JavaScriptによる置換パズルとそのソルバーの実装

大阪教育大学附属池田中学校　松川　信彦
Nobuhiko Matsukawa,
IKEDA JUNIOR HIGH SCHOOL ATTACHED TO OSAKA KYOIKU UNIVERSITY

1　はじめに

著者は GAP や magma など既存の計算群論ソフトを利用せず, 計算群論の基礎的なアルゴリズム (Schreier-Sims, coset enumeration など) を JavaScript のみのフルスクラッチで実装し, 群論の実例計算や, その応用として rubik's cube やスライドパズルのソルバーなどの可視化アプリケーションの作成などを行ってきた.

今回は, 著者のサイトにおいて, 置換パズルを数学的に定式化し, その理論的背景が明確になるよう, 計算群論の実例を実験で試せるようにしたことと, いくつかの置換パズルとそのソルバーを実装したことについて紹介する.

置換パズルは, rubik's cube や 15 パズルなどが有名であるが, 今回は html canvas を利用した平面置換パズルに限定して実装を行った. 具体的には 24 puzzle や top spin puzzle, hungarian rings など, 有名な市販パズルだけではなく, 著者が考案したパズルである, wreath type puzzle や merge type puzzle などである. これらは PC におけるマウス操作や, モバイルデバイスにおける指での操作に対応している.

ランダムな操作を複数回 (数万回) 作用させた状態を, アニメーションを経由せずに発生させることができる機能と, 本研究の目玉であるが, 計算群論的な手法による, 任意に与えられた状態からの解答手順を導き出し, それをアニメーションで表示するなどの機能を実装した.

最初に置換パズルの数学的定式化について述べる.

2　置換パズルの数学的定式化

Ω, C を有限集合とする. ($|\Omega| < \infty,\ |C| < \infty$) 有限集合 Ω にカラーパレット C の色を塗った状態全体は

$$\mathrm{Map}(\Omega, C)$$

であると解釈できる. $\mathfrak{S}_\Omega$ を Ω 上の対称群とする. $\mathfrak{S}_\Omega$ の $\mathrm{Map}(\Omega, C)$ への作用は

$$a^\sigma(\omega^\sigma) := a(\omega)$$

($\forall a \in \mathrm{Map}(\Omega, C)\ \forall \sigma \in \mathfrak{S}_\Omega\ \forall \omega \in \Omega$) と定義する.

これは場所 ω にある着色された石を 場所 ω^σ に移動させていることと解釈できる. $\mathrm{Map}(\Omega, C)$ を 状態集合 *(set of states)* といい, その元を 状態 *(state)* と呼ぶことにする.

$$a^\sigma(\omega) = a^\sigma(\omega^{\sigma^{-1}\sigma}) = a(\omega^{\sigma^{-1}})$$

であり, 従って

$$(a^\sigma)^\tau(\omega) = a^\sigma(\omega^{\tau^{-1}}) = a((\omega^{\tau^{-1}})^{\sigma^{-1}}) = a(\omega^{(\sigma\tau)^{-1}}) = a^{\sigma\tau}(\omega)$$

だから

$$(a^\sigma)^\tau = a^{\sigma\tau}$$

となり, $\mathfrak{S}_\Omega$ が $\mathrm{Map}(\Omega, C)$ へ右から作用していることが分かる. 置換パズル *(a permutation puzzle)* とは組

$$(\sigma_1, \cdots, \sigma_m, \iota) \in \mathfrak{S}_\Omega^m \times \mathrm{Map}(\Omega, C)$$

を 1 つ固定し, $\sigma_1, \cdots, \sigma_m$ によって生成された群を $G := \langle \sigma_1, \cdots, \sigma_m \rangle < \mathfrak{S}_\Omega$ とするとき ι を含む G-軌道

$$\iota^G$$

の任意の元 a について, $a = \iota^\sigma$ となる σ に対し, 分解

$$\sigma = \sigma_{i_i}^{\varepsilon_1} \cdots \sigma_{i_k}^{\varepsilon_k}$$

$(i_1, \cdots, i_k \in \{1, \cdots, m\}, \varepsilon_1, \cdots, \varepsilon_m \in \{-1, 1\})$ を見つけることである. 実際

$$\iota = (\iota^\sigma)^{\sigma^{-1}} = a^{\sigma^{-1}} = \left(\cdots \left(a^{\sigma_{i_k}^{-\varepsilon_k}} \right)^{\sigma_{i_{k-1}}^{-\varepsilon_{k-1}}} \cdots \right)^{\sigma_{i_1}^{-\varepsilon_1}}$$

は群 G の元の生成元 (およびその逆元) を a に次々に作用させて初期状態 ι に戻す作業であり, k がその手数を表している.

$\sigma \neq \tau$ であったとしても $\iota^\sigma = \iota^\tau$ となる例はある. 例えば $4 \times 4 \times 4$ なるルービックキューブでそのような例を作ることができる. このような場合を考慮したくないので, 以下では

$$C = \Omega,\ \iota = id_\Omega$$

の場合のみ考える. この場合, 任意の状態 $a \in id_\Omega^G$ に対し $a = id_\Omega^\sigma$ とすると,

$$a(\omega) = (id_\Omega^\sigma)(\omega) = id_\Omega(\omega^{\sigma^{-1}}) = \omega^{\sigma^{-1}}$$

となるから, 状態を置換とみなすとき, その逆元が σ であるとみなせる.

従って, 置換群 G の元 (の逆元) が置換パズルの状態の全てを表していることになるが, 一般的にその位数は非常に大きくなり, その全てを列挙するのは明らかに不合理である. 例えば top spin puzzle は対称群 $\mathfrak{S}_{20}$ と同型であるから,

$$20! = 2432902008176640000$$

となる. 実は置換群においても, ベクトル空間の基底に相当するものが存在し, それが *BSGS (base and strong generating set)* と呼ばれているものである. 任意の G の元は, SGS (strong generating set) の元の積で表すことができ, Minkwitz 分解によって, SGS の元は生成系 $\{\sigma_1, \cdots, \sigma_m\}$ の元およびその逆元の積によって表すことができるので, これらが構成できれば置換パズルのソルバーが完成したと言える. 以下では BSGS およびその Minkwitz 分解について述べることを目標として, 計算群論の基礎から始めることとする.

3 計算群論

計算群論の基礎中の基礎であり, 理論の根幹ともいえる, Schreier の補題について述べる. 群 G とその部分群 H について, $H\backslash G$ を H の G における右剰余類全体とする. 即ち

$$H\backslash G = \{H\sigma \mid \sigma \in G\}$$

さらに $T \subseteq G$ を $1 \in G$ であり, $T \to H\backslash G$ $(t \mapsto Ht)$ により全単射対応となるような G の部分集合とする. T は H の G における右剰余類の完全代表系であるという. 任意の $\sigma \in G$ に対し, $\overline{\sigma} \in T$ を $H\sigma = H\overline{\sigma}$ となるような元であると定義する. このとき

定理 3.1 (Schreier の補題)**.**

$$H = \langle tx\overline{tx}^{-1} \mid t \in T,\ x \in X \rangle$$

Proof. 任意の $\sigma \in G$ に対し, $H\sigma\overline{\sigma}^{-1} = H\overline{\sigma}\ \overline{\sigma}^{-1} = H$ より, $\sigma\overline{\sigma}^{-1} \in H$.

任意の $h \in H$ に対し, $h = {x_1}^{\varepsilon_1} \cdots {x_k}^{\varepsilon_k}$ と表せる. ただし $x_1, \cdots, x_k \in X$, $\varepsilon_1, \cdots, \varepsilon_k \in \{-1, 1\}$.

$$t_i := \begin{cases} e & (i = 0), \\ \overline{t_{i-1}{x_i}^{\varepsilon_i}} = \overline{{x_1}^{\varepsilon_1} \cdots {x_i}^{\varepsilon_i}} & (i \geq 1) \end{cases}$$

とおくと,

$$h = (t_0{x_1}^{\varepsilon_1}\overline{t_0{x_1}^{\varepsilon_1}}^{-1}) \cdots (t_{k-1}{x_k}^{\varepsilon_k}\overline{t_{k-1}{x_k}^{\varepsilon_k}}^{-1})t_k$$

となるが, $t_k = \overline{h} = e$ となるので,

$$H = \langle tx^{\varepsilon}\overline{tx^{\varepsilon}}^{-1} \mid t \in T,\ x \in X,\ \varepsilon \in \{-1, 1\} \rangle$$

また, $s := \overline{tx^{-1}}$ とおくとき, $Hsx = H\overline{tx^{-1}}x = Ht$ より, $\overline{sx} = t$ となるので,

$$tx^{-1}\overline{tx^{-1}}^{-1} = (\overline{tx^{-1}}xt^{-1})^{-1} = (sx\overline{sx}^{-1})^{-1}$$

だから

$$H = \langle tx\overline{tx}^{-1} \mid t \in T,\ x \in X \rangle$$

□

3.1 BSGS

以下では $\Omega := \{1, 2, \cdots, n\}$ とし, G は $X \subseteq \mathfrak{S}_\Omega$ により生成される部分群とする. $\alpha_1, \cdots, \alpha_l \in \Omega$ に対し, その固定部分群を $G_{\alpha_1,\cdots,\alpha_l}$ と表す. すなわち

$$G_{\alpha_1,\cdots,\alpha_l} = \{\sigma \in G \mid {\alpha_i}^\sigma = \alpha_i \ \forall i \in \{1, \cdots, l\}\}$$

とおく. $B = (\beta_1, \cdots, \beta_k) \in \Omega^{\times k}$ が base であるとは,

$$G_{\beta_1,\cdots\beta_k} = \{e\}$$

を満たすものと定義する. 特に $(1, \cdots, n)$ も base である. base B に対し, $G^{(i)} := G_{\beta_1\cdots,\beta_i}$ とおくとき,

$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(k)} = \{e\}$$

を stabilizer chain であるという. base B が重複なしであるとは, $G^{(i-1)} \neq G^{(i)}$ が全ての $i \in \{1, \cdots, k\}$ について成り立つことをいう. 任意の $i \in \{1, \cdots, k\}$ に対し, $G^{(i)}$ の $G^{(i-1)}$ における右剰余類の完全代表系を $U^{(i)} (\subseteq G^{(i-1)})$ とおく. $U^{(i)}$ の取り方は同一の base B に対し色々あるが, 軌道

$$\Delta^{(i)} := {\beta_i}^{G^{(i-1)}} \simeq G^{(i)} \backslash G^{(i-1)}$$

は base B に対し一意に取れる. 著者の実装は全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ $(i \in \{1, \cdots, k\})$ で任意の $\beta \in \Delta^{(i)}$ に対し, $\beta = {\beta_i}^{u_i(\beta)}$ が成り立つものを何か一組構成するアルゴリズムとなっている. base $B = (\beta_1, \cdots, \beta_k)$ と u_i たちを利用すれば, 任意の $\sigma \in \mathfrak{S}_n$ に対し, σ が G の元であるかどうかを判定し, もしそうであるならば $U^{(i)}$ の元の積に一意的に表示することができる. 具体的に pseudocode で示すと, 以下のようになる.

```
function factoring(σ ∈ 𝔖_Ω){
  var v=[], τ_0 = σ;
  for(var i=1; i≤k; i++){
    if(β_i^{τ_{i-1}} ∈ Δ^(i)){
      var σ_i = u_i(β_i^{τ_{i-1}}) ∈ U^(i);
      var τ_i = τ_{i-1}σ_i^{-1} ∈ G^(i); (∴ τ_i = σσ_1^{-1}⋯σ_i^{-1})
      v[i] = σ_i;
    }else{
      v=[], break;
    }
  }
  return v; (∴ σ = σ_k⋯σ_1)
}
```

特に, 集合として

$$G^{(i)} = U^{(k)} \times U^{(k-1)} \times \cdots \times U^{(i+1)} \quad (\forall i \in \{0, 1, \cdots, k-1\})$$

である.

$$G = U^{(k)} \times U^{(k-1)} \times \cdots \times U^{(1)} \simeq \Delta^{(k)} \times \Delta^{(k-1)} \times \cdots \times \Delta^{(1)}$$

なので, G の位数が $\prod_{i=1}^{k} |\Delta^{(i)}|$ となる.

$S \subseteq G$ が strong generating set (以下 SGS) であるとは,

$$\langle S \cap G^{(i)} \rangle = G^{(i)} \quad (\forall i \in \{0, 1, \cdots k-1\})$$

が成り立つことをいう. base B とその SGS S との 組 (B, S) を BSGS と呼ぶ. BSGS (B, S) が構成されることは, 或る全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ で $\beta = \beta_i^{u_i(\beta)}$ $(\forall \beta \in \Delta^{(i)})$ を満たすものが構成でき, $S \subseteq \bigsqcup_{i=1}^{k} U^{(i)}$ となり, $S^{(i)} = S \cap (U^{(i+1)} \sqcup \cdots \sqcup U^{(k)})$ とおくとき,

$$G^{(i)} = \langle S^{(i)} \rangle \quad (\forall i \in \{0, 1, \cdots, k-1\})$$

が成り立つことと同値である.

3.2 Jerrum's filter

現時点では実際に扱う例は $|\Omega|$ が 100 を超えることも少ないため, 各右剰余類の完全代表系 $U^{(i)}$ の大きさも小さい. そのため, SGS として

$$S = U^{(1)} \sqcup \cdots \sqcup U^{(k)}$$

を求める実装にしている. また, $B = (1, \cdots, n)$ を取って計算することが多い. X が対称群全体を生成しなければ, $G^{(k)} = \cdots = G^{(n)} = \{e\}$ となったり, $G^{(i)} = G^{(i+1)}$ ($\Leftrightarrow$ $\Delta^{(i)} = \{i\}$) となることがあるのだが, このような i は計算終了後に排除すればよい. base B としては, 増加列 $1 \leq \beta_1 < \cdots < \beta_k \leq n$ で $G = G^{(0)} > G^{(1)} > \cdots > G^{(k)} = \{e\}$ となる.

具体的な実装は次のような $i = 1, \cdots, k$ に関するループである. base $B = (\beta_1, \cdots, \beta_k)$ に対し, $G^{(i)} = G_{\beta_1, \cdots, \beta_i}$ の生成系 $X^{(i)}$ が構成されたと仮定する. ただし $X^{(0)} = X$ とする. $U^{(i)}$ は, β_i に $S^{(i)}$ の元を次々に作用させ, 軌道 $\Delta^{(i)}$ を張るとき, それと同時に作用させた元たちを掛けていったものを記録していけば構成できる. 従って, 全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ が構成されたことになる. 次に $G^{(i)}$ の構成の仕方について述べる. Schreier の補題を使って $G^{(i)}$ の生成系は

$$X^{(i)} = \{u_i(\beta)\sigma u_i(\beta^\sigma)^{-1} \mid \beta \in \Delta^i,\ \sigma \in X^{(i-1)}\}$$

となることが分かる. これは $\beta_i^{u_i(\beta)\sigma} = \beta^\sigma = \beta_i^{u_i(\beta^\sigma)}$ より $\overline{u_i(\beta)\sigma} = u_i(\beta^\sigma)$ であることよりいえる. 単純にこのまま実装すれば, $|X^{(i)}| = |X^{(i-1)}|\ |\Delta^{(i)}|$ より

$$|X^{(i)}| = |X^{(0)}|\ |\Delta^{(1)}|\ \cdots\ |\Delta^{(i)}|$$

となり, i が増加するに従って, この生成系の大きさは爆発的に大きくなる. 例えば JavaScript では Mathieu 群 M_{12} のような小さな群ならば機能するが, Mathieu 群 M_{24} ではオーバーフローを起こしてしまい使い物にならない. 実は次のよく知られた事実があり, この問題は解決される.

定理 3.2 (Jerrum's filter)**.** 対称群 $\mathfrak{S}_n$ の任意の部分群 G に対し,

$$G = \langle S \rangle, \quad |S| < n$$

となるような部分集合 $S \subseteq G$ を構成するアルゴリズムが存在する.

Proof. Cameron [2] の証明を (多少厳密化して) 紹介する.
任意の $\sigma \in \mathfrak{S}_n$ に対し, $i(\sigma) = \min\{i \mid i^\sigma \neq i\}$,

$$e(\sigma) := \{i(\sigma), i(\sigma)^\sigma\}$$

と定義する. 任意の部分集合 $S \subseteq \mathfrak{S}_n$ に対し, $\Gamma(S) := (\{1, \cdots, n\}, e(S))$ とグラフを定義する.

$$\begin{array}{l} e : S \to e(S) \text{ が単射であり, かつ} \\ \Gamma(S) \text{ が閉路を持たない (acyclic であるという)} \end{array}$$

ときに $|S| < n$ となることは明らか. この条件を $P(S)$ とおく.

$$\Lambda_n := \{T \subseteq \mathfrak{S}_\Omega \mid \neg P(T) \ \wedge \ P(T \setminus \{\sigma\}) \ (\exists \sigma \in T)\}$$

とおき, 任意の $T \in \Lambda_n$ に対し,

$$m(T) := \sum_{\sigma \in T} i(\sigma)$$

と定義すると, $|T| \leq n$ であるから, $m(T) < n^2$ となる. 従って, 任意の $T \in \Lambda_n$ に対し

$$\langle U \rangle = \langle T \rangle \ \wedge \ [P(U) \ \vee \ [U \in \Lambda_n \ \wedge \ m(T) < m(U)]]$$

なる $U \subseteq \mathfrak{S}_n$ が構成できればよい. $P(T \setminus \{\sigma\})$ なる $\sigma \in T$ をとる. $e : T \to e(T)$ が単射でないとき, $\tau \in T \setminus \{\sigma\}$ が存在し, $e(\sigma) = e(\tau)$ となるから, $U := (T \setminus \{\sigma\}) \cup \{\sigma\tau^{-1}\}$ とおくと, 明らかに $\langle U \rangle = \langle T \rangle$. また $\neg P(U)$ のとき, $P(U \setminus \{\sigma\tau^{-1}\})$ より $U \in \Lambda_n$ であり,

$$m(U) - m(T) = i(\sigma\tau^{-1}) - i(\sigma) > 0$$

より $m(T) < m(U)$. 次に $e : T \to e(T)$ が単射かつ, T が cycle をもつとき, $e(\sigma)$ を含む閉路が存在する. この閉路を,

$$i_1, \ \cdots, \ i_k, \ i_1$$

とし, $i_1 = \min\{i_s \mid s = 1, \cdots, k\}$ としてよい. $e(\sigma_s) = \{i_s, i_{s+1}\}$ $(s = 1, \cdots, k)$ (ただし $i_{k+1} = i_1$) とし,

$$\varepsilon_s := \begin{cases} 1 & (i_s < i_{s+1}) \\ -1 & (i_s > i_{s+1}) \end{cases}$$

とおくとき, $\rho := \sigma_1^{\varepsilon_1} \cdots \sigma_k^{\varepsilon_k}$ について $i(\rho) > i_1 = i(\sigma_1)$. $U := (T \setminus \{\sigma_1\}) \cup \{\rho\}$ とおくとき, $\langle U \rangle = \langle T \rangle$. $\neg P(U)$ ならば $P(U \setminus \{\rho\})$ だから $U \in \Lambda_n$ であり,

$$m(U) - m(T) = i(\rho) - i(\sigma_1) > 0$$

より $m(T) < m(U)$. □

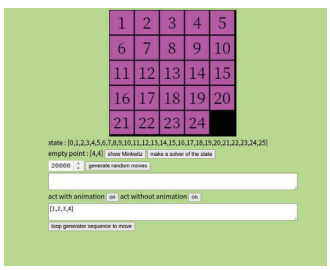
4 Minkwitz 分解アルゴリズム

Minkwitz 分解とは, 構成された置換群 G の SGS の元を G の生成系 X の元およびその逆元の積で表すアルゴリズムの一つである. Minkwitz の論文 [6] を参考に実装した. Minkwitz の論文は 7 ページと短いが, 著者にとって難解であったため web page [5] を参考に試行錯誤を重ね, とりあえず動くものができた. 著者のサイト [7] において, 下の図のように Mathieu 群 (M_{11}, M_{12}, M_{22}, M_{23}, M_{24}) などの置換群に対して試すことができる. また, 今回作成した html canvas puzzle の BSGS およびその Minkwitz 分解のデータを表示するサイト [8] において, ランダムに与えられ元に対し, それを Minkwitz 分解する実験を試すことができる.

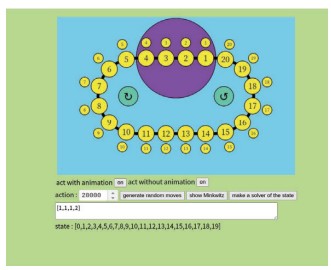
5 html canvas での実装

今回 html canvas を用いた puzzle game を実装したのでその紹介をする. 24 puzzle や top spin puzzle, hungarian rings などの既存のスライドパズルの他, wreath type puzzle や merge type puzzle など, 著者オリジナルのスライドパズルなどを著者のサイト [10] において公開している. これらについて共通しているのは, マウスやタブレット PC などにおけるタッチ操作で, ボタンによりパズルをスライドできることと, 以下にある html のボタンの機能である.

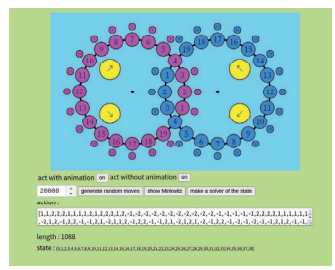
generate random moves	ランダムな操作を決められた回数だけ発生させる
act with animation	操作配列を animation つきでパズルに作用させる
act without animation	操作配列を animation なしでパズルに作用させる
make a solver of the state	ランダムな状態のパズルに対し, その解である操作配列を生成する



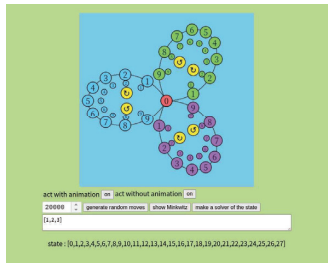
24-puzzle



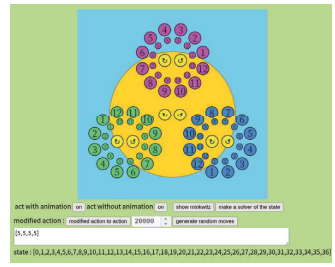
top spin puzzle



hungarian rings



wreath type puzzle



merge type puzzle

6　まとめ, 目標

置換パズルの群論的な solver は予め計算された BSGS とその Minkwitz 分解を利用して実行する. 任意の元を与えられた SGS の積で表示する実装は容易である上, JavaScript であっても非常に高速であり低コストである. それに対し BSGS を導出するコストは高く, 著者の JavaScript での実装において, $3 \times 3 \times 3$ の rubik's cube 群の BSGS を導出するには 4 分もかかる. BSGS やその Minkwitz 分解の計算は GAP や magma など高速かつ高性能なソフトに任せ, 出来上がった配列を JavaScript に書いて利用する方が, より効率的で洗練された解法を提示することが可能であると思われるが, この道を選ばない理由は, 著者が計算群論のより深い理解を動機としていることにある. web アプリとしての実装をより高速で洗練されたものにするためには rust と webassembly の利用は不可避なので, rust で基礎から計算群論プログラムを実装していかなくてはならない.

スライドパズルのシステムの雛形は一応完成したので, これを利用した思いつく限りのスライドパズルを作成したい. 既存のスライドパズルでは実現できないようなパズル (例えば Moebius strip や torus のスライドパズル, 正多面体の rubik's like puzzle を球面に膨らませ, それを平面に立体射影したスライドパズル など) を作成し, 公開していきたい.

参考文献

[1] Gregory Butler, Fundamental Algorithms for Permutation Groups (Lecture Notes in Computer Science), Springer, 1991.

[2] P. J. Cameron, Permutation groups (London Mathematical Society Student Texts, vol. 45, Cambridge University Press), Cambridge, 1999.

[3] 藤本光史, Computational Group Theory への招待, 数理解析研究所講究録 941 巻, 1996 年, 73-83.

[4] D. L. Johnson, Topics in the Theory of Group Presentations, (London Mathematical Society Lecture Note Series, vol. 42, Cambridge University Press), Cambridge, 1980.

[5] Mathematics_and_Such,
https://mathstrek.blog/2018/06/21/solving-permutation-based-puzzles/

[6] T. Minkwitz, An Algorithm for Solving the Factorization Problem in Permutation Groups, J. Symbolic Computation (1998) 26, 89–95

[7] Minkwitz 分解の実装,
https://noblegarden-math.jp/math/notes/Permutation_Puzzles/MinkwitzFactorization/

[8] 与えられた BSGS データを Minkwitz 分解する実装, https://noblegarden-math.jp/math/notes/Permutation_Puzzles/Minkwitz/

[9] Scott H. Murray, The Schreier-Sims algorithm, 2003,

[10] permutation puzzles 実装, https://noblegarden-math.jp/math/notes/Permutation_Puzzles/