G - large (but finite) group

$G = \langle S \rangle$ elements of S - permutations.
(of large degree).

Aims: • Compute the order of G.

• find out if given permutation σ actually belongs to G
(membership test).

usually hard when G is given abstractly.

sometimes easy when G is given by property

Anti-aims:

• enumerating / storing all of elements of G.

In general we may want to store $O(|S|)$ additional elements to speed up the computations

(Note: usually $|G| \sim O(2^{|S|})$).

Basis and stabilizer chains.

Let (G, S) be given as previously and let $G \curvearrowright \Omega$.

<u>Defn</u>: A sequence/vector/tuple/list of points
$$(\beta_1, \dots, \beta_m) \in \Omega^m$$
is called a <u>basis</u> iff every $\sigma \in G$ can be uniquely determined by
$$(\beta_1^\sigma, \dots, \beta_m^\sigma)$$

<u>Ex</u>:

$$\sigma = (1,2)(3,4) \dots (999,1000)$$
$$\tau = (1,2)(3,4), \dots, (999,1000,1001)$$

$G = \langle \sigma, \tau \rangle \subset \mathrm{Sym}(1001)$ but it's enough to observe the action of $g \in G$ on $(\beta_1, \dots, \beta_3) = (999, 1000, 1001)$.

Suppose that such $(\beta_1, \dots, \beta_m)$ is given and $(\alpha_1, \dots, \alpha_m)$ is supplied.

Can we determine the permutation $\sigma \in G$ that takes $(\beta_1 \dots, \beta_m) \longrightarrow (\alpha_1, \dots, \alpha_m)$?

Consider

$$G = G^{(0)} > G^{(1)} > \dots > G^{(m)} = \{id\}$$
where $G^{(i)} = \mathrm{Stab}_{G^{(i-1)}}(\beta_i)$.

$(\beta_1, \dots, \beta_m)$

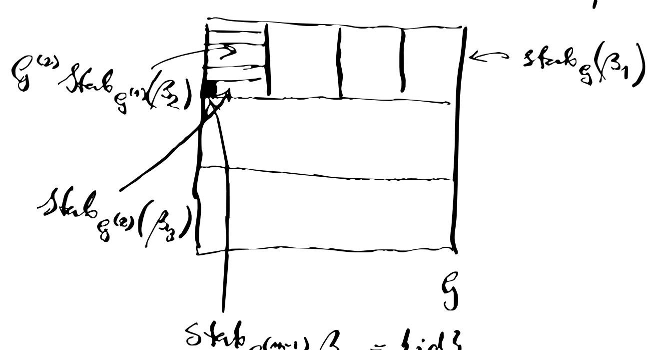
- only id stabilizes all of them.
- pick β_1 and let $\mathcal{G}^{(1)} < \mathcal{G}$ be its stabilizer

By Orbit - Stabilizer we can divide $\mathcal{G}$ into $\mathrm{Stab}_{\mathcal{G}}(\beta_1)$ - cosets - given by the orbit $\beta_1^{\mathcal{G}}$

$$\beta_1 \to \beta_1^{g_1} \to \dots \to \beta_1^{g_k}$$

$\mathcal{G}^{(1)} = \mathrm{Stab}_{\mathcal{G}}(\beta_1)$ $\quad \mathrm{Stab}_{\mathcal{G}}(\beta_1) g_1$

Inside $\mathrm{Stab}_{\mathcal{G}}(\beta_1)$ find the stabilizer of β_2



- every element of $\mathrm{Stab}_{\mathcal{G}^{(1)}}(\beta_2)$ fixes β_1 <u>and</u> β_2
- all elements that fix β_1 can be divided into subsets based on where do these send β_2.

Let $\sigma \in \mathcal{G}$

- identify $\beta_1^{\sigma} \longleftrightarrow r_1$ (pt on the orbit) ← coset representative for $\mathrm{stab}_{\mathcal{G}}(\beta_1) \backslash \mathcal{G}$, $\mathrm{stab}_{\mathcal{G}}(\beta_1) = \mathcal{G}^{(1)}$

- set $\sigma_1 = \sigma \cdot r_1^{-1} \in \mathcal{G}^{(1)}$

- identify $\beta_2^{\sigma_1} \longleftrightarrow r_2$ ← coset representative for $\mathrm{stab}_{\mathcal{G}}(\beta_2) \backslash \mathcal{G}^{(1)}$, $\mathrm{stab}_{\mathcal{G}}(\beta_2) = \mathcal{G}^{(2)}$

- set $\sigma_2 = \sigma_1 \cdot r_2^{-1} = \sigma \cdot r_1^{-1} \cdot r_2^{-1} \in \mathcal{G}^{(2)}$

$\vdots$

Play the same game until we have found $\sigma_m = \sigma \cdot r_1^{-1} \cdot r_2^{-1} \cdot \ldots \cdot r_m^{-1} \in \mathcal{G}^{(m)} = \{id\}$.

we recover $\sigma = r_m \cdot r_{m-1} \cdot \ldots \cdot r_1$.

ALGORITHM: Sift / membership test

INPUT:
- $(\beta_1, \dots, \beta_m)$ — basis for $\mathcal{G} \subset \mathrm{Sym}(d)$
- $g \in \mathrm{Sym}(d)$

OUTPUT:
- $L = [b_1, \dots, b_m]$ of coset representatives for $\mathcal{G} = \mathcal{G}^{(0)} > \mathcal{G}^{(1)} > \dots > \mathcal{G}^{(m)} = \{1\}$
- $r \in \{\mathrm{Sym}(d) \setminus \mathcal{G}\} \cup \{e\}$ s.t. $g = r \cdot b_m \cdot \dots \cdot b_1$

```
begin
  L = [ ]
  G^0 = G
  r = g
  for i in 1:m
      T = transversal(β_i, G^{i-1})
      δ = β_i^r
      if δ ∉ T
          return L, r          // r ≠ e ; length(L) = i-1
      end
      push b_i to L
      r = r·b_i^{-1}
      if r = e
          return L, r          // length(L) = i
      else
      G^i = Stab_{G^{i-1}}(β_i)
  end
  return L, r               // happens only when g ∉ G
end                         //     and then r ≠ e
                            // note: length(L) = m here.
```

Notes:

- basis, transversals and stabilizers are interconnected, so we will be building them together at the same time as a Stabilizer Chain structure.
- We shouldn't use Schreier generators though: by the time we finish we'll end up with $\mathcal{O}(2^{|S|})$ of them!
- we will usually take β_i = first(T_i) (the first element on the orbit)

A partial stabilizer chain is a sequence

$$\mathcal{C} = \{ G^{(0)} \geq G^{(1)} \geq \dots \geq G^{(n)} = \{id\} \}$$

such that $Stab_{G^{(i-1)}}(\beta_i) \geq G^{(i)}$.

A stabilizer chain (proper, complete) is a similar sequence where $Stab_{G^{(i-1)}}(\beta_i) = G^{(i)}$.

Note: partial stabilizer chain is proper iff $|G| = |\Delta_1| \cdot |G^{(1)}| = |\Delta_1| \cdot |\Delta_2| \cdot |G^{(2)}| = \prod_i |\Delta_i|$

(If we know $|G|$ already that's easy to verify)

Data structures for Stabilizer chain:

```
struct PointStabilizer
    S :: Vector{Permutation}   // the generating set
    x :: Int        // point
    T :: Transversal{...}      // the transversal /orbit
    Stab :: PointStabilizer       of x under S
end
```

```
struct StabilizerChain
    S :: Vector{...}        // vector of generating
                                               sets
    β :: Vector{Int}        // the first pts of orbits
                               (or: the basis)
    T :: Vector{Transversals}  // Transversals: T[i]
end                                is the transversal
                                    of β[i] under
                                              S[i]
```

How to complete a partial stabiliser chain?

Gien a generator of G we sift it through the chain, extending it when necessary.

ALGORITHM : stabilizer-chain

INPUT : • S - a generating set for G

OUTPUT: • $\mathcal{C}$ - a stabilizer chain for G
(complete, proper

```
begin
    C = ...    // initialize the data structure
                  for s.c.
    for g ∈ S
        L, r = sift(C, g)
        if r ≠ id    // g is not contained in C
            push r to C
        end
    end
    return C
end
```

There are two possibilities for the
push r to $\mathcal{C}$
depending on the data structure:

- a recursive push!(ps::PointStabilizer, g::Permutation)
- an iterative
  push!(sc::StabilizerChain, g::Permutation, depth::Int)
  where we need to take care of the dept "manually".

ALGORITHM: push!($\mathcal{C}$, g, d)

INPUT: • $\mathcal{C}$ :: Stabilizerchain a (partial) stabilizerchain
• g – permutation
• d=1 depth (a non-negative integer)

OUTPUT: • $\mathcal{C}$ – (partial) stabilizer chain containing g

```
begin
    assert C.β[i]^g = C.β[i] for all  i < d
    if d > length(C)                    // add new layer
                                        to C
        β = first_moved(g)
        S = [g]
        T = Transversal(β, S)
        extend C by (S, β, T)
        if length(T) < order(g)  // some power of g
                                     stabilizes β
            k = length(T)
            push!(C, g^k, d+1)
        end
    else
        push!(C.S[d], g)
        // since we extended the generator set at
        // level d we need to
        // 1) update the transversal
        C.T[d] = Transversal(C.β[d], C.S[d])
        //   sift any new schreier generator
        //    that arises from g down the chain
        for s in schreier_generators(C.T[d], C.S[d])
            L, r = sift(C, s, ...)  // start sifting at
                                         depth d+1
            if r ≠ id
                push!(C, s, d+1)
            end
        end
    end
    return C
end
```

Algorithm: push!

Input: • $\mathcal{C}$ ::PointStabilizer - a (partial) stabilizer chain
• g - a permutation

Output: • $\mathcal{C}$ - a (partial) stab. chain containing g.

```
begin
    if isempty(C.S)            // are we at the bottom
                               // of the chain?
        β = first_moved(g)     // then we need to extend
                               // it!
        S = [g]
        T = Transversal(β, S)
        initialize C with (β, S, T)
                               ← this makes C.stab.S
                                 empty!

        if length(C.T) < order(g)   // a power of g
                                    // stabilizes C.β
            h = length(C.T)
            push!(C.stab, g^h)
        end
    else
        push g to C.S                   recompute the
        C.T = Transversal(C.β, C.S)     transversal
        for s in schreier_generators(C.T, C.S)
            h, r = sift(C.stab, s)      process the
            if r ≠ id                   new schreier
                push!(C.stab, r)        generators
            end
        end
    end
    return C
end
```

Defn: A <u>strong generating set</u> (sgs) for G is a set S such that $G = \langle S \rangle$ and

$$G^{(i)} = \langle S \cap G^{(i)} \rangle.$$

If $\mathcal{C}$ is a completed stabilizer chain for G, then $S = \bigcup_{i=1}^{d} S_i$ is a sgs.

In the other direction: Given a sgs (and the corresponding basis) we can rebuild the stabilizer chain by simply computing the transversals.

Performance notes:

- There is no need to compute <u>all</u> Schreier generators when recomputing the transversal happens.
- Unfortunate choice for generators may lead to <u>very</u> long T_i's on each level.

Ex: $G = \langle a = (1, \dots, 100),\ b = (1,2) \rangle$

$\beta_1 = 1$, representative $= a^i$, $-50 < i < 50$.

better generating set:

$\langle \qquad\qquad \rangle$

How?

- Expensive operations:
  - permutation multiplication:

    every $a \cdot b$ allocates!

    → store the products as words in generators.

→ If $H < G$ and basis β for G is known we can always store β^g instead of g!

then $g \cdot h$ is $(\beta^g)^h$.

the cost of multiplication:

$$O(\text{degree}(G)) \longrightarrow O(\text{length}(\text{basis})).$$

If $|G|$ is known beforehand (eg. were recomputing the chain) then we could quickly terminate as soon as $\prod_{i=1}^{d} |T_i| = |G|$.
This usually avoids sifting of most of the generators.

Let $\mathcal{G} = \langle \underset{a}{(1,3,5,7)(2,4,6,8)}, \underset{b}{(1,3,8)(4,5,7)} \rangle$

$\beta_1 = 1, \quad S_1 = [a, b]$

$\Delta_1 = [1, 3, 5, 8, 7, 2, 4, 6]$

$T_1 = [e, a, a^2, ab, a^3, aba, a^3b \ a^3ba]$

$s_1 = e \cdot a \cdot \overline{e \cdot a}^{-1} = e$

$s_2 = e \cdot b \cdot \overline{e \cdot b}^{-1} = b \cdot a^{-1} = \underbrace{(2,8,7)(3,6,4)}_{c}$

$\beta_2 = 2, \quad S_2 = [c]$ push!$(e, c, 2)$

$\Delta_2 = [2, 8, 7]$

$T_2 = [e, c \ c^2]$

We'd need to go back and start processing

$s_3 = a \cdot a \cdot \overline{a \cdot a}^{-1} = e$

$s_4 = a \cdot b \cdot \overline{a \cdot b}^{-1} = e$

$\vdots$

Had we knew that $|\mathcal{G}| = 24$, we could have observed:

$$|\Delta_1| \cdot |\Delta_2| = 8 \cdot 3 = 24 = |\mathcal{G}|$$

so the chain is complete and we're done!

Lemma: If $\mathcal{C}$ is a partial stabilizer chain for G then chosen uniformly at random $g \in G$ fails the membership test with $\mathcal{C}$ with probability at least $\frac{1}{2}$.

Corollary:
If elements g are chosen uniformly at random from G, then the probability of n of them passing the membership test with an incomplete chain is at most $\left(1-\frac{1}{2}\right)^n$.

Achievements unlocked by the Schreier-Sims algorithm:

- membership test for G
- compute $|G|$ as $\prod_{i=1}^{d} |T_i|$
- Given $\gamma = (\gamma_1, \dots, \gamma_d)$ find $g \in G$ s.t. $\beta^g = \gamma$.
- Normal closure as the stabilizer of H under the action $(g, H) \mapsto g^{-1} H g$.
- derived series: $D_0 = G$; $D_i = D_{i-1}'$ ← the commutator subgroup
- lower central series: $L_0 = G$; $L_i = [G, L_{i-1}]$
- test whether two elements are in the same coset of a subgroup
- Determine the permutation action on the cosets of a subgroup
- Determine point-wise stabilizer of a set
- enumerate G
- Obtain random elements from G with <u>guaranteed uniform distribution</u>.

Other topics:

Factorisation into generators.

$$g = r_1 \cdot \ldots \cdot r_k = \underbrace{s_{11} s_{12} \cdots s_{1m_1}}_{r_1} \cdot \underbrace{s_{21} \ldots s_{2m_2}}_{r_2} \cdot \ldots \cdot \underbrace{s_{k1} \cdots s_{km_k}}_{r_k}$$

this is usually very far from minimal.

solution: minimize r_i by e.g. flattening the schreier trees.
(but this still will not give you minimality).

Homomorphisms:

If we know (sgs, basis) for $G = \langle S_G \rangle$
have a homomorphism $\varphi : G \to H$
we can quickly evaluate it by

- starting with $\{(s, \varphi(s))\}_{s \in S_G} \subset G \times H$
- doing the computation of sgs in G and mirroring the group operations on H part
- If $g \in G$, $g = r_1 \cdot \ldots \cdot r_k \Rightarrow$ the computations gives us $\varphi(r_1), \ldots, \varphi(r_k)$

If H is a permutation group then
$G \times H$ is also: $\Rightarrow$

$$G \times H \overset{i}{\hookrightarrow} \mathrm{Sym}(\mathrm{degree}(G) + \mathrm{degree}(H)) \curvearrowright \underbrace{i(\Omega_G)}_{\deg(G)} \underbrace{i(\Omega_H)}_{\deg(H)}$$

$\ker \varphi \cong$ pointwise stabilizer of $i(\Omega_H)$.
$(1 : \deg(G)$ - projection$)$.