Project

February 11, 2024

0.0.1 Sage Code

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from time import time
def is_GroupByGenerators(group, generators, debug=False):
    """
    Check if a `group` is generated by given `generators`.

    INPUT:

    - `group` -- a group object.
    - `generators` -- a list or tuple of elements that generate the group.

    OUTPUT:

    Boolean.

    EXAMPLES:

        sage: G = SymmetricGroup(3)
        sage: generators = [(1,2), (1,2,3)]
        sage: is_GroupByGenerators(G, generators)
        True
    """
    from sage.libs.gap.element import GapElement
    if not isinstance(group, GapElement):
        group = group._libgap_()
    x = set(group.AsList()) == set(libgap.GroupByGenerators(generators).
 ↪AsList())
    if debug:
        print(group.AsList(),libgap.GroupByGenerators(generators).AsList(),x)
    return x


def minimum_generating_set(group, gap_based=False, debug = False):
    """
    Return a minimum generating set of the `group`.
```

```
    INPUT:

    - `group` -- a group object.
    - `gap_based` -- boolean (default: False). If True, the output is GAP based.

    OUTPUT:

    A set of elements that generate the group.

    EXAMPLES::

        sage: G = SymmetricGroup(3)
        sage: minimum_generating_set(G)
        {[1, 3, 2], [2, 3, 1]}

        sage: G = GL(2,GF(3))
        sage: s = minimum_generating_set(G, gap_based=True); s
        {[ [ Z(3)^0, Z(3)^0 ], [ Z(3), 0*Z(3) ] ],
         [ [ Z(3), 0*Z(3) ], [ 0*Z(3), Z(3)^0 ] ]}
        sage: type(list(s)[0])
        <class 'sage.libs.gap.element.GapElement_List'>
    """
    from sage.misc.functional import log
    from sage.libs.gap.element import GapElement

    if not isinstance(group, GapElement):
        group = group._libgap_()
    if not group.IsFinite().sage():
        raise NotImplementedError("Implemented for finite group only")

    group_elements = group.AsList()
    if debug:
        print("\nFinding mingen for G =",group," of length␣
↪",len(group_elements))

    if group.IsCyclic().sage():
        if debug:
            print("Group is cyclic.")
        for ele in group_elements:
            if is_GroupByGenerators(group, [ele]):
                if gap_based:
                    ret = set([ele])
                else:
                    ret = set([ele.sage()])
                if debug:
                    print("mingen : ",ret)
```

```
                return ret

    if group.IsSimple().sage():
        if debug:
            print("Group is simple.")
        n = len(group_elements)
        for i in range(n):
            for j in range(i+1, n):
                if is_GroupByGenerators(group,[group_elements[i],␣
↪group_elements[j]]):
                    if gap_based:
                        ret = set([group_elements[i], group_elements[j]])
                    else:
                        ret= set([group_elements[i].sage(), group_elements[j].
↪sage()])
                    if debug:
                        print("mingen :",ret)
                    return ret

    # The MinimalNormalSubgroups method returns a list of all minimal normal␣
↪subgroups
    # but for this algorithm we need only one minimal normal subgroup (which is␣
↪not trivial).
    # TODO: Replace the function with the one that gives only one minimal␣
↪normal subgroup
    N = group.MinimalNormalSubgroups()[0]
    if debug:
        print("N:",N,len(N.AsList()))
    n = N.SmallGeneratingSet()
    if debug:
        print("n:",n,len(n))
    phi = group.NaturalHomomorphismByNormalSubgroup(N)
    GbyN = phi.ImagesSource()
    if debug:
        print("GbyN:",GbyN,len(GbyN.AsList()))
    GbyN_mingenset = minimum_generating_set(GbyN, gap_based=True,debug=debug)
    if debug:
        print("\nmingen(GbyN) of length",len(GbyN_mingenset),":",GbyN_mingenset)
    g = [phi.PreImagesRepresentative(g) for g in list(GbyN_mingenset)]
    l = len(g)
    if debug:
        print("g of length ",len(g),":",g)

    if N.IsAbelian().sage():
        if debug:
            print("N is abelian")
        if is_GroupByGenerators(group, g):
```

```
            if gap_based:
                ret = set(g)
            else:
                ret = set([ele.sage() for ele in g])
            if debug:
                print("mingen:",ret)
            return ret
        for i in range(l):
            for j in range(len(n)):
                modifeid_g = g[:i] + [g[i]*n[j]] + g[i+1:]
                if is_GroupByGenerators(group, modifeid_g):
                    if gap_based:
                        ret= set(modifeid_g)
                    else:
                        ret= set([ele.sage() for ele in modifeid_g])
                    if debug:
                        print("mingen:",ret)
                    return ret
        if debug:
            print("none of the mmodified g worked.")
        if gap_based:
            ret = set(g+[n[0]])
        else:
            ret = set([ele.sage() for ele in g] + [n[0].sage()])
        if debug:
            print("mingen:",ret)
        return ret

    def gen_combinations(g, N_old, t, debug=False):
        # This function is used to generate some combinations (which are␣
↪required for the algorithm)
        # of the elements of N_old and g.
        L = [g]
        N = [ele for ele in N_old]  # This line is included because N_old does␣
↪not have slicing method
        if debug:
            print("\n finding combinations for N=",N," and g=",g)
        N = N[1:]

        for i in range(t):
            newL = []
            for g in L:
                for j in range(len(N)):
                    x = g[:i]
                    y = g[i]
                    y = y * (N[j])
                    x = x + [y]
```

```
                    x = x + g[i+1:]
                    newL.append(x)
            L = L + newL
            if debug:
                print(f"after iteration number {i+1}:",L)
        return L

    def explode(g,N,t):
        t = -int(-t)
        if t>len(g):
            t = len(g)
        if t<=0:
            yield g
        for go in explode(g,N,t-1):
            for j in range(len(N)):
                gm = go[:t-1] + [go[t-1]*N[j]] + go[t:]
                yield gm


    t = -int(-(13/5 + log(group.Size().sage(), 2)/log(N.Size().sage(), 2)))
    if debug:
        print("t = ",t,", l =",l)
    """
    if t <= l:
        for gens in gen_combinations(g, N.AsList(), t):
            if is_GroupByGenerators(group, gens):
                if gap_based:
                    ret =  set(gens)
                else:
                    ret = set([ele.sage() for ele in gens])
                if debug:
                    print("mingen:",ret)
                return ret
    """
    for gens in explode(g,N.AsList(),t):
        if is_GroupByGenerators(group,gens):
            if gap_based:
                ret = set(gens)
            else:
                ret = set([ele.sage() for ele in gens])
            if debug:
                print("mingen:",ret)
            return ret
    for raw_gens in explode(g, N.AsList(), l):
        for nl in [ele for ele in N.AsList()][1:]:
            if nl in raw_gens:
                continue
```

```
            gens = raw_gens+[nl]
            if is_GroupByGenerators(group, gens):
                if gap_based:
                    ret = set(gens)
                else:
                    ret = set([ele.sage() for ele in gens])
                if debug:
                    print("raw_gens",raw_gens)
                    print("nl:",nl)
                    print("mingen:",ret)
                return ret

def Z_p_S_3(p,q=3):
    S = SymmetricGroup(q)._libgap_()
    Z = PermutationGroup([tuple((i+1 for i in range(p)))])._libgap_()
    SZ = libgap.DirectProduct(S,Z)
    return SZ

def Z_2_to_n(n):
    return PermutationGroup([(2*i+1,2*i+2) for i in range(n)])._libgap_()

def Z_n(n):
    Z = PermutationGroup([tuple((i+1 for i in range(n)))])._libgap_()
    return Z

def S_n(n):
    S = SymmetricGroup(n)._libgap_()
    return S

def D_n(n):
    D = DihedralGroup(n)._libgap_()
    return D

def TimeAndPlot():
    D = {
        Z_p_S_3:(10,2,1,3,r"Z_3\times S_n"),
        Z_2_to_n:(10,2,2,5,"Z_2^n"),
        S_n:(7,2,1,3,'S_n'),
        Z_n:(30,2,1,10,'Z_n'),
        D_n:(30,1,1,10,'D_n')
    }
    print("| Group Type",";en(G)","len(mingen(G))","mingen(g) |",sep=' | ')
    for Gfunc in D:
        N,N0,d,iterations,name = D[Gfunc]
        plt.figure()

        y = []
```

```
        x = []
        for n in range(N0,N,d):
            G = Gfunc(n)
            assert G is not None
            to = time()
            for _ in range(iterations):
                g = minimum_generating_set(G)
            print("| $$ "+name+" $$",len(G.AsList()),len(g),",".
↪join([str(tuple(x)) for x in g])+" |",sep=' | ')
            y.append((time()-to)/iterations)
            x.append(len(G.AsList()))
        y = np.log(np.array(y))
        x = np.array(x)
        plt.plot(x,y,'-o')

        # Curve fitting
        Ln = np.log(x)
        X = np.array([[ln,1] for ln in Ln])
        XT = X.T
        XTX = X.T @ X
        XTXi = np.linalg.inv(XTX)
        pseudo_inverse = XTXi @ XT
        theta = np.dot(pseudo_inverse,y)
        a,b = theta
        x = np.linspace(x[0],x[-1],1000)
        Ln = np.log(x)
        L_pred = Ln*a + b
        plt.plot(x,L_pred)

        plt.xlabel("$ |G| $")
        plt.ylabel(r"$\ln(t)$ ($ t $ in seconds)")
        plt.title(f"Time ($ t $) to find minimum generating set for $ G =␣
↪{name} $")
        plt.legend(["Actual",f"logarithmic curve fitted"])
    #plt.show()

TimeAndPlot()
```

0.0.2 Output

$G \cong$	card(G)	card(mingen(G))	mingen(G)
$Z_3 \times S_n$	12	2	(1, 3, 2),(2, 3, 1, 5, 4)
$Z_3 \times S_n$	18	2	(1, 3, 2),(2, 3, 1, 5, 6, 4)
$Z_3 \times S_n$	24	2	(2, 3, 1, 5, 6, 7, 4),(1, 3, 2)
$Z_3 \times S_n$	30	2	(2, 3, 1),(1, 3, 2, 5, 6, 7, 8, 4)
$Z_3 \times S_n$	36	2	(2, 3, 1, 9, 4, 5, 6, 7, 8),(1, 3, 2)
$Z_3 \times S_n$	42	2	(2, 3, 1),(1, 3, 2, 5, 6, 7, 8, 9, 10, 4)
$Z_3 \times S_n$	48	2	(1, 3, 2),(2, 3, 1, 5, 6, 7, 8, 9, 10, 11, 4)
$Z_3 \times S_n$	54	2	(1, 3, 2),(2, 3, 1, 5, 6, 7, 8, 9, 10, 11, 12, 4)
Z_2^n	4	2	(1, 2, 4, 3),(2, 1)
Z_2^n	16	4	(1, 2, 4, 3),(1, 2, 3, 4, 6, 5),(1, 2, 3, 4, 5, 6, 8, 7),(2, 1)
Z_2^n	64	6	(1, 2, 3, 4, 5, 6, 8, 7),(2, 1),(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11),(1, 2, 4, 3),(1, 2, 3, 4, 6, 5),(1, 2, 3, 4, 5, 6, 7, 8, 10, 9)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
Z_2^n	256	8	(1, 2, 3, 4, 5, 6, 8, 7),(2, 1),(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11),(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 15),(1, 2, 4, 3),(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 13),(1, 2, 3, 4, 6, 5),(1, 2, 3, 4, 5, 6, 7, 8, 10, 9)
S_n	2	1	(2, 1)
S_n	6	2	(2, 3, 1),(1, 3, 2)
S_n	24	2	(2, 3, 1),(1, 2, 4, 3)
S_n	120	2	(2, 3, 4, 1),(1, 2, 3, 5, 4)
S_n	720	2	(1, 2, 3, 4, 6, 5),(2, 3, 4, 5, 6, 1)
Z_n	2	1	(2, 1)
Z_n	3	1	(2, 3, 1)
Z_n	4	1	(2, 3, 4, 1)
Z_n	5	1	(2, 3, 4, 5, 1)

$G \cong$	$\mathrm{card}(G)$	$\mathrm{card}(\mathrm{mingen}(G))$	$\mathrm{mingen}(G)$
Z_n	6	1	(2, 3, 4, 5, 6, 1)
Z_n	7	1	(2, 3, 4, 5, 6, 7, 1)
Z_n	8	1	(2, 3, 4, 5, 6, 7, 8, 1)
Z_n	9	1	(2, 3, 4, 5, 6, 7, 8, 9, 1)
Z_n	10	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 1)
Z_n	11	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1)
Z_n	12	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1)
Z_n	13	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1)
Z_n	14	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1)
Z_n	15	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1)
Z_n	16	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1)
Z_n	17	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1)

$G \cong$	$\mathrm{card}(G)$	$\mathrm{card}(\mathrm{mingen}(G))$	$\mathrm{mingen}(G)$
Z_n	18	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1)
Z_n	19	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1)
Z_n	20	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1)
Z_n	21	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1)
Z_n	22	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 1)
Z_n	23	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 1)
Z_n	24	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 1)
Z_n	25	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 1)
Z_n	26	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 1)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
Z_n	27	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 1)
Z_n	28	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 1)
Z_n	29	1	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 1)
D_n	2	1	(2, 1)
D_n	4	2	(1, 2, 4, 3),(2, 1)
D_n	6	2	(2, 3, 1),(1, 3, 2)
D_n	8	2	(2, 3, 4, 1),(1, 4, 3, 2)
D_n	10	2	(1, 5, 4, 3, 2),(2, 3, 4, 5, 1)
D_n	12	2	(1, 6, 5, 4, 3, 2),(6, 1, 2, 3, 4, 5)
D_n	14	2	(1, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 1)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
D_n	16	2	(1, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 1)
D_n	18	2	(1, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 1)
D_n	20	2	(8, 9, 10, 1, 2, 3, 4, 5, 6, 7),(1, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	22	2	(1, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1)
D_n	24	2	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1),(1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	26	2	(1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1)
D_n	28	2	(10, 11, 12, 13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9),(1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	30	2	(1, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(14, 15, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
D_n	32	2	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1),(1, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
D_n	34	2	(1, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1)
D_n	36	2	(12, 13, 14, 15, 16, 17, 18, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11),(1, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	38	2	(1, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1)
D_n	40	2	(10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2, 3, 4, 5, 6, 7, 8, 9),(1, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	42	2	(11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10),(1, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	44	2	(14, 15, 16, 17, 18, 19, 20, 21, 22, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13),(1, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
D_n	46	2	(1, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 1)
D_n	48	2	(1, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(18, 19, 20, 21, 22, 23, 24, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17)
D_n	50	2	(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 1),(1, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
D_n	52	2	(16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15),(1, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)

$G \cong$	card(G)	card(mingen(G))	mingen(G)
D_n	54	2	(1, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 1)
D_n	56	2	(1, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(26, 27, 28, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25)
D_n	58	2	(1, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2),(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 1)