1, 2, 4, 3]]

In [6]: # What are the stabilizers?

In [7]: D4.stabilizer(1)

Out[7]: Subgroup generated by [(2,4)] of (Dihedral group of order 8 as a permutation group)

In [8]: list(D4.stabilizer(1))

Out[8]: [(), (2,4)]

In [9]: list(D4.stabilizer(2))

Out[9]: [(), (1,3)]

In [10]: # Are they all isomorphic?

In [11]: D41 = D4.stabilizer(1); D42 = D4.stabilizer(2)

In [12]: D41.is_isomorphic(D42)

Out[12]: True

In [13]: # What are the fixpoint sets?

In [16]: for g in D4:
             print g, D4.subgroup([g]).fixed_points()
```

```
() [1, 2, 3, 4]
(1,3)(2,4) []
(1,4,3,2) []
(1,2,3,4) []
(2,4) [1, 3]
(1,3) [2, 4]
(1,4)(2,3) []
(1,2)(3,4) []
```

In [17]:
```
# Verify Burnside's thm
```

In [20]:
```
su,si = add([len(D4.subgroup([g]).fixed_points()) for g in D4]) , D4.order()
```

In [21]:
```
su,si, su/si
```

Out[21]:
```
(8, 8, 1)
```

In [23]:
```
# Let D4 act on colorings with k colors, how many orbits?
```

In [ ]:
```
# Use Burnside
```

In [25]:
```
var('s,t,u,k')
s=0
for g in D4:
    u = g.cycle_type()
    t = k^len(u)
    s = s+ t
    print g,u,t
```

```
() [1, 1, 1, 1] k^4
(1,3)(2,4) [2, 2] k^2
(1,4,3,2) [4] k
(1,2,3,4) [4] k
(2,4) [2, 1, 1] k^3
(1,3) [2, 1, 1] k^3
(1,4)(2,3) [2, 2] k^2
(1,2)(3,4) [2, 2] k^2
```

In [32]:
```
print s/D4.order()
```

```
1/8*k^4 + 1/4*k^3 + 3/8*k^2 + 1/4*k
```

In [33]:
```
[[j,s.subs(k=j)/D4.order()] for j in range(1,10)]
```

Out[33]:
```
[[1, 1],
 [2, 6],
 [3, 21],
```

```
 [4, 55],
 [5, 120],
 [6, 231],
 [7, 406],
 [8, 666],
 [9, 1035]]
```

```
In [22]: # Do the same for D5, D6, D7
```

## 2 Exercise 2

```
In [34]: # Exercise 2, rotations of the cube

In [35]: # Label the vertices of the cube, first the top, cc, then the bottom, cc

In [38]: # Verify that these two 90 deg rotations generate the group

In [46]: rot1 = Permutation('(1,2,3,4)(5,6,7,8)'); rot1

Out[46]: [2, 3, 4, 1, 6, 7, 8, 5]

In [48]: rot2 = Permutation('(1,4,8,5)(2,3,7,6)'); rot2

Out[48]: [4, 3, 7, 8, 1, 2, 6, 5]

In [49]: cube = PermutationGroup([rot1,rot2])

In [50]: cube.order()

Out[50]: 24

In [51]: # What are the stabilizers? Are they isomorphic?

In [52]: S1 = cube.stabilizer(1); list(S1)

Out[52]: [(), (2,5,4)(3,6,8), (2,4,5)(3,8,6)]

In [55]: # What are the fixpoints? If you fix 1, need you also fix its antipodal?

In [56]: S1.fixed_points()

Out[56]: [1, 7]

In [57]: # Verify Burnside

In [58]: su,si = add([len(cube.subgroup([g]).fixed_points()) for g in cube]) , cube.order()

In [59]: (su,si,su/si)

Out[59]: (24, 24, 1)

In [60]: # In how many ways can you color the vertices of the cube, up to rotational symmetry, u
```

## 3 Exercise 3

```
In [61]: # The full symmetry group of the cube (including refelctions) can be generated by addin

In [62]: ap = Permutation('(1,7)(2,8)(3,5)(4,6)'); ap

Out[62]: [7, 8, 5, 6, 3, 4, 1, 2]

In [66]: fullcube = PermutationGroup(cube.gens() + [ap])

In [67]: fullcube.gens()

Out[67]: [(1,2,3,4)(5,6,7,8), (1,4,8,5)(2,3,7,6), (1,7)(2,8)(3,5)(4,6)]

In [68]: fullcube.order()

Out[68]: 48

In [70]: cube.is_normal(fullcube)

Out[70]: True

In [72]: list(fullcube.stabilizer(1))

Out[72]: [(), (2,5,4)(3,6,8), (2,4,5)(3,8,6), (3,6)(4,5), (2,5)(3,8), (2,4)(6,8)]

In [73]: # Do the exercises from EX2 for the full symmetry group

In [74]: # Also: can the full symmetry group be generated by only two elements?
```

## 4 Exercise 4

```
In [75]: # Consider the following arrangement of 16 points

In [45]: ra = 0.3

In [46]: C0 = circle((0,0),1)

In [47]: C1 = circle((1,0),ra)

In [48]: C2 = circle((0,1),ra)

In [49]: C3 = circle((-1,0),ra)

In [50]: C4 = circle((0,-1),ra)

In [51]: # patience...

In [52]: PP = point([(ra,0),(0,ra),(-ra,0),(0,-ra)])

In [67]: PP1 = point([(1+ra,0),(1+0,ra),(1-ra,0),(1+0,-ra)], color='red',size=30)
```

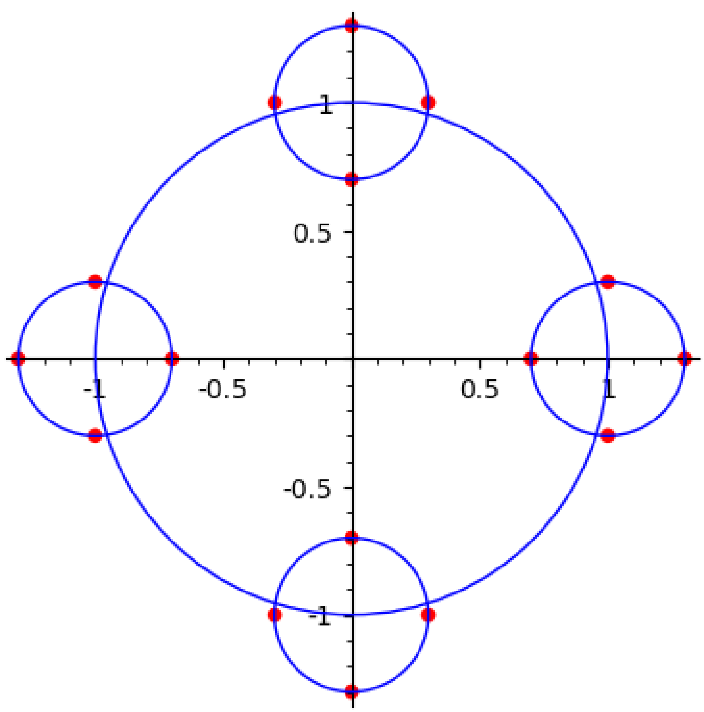
In [68]:
```
PP2 = point([(ra,1+0),(0,1+ra),(-ra,0+1),(0,-ra+1)], color='red',size=30)
```

In [69]:
```
PP3 = point([(ra-1,0),(0-1,ra),(-ra-1,0),(0-1,-ra)], color='red',size=30)
```

In [70]:
```
PP4 = point([(ra,0-1),(0,ra-1),(-ra,0-1),(0,-ra-1
                                    )], color='red',size=30)
```

In [71]:
```
C0 + C1 + C2 + C3 + C4 + PP1 + PP2 + PP3 + PP4
```

Out[71]:




In [64]:
```
# Label the vertices 1-16, cc in each small circle
```

In [122]:
```
# We are allowed to rotate the whole shebang:
```

In [1]:
```
R = Permutation('(1,5,9,13)(2,6,10,14)(3,7,11,15)(4,8,12,16)')
```

In [124]:
```
# There are also rotations of each small circle
```

In [2]:
```
r1 = Permutation((1,2,3,4))
r2 = Permutation((5,6,7,8))
r3 = Permutation((9,10,11,12))
r4 = Permutation((13,14,15,16))
```

In [127]:
```
# However, we are not allowed to perform these small rotations individually
```

```
In [128]: # What we may do, is to first rotate all small circles
          # Then every circle but the first
          # Then the last two
          # Finally the last
```

```
In [3]: s = r1*r2^2*r3^3*r4^4; s.cycle_string()
```

```
Out[3]: '(1,2,3,4)(5,7)(6,8)(9,12,11,10)'
```

```
In [131]: # This, then, is the group of allowed rotations, acting on the vertices
```

```
In [4]: StrangeGroup = PermutationGroup([R,s],canonicalize=False)
```

```
In [5]: StrangeGroup.order()
```

```
Out[5]: 64
```

```
In [6]: StrangeGroup.is_abelian()
```

```
Out[6]: False
```

```
In [182]: #
```

```
In [136]: # What is the stabilizer of vertex 1?
```

```
In [7]: ST1 = StrangeGroup.stabilizer(1); list(ST1)
```

```
Out[7]: [(),
         (5,7)(6,8)(13,15)(14,16),
         (5,6,7,8)(9,11)(10,12)(13,16,15,14),
         (5,8,7,6)(9,11)(10,12)(13,14,15,16)]
```

```
In [139]: # I want actually perform the sequence of allowed rotations to achieve these group ele
```

```
In [8]: var('x,y'); x,y = StrangeGroup.gens()
```

```
In [9]: x,y
```

```
Out[9]: ((1,5,9,13)(2,6,10,14)(3,7,11,15)(4,8,12,16), (1,2,3,4)(5,7)(6,8)(9,12,11,10))
```

```
In [10]: R.cycle_string(),s.cycle_string()
```

```
Out[10]: ('(1,5,9,13)(2,6,10,14)(3,7,11,15)(4,8,12,16)',
          '(1,2,3,4)(5,7)(6,8)(9,12,11,10)')
```

```
In [11]: h = list(ST1)[1]; h
```

```
Out[11]: (5,7)(6,8)(13,15)(14,16)
```

```
In [12]: h.word_problem([x,y])
```

```
(x2^-1*x1^-2)^2
[['((1,2,3,4)(5,7)(6,8)(9,12,11,10)', -1]]
```

```
Out[12]: ('(x2^-1*x1^-2)^2',
          '((1,2,3,4)(5,7)(6,8)(9,12,11,10)^-1*(1,5,9,13)(2,6,10,14)(3,7,11,15)(4,8,12,16)^-2)^2
```

```
In [13]: ((s^(-1)*R^(-2))^2).cycle_string()
```

```
Out[13]: '(5,7)(6,8)(13,15)(14,16)'
```

```
In [176]: # So, first three compound small circle moves
          # then two wholeshebangs
          # then repeat?
          # Or is that backwards?
```

```
In [177]: # Check the other elements of the stabilizer of vertex one
```

```
In [178]: # Also check if (1,2,3) is the group
```

```
In [14]: dubious = Permutation('(1,2,3)')
```

```
In [72]: StrangeGroup(dubious)
```

```
In [17]: #
```

```
In [ ]: # In how many ways can we color the vertices with k colors, if two colorings
        # are considered equivalent if they can be transformed into each other using
        # the symmetries of StrangeGroup?
```

## 5 Exercise 5

```
In [18]: # An undirected graph on [n] is a subset of the set
         # binomial([n],2) of potential edges
```

```
In [19]: # The natural action of Sn on [n] extends to an action on binomial([n],2)
         # and to its power set
```

```
In [111]: n=2
```

```
In [112]: print "Graphs with ", n, " vertices"
```

```
Graphs with  2  vertices
```

```
In [92]: Gn = SymmetricGroup(n); Gn
```

```
Out[92]: Symmetric group of order 3! as a permutation group
```

```
In [86]: Xn = range(1,n+1); Xn
```

```
Out[86]: [1, 2, 3]

In [96]: EDGES = Subsets(Xn,2,submultiset=True).list(); EDGES

Out[96]: [[1, 2], [1, 3], [2, 3]]

In [99]: gap("Orbits(" + str(Gn._gap_()) + "," + str(EDGES) + ",OnSets)")

Out[99]: [ [ [ 1, 2 ], [ 2, 3 ], [ 1, 3 ] ] ]

In [100]: GRAPHS=list(subsets(EDGES)); GRAPHS

Out[100]: [[],
           [[1, 2]],
           [[1, 3]],
           [[1, 2], [1, 3]],
           [[2, 3]],
           [[1, 2], [2, 3]],
           [[1, 3], [2, 3]],
           [[1, 2], [1, 3], [2, 3]]]

In [102]: ISOCLASSES=gap("Orbits(" + str(Gn._gap_()) + "," + str(GRAPHS) + ",OnSetsSets)")

In [109]: for c in ISOCLASSES:
              print "Graphs with ", len(c[1]), "edges"
              print "Isoclass has this many graphs: ",len(c)
              print "It contains: ", c

Graphs with  0 edges
Isoclass has this many graphs:  1
It contains:  [ [  ] ]
Graphs with  1 edges
Isoclass has this many graphs:  3
It contains:  [ [ [ 1, 2 ] ], [ [ 2, 3 ] ], [ [ 1, 3 ] ] ]
Graphs with  2 edges
Isoclass has this many graphs:  3
It contains:  [ [ [ 1, 2 ], [ 1, 3 ] ], [ [ 1, 2 ], [ 2, 3 ] ], [ [ 1, 3 ], [ 2, 3 ] ] ]
Graphs with  3 edges
Isoclass has this many graphs:  1
It contains:  [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ] ]


In [ ]: for n in range(2,5+1):

            Gn = SymmetricGroup(n);
            Xn = range(1,n+1);
            EDGES = Subsets(Xn,2,submultiset=True).list();
            GRAPHS=list(subsets(EDGES));
            ISOCLASSES=gap("Orbits(" + str(Gn._gap_()) + "," + str(GRAPHS) + ",OnSetsSets)")
```

```
            print "Graphs with ", n, " vertices has ", len(ISOCLASSES), " isoclasses of graphs."
            for c in ISOCLASSES:
                print "Graphs with ", n, "vertices and ", len(c[1]), "edges"
                print "Isoclass has this many graphs: ",len(c)
                print "It contains: ", c
            print
            print
```

```
In [ ]: # Fix the output so that it says
        # For n vertices, there are
        # c(n,k) non-isomorphic graphs with k edges
        # here is a representative for each isoclass
        # If you can, feed it to SAGE and have SAGE plot the representative!
```

## 6 Exercise 6 (hors competition)

```
In [ ]: # What if the graphs are directed, and may contain loops?

In [16]: n=3; n

Out[16]: 3

In [17]: VER = range(1,n+1); VER

Out[17]: [1, 2, 3]

In [14]: Gn = SymmetricGroup(n); Gn

Out[14]: Symmetric group of order 3! as a permutation group

In [4]: DIEDGES = IntegerVectors(length=2, min_part=1,max_part=n).list();  len(DIEDGES)

Out[4]: 9

In [5]: DIEDGES

Out[5]: [[1, 1], [2, 1], [1, 2], [3, 1], [2, 2], [1, 3], [3, 2], [2, 3], [3, 3]]

In [6]: DIGRAPHS = list(subsets(DIEDGES)); len(DIGRAPHS)

Out[6]: 512

In [7]: ISOCLASSES=gap("Orbits(" + str(Gn._gap_()) + "," + str(DIGRAPHS) + ",OnSetsTuples)")

In [8]: len(ISOCLASSES)

Out[8]: 427

In [9]: ISOCLASSES[5]
```

```
Out[9]: [ [ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ],
          [ [ 2, 1 ], [ 2, 2 ], [ 2, 3 ], [ 3, 1 ] ],
          [ [ 1, 2 ], [ 3, 1 ], [ 3, 2 ], [ 3, 3 ] ],
          [ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 3, 2 ] ],
          [ [ 2, 1 ], [ 3, 1 ], [ 3, 2 ], [ 3, 3 ] ],
          [ [ 1, 3 ], [ 2, 1 ], [ 2, 2 ], [ 2, 3 ] ] ]
```

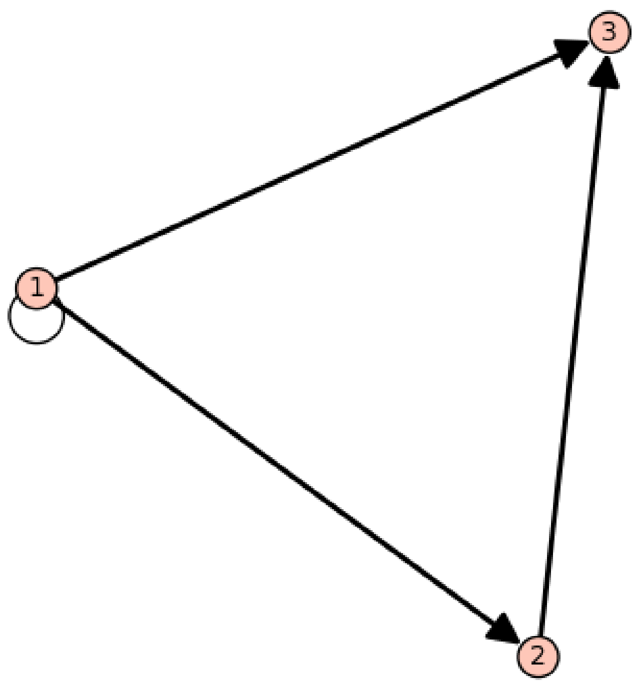
```
In [10]: adi =ISOCLASSES[5][1]; adi
```

```
Out[10]: [ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ]
```

```
In [38]: DI=DiGraph(adi,format='list_of_edges',loops=True)
```

```
In [39]: DI.plot()
```

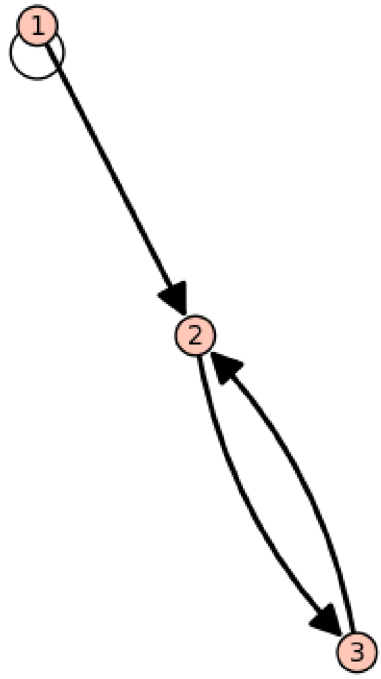
```
Out[39]:
```




```
In [ ]: # Another...
```

```
In [41]: DiGraph(ISOCLASSES[60][1],format='list_of_edges',loops=True).plot()
```

```
Out[41]:
```




```
In [42]: # How many non-isomorphic digraphs with 3,4,5 vertices are there?
         # How many with specific number of directed edges?
         # Plot all representatives for n=3

In [ ]:
```