Eötvös Loránd Tudományegyetem
Természettudományi Kar

Nagy Gergely Gábor

Csoportelméleti algoritmusok

Szakdolgozat
Matematika BSc

Témavezető: Pelikán József, egyetemi adjunktus
Algebra és Számelmélet Tanszék

Budapest, 2012.

Tartalomjegyzék

Bevezetés

Az algoritmikus csoportelmélet a matematikának egy olyan ága, amivel a XX. század eleje óta foglalkoznak, de csak a '60-'70-es években kezdett el nagyon fejlődni. A szakdolgozat kitűzött célja az, hogy véges csoportoknak számítógéppel tudjuk kiszámolni a karaktertáblázatát. Ez egy elég összetett feladat ahhoz, hogy sok más csoportelméleti algoritmus is kelljen hozzá, ezekről fogunk beszélni.

A legtöbb csoportelméleti algoritmus a csoport számítógépes megadásától függ, ugyanis a csoportot megadhatjuk permutációcsoportként vagy mátrixcsoportként; policiklikus reprezentációval vagy véges generátorrendszerrel és relációkkal is. Vannak azonban olyan algoritmusok is, amelyek nem függnek a megadás módszerétől, itt a csoportra, mint egy fekete dobozra tekintünk, aminek nem ismerjük a szerkezetét. Innen jön az ilyen csoportokra a black-box elnevezés. Ezekre vonatkozó algoritmusokról szól az 1. fejezet.

A megadási módszerek közül a permutációcsoportokat választottam ki, hatékony használatukról és hozzájuk kapcsolódó algoritmusokról szól a 2. fejezet.

Karaktertáblázatok kiszámolására jelenleg kettő, egymástól teljesen eltérő ötletre alapuló algoritmus ismert. Ezek közül amiről mi beszélünk, az a Burnside-Dixon-Schneider algoritmus, ami egy lineáris algebrai megközelítést alkalmaz, erről szól a 3. fejezet. A másik algoritmusról nem fogunk részletesen beszélni, itt szeretnék pár mondatot írni róla. 1986-ban Slattery [Sla86] publikált egy eljárást, ami p-csoportokra meg tudja mondani, hogy hányadrendű irreducibilis reprezentációból hány darab van. Conlon 1990-ben [Con90] ezt az ötletet továbbfejlesztve ki tudta számolni p-csoportokra a karaktertáblázatot. A minden csoportra alkalmazható algoritmus Unger-nek köszönhető 2006-ból [Ung06], az ötlet Brauer tételén alapszik, miszerint az általánosított karakterek gyűrűjét generálják az úgynevezett elemi részcsoportok irreducibilis karaktereiből indukált karakterek. Elemi részcsoportoknak azokat a részcsoportokat hívjuk, amelyek előállnak egy p-csoport és egy ciklikus csoport direktszorzataként, ezeknek tehát Conlon algoritmusának segítségével ki tudjuk számolni az irreducibilis karaktereit. Miután az algoritmus kiszámolta a gyűrű egy generátorrendszerét, LLL-redukció [LLL82] segítségével megkeresi az 1-normájú karaktereket, ezek lesznek az irreducibilis karakterek.

A szakdolgozat függelékében megtalálható az algoritmusok nagyrészének implementációja Mathematica 8 környezetben. Azért a Mathematica-ra esett a választásom, mivel az az egyik legnagyobb, legjobban megírt általános célú matematikai programcsomag, de az algebrai alkalmazások terén azonban nagyon elmaradott. A 2010-ben kiadott 8-as verzió volt az első, amibe már valamennyi csoportelméleti

funkciót beleraktak, azonban csak permutációcsoportokat tud kezelni, és azokra is csak nagyon kevés algoritmus van benne. Az implementáció a meglevő függvényeket már felhasználja, így régebbi verziókkal nem kompatibilis. Mivel publikussá szeretném tenni a programomat, ezért annak a dokumentációja, valamint a forráskódbeli megjegyzések angol nyelvűek. A részletezett algoritmusokra példák a dokumentációban találhatóak.

A szakdolgozat előismeretnek tekinti a Matematika BSc négy félévnyi Algebra képzésének anyagát, valamint általában algoritmusokkal kapcsolatos jártasságot. Akinek a téma felkeltette az érdeklődését az algoritmikus csoportelmélet iránt, az részletesebben olvashat erről [HEO05]-ben, valamint kifejezetten permutációcsoport-algoritmusokról [Ser03]-ban.

Ezúton szeretnék köszönetet mondani témavezetőmnek, Pelikán Józsefnek, aki a négy félév Algebra előadásaival és gyakorlataival felkeltette érdeklődésemet a téma iránt, valamint szakirodalom ajánlásával és értékes észrevételeivel nagyban segítette a szakdolgozat elkészítését.

1. Black-box csoportok

A black-box csoportok eredeti ötlete és definíciója magyar matematikusoktól származik, Babai László és Szemerédi Endre publikálta [BS84]-ben. Ma a legtöbb helyen inkább kicsit általánosabb definíciókat használnak, amivel kevésbé körülményes az egyes csoportok tényleges leírása. Az itt leírt definíció megegyezik a [Ser03]-ban olvashatóval.

A black-box csoport egy olyan csoport, aminek elemeit a véges Σ ábécé feletti legfeljebb N hosszú szavakkal azonosítunk. Nincs megkövetelve, hogy egy elemnek csak egy szó felelhessen meg, se az hogy minden szó hozzátartozzon egy elemhez. A csoportműveleteket egy orákulum végzi. Ha adott két szó, amik a $g, h \in G$ elemeket reprezentálják, akkor meg tudjuk állapítani, hogy $g = 1$ igaz-e, valamint ki tudjuk számolni a g^{-1}-hez és a gh-hoz tartozó szavakat. Általában a csoport megadása egy generátorrendszer segítségével történik, az elemeihez tartozó szavak megadásával.

Előfordulhat, hogy az orákulum egy $\overline{G} \geq G$ nagyobb csoport elemeihez tartozó szavakat fogad el, ilyenkor csak azt tesszük fel, hogy azt tudjuk megállapítani, hogy a szó $\overline{G}$-beli-e, azt nem, hogy G vagy $\overline{G} \setminus G$ eleme.

Erre példa egy véges F test feletti G $n \times n$-es mátrixcsoport. $\Sigma = F$ adja az ábécét, a szavak n^2 hosszúak, valamint $\overline{G}$ az összes invertálható F feletti $n \times n$-es mátrix csoportja. Természetesen a permutációcsoportok, illetve a policiklikus csoportok is könnyen leírhatók black-box csoportokként.

A black-box csoportokra vonatkozó algoritmusoknak egyik leggyakoribb típusa az általunk később is sokat használt orbit-számoló algoritmusok, amik közül párat az 1.1. alfejezetben részletezünk. A másik általunk itt leírt algoritmus konjugáltosztályokra tudja bontani a csoportot, ami a reprezentációelméletben nagyon fontos, erről szól az 1.2. alfejezet.

1.1. Orbit-algoritmusok

Gyakori eset, hogy G hatását nézzük egy Ω halmazon, és ott egy $\alpha \in \Omega$ elem orbitját, vagyis $\alpha^G = \{\alpha^g \mid g \in G\}$-t keressük. Ilyen előfordulhat, ha $G \leq \mathrm{Sym}(\Omega)$, de például az is ide tartozik, ha $g \in G$ konjugáltosztályát szeretnénk meghatározni, ugyanis választhatjuk $\Omega = G$-t és hatásnak a konjugálást tekinthetjük.

Feltesszük, hogy adott $\beta \in \Omega$ és $g \in G$-re meg tudjuk határozni a β^g képet, valamint Ω-beli elemeket össze tudunk hasonlítani egymással. Lehetséges, hogy Ω nagyon nagy, így nem tesszük fel, hogy fel tudjuk sorolni az elemeit.

Legyen $G = \langle S \rangle$. α^G az a legszűkebb részhalmaza Ω-nak, ami tartalmazza α-t,

valamint zárt az S-beli csoportelemek hatására nézve. Vegyük azt az irányított gráfot, aminek csúcsai Ω elemei és β-ból γ-ba akkor megy él, ha van olyan S-beli elem, ami β-t γ-ba viszi. Ebben a gráfban kell megkeresnünk az α-t tartalmazó összefüggő komponenst. Mivel G véges rendű, így $\beta^g = \gamma$ esetén létezik l, amire $\gamma^{g^l} = \beta$, vagyis az összefüggő komponensek erősen összefüggők. A komponenst meg tudjuk találni szélességi kereséssel $\mathcal{O}(|S||\alpha^G|)$ idő alatt, $\mathcal{O}(|\alpha^G|)$ memória használatával.

Most nézzünk egy általánosabb verziót. Legyen adott Ω-n egy algebrai struktúra, és tegyük fel, hogy G hatása olyan, hogy az minden $*$ Ω-beli műveletre nézve disztributív, vagyis $(\alpha * \beta)^g = \alpha^g * \beta^g$. $A \subseteq \Omega$-ra szeretnénk kiszámolni $\langle A^G \rangle$-t, vagyis Ω-nak azt a legszűkebb részhalmazát, ami tartalmazza A minden elemét és zárt G hatására és az Ω-beli művelet(ek)re is. Ha az előző konjugálós példát nézzük úgy, hogy Ω-n is nézzük a csoportműveletet, akkor ezzel A normális lezártját szeretnénk meghatározni. Ha $G' = [G, G]$ kommutátor részcsoportot szeretnénk meghatározni, akkor azt így tudjuk megtenni, hiszen $G' = \langle [s, t] \mid s, t \in S \rangle_N$, vagyis az S-beli elemek kommutátorainak normális lezártja. Ezzel tudunk feloldhatóságot valamint nilpotenciát is ellenőrizni.

Mivel a végeredmény zárt az Ω-beli művelet(ek)re, így célszerű az algoritmusnak csak egy generátorrendszert előállítani. Feltesszük, hogy tudjuk ellenőrizni, hogy Ω egy eleme benne van-e pár Ω-beli elem által generált struktúrában. Az algoritmus az előzőnek egy egyszerű változtatásával kapható, egyrészt az elején a szélességi bejárást A minden eleméből egyszerre kell indítani, másrészt ha egy új csúcshoz érünk, akkor csak akkor kell felírnunk a generátorelemek listájába, ha az addigiak által generált struktúrában nincs benne. Érdemes itt megjegyeznünk, hogy ha Ω-n legalább egy csoportstruktúra van, akkor minden új generátorelem a csoport elemszámát legalább megduplázza, vagyis legfeljebb $\mathcal{O}(\log |\langle A^G \rangle|)$ generátorelemet írunk fel. A számítási idő nagyrésze jellemzően arra a számolásra megy el, amikor egy elemről megpróbáljuk eldönteni, hogy benne van-e a már felírt elemek által generált struktúrában. Vannak olyan randomizált algoritmusok is erre a célra, amikhez nem szükséges feltennünk, hogy ezt el tudjuk dönteni, ezekről [Ser03]-ban olvashatunk.

1.2. Konjugáltosztályok megtalálása

Tegyük fel, hogy a $g, h \in G$ csoportelemekről valahogy el tudjuk dönteni, hogy konjugáltak-e, meg tudjuk határozni $C_G(g)$-t, vagyis a g elem centralizátorát, valamint, hogy tudunk részcsoportokban (közel) egyenletes eloszlású véletlen elemet kiválasztani. Ezeket permutációcsoportoknál meg fogjuk mutatni, hogyan tudjuk megcsinálni. Feladatunk, hogy megtaláljuk a konjugáltosztályokat a csoportban, vagyis

az osztályoknak megtaláljuk egy-egy reprezentáns elemét. Bár sok mindent felhasználunk, amit csak speciális csoportoknál tudunk megtenni, mégis az osztályok megtalálása minden csoport esetén ugyanazzal az algoritmussal történhet, azért írom a black-box csoportok fejezetében.

A legegyszerűbb algoritmus annyiból áll, hogy sorban választunk véletlenszerű elemeket a csoportból, megnézzük, hogy az új elem konjugált-e az eddig megtalált reprezentáns elemek valamelyikével, és ha nem, akkor bevesszük a listába. Minden megtalált reprezentáns elemnek kiszámolhatjuk a konjugáltosztályának a méretét, ugyanis az a centralizátorának indexével egyenlő. Akkor fejezzük be az algoritmust, amikor az osztályok méretének összege eléri a csoport rendjét. Ennek az algoritmusnak a nagy hátránya, hogy ha egy nagy csoportban van egy kis elemszámú konjugáltosztály, akkor azt csak nagyon nehezen tudjuk megtalálni.

Másik véletlenszerű elemválasztáson alapuló algoritmust talált ki Mark Jerrum 1995-ben [Jer95]. Konstruáljunk egy M Markov-láncot a következőképpen. Az állapotok halmaza legyen G, a P átmeneti valószínűségek mátrixának $g, h \in G$-hez tartozó eleme pedig

$$p_{g,h} = \begin{cases} 1/|C_G(g)| & \text{ha } h \in C_G(g) \\ 0 & \text{különben} \end{cases}$$

G elemein nézzünk egy véletlen sétát M szerint. Először választunk egy $x_0 \in G$ kezdőállapotot, majd miután $(x_0, \ldots, x_k)$ már definiált, legyen x_{k+1} véletlen elem $C_G(x_k)$-ból. Mivel minden $g, h \in G$-re $1 \in C_G(g)$ és $h \in C_G(1)$, ezért akárhonnan akárhova legfeljebb két lépéssel eljuthatunk, tehát M irreducibilis. Minden g-re $p_{g,g} = 1/|C_G(g)| > 0$, tehát minden állapot aperiodikus. Így alkalmazhatjuk a Markov-láncok alaptételét, miszerint létezik (és csak egy létezik) stacionárius eloszlás ($v = (v_g \mid g \in G)$) és $\delta \in (0,1)$, amire minden $k \geq 0$-ra és $g \in G$-re $|\operatorname{Prob}(x_k = g) - v_g| \leq \delta^k$. Jelöljük r-rel a konjugált osztályok számát, és legyen $v_g = C_G(g)/(r|G|)$. Könnyen látható, hogy $\sum_{g \in G} v_g = 1$, valamint hogy $v = vP$, tehát ez a v a stacionárius eloszlás. Ha akármelyik konjugáltosztályra összeadjuk az abba tartozó g-kre v_g-ket, akkor $1/r$-et kapunk, tehát elég nagy m-re x_m közel egyenlő valószínűséggel lehet az egyes konjugáltosztályokban. Tehát az előző algoritmusunkat változtassuk annyival, hogy ne teljesen véletlenszerű csoportelemeket vegyünk mindig, hanem mindig az előző centralizátorából válasszunk csak, és így gyorsabban megtalálhatjuk minden osztály reprezentáns elemét.

Vannak teljesen más elven működő további algoritmusok is erre a célra, de azokat nem részletezzük.

2. Permutációcsoportok

Minden véges csoport felírható permutációcsoportként, így a permutációcsoportok hatékony használata kiemelten fontos. A ma is használt módszer Charles C. Simstől származik 1970-ből [Sim70]. A 2.1. alfejezet az alapvető definíciókról szól, amik szükségesek Sims módszerének megértéséhez, a 2.2. alfejezet arról szól, hogy hogyan tudjuk egy tetszőleges módon megadott permutációcsoportnak a hatékony megadását megkonstruálni, míg a 2.3. rész az itt előforduló főként backtrack-jellegű algoritmusokat részletezi. Ez a fejezet egy nagyon rövid kivonata lényegében Seress Ákos közel 300 oldalas könyvének [Ser03], ami részletesebben tárgyal erről a témakörről.

2.1. Bázisok és erős generátorrendszerek

Legyen $G \leq \mathrm{Sym}(\Omega)$. A $B = (\beta_1, \ldots, \beta_m)$ különböző Ω-beli elemekből álló sorozatot G bázisának hívjuk, ha G-nek egyetlen olyan eleme van (mégpedig az egységelem), ami pontonként fixen tartja B-t, vagyis ha $G_B = \{1\}$. Egy bázis mindig definiál egy

$$G = G^{[1]} \geq G^{[2]} \geq \ldots \geq G^{[m]} \geq G^{[m+1]} = \{1\} \tag{2.1}$$

részcsoport-láncot, ahol $G^{[i]} = G_{(\beta_1,\ldots,\beta_{i-1})} = G_{\beta_1} \cap \ldots \cap G_{\beta_{i-1}}$, vagyis a B első $i - 1$ elemét fixen tartó csoportelemek részcsoportja. A bázis irredundáns, ha minden $1 \leq i \leq m$-re $G^{[i]} > G^{[i+1]}$, vagyis ha a lánc $m + 1$ különböző részcsoportból áll. Mostantól csak irredundáns bázisokkal foglalkozunk. $G^{[i]}$-ben $G^{[i+1]}$ mellékosztályai a $\beta_i^{G^{[i]}}$ orbit elemeinek felel meg, így $2 \leq \left|G^{[i]} : G^{[i+1]}\right| \leq |\Omega|$. Ezt felhasználva

$$2^m \leq |G| = \prod_{i=1}^{m} \left|G^{[i]} : G^{[i+1]}\right| \leq |\Omega|^m \tag{2.2}$$

Logaritmust véve (mint más algoritmusokkal kapcsolatos leírások esetén is, mi is mindig 2-es alapú logaritmust használunk), majd átrendezve

$$\frac{\log |G|}{\log |\Omega|} \leq m = |B| \leq \log |G| \tag{2.3}$$

Különböző irredundáns bázisok lehetnek különböző méretűek, de nem lehet egy bázis sem "túl nagy" emiatt.

Az $S \subseteq G$ részhalmazt G erős generátorrendszerének hívjuk (a B bázisra nézve), ha minden $1 \leq i \leq m + 1$-re $\langle S \cap G^{[i]} \rangle = G^{[i]}$. Ha adott egy erős generátorrendszer, akkor az 1.1. fejezetben leírt algoritmussal a $\beta_i^{G^{[i]}}$ orbitok könnyen kiszámolhatóak.

Ezeket az orbitokat fundamentális orbitoknak hívjuk. Ha a kiszámolásuk során az orbit minden pontjához felírjuk, hogy melyik $G^{[i]}$-beli elem viszi oda β_i-t, akkor megkapjuk az R_i transzverzálist, vagyis $G^{[i+1]}$ mellékosztályainak reprezentáns elemeit. Ha az R_i transzverzálist explicit számolnánk ki és tárolnánk, akkor $\Theta(|\Omega|^2)$ idő és memória kellene hozzá. Ezt elkerülhetjük az úgy nevezett Schreier-fák használatával. A Schreier-fa adatszerkezet egy T_i sorozat, minden β_i bázisponthoz tartozik egy. T_i egy irányított cimkézett fa, aminek pontjai a $\beta_i^{G^{[i]}}$ fundamentális orbit elemeinek felelnek meg. Minden él a β_i gyökér fele mutat és meg van cimkézve S egy elemével. Ha γ_1-ből γ_2-be megy egy él h cimkével, az azt jelenti, hogy $\gamma_1^h = \gamma_2$. Így ha akármelyik γ-ból végigmegyünk éleken β_i-ig és a cimkéket összeszorozzuk, akkor megkapjuk hogy melyik permutáció viszi γ-t β_i-be. Így T_i meghatározza a $G^{[i+1]}$ részcsoport mellékosztályainak reprezentáns elemeinek az inverzét $G^{[i]}$-ben. Azért az inverzekkel csináltuk, mert az alább leírt, úgynevezett szitáló eljáráshoz az inverzekre lesz szükségünk. A gyakorlatban S felírása után (ehhez $\mathcal{O}(|S||\Omega|)$ memória szükséges) T_i-t el tudjuk tárolni egy $|\Omega|$-hosszú V_i tömbben. $\gamma \in \Omega$-ra $V_i[\gamma]$-t akkor és csak akkor definiáljuk, ha $\gamma \in \beta_i^{G^{[i]}}$, és ilyenkor $V_i[\gamma]$ azt tartalmazza, hogy S hányadik eleme van a γ-ból kiinduló egyetlen él cimkéjén. Emiatt a tárolási mód miatt Sims eredetileg Schreier-vektoroknak hívta az adatszerkezetet, és ennek használatával S és az összes transzverzális összesen $\Theta((|S| + m)|\Omega|)$ memóriában eltárolható.

Feltehetjük, hogy $1 \in R_i$, vagyis hogy az egységelemet írtuk fel olyan elemnek ami β_i-t önmagába viszi. Ezeknek a transzverzálisoknak a segítségével minden $g \in G$ kanonikus alakra hozható, azaz egyértelműen felírható ilyen elemek szorzataként. Precízebben megfogalmazva minden $g \in G$-nek pontosan egy olyan szorzatalakja létezik, amire $g = r_m r_{m-1} \dots r_1$, ahol $r_i \in R_i$. Ez a szorzatalak algoritmikusan könnyen megtalálható a következőképpen: Adott $g \in G$-re először megkeressük azt az $r_1 \in R_1$-et, amire $\beta_1^g = \beta_1^{r_1}$. Ezután $g_2 = g r_1^{-1}$-zel folytatjuk, megkeressük azt az $r_2 \in R_2$-t, amire $\beta_2^{g_2} = \beta_2^{r_2}$. $g_3 = g_2 r_2^{-1}$, és így tovább, folytatjuk amíg végig nem érünk. $1 = g_{m+1} = g_m r_m^{-1}$ lesz az utolsó lépés, visszaszámolva $g = r_m r_{m-1} \dots r_1$-et kapjuk. Ezt az eljárást szitáló eljárásnak hívjuk.

A szitálás alkalmas egyben arra is, hogy egy $h \in \mathrm{Sym}(\Omega)$-beli elemről eldöntsük, hogy G-ben van-e. Ha megpróbáljuk h-ra alkalmazni az eljárást és sikerrel járunk, akkor $h \in G$. Két rész van az algoritmusban ahol elakadhatunk. Lehet, hogy valamely $i \leq m$-re $h_i = h r_1^{-1} r_2^{-1} \dots r_{i-1}^{-1}$-re nem találunk megfelelő r_i-t, mert $\beta_i^{h_i} \notin \beta_i^{G^{[i]}}$. Lehet az is, hogy a végén $h_{m+1} \neq 1$. Mindkét ilyen esetben nyilvánvalóan $h \notin G$, ilyenkor az utoljára kiszámolt h_i-t vagy h_{m+1}-et maradéknak hívjuk.

G rendjét is könnyen kiszámolhatjuk egy erős generátorrendszer ismeretében, ugyanis $|G| = |\beta_1^{G^{[1]}}||\beta_2^{G^{[2]}}| \dots |\beta_m^{G^{[m]}}| = |R_1||R_2| \dots |R_m|$.

2.2. Schreier-Sims algoritmus

Ha megadnak nekünk egy $G \leq \mathrm{Sym}(\Omega)$ permutációcsoportot valahogyan (black-box csoportnak is tekinthetjük akár), akkor ki kell tudnunk számolni egy bázisát és egy erős generátorrendszert ahhoz, hogy hatékonyan tudjunk vele dolgozni. Ez a Schreier-Sims algoritmussal történik, ami a következő két lemmán alapszik.

2.1. Lemma (Schreier)**.** *Legyen $H \leq G = \langle S \rangle$ és legyen R jobb transzverzálisa H-nak G-ben, amire $1 \in R$. Jelöljük $g \in G$-re $Hg \cap R$ egyetlen elemét $\overline{g}$-vel. Ilyenkor a*

$$T = \left\{ rs(\overline{rs})^{-1} \mid r \in R, s \in S \right\}$$

halmaz H-t generálja, vagyis $H = \langle T \rangle$. A T halmaz elemeit H Schreier-generátorainak hívjuk.

Bizonyítás. Definíció szerint $T \subseteq H$, így elég belátni, hogy $H \leq \langle T \rangle$. Legyen $h \in$ $\in H$ tetszőleges, felírhatjuk $h = s_1 s_2 \dots s_k$ alakban, ahol $s_i \in S$. Sorban definiáljuk $1 \leq i \leq k$-ra r_i-t és t_i-t, úgy hogy $h = t_1 t_2 \dots t_i r_i s_{i+1} s_{i+2} \dots s_k$ igaz legyen $\forall i$-re. Kezdőértéknek vegyük $r_0 = 1$-et, ezzel $h = r_0 s_1 s_2 \dots s_k$. Ha r_{i-1} már definiált, akkor legyen $t_i = r_{i-1} s_i (\overline{r_{i-1} s_i})^{-1} \in T$ és $r_i = \overline{r_{i-1} s_i} \in R$. Ezekre indukció szerint $h =$ $= t_1 t_2 \dots t_i r_i s_{i+1} s_{i+2} \dots s_k$ teljesül. Ha végigértünk, akkor $h = t_1 t_2 \dots t_k r_k$ alakot kapunk. Mivel $h \in H$ és $t_1 t_2 \dots t_k \in \langle T \rangle \leq H$, ezért $r_k \in H \cap R = \{1\}$. Így $h \in \langle T \rangle$. □

2.2. Lemma. *Legyen $\{\beta_1, \dots, \beta_m\} \subseteq \Omega$ és $G \leq \mathrm{Sym}(\Omega)$. Legyen $S_i \subseteq G^{[i]} =$ $= G_{(\beta_1,\dots,\beta_{i-1})}$ minden $1 \leq i \leq m$-re. A rövidség kedvéért vezessük be az $S_{m+1} =$ $= \emptyset$ jelölést. Ha $\langle S_1 \rangle = G$ és $\langle S_i \rangle_{\beta_i} = \langle S_{i+1} \rangle$ teljesül $1 \leq i \leq m$-re, akkor $B =$ $= (\beta_1, \dots, \beta_m)$ G-nek bázisa, és $S = \bigcup_{i=1}^{m} S_i$ erős generátorrendszer B-re nézve.*

Bizonyítás. Teljes indukciót alkalmazunk, az indukciós feltevésünk az, hogy a $G^* =$ $= \langle S_2 \rangle = G_{\beta_1}$ csoportnak $S^* = \bigcup_{i=2}^{m} S_i$ erős generátorrendszere a $B^* = (\beta_2, \dots, \beta_m)$ bázisra nézve. Ellenőriznünk kell, hogy minden $1 \leq i \leq m+1$-re $\langle S \cap G^{[i]} \rangle = G^{[i]}$. $G^{[1]} = G$, így $i = 1$-re triviális. A lemma feltevéséből $G^{[2]} = G_{\beta_1} = \langle S_1 \rangle_{\beta_1} =$ $= \langle S_2 \rangle \leq \langle S \cap G^{[2]} \rangle \leq G_{\beta_1} = G^{[2]}$, vagyis $i = 2$-re is készen vagyunk. $i > 2$-re az indukciós feltevés miatt $G^{[i]} \geq \langle S \cap G_{(\beta_1,\dots,\beta_{i-1})} \rangle \geq \langle S^* \cap G^*_{(\beta_2,\dots,\beta_{i-1})} \rangle = G^*_{(\beta_2,\dots,\beta_{i-1})} =$ $= G_{(\beta_1,\dots,\beta_{i-1})} = G^{[i]}$, tehát $i > 2$-re is készen vagyunk. Mivel az indukciós feltevést csak $m \geq 2$ esetén használtuk, a bizonyítással készen vagyunk. □

Ha adott $G = \langle T \rangle$, akkor annak egy erős generátorrendszerét a következőképpen kaphatjuk. Az algoritmus futása közben nyilvántartunk egy $B = (\beta_1, \dots, \beta_m)$

listát egy irredundáns bázis már ismert elemeiről, és minden $1 \leq i \leq m$-re egy $S_i \subseteq G_{(\beta_1,\ldots,\beta_{i-1})}$ halmazt, amikre a 2.2. lemmát szeretnénk majd alkalmazni. Az algoritmus alatt S_i-ket fogjuk növelni, illetve új báziselemeket fogunk B-hez adni (ezáltal m-et is növelve), így ha eleinte $\langle S_1 \rangle = G$ teljesül, akkor abban a helyzetben vagyunk készen, amikor minden $1 \leq i \leq m$-re $\langle S_i \rangle_{\beta_i} = \langle S_{i+1} \rangle$ teljesül. Végig fent fogjuk tartani, hogy $\langle S_i \rangle \geq \langle S_{i+1} \rangle$ is fennáll minden i-re. Kezdetben legyen $B = (\beta_1)$, ahol $\beta_1 \in \Omega$ olyan hogy T-nek legalább egy eleme mozgatja, és legyen $S_1 = T$, ezzel garantálva, hogy $\langle S_1 \rangle = G$ teljesüljön. Azt mondjuk, hogy az algoritmus az l. szinten tart, ha minden $l < i \leq m$-re teljesül $\langle S_i \rangle_{\beta_i} = \langle S_{i+1} \rangle$. Kezdetben az első szintről indulunk, az algoritmus futása akkor ér véget, amikor a nulladik szintre jutunk.

Amikor az l. szinten vagyunk, a következő történik: Megnézzük, hogy $\langle S_l \rangle_{\beta_l} = \langle S_{l+1} \rangle$ teljesül-e. Mivel $\langle S_l \rangle \geq \langle S_{l+1} \rangle$, valamint $S_l \subseteq G_{(\beta_1,\ldots,\beta_{l-1})}$, ezért elegendő a $\langle S_l \rangle_{\beta_l} \leq \langle S_{l+1} \rangle$ irányú tartalmazást ellenőrizni. A 2.1. lemma alapján $\langle S_l \rangle_{\beta_l} = \langle rs(\overline{rs})^{-1} \mid r \in R_l, s \in S_l \rangle$, ahol R_l az $\langle S_l \rangle_{\beta_l}$ részcsoport transzverzálisa $\langle S_l \rangle$-ben. Meg kell néznünk, hogy az összes Schreier-generátor benne van-e $\langle S_{l+1} \rangle$-ben. Ezt az előző részben leírt szitálással tudjuk ellenőrizni, mivel a 2.2. lemma szerint $\langle S_{l+1} \rangle$-nek már ismerjük egy erős generátorrendszerét. Ha mind benne van, vagyis ha $\langle S_l \rangle_{\beta_l} = \langle S_{l+1} \rangle$ teljesül, akkor az $l-1$. szintre lépünk. Ha nincs benne mind, vagyis $\langle S_l \rangle_{\beta_l} > \langle S_{l+1} \rangle$, akkor a szitálás során, amit kaptunk maradékot arra a Schreier-generátorra, ami nincs benne $\langle S_{l+1} \rangle$-ben, azt vegyük hozzá az S_{l+1} halmazhoz. Ettől a hozzávételtől $\langle S_1 \rangle \geq \langle S_2 \rangle \geq \ldots \geq \langle S_{m+1} \rangle$ továbbra is fennáll, valamint $S_{l+1} \subseteq G_{(\beta_1,\ldots,\beta_l)}$ is igaz marad. $l = m$ esetén B-hez vegyünk hozzá egy olyan Ω-beli elemet, amit a maradék nem hagy helyben. Ezután az algoritmus az $l+1$. szinten folytatódik.

Az algoritmus mindenképpen véges, hiszen Ω véges, így csak véges sokszor tudtuk B-t növelni. Ha a transzverzálisok számolásakor explicit leírunk minden elemet, akkor $\mathcal{O}(|\Omega|^2 \log^3 |G| + |T||\Omega|^2 \log |G|)$ a futásidő, és $\mathcal{O}(|\Omega|^2 \log |G| + |T||\Omega|)$ a memóriaigény. Ha Schreier-fákkal számoltunk, akkor $\mathcal{O}(|\Omega|^3 \log^3 |G| + |T||\Omega|^3 \log |G|)$ a futásidő, és $\mathcal{O}(|\Omega| \log^2 |G| + |T||\Omega|)$ a memóriaigény. Ezeknek a bizonyítása megtalálható [Ser03]-ban, valamint [Mur03]-ban olvashatjuk az algoritmusnak rengeteg változatát részletesen elemezve, hogy mikor melyik a legalkalmasabb.

Sokszor elő fog fordulni, hogy az erős generátorrendszert nem akármilyen bázishoz szeretnénk megkapni. Ilyenkor kétféle lehetőségünk van. Ha megadnak nekünk egy $(\Omega, \prec)$ rendezést, amilyen sorrendben szeretnék a báziselemeket megkapni, akkor amikor az algoritmus új báziselemet választ, akkor mindig a lehető legelsőt választjuk, illetve megnézzük a többi Schreier-generátort is, hogy azoknak a maradékával be tudunk-e venni előbbi báziselemet. Másik lehetőség az úgynevezett bázisváltás,

hogy ha már végigszámoltuk az algoritmust egy tetszőleges bázisban, akkor annak az eredményét felhasználva gyorsabban is ki lehet számolni az új erős generátorrendszert. A módszer azon alapszik, hogy ha $(\beta_1, \ldots, \beta_m)$ bázisra nézve már ismert az erős generátorrendszer és az R_j transzverzálisok, akkor valamilyen i-re ki tudjuk számolni a $(\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \beta_i, \beta_{i+2}, \ldots, \beta_m)$ bázishoz tartozót is, vagyis két egymásutáni báziselemet fel tudunk cserélni. Ezzel a művelettel akármilyen más bázisba is eljuthatunk, hiszen új báziselemet is fel tudunk venni a listába olyan helyen, hogy az redundáns legyen. Ez a módszer méghatásosabb, ha nem az egész bázist adták meg, hanem csak az első pár báziselemet. Erre példa, ha egy G_ω stabilizátort szeretnénk kiszámolni, ilyenkor csak az első báziselem fontos. Nézzük meg hogyan cserélhetünk ki két báziselemet.

Legyen $B = (\beta_1, \ldots, \beta_m)$, S az ehhez tartozó erős generátorrendszer, R_j-k pedig a transzverzálisak. $\overline{B} = (\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \beta_i, \beta_{i+2}, \ldots, \beta_m)$-ra ki szeretnénk számolni a hozzátartozó $\overline{S}$ erős generátorrendszert, valamint az új $\overline{R}_j$ transzverzálisokat. Minden $1 \leq j < i$-re, valamint $i+1 < j \leq m$-re $\overline{R}_j = R_j$, tehát azokkal készen vagyunk. Mivel $\overline{G}^{[i]} = G^{[i]}$, az új i-edik fundamentális orbit $\beta_{i+1}^{\langle S \cap G^{[i]} \rangle}$, ezért az $\overline{R}_i$ transzverzálist könnyen kiszámolhatjuk. A számolás neheze $\overline{R}_{i+1}$-nél jön. Legyen $\overline{S} = S$ eleinte, és kezdetben tekintsük az új $i+1$-edik fundamentális orbitot $\overline{\Delta}_{i+1} = \{\beta_i\}$-nek. Ezekután tekinthetjük az $\overline{R}_i$ és $\overline{S} \cap \overline{G}^{[i]}$ által alkotott Schreier-generátorokat, amik $\overline{G}^{[i+1]} = G_{(\beta_1,\ldots,\beta_{i-1},\beta_{i+1})}$ generátorrendszerét adják. Ha ezek közül valamelyik β_i-t kiviszi $\overline{\Delta}_{i+1}$-ből, akkor azt az elemet $\overline{S}$-hoz adjuk hozzá és a fundamentális orbitot számoljuk újra: $\overline{\Delta}_{i+1} = \beta_i^{\langle \overline{S} \cap \overline{G}^{[i+1]} \rangle}$. Mivel $|\overline{\Delta}_{i+1}| = |\overline{R}_{i+1}| = |R_i||R_{i+1}|/|\overline{R}_i|$ előre ismert, ezért nem kell minden Schreier-generátort megnéznünk. Legfeljebb $\log |\overline{G}^{[i+1]}|$ elemet adtunk hozzá S-hez, mivel minden elem hozzáadása $\langle \overline{S} \cap \overline{G}^{[i+1]} \rangle$ elemszámát legalább megduplázta, így nem lesz túl nagy az új erős generátorrendszerünk. $\overline{R}_{i+1}$ Schreier-fa ábrázolása $\overline{S}$ segítségével kiszámolható.

2.3. Backtrack módszerek

Ebben a részben legyen adott G és legyen $\mathcal{P}$ egy tulajdonság, meg kell találnunk $G(\mathcal{P})$-t, vagyis G azon elemeit, amik $\mathcal{P}$ tulajdonságúak. Ehhez feltesszük, hogy minden $g \in G$-re el tudjuk dönteni, hogy $\mathcal{P}$ tulajdonságú-e. Néhány tulajdonságra nincs ismert jobb algoritmus annál, mint hogy végigmegyünk G elemein, és mindegyiket ellenőrizzük. Gyakori eset azonban, hogy $G(\mathcal{P})$, ha nem üres, akkor G egy részcsoportja, esetleg egy részcsoport mellékosztálya. Például, ha egy elem vagy egy részhalmaz centralizátorát szeretnénk megtalálni, akkor egy részcsoportot keresünk. Mellékosztályra példa, ha $a, b \in G$-re azokat a $g \in G$-ket keressük, amelyek a-t

b-be konjugálják, vagyis amelyekre $a^g = g^{-1}ag = b$. Gyakran elegendő azt eldöntenünk, hogy $G(\mathcal{P})$ üres-e, az előző példánál ez annak eldöntését jelenti, hogy a és b konjugáltak-e. Ebben a részben feltesszük, hogy $G(\mathcal{P})$ egy részcsoport, egy részcsoport mellékosztálya vagy üres.

Legyen $B = (\beta_1, \ldots, \beta_m)$ a $G \leq \mathrm{Sym}(\Omega)$ csoport egy bázisa, és legyen $G = G^{[1]} \geq G^{[2]} \geq \ldots \geq G^{[m]} \geq G^{[m+1]} = \{1\}$ az ehhez tartozó részcsoportlánc. Ω-n definiáljunk egy $\prec$-rendezést, amire $\beta_1 \prec \beta_2 \prec \ldots \prec \beta_m$, valamint minden $\omega \in \Omega \setminus B$-re $\beta_m \prec \omega$. G minden elemét egyértelműen meghatározza az, hogy B elemeit hova viszi, így a báziselemek képeinek sorozátaval azonosíthatjuk a csoportelemeket. Ez az azonosítás egy rendezést is indukál G-n, $g, h \in G$-re $g \prec h$ akkor és csak akkor teljesül, ha létezik $i \in \{1, \ldots, m\}$, amire minden $j < i$-re $\beta_j^g = \beta_j^h$, valamint $\beta_i^g \prec \beta_i^h$. A báziselemek képeinek meghatározásával szeretnénk csoportelemeket keresni. $(\beta_1, \ldots, \beta_l)$ lehetséges képei $G^{[l+1]}$ mellékosztályait határozzák meg. Ezeknek a képeknek a halmazát jelöljük $\mathcal{T}_l$-lel. Legyen $\mathcal{T} = \bigcup_{i=1}^{m} \mathcal{T}_i$, ezen nézzük a tartalmazást, mint természetes részbenrendezést, amivel $\mathcal{T}$-t egy keresőfának tekinthetjük. $\mathcal{T}$ gyökere (a () sorozat) G-nek felel meg, a levelei (az m-hosszú sorozatok) a csoportelemeknek. Jelöljük $t \in \mathcal{T}$-re $\mathcal{T}(t)$-vel a t-ből induló részfát. Legyen $\varphi : \mathcal{T} \to P(G)$, amire $\varphi((\gamma_1, \ldots, \gamma_l)) = \{g \in G \mid \beta_i^g = \gamma_i \quad \forall 1 \leq i \leq l\}$. Az algoritmusunk ezt a keresőfát járja be mélységi bejárással (backtrack), így keressük meg $G(\mathcal{P})$-t. Amikor $t = (\gamma_1, \ldots, \gamma_{l-1}) \in \mathcal{T}_{l-1}$-nél tartunk a bejárásban, akkor γ_l-nek nem kell minden Ω-beli elemet végigpróbálnunk, hanem elég azokat, amikre $\varphi((\gamma_1, \ldots, \gamma_l))$ nem üres. Mivel $\varphi(t)$ $G^{[l]}$-nek egy mellékosztálya, ezért ha veszünk egy tetszőleges $g \in \varphi(t)$ csoportelemet, akkor a szóbajövő γ_l-ek azok, amikre $\gamma_l \in (\beta_l^{G^{[l]}})^g$. Ez a bejárás a csoportelemeket mellesleg $\prec$-lexikografikus sorrendben találja meg. Ha menet közben $t \in \mathcal{T}$-nél tartunk és $\varphi(t) \cap G(\mathcal{P})$-ről be tudjuk látni, hogy vagy üres, vagy hogy már minden elemét ismerjük, akkor a mélységi bejárást attól a ponttól nem kell mélyebbre folytatnunk. Akkor lesz hatékony az algoritmusunk, ha minél nagyobb részfákat el tudunk így dobni. Vannak a $\mathcal{P}$ tulajdonságtól független és attól függő elvek, amik alapján egy részfát eldobhatunk. A következő lemma, az egyik legegyszerűbb $\mathcal{P}$-től független ilyen elvet mutatja be.

2.3. Lemma. *Tegyük fel, hogy $G(\mathcal{P}) \leq G$, a $K = G(\mathcal{P}) \cap G^{[l+1]}$ részcsoportot már ismerjük valamilyen $l \in \{1, \ldots, m\}$-re és hogy most valamelyik $t \in \mathcal{T}_l$-re $\mathcal{T}(t)$ belsejében vagyunk. Ha találunk egyetlen $g \in \varphi(t) \cap G(\mathcal{P})$-t, akkor $\varphi(t) \cap G(\mathcal{P}) \subseteq \subseteq \langle g, K \rangle \subseteq G(\mathcal{P})$, így kihagyhatjuk $\mathcal{T}(t)$ maradék részét.*

Bizonyítás. Minden $h \in \varphi(t) \cap G(\mathcal{P})$ a Kg mellékosztályban található. □

A következő lemmához tegyük fel, hogy ismerünk olyan $K, L \leq G$ részcsoportokat, amikre igaz, hogy minden $g \in G$-re $g \in G(\mathcal{P})$ akkor és csak akkor, ha a kettős mellékosztály $KgL \subseteq G(\mathcal{P})$. Ha $G(\mathcal{P})$ részcsoport, akkor tekinthetjük K-nak és L-nek $G(\mathcal{P})$ már ismert részcsoportját. P-től függően azonban lehet, hogy más K-t meg L-et is választhatunk. Például, ha P azon elemekre teljesül, amik a-t b-be konjugálják, akkor választhatjuk $K = \langle a \rangle$-t és $L = \langle b \rangle$-t. Mivel minden KgL kettős mellékosztálynak elég egyetlen elemét megtalálnunk, és a bejárás $\prec$-lexikografikus sorrendben találja meg az elemeket, ezért a $\mathcal{T}(t)$ részfát kihagyhatjuk, ha tudjuk, hogy egyik $g \in \varphi(t)$ sem lehet a saját KgL kettős mellékosztályának első eleme. Sajnos egy kettős mellékosztály első elemét megtalálni NP-nehéz probléma [Luk93], ezért kicsit másképp kell csinálnunk, erre való a következő 3 lemma.

2.4. Lemma. *Legyen $t = (\gamma_1, \ldots, \gamma_l) \in \mathcal{T}$. Ha $g \in \varphi(t)$ a KgL kettős mellékosztály lexikografikusan első eleme, akkor γ_l a $\gamma_l^{L_{(\gamma_1,\ldots,\gamma_{l-1})}}$ orbit első eleme.*

Bizonyítás. Indirekten tegyük fel, hogy van olyan $\gamma \in \gamma_l^{L_{(\gamma_1,\ldots,\gamma_{l-1})}}$, amire $\gamma \prec \gamma_l$. Legyen h az az $L_{(\gamma_1,\ldots,\gamma_{l-1})}$-beli elem, amire $\gamma_l^h = \gamma$. Ilyenkor $gh \in KgL$ és $gh \prec g$, tehát ellentmondásra jutottunk. □

Eszerint a lemma szerint, amikor a $t' = (\gamma_1, \ldots, \gamma_{l-1}) \in \mathcal{T}_{l-1}$ részfát nézzük, akkor elég $L_{(\gamma_1,\ldots,\gamma_{l-1})}$ orbitjainak első elemeivel folytatnunk. Ahhoz, hogy az orbitokat kiszámoljuk L-ben bázist kell váltanunk, azaz L-nek egy $(\gamma_1, \ldots, \gamma_{l-1})$-gyel kezdődő bázishoz tartozó erős generátorrendszerét kell kiszámolnunk. Emiatt ez a kritérium nem annyira gyorsan ellenőrizhető, sok implementációban csak kis l-ekre alkalmazzák.

Az előző lemma egy általánosabb alakja a következő:

2.5. Lemma. *Tegyük fel, hogy $\beta_l \in \beta_k^{K_{(\beta_1,\ldots,\beta_{k-1})}}$ valamilyen $k \leq l$-re. Legyen $t = (\gamma_1, \ldots, \gamma_l) \in \mathcal{T}$. Ha $g \in \varphi(t)$ a KgL kettős mellékosztály lexikografikusan első eleme, akkor $\gamma_k \preceq \min_{\prec}(\gamma_l^{L_{(\gamma_1,\ldots,\gamma_{k-1})}})$.*

Bizonyítás. Legyen h_1 az a $K_{(\beta_1,\ldots,\beta_{k-1})}$-beli elem, amire $\beta_k^{h_1} = \beta_l$. Indirekten tegyük fel, hogy van olyan $\gamma \in \gamma_l^{L_{(\gamma_1,\ldots,\gamma_{k-1})}}$, amire $\gamma \prec \gamma_k$. Legyen h_2 az az $L_{(\gamma_1,\ldots,\gamma_{k-1})}$-beli elem, amire $\gamma_l^{h_2} = \gamma$. $h_1 g h_2 \in KgL$ és $h_1 g h_2 \prec g$, mivel $i < k$-ra $\beta_i^{h_1 g h_2} = \beta_i^{g h_2} = \gamma_i^{h_2} = \gamma_i = \beta_i^g$, valamint $\beta_k^{h_1 g h_2} = \beta_l^{g h_2} = \gamma_l^{h_2} = \gamma \prec \gamma_k = \beta_k^g$. Ellentmondásra jutottunk. □

Ez $k = l$ esetén az ezelőtti lemmát adja vissza, $k < l$-re pedig azt jelenti, hogy $\beta_l \in \beta_k^{K_{(\beta_1,\ldots,\beta_{k-1})}}$ esetén csak azok a $t = (\gamma_1, \ldots, \gamma_l) \in \mathcal{T}_l$-ek érdekesek, amikre $\gamma_l \succ \succ \gamma_k$.

2.6. Lemma. *Legyen* $s = |\beta_l^{K_{(\beta_1,\ldots,\beta_{l-1})}}|$, *azaz* K l*-edik fundamentális orbitjának mérete. Legyen* $t = (\gamma_1, \ldots, \gamma_l) \in \mathcal{T}$. *Ha* $g \in \varphi(t)$ *a* KgL *kettős mellékosztály lexikografikusan első eleme, akkor* γ_l *nem lehet a* $\gamma_l^{G_{(\gamma_1,\ldots,\gamma_{l-1})}}$ *orbit utolsó* $s-1$ *eleme között.*

Bizonyítás. A $\Gamma = \{\beta_l^{hg} \mid h \in K_{(\beta_1,\ldots,\beta_{l-1})}\}$ halmaz s elemű és $\gamma_l = \beta_l^g \in \Gamma$. Minden $\gamma = \beta_l^{hg} \in \Gamma$ $G_{(\gamma_1,\ldots,\gamma_{l-1})}$-nek ugyanahhoz az orbitjához tartozik, mivel $\gamma^{g^{-1}h^{-1}g} = \gamma_l$, ahol $g^{-1}h^{-1}g \in G_{(\gamma_1,\ldots,\gamma_{l-1})}$. $hg \in KgL$ és g minimalitása miatt $\gamma_l = \min_{\prec} \Gamma$, ezért legalább $s-1$ elem jön később KgL-ben. □

Emiatt amikor a $t' = (\gamma_1, \ldots, \gamma_{l-1}) \in \mathcal{T}_{l-1}$ részfát nézzük, akkor $G_{(\gamma_1,\ldots,\gamma_{l-1})}$ minden orbitjának utolsó $s-1$ elemét kihagyhatjuk.

Most nézzünk $\mathcal{P}$-specifikus kritériumokat. Nézzük először a centralizátor problémát, legyen adott $c \in G$, $C_G(c) = G(\mathcal{P})$-t kell megtalálnunk. Mivel $C_G(c) = C_G(\langle c \rangle)$, ezért minden $\omega \in \Omega, g \in C_G(c), k \in \mathbb{Z}$-re igaz, hogy $(\omega^{c^k})^g = (\omega^g)^{c^k}$. Ez két dolgot is jelent, egyrészt hogy ha ω g általi képét meghatároztuk már, akkor az az ω^{c^k} alakú elemek képeit is meghatározza, másrészt pedig hogy ha ω egy l-hosszú ciklus része c-ben, akkor ω^g is csak egy l-hosszú ciklus része lehet c-ben. Ezeket a feltételeket akkor tudjuk hatékonyan kihasználni, hogyha olyan bázisban írjuk fel az erős generátorrendszert, amiben c egyes ciklusainak az elemei egymást követik. Így tudunk $\omega, \omega^c, \omega^{c^2}, \ldots$ közül minél többből báziselemet csinálni, tehát így lesz minél több $\mathcal{T}(t)$ eldobható.

Másik példának nézzük a konjugáltság-ellenőrzés esetét, adott $a, b \in G$, $G(\mathcal{P}) = \{g \in G \mid a^g = b\}$. Ebben a problémában $G(\mathcal{P})$ vagy üres, vagy $C_G(a)$ egy jobboldali mellékosztálya, ezért hasonlít ez az eset a centralizátor-problémára, ahol hasonló feltételek jöttek ki. g-nek a ciklusait b ugyanolyan hosszú ciklusaihoz kell rendelnie, valamint ha a egy ciklusának egy elemének a képét meghatározzuk, az az egész ciklus képét is megmondja. Itt is speciális bázisban érdemes felírni az erős generátorrendszert, mégpedig olyanban amiben a ciklusainak elemei egymást követik.

Másik fontos itt említendő probléma a következő: Legyen adottak a $G, H \leq \leq \mathrm{Sym}(\Omega)$ csoportok, a $G \cap H$ csoportnak szeretnénk egy generátorrendszerét meghatározni. Feltehetjük, hogy $|G| \leq |H|$. Először számoljuk ki G-nek egy $B = (\beta_1, \ldots, \beta_m)$ bázisát, és definiáljuk $\mathcal{T}$-t és φ-t, mint eddig. Számoljuk ki H-nak egy olyan bázisát, ami B-vel kezdődik és egy ehhez tartozó erős generátorrendszert. Legyen $\psi : \mathcal{T} \to P(H)$ φ-hez analóg módon definiálva H-ra. Ha $t = (\gamma_1, \ldots, \gamma_{l-1}) \in \in \mathcal{T}_{l-1}$ részfáját nézzük, akkor csak azokkal a γ_l elemekkel kell folytatnunk, amik nem csak G-ben lehetséges folytatások, hanem H-ban is, vagyis a $\gamma_l \in (\beta_l^{G^{[l]}})^g \cap (\beta_l^{H^{[l]}})^h$ alakúakat, ahol $g \in \varphi(t)$ és $h \in \psi(t)$. Így minden $\mathcal{T}_m$-beli elem, amihez eljutunk,

egy olyan G-beli elemet reprezentál, ami H-ban is benne van, vagyis a menetközben felírt generátorelemei $G(\mathcal{P})$-nek azok pont $G \cap H$-nak lesznek a generátorelemei.

3. Burnside-Dixon-Schneider algoritmus

Ebben a fejezetben a véges csoportok karaktertáblázatának kiszámítására alkalmas Burnside-Dixon-Schneider algoritmusról lesz szó. A csoport megadásának módját most figyelmen kívül hagyjuk, de feltesszük hogy pár alapvető algoritmus már rendelkezésünkre áll. Permutációcsoportoknál ezeket már részleteztük, de más megadásoknál is megoldhatóak.

Az eredeti algoritmus Burnside-tól származik [Bur11]. Az első alfejezetben ismertetjük az eredeti algoritmust, a többi alfejezet ennek az algoritmusnak módosításairól, javításairól szól.

3.1. Burnside eredeti algoritmusa

Legyen G tetszőleges véges csoport, jelöljuk a konjugáltosztályainak számát r-rel, az osztályokat $C_1, \ldots, C_r$-rel, ezeknek egy-egy reprezentáns elemét $g_1, \ldots, g_r$-rel. Legyen $i \in \{1, \ldots, r\}$-re $h_i = |C_i| = |G : C_G(g_i)|$, a konjugáltosztályok elemszáma. Természetesen választhatjuk g_1-et 1-nek, így $h_1 = 1$.

Feltételezzük, hogy a 2. fejezetben leírtak szerint g_i-k, h_i-k már ki vannak számolva. Azt is fel kell tennünk, hogy minden $g \in G$-re gyorsan meg tudjuk állapítani, hogy melyik konjugáltosztályba tartozik. Kisebb csoportok esetében, ha van elég memória, érdemes minden g-re előre kiszámolnunk ezt.

Jelöljük G irreducibilis karaktereit $\chi^1, \ldots, \chi^r$-rel, és a rövidség kedvéért legyen $\chi^i_j = \chi^i(g_j)$. Legyen $d_i = \chi^i_1$, azaz χ^i foka.

$1 \leq j, k, l \leq r$-re legyen c_{jkl} azon elempárok száma, ahol az egyik elem C_j-beli, a másik elem C_k-beli és a szorzatuk g_l. Ismert, hogy c_{jkl} független g_i-k megválasztásától, valamint hogy teljesül a következő egyenlőség minden $1 \leq i, j, k \leq r$-re:

$$\frac{h_j \chi^i_j}{d_i} \frac{h_k \chi^i_k}{d_i} = \sum_{l=1}^{r} c_{jkl} \frac{h_l \chi^i_l}{d_i} \tag{3.1}$$

Legyen M_j az az $r \times r$-es mátrix, aminek (k, l)-edik eleme c_{jkl}, ezeket könnyen ki tudjuk számolni. Legyen $v^i = [h_1 \chi^i_1/d_i, \ldots, h_r \chi^i_r/d_i]^T$. Ha i, j-t lerögzítjük, akkor ezekkel a jelölésekkel a (3.1) egyenlet átírható a következő alakba:

$$\frac{h_j \chi^i_j}{d_i} v^i = M_j v^i \tag{3.2}$$

Vagyis v^i jobboldali sajátvektora M_j-nek minden i, j-re. Van tehát r darab közös sajátvektorunk minden M_j-hez, amik közül semelyik kettőhöz sem tartozhat minden

M_j esetén ugyanaz a sajátérték, így ezeket a sajátvektorokat M_j-kből konstansszorzó erejéig egyértelműen meghatározhatjuk lineáris algebrai ismereteink alapján. A helyes konstansszorzókat könnyen megkaphatjuk, hiszen $v_1^i = h_1\chi_1^i/d_i = 1d_i/d_i = 1$, így M_j-kből meg tudjuk állapítani v_j^i-ket.

Kérdés, hogy v_j^i-k ismeretében hogyan állapíthatjuk meg χ_j^i-ket. h_j-ket ismerjük, tehát elegendő d_i-k kiszámolása. Szintén ismert, hogy a komplex-értékű osztályfüggvények terén értelmezhető egy természetes skalárszorzat a következőképpen:

$$\langle \alpha, \beta \rangle = \frac{1}{|G|} \sum_{g \in G} \alpha(g)\overline{\beta(g)} = \frac{1}{|G|} \sum_{j=1}^{r} h_j \alpha(g_j)\overline{\beta(g_j)} \tag{3.3}$$

A karakterekre vonatkozó ortogonális relációk alapján $\langle \chi^i, \chi^j \rangle = \delta_{ij}$, ami alapján

$$1 = \langle \chi^i, \chi^i \rangle = \frac{1}{|G|} \sum_{j=1}^{r} h_j |\chi_j^i|^2 = \frac{1}{|G|} \sum_{j=1}^{r} h_j \left| \frac{d_i v_j^i}{h_j} \right|^2 = \frac{d_i^2}{|G|} \sum_{j=1}^{r} \frac{|v_j^i|^2}{h_j} \tag{3.4}$$

Ezalapján d_i kifejezhető, tehát sikerült χ_j^i-ket kiszámolnunk.
Összefoglalva az algoritmust:

1. Kiszámoljuk r, C_i, g_i, h_i, M_i-ket
2. Kiszámoljuk M_i-knek az r darab közös sajátvektorát, majd ezekből v_j^i-ket
3. Kiszámoljuk d_i-ket, és végül ebből χ_j^i-ket

3.2. Schneider első módosítása

Ezt a módosítást Schneider írta le [Sch90]-ben. Nem annyira a sebességén javít az algoritmusnak, hanem inkább egyszerűbbé teszi azt, ezért ezt vesszük előre, annak ellenére is, hogy a 3.3. részben leírt módosítást Dixon már jóval előtte kitalálta.

Jelöljük $1 \leq j \leq r$-re j'-vel g_j^{-1} konjugáltosztályának a számát, vagyis amire $C_{j'} = C_j^{-1}$. Mivel $|C_l| = h_l$, így azon $(x, y, z) \in C_j \times C_k \times C_l$ hármasok száma, amire $xy = z$ egyenlő $h_l c_{jkl}$, de szintén egyenlő azon hármasok számával, amikre $x^{-1}z = y$, vagyis $h_l c_{jkl} = h_k c_{j'lk}$. Ezt a (3.1) egyenlet jobb oldalába beírva, majd j-t és j'-t felcserélve kapjuk, hogy

$$\frac{h_{j'}\chi_{j'}^i}{d_i}\chi_k^i = \sum_{l=1}^{r} c_{jlk}\chi_l^i \tag{3.5}$$

Vagyis a $[\chi_1^i, \ldots, \chi_r^i]$ sorvektor baloldali sajátvektora M_j-nek.

Konstansszorzók erejéig most is egyértelműen meghatározhatjuk M_j-kből ezeket a vektorokat, most viszont a konstansszorzó megállapításához kell használnunk a

karakterek ortonormáltságát. Tegyük fel, hogy találtunk r darab közös sajátvektort, jelöljük ezeket w^i-vel. A keresett szorzó d_i/w_1^i, mert az első elemnek d_i-nek kell lennie.

$$1 = \langle \chi^i, \chi^i \rangle = \frac{1}{|G|} \sum_{j=1}^{r} h_j |\chi_j^i|^2 = \frac{1}{|G|} \sum_{j=1}^{r} h_j \left| \frac{d_i}{w_1^i} w_j^i \right|^2 = \frac{d_i^2}{|G||w_1^i|^2} \sum_{j=1}^{r} h_j |w_j^i|^2 \qquad (3.6)$$

Ezalapján d_i-ket megállapíthatjuk, amiből meg χ_j^i-ket megkapjuk.

3.3. Dixon módosítása

Jelöljük G exponensét, vagyis az elemrendek legkisebb közös többszörösét e-vel. Ismert, hogy χ_j^i-k algebrai egészek, sőt éppen d_i darab komplex e-edik egységgyök összege, vagyis az algoritmustól elvárható kell, hogy legyen, hogy a karaktertáblázat elemeit ne csak közelítő értékekkel, hanem pontos értékekkel kapjuk meg. Az eddig leírt módszer azonban csak közelítésre volt alkalmas, ugyanis az M_j mátrixok közös sajátértékeit csak úgy tudjuk meghatározni. Erre Dixon talált egy megoldást, amit [Dix67]-ben írt le. Az alapötlet az, hogy válasszunk egy alkalmas p prímet, minden számítást $\mathbb{F}_p$ felett végezzünk el, majd a kapott eredményt emeljük vissza $\mathbb{C}$-be.

Jelöljünk egy e-edik komplex primitív egységgyököt ζ-val. Minden karaktertáblázatbeli elem $\mathbb{Z}[\zeta]$-beli. $h_j\chi_j^i/d_i$-ről is ismert, hogy algebrai egész, tehát $h_j\chi_j^i/d_i \in$ $\in \mathbb{Q}[\zeta] \cap \mathbb{A} = \mathbb{Z}[\zeta]$, így a (3.1), (3.5) egyenletek is $\mathbb{Z}[\zeta]$-beliek.

Válasszunk egy olyan p prímet, amire $e|p-1$ és $p > 2d_i$ minden $1 \leq i \leq r$-re. Mivel d_i-ket nem ismerjük még, de tudjuk, hogy $\sum_{i=1}^{r} d_i^2 = |G|$, ezért $p > 2\lfloor\sqrt{|G|}\rfloor$ elegendő. Mivel $e|p-1$, ezért $\mathbb{F}_p$-ben van e-rendű elem, jelöljünk egy ilyet ω-val. Így adódik egy $\Theta : \mathbb{Z}[\zeta] \to \mathbb{F}_p$ gyűrűhomomorfizmus, a következő hozzárendeléssel:

$$\sum_{i=0}^{e-1} a_i \zeta^i \mapsto \sum_{i=0}^{e-1} a_i \omega^i$$

Θ-t alkalmazva az (3.5) egyenletre $\mathbb{F}_p$ feletti egyenletrendszert kapunk, tehát a $[\Theta(\chi_1^i), \ldots, \Theta(\chi_r^i)]$ sorvektorok $\mathbb{F}_p$ felett baloldali sajátvektorai $\Theta(M_j)$-knek. Be kell látnunk, hogy ezek a sorvektorok lineárisan függetlenek $\mathbb{F}_p$ felett. Legyen X az a mátrix, aminek (i,j)-edik eleme χ_j^i, vagyis X maga a karaktertáblázat. Legyen Y az a mátrix, aminek (j,i)-edik eleme $h_j\chi_j^i$. A karakterek ortonormáltsága miatt $XY =$ $= |G|I_r$. Erre alkalmazva Θ-t azt kapjuk, hogy $\Theta(X)\Theta(Y) = \Theta(XY) = \Theta(|G|I_r) =$ $= \Theta(|G|)I_r$. Mivel $|G|$ minden q prímosztójára $q|e$, ezért $q|p-1$, vagyis $\Theta(|G|) \neq 0$, tehát $\Theta(X)$ sorai tényleg lineárisan függetlenek $\mathbb{F}_p$ felett. Az i_1 és i_2-edik sorokhoz

tartozó sajátértékek nem mind egyezhetnek meg, ugyanis akkor az a két sor lineárisan összefüggne. Így a $\Theta(M_j)$ mátrixok $\mathbb{F}_p$ felett is skalárszorzó erejéig egyértelműen meghatározzák a közös sajátvektoraikat.

Tehát miután kiszámoltuk $\Theta(M_j)$-ket, kiszámolhatjuk a karakterisztikus polinomjaikat, amit faktorizálva megkapjuk a sajátértékeket, amiből végül meghatározhatjuk a baloldali sajátvektorokat. Ezeket a műveleteket véges testek felett mind könnyen el tudjuk végezni, a részletekbe most nem megyünk bele.

A kapott sajátvektorokat normalizálhatjuk úgy, hogy megszorozzuk az első elemük inverzével, vagyis azokat a sajátvektorokat nézzük, amiknek az első eleme 1. Ezeket a sajátvektorokat jelöljük v^i-vel. Mivel $\Theta(\chi_1^i) = \Theta(d_i) = d_i$, így $\Theta(\chi_j^i) = = d_i v_j^i$, tehát d_i-t kell meghatároznunk. Az ismert $\chi^i(g^{-1}) = \overline{\chi^i(g)}$ azonosságból kapjuk, hogy

$$1 = \langle \chi^i, \chi^i \rangle = \frac{1}{|G|} \sum_{j=1}^{r} h_j \chi_j^i \overline{\chi_j^i} = \frac{1}{|G|} \sum_{j=1}^{r} h_j \chi_j^i \chi_{j'}^i \tag{3.7}$$

$|G|$-vel szorozva, majd Θ-t ráalkalmazva

$$|G| \equiv \sum_{j=1}^{r} h_j \Theta(\chi_j^i)\Theta(\chi_{j'}^i) \equiv d_i^2 \sum_{j=1}^{r} h_j v_j^i v_{j'}^i \pmod{p} \tag{3.8}$$

Ebből d_i^2 p-s maradékát megkapjuk, amiből d_i egyértelmű, mert $0 < d_i < p/2$ lehet csak.

$\Theta(\chi_j^i)$-ket már tudjuk, az algoritmus utolsó lépése, hogy visszakapjuk χ_j^i-ket, vagyis a komplex értékeket. Mivel χ_j^i d_i darab ζ-hatvány összege, ezért léteznek olyan $0 \leq m_{ijk} \leq d_i < p$ egészek, amikre:

$$\chi_j^i = \sum_{k=0}^{e-1} m_{ijk} \zeta^k \tag{3.9}$$

Jelöljük $j(l)$-lel annak a konjugáltosztálynak a számát, amiben g_j^l van. Az előző egyenletből $\chi_{j(l)}^i$-t is ki tudjuk fejezni, mégpedig:

$$\chi_{j(l)}^i = \sum_{k^*=0}^{e-1} m_{ijk^*} \zeta^{k^* l} \tag{3.10}$$

Ha a következő összeget tekintjük, majd ezt belehelyettesítjük:

$$\frac{1}{e} \sum_{l=0}^{e-1} \chi_{j(l)}^i \zeta^{-kl} = \frac{1}{e} \sum_{l=0}^{e-1} \sum_{k^*=0}^{e-1} m_{ijk^*} \zeta^{(k^*-k)l} = \frac{1}{e} \sum_{k^*=0}^{e-1} m_{ijk^*} \sum_{l=0}^{e-1} \zeta^{(k^*-k)l} =$$

$$= \frac{1}{e} \sum_{k^*=0}^{e-1} m_{ijk^*} e \delta_{kk^*} = m_{ijk} \tag{3.11}$$

Θ-t ráalkalmazzuk az egyenletre:

$$m_{ijk} = \Theta(m_{ijk}) = \Theta(e)^{-1} \sum_{l=0}^{e-1} \Theta(\chi^i_{j(l)}) \omega^{-kl} \tag{3.12}$$

Ezzel $\Theta(\chi^i_j)$-kből ki tudtuk fejezni χ^i_j-ket, vagyis készen vagyunk.

3.4. Schneider második módosítása

Nem esett szó eddig arról, hogy hogyan lehet az M_j mátrixokat kiszámolni. Egyszerre M_j egy oszlopát (legyen most ez az l-edik) tudjuk megadni, mégpedig úgy hogy minden $x \in C_j$-re kiszámoljuk $y = x^{-1} g_l$-t és azt, hogy y melyik konjugáltosztályba esik. M_j (k,l)-edik eleme azon x-ek száma, amikre $y \in C_k$. Általában az egész algoritmus futásának legnagyobb részét ez a számolás teszi ki, ezért arra kell törekednünk, hogy minél kevesebb oszlop kiszámolásával meghatározhatóak legyenek a sajátvektorok. Vegyük észre, hogy M_j első oszlopának j'-edik eleme h_j, az összes többi eleme 0, így az első oszlopokat már ismerjük. M_1-et is ismerjük, hiszen annak (k,l)-edik eleme δ_{kl}, vagyis $M_1 = I_r$.

Ebben a részben a sajátvektorok kiszámításához mutatunk egy olyan módszert, amihez nem kell mindegyik M_j mátrixot teljesen kiszámolnunk. Nevezzük karakteraltereknek $\mathbb{F}_p^r$ azon altereit, amelyeket néhány irreducibilis karakter generál, vagyis a $\langle \Theta(\chi^i) \mid i \in I \rangle$ alakban felírhatóakat, ahol $I \subseteq \{1, \ldots, r\}$. A sajátvektorokat a következőképpen szeretnénk kiszámolni: kiindulunk egyetlen karakteraltérből (a teljes $\mathbb{F}_p^r$-ről), és minden lépésben egy legalább két dimenziós karakteralteret egy M_j segítségével felbontunk több kisebb dimenziós karakteraltér direkt összegére, úgy hogy a karakteraltérnek nézzük a metszetét M_j sajátaltereivel. Ezt addig ismételjük, amíg csak 1-dimenziós karakteraltereink maradnak, ezek a keresett közös sajátvektorok lesznek. Egy alteret mindig egy sor-redukált bázisként tekintünk és tárolunk, ami azt jelenti, hogy a generátorvektorokat egy mátrix soraiként leírva olyat kapunk, amire teljesül, hogy minden sor első nemnulla eleme 1, valamint ha az i-edik sorban a j-edik elem az első nemnulla elem, akkor minden $i' > i, j' \leq j$-re az (i', j') helyeken 0 van. Egy tetszőlegesen megadott generátorrendszerből Gauss-eliminációval, majd az üres sorok törlésével sor-redukált bázist kaphatunk.

Ahhoz, hogy eldöntsük, hogy melyik M_j-vel érdemes próbálkozni egy adott V karakteraltér felbontásánál, a következő lemmára lesz szükségünk:

3.1. Lemma. *Legyen $b_1, \ldots, b_s$ a V karakteraltér sor-redukált bázisa. Akkor és csak akkor esik V a $\Theta(M_j)$ mátrix egyetlen baloldali sajátalterébe, ha minden olyan V-beli vektorra, aminek az első koordinátája 0, teljesül az, hogy jobbról szorozva $\Theta(M_j)$-vel a kapott vektornak is 0 az első koordinátája.*

Bizonyítás. V karakteraltér, tehát van benne olyan vektor, aminek első koordinátája nemnulla, így b_1 első koordinátája 1. Ha V egy sajátaltérbe esik, akkor minden V-beli vektor sajátvektor, vagyis ha egy vektor első koordinátája 0, akkor a szorzatnak is. A másik irány bizonyításához tegyük fel indirekten, hogy V-be belemetsz két különböző sajátértékhez (λ_1, λ_2) tartozó sajátaltér is. Mivel V és egy sajátaltér metszete karakteraltér, ezért van olyan vektor a metszetben, aminek első koordinátája 1. Legyen egy λ_i-hez tartozó ilyen vektor v_i. $b_1 = v_i + w_i$ alakba írható, ahol w_i első koordinátája 0. Ezt jobbról $\Theta(M_j)$-vel szorozva azt kapjuk, hogy $b_1\Theta(M_j) = = v_i\Theta(M_j) + w_i\Theta(M_j) = \lambda_i v_i + w_i\Theta(M_j)$. Ha csak az első koordinátákat nézzük, akkor azt kapjuk, hogy $b_1\Theta(M_j)$ első koordinátája λ_i. Ezt $i = 1{,}2$-re is megkaptuk, vagyis $\lambda_1 = \lambda_2$, ami ellentmondás. □

Ez azt jelenti, hogy egy $V = \langle b_1, \ldots, b_s \rangle$ karakteralteret M_j-vel akkor tudunk felbontani, ha van olyan V-beli vektor, aminek első koordinátája 0, de $\Theta(M_j)$-vel vett szorzatának első koordinátája nem 0. Ez ekvivalens azzal, hogy létezik $2 \leq \leq i \leq s$, amire $b_i\Theta(M_j)$ első koordinátája nem 0. A szorzat első koordinátája b_i és a mátrix első oszlopának skalárszorzata, és mivel tudjuk, hogy az első oszlopban pontosan egy nemnulla elem van, mégpedig a j'-edik helyen, ezért az kell, hogy b_i j'-edik koordinátája ne legyen 0. Összefoglalva, akkor tudjuk V-t felbontani M_j-vel, ha létezik $2 \leq i \leq s$, amire b_i j'-edik koordinátája nem 0.

Ha $V = \langle b_1, \ldots, b_s \rangle$ sor-redukált alakban van megadva, b_i első nemnulla koordinátája a c_i-edik, akkor a $\Theta(M_j)$ mátrix hatását V-n ki tudjuk számolni csak a $c_1, \ldots, c_s$-edik oszlopok ismeretével. Ugyanis egy V-beli vektort egyértelműen meghatároznak a $c_1, \ldots, c_s$-edik koordinátái, vagyis ha minden $1 \leq i \leq s$-re $b_i\Theta(M_j)$-nek kiszámoljuk a $c_1, \ldots, c_s$-edik koordinátáit, akkor azok alapján a szorzatot fel tudjuk írni b_i-k lineáris kombinációjaként, vagyis ezek alapján az oszlopok alapján fel tudunk írni egy olyan $s \times s$-es A mátrixot, ami a b_i bázisban felírt $\Theta(M_j)$-vel való szorzásnak felel meg. Ennek kell a sajátaltereit meghatároznunk. Ehhez kiszámoljuk a karakterisztikus polinomját, azt $\mathbb{F}_p$ felett faktorizálni tudjuk például a Cantor-Zassenhaus algoritmussal [CZ81], így megkaptuk a λ_i sajátértékeket. $A - \lambda_i I$ baloldali nulltere lesz az i-edik sajátaltér, amit az eredeti bázisba visszaírva megkapjuk V és $\Theta(M_j)$ i-edik baloldali sajátalterének metszetét.

3.5. További módosítások

Ha Schneider második módosítását követjük, akkor segít az algoritmusnak, ha előre ki tudunk számolni pár irreducibilis karaktert. A triviális (csupa 1) karaktert például mindig fel tudjuk használni. Általában megéri, hogy a lineáris karaktereket, illetve azok Θ-általi képeit előre kiszámoljuk. Először nézzük meg, hogyan tudjuk kiszámolni ezeket Abel-csoportokra. (Valójában Abel-csoportoknál csak ezeket kell kiszámolni, hiszen minden irreducibilis karakter lineáris.) Ilyenkor minden elem egy önálló konjugáltosztályt alkot. Ha G egy ciklikus csoport ($G \simeq Z_n$), akkor minden irreducibilis karakter úgy néz ki, hogy a generátorelemhez hozzárendel egy n-edik komplex egységgyököt, a hatványaihoz pedig az egységgyök megfelelő hatványait. Ha G Abel-csoport, akkor a véges Abel-csoportok alaptétele szerint felírható prímhatványrendű ciklikusok direkt szorzataként, így ezt a felbontást megkeresve a tényezők irreducibilis karaktereit ki tudjuk számolni, majd ezek direkt szorzataként megkapjuk G irreducibilis karaktereit. Abban az esetben, ha G nem kommutatív, G abelizálását kell tekintenünk, vagyis a $G^{\text{ab}} = {}^{G}\!/_{G'} = {}^{G}\!/_{[G,G]}$ Abel-csoportnak kell meghatároznunk a karaktertábláját, majd ezen karaktereket G-re visszaemelve megkaphatjuk G lineáris irreducibilis karaktereit.

Lehet bizonyos esetekben gyorsabb módszert találni χ_j^i-k kiszámolására is $\Theta(\chi_j^i)$-kből. Miután d_i-t már kiszámoltuk és látjuk, hogy d_i elég kicsi, akkor a leírt módszernél gyorsabb lehet minden lehetséges d_i darab ζ-hatvány összegére ellenőrizni, hogy melyikekre alkalmazva Θ-t kapjuk a már kiszámolt $\Theta(\chi_j^i)$-t. Ha csak egy lehetséges összeg van, akkor megkaptuk χ_j^i-t. Ez e^{d_i} féle összeg ellenőrzését jelenti, míg a leírt módszer az összes χ_j^i-t $\mathcal{O}(e^4)$ időben kiszámolja, ezért tényleg csak kis d_i-kre érheti meg ez. Ugyanitt másik gyorsítási lehetőség, hogy ha χ^i-t egy konjugáltosztályra kiszámoltuk, akkor az inverz konjugáltosztályon felvett értékét is rögtön tudjuk, hiszen $\chi_{j'}^i = \overline{\chi_j^i}$.

A. Mathematica 8 implementáció

Az alábbi implementáció Mathematica 8 (vagy annál újabb) környezethez készült, anélkül nem használható. Letölthető és installálható [Nag12]-ben leírt módon. Permutációcsoportok kezelésére pár függvény már alapból benne van Mathematica-ban, definiálni tudunk csoportokat, pár alap csoport előre benne van (szimmetrikus, alternáló, ciklikus, diéder és ábel-csoportok, valamint a sporadikus csoportok nagy része). Le tudjuk kérdezni egy csoport erős generátorrendszerét (Schreier-Sims algoritmusnak egy változata bele van írva), valamint egy elem centralizátorát. Ezeken kívül a többi algoritmust az én implementációm biztosítja. A függelék innentől angol nyelven folytatódik.

A.1. License

This package can be used under the rules of the 2-clause BSD license:

Copyright (c) 2011, Gergely Nagy
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.2. Documentation

– NullQ[expr] gives True if expr is Null, and False otherwise.

- Test whether Null is Null:

```
In[1]:= NullQ[Null]
Out[1]:= True
```

- Test whether "Null" is Null:

```
In[2]:= NullQ["Null"]
Out[2]:= False
```

– GroupQ[expr] gives True if expr is a group, and False otherwise.

- Test whether SymmetricGroup[3] is a group:

```
In[1]:= GroupQ[SymmetricGroup[3]]
Out[1]:= True
```

– GroupActionSetSort[actset] sorts the elements of actset into an order in which elements of option GroupActionBase are the first ones and then other elements follow.

- If there are no base given, it is just a normal sort:

```
In[1]:= GroupActionSetSort[{3,1,5,2,7,9,6}]
Out[1]:= {1, 2, 3, 5, 6, 7, 9}
```

- If base is given, then base elements come first:

```
In[2]:= GroupActionSetSort[{3,1,5,2,7,9,6}, GroupActionBase
         -> {2,7,4,6}]
Out[2]:= {2, 7, 6, 1, 3, 5, 9}
```

– CyclesActionSet[elem] gives the action set of a group element.

- Action set for Cycles[1,3,2,7]:

```
In[1]:= CyclesActionSet[Cycles[{{1,3},{2,7}}]]
Out[1]:= {1, 2, 3, 7}
```

- We can use GroupActionBase to specify order:

```
In[2]:= CyclesActionSet[Cycles[{{1,3},{2,7}}],
         GroupActionBase -> {1,7,5,3}]
Out[2]:= {1, 7, 3, 2}
```

– GroupActionSet[group] gives the action set of a group.

  - Action set for a group:

```
In[1]:= GroupActionSet[MathieuGroupM11[]]
Out[1]:= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

  - We can use GroupActionBase to specify order:

```
In[2]:= GroupActionSet[SymmetricGroup[4], GroupActionBase ->
          {3,2}]
Out[2]:= {3, 2, 1, 4}
```

– GroupExponent[group] gives the exponent of the group.

  - Compute the exponent of a group:

```
In[1]:= GroupExponent[DihedralGroup[8]]
Out[1]:= 8
```

  - The exponent of a group is always a divisor of its order:

```
In[2]:= GroupOrder[DihedralGroup[8]]/GroupExponent[
         DihedralGroup[8]]
Out[2]:= 2
```

– GroupElementFromImage[group, a, b] gives an element of the group which moves a to b, or Null if there is no such element.

  - We can get an element of CyclicGroup[5] which moves 1 to 3:

```
In[1]:= GroupElementFromImage[CyclicGroup[5], 1, 3]
Out[1]:= Cycles[{{1, 3, 5, 2, 4}}]
```

  - If there are no such element it gives Null:

```
In[2]:= NullQ[GroupElementFromImage[CyclicGroup[5], 1, 6]]
Out[2]:= True
```

– GroupIrredundantStabilizerChain[group] gives a stabilizer chain of the group according to the option GroupActionBase, but skips redundant base elements.

  - The built-in GroupStabilizerChain function can give redundant stabilizer chain:

```
In[1]:= GroupStabilizerChain[CyclicGroup[5], GroupActionBase
          -> {1, 3}]
Out[1]:= {{} -> PermutationGroup[{Cycles[{{1, 2, 3, 4,
          5}}]}], {1} -> PermutationGroup[{}], {1, 3} ->
          PermutationGroup[{}]}
```

- However, GroupIrredundantStabilizerChain always gives irredundant:

```
In[2]:= GroupIrredundantStabilizerChain[CyclicGroup[5],
         GroupActionBase -> {1, 3}]
Out[2]:= {{} -> PermutationGroup[{Cycles[{{1, 2, 3, 4,
         5}}]}], {1} -> PermutationGroup[{}]}
```

– GroupConjugatesQ[group, elem1, elem2] gives True if elem1 and elem2 are conjugates in the group, and False otherwise.

- Cycles[1,2,3] and Cycles[1,3,2] are conjugates in SymmetricGroup[4]:

```
In[1]:= GroupConjugatesQ[SymmetricGroup[4], Cycles
         [{{1,2,3}}], Cycles[{{1,3,2}}]]
Out[1]:= True
```

- But they are not conjugates in AlternatingGroup[4]:

```
In[2]:= GroupConjugatesQ[AlternatingGroup[4], Cycles
         [{{1,2,3}}], Cycles[{{1,3,2}}]]
Out[2]:= False
```

- We can check conjugacy not just for group elements, in that case it gives True iff there is a group element that conjugates elem1 to elem2:

```
In[3]:= GroupConjugatesQ[SymmetricGroup[4], Cycles
         [{{1,2,3},{5,6}}], Cycles[{{1,3,2},{5,6}}]]
Out[3]:= True
```

– GroupConjugacyClassRepresentatives[group] gives a list of group elements which represent the conjugacy classes.

- We can get class representatives for a group:

```
In[1]:= GroupConjugacyClassRepresentatives[AlternatingGroup
         [4]]
Out[1]:= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
         3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupNumOfConjugacyClasses[group] gives the number of conjugacy classes in the group.

- The number of conjugacy classes in SymmetricGroup[n] is always PartitionsP[n]:

```
In[1]:= Table[GroupNumOfConjugacyClasses[SymmetricGroup[n]],
          {n, 6}]
Out[1]= {1, 2, 3, 5, 7, 11}
In[2]:= Table[PartitionsP[n], {n, 6}]
Out[2]= {1, 2, 3, 5, 7, 11}
```

– GroupConjugacyClassSizes[group] gives the list of sizes of the conjugacy classes (in the same order as GroupConjugacyClassRepresentatives[group] gives the elements.

- We can get sizes of conjugacy classes:

```
In[1]:= GroupConjugacyClassSizes[AlternatingGroup[4]]
Out[1]= {1, 3, 4, 4}
```

- The ordering corresponds to GroupConjugacyClassRepresentatives:

```
In[2]:= GroupConjugacyClassRepresentatives[AlternatingGroup
          [4]]
Out[2]= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
          3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupConjugacyClassInverses[group] gives the list whose k-th element is the index of the conjugacy class in which the inverses of the elements of the k-th conjugacy class are.

- We can get indices of inverse conjugacy classes:

```
In[1]:= GroupConjugacyClassInverses[AlternatingGroup[4]]
Out[1]= {1, 2, 4, 3}
```

- The ordering corresponds to GroupConjugacyClassRepresentatives:

```
In[2]:= GroupConjugacyClassRepresentatives[AlternatingGroup
          [4]]
Out[2]= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
          3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupConjugacyClassNum[group, elem] gives the index of the conjugacy class of elem in group.

- We can get index of the conjugacy class of an element:

```
In[1]:= GroupConjugacyClassNum[AlternatingGroup[4], Cycles
          [{{1,4,2}}]]
Out[1]= 4
```

- The index corresponds to GroupConjugacyClassRepresentatives:

```
In[2]:= GroupConjugacyClassRepresentatives[AlternatingGroup
          [4]]
Out[2]:= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
          3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupConjugacyClass[group, n] gives the full list of elements in the n-th conjugacy class.

- We can get a full conjugacy class:

```
In[1]:= GroupConjugacyClass[AlternatingGroup[4], 4]
Out[1]:= {Cycles[{{1, 4, 2}}], Cycles[{{2, 4, 3}}], Cycles
          [{{1, 2, 3}}], Cycles[{{1, 3, 4}}]}
```

- The ordering corresponds to GroupConjugacyClassRepresentatives:

```
In[2]:= GroupConjugacyClassRepresentatives[AlternatingGroup
          [4]]
Out[2]:= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
          3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupCharacterScalarProduct[group, chi, psi] gives the scalar product of two characters (chi and psi) of the group given by a list of values in the conjugacy classes.

- We can compute scalar product of two characters:

```
In[1]:= GroupCharacterScalarProduct[AlternatingGroup[4], {1,
          1, 1, 1}, {4, 0, 0, 0}]
Out[1]:= 1/3
```

- Irreducible characters form an ortonormal base in the space of characters:

```
In[2]:= g = AlternatingGroup[4]; tbl = GroupCharacterTable[g
          ]; FullSimplify[Table[GroupCharacterScalarProduct[g
          , tbl[[i]], tbl[[j]]], {i, Length[tbl]}, {j, Length
          [tbl]}]]
Out[2]:= {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
          1}}
```

- We can compute it over a finite field:

```
In[2]:= GroupCharacterScalarProduct[AlternatingGroup[4], {1,
          1, 1, 1}, {4, 0, 0, 0}, Modulus -> 7]
Out[2]:= 5
```

– GroupDixonPrime[group] gives the smallest prime number (p) such that GF[p] can be used to represent all the complex characters in.

  - It gives the smallest p prime of the form k*GroupExponent[group]+1 such that p > 2*Sqrt[GroupOrder[group]]:

```
In[1]:= GroupExponent[MathieuGroupM11[]]
Out[1]= 1320
In[2]:= N[2*Sqrt[GroupOrder[MathieuGroupM11[]]]]
Out[2]= 177.989
In[3]:= GroupDixonPrime[MathieuGroupM11[]]
Out[3]= 1321
```

  - Another example:

```
In[4]:= GroupDixonPrime[CyclicGroup[7]]
Out[4]= 29
```

– GroupCharacterTableOverFiniteField[group] gives the character table of the group over GF[p] where p is given by GroupDixonPrime[group].

  - We can calculate the character table of a group over GF[GroupDixonPrime[group]]:

```
In[1]:= GroupDixonPrime[AlternatingGroup[4]]
Out[1]= 7
In[2]:= MatrixForm[GroupCharacterTableOverFiniteField[
          AlternatingGroup[4]]]
Out[2]= {{1, 1, 1, 1}, {1, 1, 2, 4}, {1, 1, 4, 2}, {3, 6, 0,
          0}}
In[3]:= MatrixForm[GroupCharacterTable[AlternatingGroup[4]]]
Out[3]= {{1, 1, 1, 1}, {1, 1, (-1)^(2/3), -(-1)^(1/3)}, {1,
          1, -(-1)^(1/3), (-1)^(2/3)}, {3, -1, 0, 0}}
```

  - The ordering of the columns corresponds to GroupConjugacyClassRepresentatives:

```
In[4]:= GroupConjugacyClassRepresentatives[AlternatingGroup
          [4]]
Out[4]= {Cycles[{}], Cycles[{{1, 4}, {2, 3}}], Cycles[{{1,
          3, 2}}], Cycles[{{1, 3, 4}}]}
```

– GroupCharacterTable[group] gives the character table of the group.

  - We can calculate the character table of a group:

```
In[1]:= GroupCharacterTable[DihedralGroup[5]]
Out[1]= {{1, 1, 1, 1}, {1, 1, 1, -1}, {2, (-1 - Sqrt[5])/2,
         (-1 + Sqrt[5])/2, 0}, {2, (-1 + Sqrt[5])/2, (-1 -
         Sqrt[5])/2, 0}}
```

- The ordering of the columns corresponds to GroupConjugacyClassRepresentatives:

```
In[2]:= GroupConjugacyClassRepresentatives[DihedralGroup[5]]
Out[2]= {Cycles[{}], Cycles[{{1, 2, 3, 4, 5}}], Cycles[{{1,
         3, 5, 2, 4}}], Cycles[{{1, 3}, {4, 5}}]}
```

A.3. Source code

```
1  BeginPackage["GroupExt`"]
2
3  (* list of public functions with usages *)
4  NullQ::usage = "NullQ[expr] gives True if expr is Null, and False
       otherwise."
5  GroupQ::usage = "GroupQ[expr] gives True if expr is a group, and False
       otherwise."
6  GroupActionSetSort::usage = "GroupActionSetSort[actset] sorts the
       elements of actset into an order in which elements of option
       GroupActionBase are the first ones and then other elements follow."
7  CyclesActionSet::usage = "CyclesActionSet[elem] gives the action set of
        a group element."
8  GroupActionSet::usage = "GroupActionSet[group] gives the action set of
       a group."
9  GroupExponent::usage = "GroupExponent[group] gives the exponent of the
       group."
10 GroupElementFromImage::usage = "GroupElementFromImage[group, a, b]
       gives an element of the group which moves a to b, or Null if there
       is no such element."
11 GroupIrredundantStabilizerChain::usage = "
       GroupIrredundantStabilizerChain[group] gives a stabilizer chain of
       the group according to the option GroupActionBase, but skips
       redundant base elements."
12 GroupConjugatesQ::usage = "GroupConjugatesQ[group, elem1, elem2] gives
       True if elem1 and elem2 are conjugates in the group, and False
       otherwise."
13 GroupConjugacyClassRepresentatives::usage = "
       GroupConjugacyClassRepresentatives[group] gives a list of group
       elements which represent the conjugacy classes."
14 GroupNumOfConjugacyClasses::usage = "GroupNumOfConjugacyClasses[group]
       gives the number of conjugacy classes in the group."
15 GroupConjugacyClassSizes::usage = "GroupConjugacyClassSizes[group]
       gives the list of sizes of the conjugacy classes (in the same order
        as GroupConjugacyClassRepresentatives[group] gives the elements."
16 GroupConjugacyClassInverses::usage = "GroupConjugacyClassInverses[group
       ] gives the list whose k-th element is the index of the conjugacy
       class in which the inverses of the elements of the k-th conjugacy
       class are."
17 GroupConjugacyClassNum::usage = "GroupConjugacyClassNum[group, elem]
       gives the index of the conjugacy class of elem in group."
18 GroupConjugacyClass::usage = "GroupConjugacyClass[group, n] gives the
       full list of elements in the n-th conjugacy class."
```

```
19 GroupCharacterScalarProduct::usage = "GroupCharacterScalarProduct[group
     , chi, psi] gives the scalar product of two characters (chi and psi
     ) of the group given by a list of values in the conjugacy classes."
20 GroupDixonPrime::usage = "GroupDixonPrime[group] gives the smallest
     prime number (p) such that GF[p] can be used to represent all the
     complex characters in."
21 GroupCharacterTableOverFiniteField::usage = "
     GroupCharacterTableOverFiniteField[group] gives the character table
      of the group over GF[p] where p is given by GroupDixonPrime[group
     ]."
22 GroupCharacterTable::usage = "GroupCharacterTable[group] gives the
     character table of the group."
23
24 Begin["GroupExt`Private`"]
25
26 (* there is no proper way to determine if something is null *)
27 NullQ[x_] := ToString[x] == "Null" && !StringQ[x]
28
29 (* only way to check if something is a group is with a list like this
     *)
30 GroupQ[g_] := MemberQ[{
31   PermutationGroup, GroupStabilizer, GroupSetwiseStabilizer,
       GroupCentralizer,
32   SymmetricGroup, AlternatingGroup, CyclicGroup, AbelianGroup,
33   DihedralGroup, MathieuGroupM11, MathieuGroupM12, MathieuGroupM22,
34   MathieuGroupM23, MathieuGroupM24, JankoGroupJ1, JankoGroupJ2,
35   JankoGroupJ3, JankoGroupJ4, HigmanSimsGroupHS, ConwayGroupCo1,
36   ConwayGroupCo2, ConwayGroupCo3, McLaughlinGroupMcL, SuzukiGroupSuz,
37   HeldGroupHe, RudvalisGroupRu, FischerGroupFi22, FischerGroupFi23,
38   FischerGroupFi24Prime, TitsGroupT, ONanGroupON, HaradaNortonGroupHN,
39   ThompsonGroupTh, LyonsGroupLy, BabyMonsterGroupB, MonsterGroupM
40 }, Head[g]]
41
42 (* in Mathematica 8.0.0 there is a bug that crashes GroupCentralizer[]
     if called with the identity element (fixed in 8.0.1) *)
43 Off[General::shdw]
44 GroupExt`GroupCentralizer[g_?GroupQ, x_Cycles] := If[Cycles[{}] == x, g
     , System`GroupCentralizer[g, x]]
45 On[General::shdw]
46
47 (* we extend some operators here to work with permutations and groups
     *)
48 ClearAttributes[NonCommutativeMultiply, Protected]
```

```
49    (* multiplication of two permutations with ** operator (we can't use
          *, because Mathematica assumes * is commutative), example: Cycles
          [{{1,2}}]** Cycles[{{2,3}}] = Cycles[{{1,3,2}}] *)
50    NonCommutativeMultiply[x_Cycles, y_Cycles] := PermutationProduct[x, y
          ]
51  SetAttributes[NonCommutativeMultiply, Protected]
52
53  ClearAttributes[Power, Protected]
54    (* raising permutations to a power with ^ operator, example: Cycles
          [{{1,2}}]^3 = Cycles[{{1,2}}] *)
55    Power[x_Cycles, n_Integer] := PermutationPower[x, n]
56
57    (* determine where an element is moved by a permutation with ^
          operator, example 1^Cycles[{{1,2}}] = 2 *)
58    Power[n_Integer, a_Cycles] := PermutationReplace[n, a]
59
60    (* calculate orbit of an element with ^ operator, example 1^
          PermutationGroup[{ Cycles[{{1,2}}], Cycles[{{3,4}}]}] = {1,2} *)
61    Power[n_Integer, g_?GroupQ] := GroupOrbits[g, {n}][[1]]
62
63    (* conjugaction of elements with ^ operator, example: Cycles
          [{{1,2}}]^Cycles[{{2,3}}] = Cycles[{{1,3}}] *)
64    Power[x_Cycles, y_Cycles] := y^(-1)**x**y
65  SetAttributes[Power, Protected]
66
67  (* it can sort a subset of the group's action set by a base *)
68  Options[GroupActionSetSort] = {GroupActionBase -> {}}
69  GroupActionSetSort[actset_List, OptionsPattern[]] := Module[{base},
70    base = OptionValue[GroupActionBase];
71    Join[Sort[Intersection[actset, base], (Position[base, #1][[1,1]] <
          Position[base, #2][[1,1]])&], Sort[Complement[actset, base]]]
72  ]
73
74  (* it gives the set a groupelement acts on (listable) *)
75  CyclesActionSet[a_Cycles, opts:OptionsPattern[GroupActionSetSort]] :=
        GroupActionSetSort[Flatten[a[[1]]], opts]
76  SetAttributes[CyclesActionSet, Listable]
77
78  (* it gives the set the group acts on *)
79  GroupActionSet[g_?GroupQ, opts:OptionsPattern[GroupActionSetSort]] :=
        GroupActionSetSort[Apply[Union, CyclesActionSet[GroupGenerators[g
          ]]], opts]
80
```

```
(* exponent is the least common multiplier of orders of the class
    representatives *)
GroupExponent[g_?GroupQ] := GroupExponent[g] = Apply[LCM, Map[
    PermutationOrder, GroupConjugacyClassRepresentatives[g]]]

(* breadth-first-search for an element that moves a to b *)
GroupElementFromImage[g_?GroupQ, a_Integer, b_Integer] := Module[{lk =
    1, list, c, x, i},
  Catch[
    (* if a == b then the identity is good *)
    If[a == b, Throw[Cycles[{}]]];
    s = GroupGenerators[g];
    (* we start bfs from the identity which moves a to a *)
    list = {Cycles[{}] -> a};
    lk = 1;
    (* while we have new elements where we can move a to *)
    While[lk <= Length[list],
      (* we try every generator *)
      Do[
        (* we compute the new group element (c), and where that moves a
            to (x) *)
        c = list[[lk, 1]]**s[[i]];
        x = list[[lk, 2]]^s[[i]];
        (* if it moves a to b then we're done *)
        If[x == b, Throw[c]];
        (* if we couldn't get to x yet then we add c->x to our list *)
        If[Count[list, x, 2] == 0, list = Append[list, c -> x]]
      , {i, Length[s]}];
      lk = lk + 1
    ];
    (* if we can't get to b then we return Null *)
    Null
  ]
]

(* the builtin GroupStabilizerChain can give redundant base *)
GroupIrredundantStabilizerChain[g_?GroupQ, opts:OptionsPattern[
    GroupStabilizerChain]] := Module[{sc, ret, i},
  (* first we call the original version *)
  sc = GroupStabilizerChain[g, opts];
  (* the first element is always necessary *)
  ret = {First[sc]};
  Do[
```

```
119       (* we add an element if the group is not the same as the previously
              added, and we only add the last base element to the list *)
120       If[ret[[-1, 2, 1]] != sc[[i, 2, 1]], ret = Append[ret, Append[ret
              [[-1, 1]], sc[[i, 1, -1]]] -> sc[[i, 2]]]]
121     , {i, 2, Length[sc]}];
122     ret
123   ]
124
125   (* private function: Sifting procedure to get an element from some base
          images *)
126   GroupElementFromBaseImages[sc_, base_, img_, imgn_] := Module[{i, x,
        ret},
127     Catch[
128       (* we start with the identity *)
129       ret = Cycles[{}];
130       (* we iterate through given images *)
131       Do[
132         (* we look for an element that stabilizes the first i-1 base
              elements but moves base[[i]] to img[[i]]^(ret^(-1)) (this way
               x**ret moves base[[i]] to img[[i]] *)
133         x = GroupElementFromImage[sc[[i,2]], base[[i]], img[[i]]^(ret
              ^(-1))];
134         (* if there are no such elemet then we return Null *)
135         If[NullQ[x], Throw[Null]];
136         (* we continue with x**ret *)
137         ret = x**ret
138       , {i, imgn}];
139       (* if we are done with all the given images then ret is good *)
140       ret
141     ]
142   ]
143
144   (* we use a backtrack method to check if a and b are conjugates *)
145   GroupConjugatesQ[g_?GroupQ, a_Cycles, b_Cycles] := Module[{acycles,
        bcycles, sc, base, pos, p, borbitfirst}, Catch[
146     (* we sort the cycles of a and b by length *)
147     acycles = Sort[a[[1]], (Length[#1] > Length[#2]) &];
148     bcycles = Sort[b[[1]], (Length[#1] > Length[#2]) &];
149     (* if a and b has cycles of different length then they are not
          conjugates *)
150     If[Map[Length, acycles] != Map[Length, bcycles], Throw[False]];
151     (* we compute the stabilizer chain with a base in which elements of a
          's cycles  *)
```

```
152     sc = GroupIrredundantStabilizerChain[g, GroupActionBase -> Flatten[
            acycles]];
153     (* we compute its base *)
154     base = sc[[-1, 1]];
155     (* we compute positions of base elements in a, {x, y, z} means it's
            the y-th in the x-th cycle (which is z long); {0, 0, 1} means it'
            s stabilized by a *)
156     pos = Map[(p = Position[a, #, {3}]; If[Length[p] == 0, {0, 0, 1}, {p
            [[1, 2]], p[[1, 3]], Length[a[[1, p[[1, 2]]]]]}])&, base];
157     (* we compute <b>'s orbits' first elements *)
158     borbitfirst = Union[Complement[GroupActionSet[g], Flatten[b[[1]]],
            Map[First[GroupActionSetSort[#, GroupActionBase -> base]]&, b
            [[1]]]];
159     (* we call our backtrack function with initially no base images *)
160     GroupConjugatesQBT[a, b, sc, base, Length[base], {}, 0, pos,
            borbitfirst]
161 ]]
162
163 (* private function: the backtrack function *)
164 GroupConjugatesQBT[a_, b_, sc_, base_, basen_, img_, imgn_, pos_,
        borbitfirst_] := Module[{try, elem, l, p}, Catch[
165     (* elem is an element whose base images start with img *)
166     elem = GroupElementFromBaseImages[sc, base, img, imgn];
167     (* if we have all the images then we check if it's good *)
168     If[imgn == basen, Throw[a^elem == b]];
169     (* l is the next base element's number *)
170     l = imgn+1;
171     (* try will be the list of possible images for the next base element
            *)
172     (* if base[[l]] is in the same cycle in a as base[[imgn]] then we
            know the next image *)
173     If[imgn > 0 && pos[[l, 1]] > 0 && pos[[l, 1]] == pos[[imgn, 1]],
174       p = Position[b, img[[imgn]], {3}];
175       try = {b[[1, p[[1, 2]], Mod[pos[[l, 2]]-pos[[imgn, 2]]+p[[1, 3]]-1,
                pos[[l, 3]]]+1]]}
176     , (* else *)
177       (* the elements whose images start with img are a coset of sc[[imgn
              +1, 2]] and elem is one of them, so these are the possible
              images for base[[imgn+1]] *)
178       try = (base[[l]]^sc[[l, 2]])^elem;
179       (* if base[[l]] is stabilized by a then img[[l]] must be stabilized
              by b *)
180       If[pos[[l, 1]] == 0,
181         try = Select[try, (Count[b, #, {3}] == 0)&]
```

```
182       , (* else *)
183         (* otherwise cycle-length must be the same *)
184         try = Select[try, (p = Position[b, #, {3}]; Length[p] > 0 &&
                Length[b[[1, p[[1, 2]]]]] == pos[[l, 3]])&]
185       ];
186       (* if b stabilizer all images thus far (if borbitfirst != {}) then
             we only have to check <b>'s orbits' first element *)
187       If[Length[borbitfirst] > 0, try = Intersection[try, borbitfirst]];
188     ];
189     (* we try every possible image for base[[l]] *)
190     Do[
191       (* if it's good then we return True *)
192       If[GroupConjugatesQBT[a, b, sc, base, basen, Append[img, try[[i]]],
              l, pos, If[Count[b, try[[i]], {3}] == 0, borbitfirst, {}]],
            Throw[True]]
193     , {i, Length[try]}];
194     (* if we didn't return True then it's False *)
195     False
196   ]]
197
198   (* We find conjugacy classes by testing random elements, each taken
        from the last element's centralizer *)
199   GroupConjugacyClassRepresentatives[g_?GroupQ] :=
        GroupConjugacyClassRepresentatives[g] = Module[{repr, n, sum, x, k,
         cent, centorder},
200     n = GroupOrder[g];
201     (* at the beginning we only know the identity as a group
          representative *)
202     (* at each step, we have a group element (x), calculate its
          centralizer (cent) and its order (centorder) *)
203     x = Cycles[{}];
204     cent = g;
205     centorder = n;
206     (* we will store the representatives in repr *)
207     repr = {x};
208     (* k is the number of already found conjugacy classes *)
209     k = 1;
210     (* sum is the total number of elements in the known classes *)
211     sum = 1;
212     (* we are done when we find all elements, so we repeat until sum == n
          *)
213     While[sum < n,
214       (* we take 3 random elements before actually testing it *)
215       Do[
```

```
216         (* we get a random element from the last elements centralizer *)
217           x = First[GroupElements[cent, {RandomInteger[{1, centorder
                }]}]];
218         cent = GroupCentralizer[g, x];
219         centorder = GroupOrder[cent];
220       , {3}];
221       Catch[
222         (* we step out of the Catch[] block if x is in an already known
              conjugacy class *)
223         Do[If[GroupConjugatesQ[g, repr[[i]], x], Throw[True]], {i, 1, k
              }];
224         (* if we are here, we found a new class *)
225         k = k+1;
226         repr = Append[repr, x];
227         sum = sum + n/centorder
228       ]
229     ];
230     repr
231   ]
232
233   (* we determine numbers of conjugacy classes by computing the
        representatives and counting them *)
234   GroupNumOfConjugacyClasses[g_?GroupQ] := GroupNumOfConjugacyClasses[g]
        = Length[GroupConjugacyClassRepresentatives[g]]
235
236   (* we compute |G : C_G(a)| for all representatives and return them in a
        list *)
237   GroupConjugacyClassSizes[g_?GroupQ] := GroupConjugacyClassSizes[g] =
        Module[{n},
238     n = GroupOrder[g];
239     Map[(n/GroupOrder[GroupCentralizer[g, #]])&,
          GroupConjugacyClassRepresentatives[g]]
240   ]
241
242   (* we compute indices of the conjugacy classes' inverses *)
243   GroupConjugacyClassInverses[g_?GroupQ] := GroupConjugacyClassInverses[g
        ] = Map[GroupConjugacyClassNum[g, #^(-1)]&,
        GroupConjugacyClassRepresentatives[g]]
244
245   (* we determine number of the conjugacy class of a specific element by
        computing representatives and then try if they are conjugates *)
246   GroupConjugacyClassNum[g_?GroupQ, a_Cycles] := Module[{repr},
247     (* we calculate representatives *)
248     repr = GroupConjugacyClassRepresentatives[g];
```

```
249     Catch[
250       (* we iterate over them *)
251       Do[
252         (* if a~repr[[i]] then we return i *)
253         If[GroupConjugatesQ[g, a, repr[[i]]], Throw[i]]
254       , {i, Length[repr]}];
255       (* if we didn't found it then a is not in g *)
256       Null
257     ]
258   ]
259
260   (* breadth-first-search for elements in the k-th conjugacy class *)
261   GroupConjugacyClass[g_?GroupQ, k_Integer] := GroupConjugacyClass[g, k]
        = Module[{list, i, next, s},
262     (* we calculate the generators *)
263     s = GroupGenerators[g];
264     (* list contains found elements *)
265     list = {GroupConjugacyClassRepresentatives[g][[k]]};
266     (* next-th element of the list is coming *)
267     next = 1;
268     (* we go through the elements of the list *)
269     While[next <= Length[list],
270       Do[
271         (* x is the new element we get by conjugating the next-th element
              with the i-th generator *)
272         x = list[[next]]^s[[i]];
273         (* if we haven't found x yet then we add it to the list *)
274         If[Count[list, x] == 0, list = Append[list, x]]
275       , {i, Length[s]}];
276       next = next+1
277     ];
278     list
279   ]
280
281   (* private function: MT_i(k, j) = $|\{(a,b) \in G_1 \times G_2 \mid a \
        in C_i, b \in C_j, a*b=g_k\}|$ *)
282   GroupMTTableRow[g_, i_, k_] := GroupMTTableRow[g, i, k] = Module[{repr,
        inv, ret, iclass, j, l},
283     (* we calculate the representatives and inverses *)
284     repr = GroupConjugacyClassRepresentatives[g];
285     inv = GroupConjugacyClassInverses[g];
286     (* we will return a Length[repr] long array, initially it's all zero
          *)
287     ret = ConstantArray[0, Length[repr]];
```

```
288     (* if i == 1 then MT_i is the identity matrix, so in its kth row
            there is only one non-zero element, the kth *)
289     If[i == 1, Return[ReplacePart[ret, k -> 1]]];
290     (* the first row always has only one non-zero element, the inv[[i]]-
            th element is the size of its conjugacy class *)
291     If[k == 1, Return[ReplacePart[ret, inv[[i]] ->
            GroupConjugacyClassSizes[g][[i]]]]];
292     (* otherwise we compute an element from the kth class and iterate
            through the ith class *)
293     iclass = GroupConjugacyClass[g, i];
294     Do[
295       j = GroupConjugacyClassNum[g, iclass[[l]]^(-1)**repr[[k]]];
296       (* we increase the j-th element by 1 *)
297       ret = ReplacePart[ret, j -> ret[[j]]+1]
298     , {l, Length[iclass]}];
299     ret
300   ]
301
302   (* we calculate scalar product as 1/|G|\sum_{g \in G}a(g)b(g^-1) and
        not using conjugates as in the definition, because it works with
        finite fields as well *)
303   Options[GroupCharacterScalarProduct] = {Modulus -> 0}
304   GroupCharacterScalarProduct[g_?GroupQ, a_, b_, OptionsPattern[]] :=
        Module[{sizes, inv, mod, ret},
305     sizes = GroupConjugacyClassSizes[g];
306     inv = GroupConjugacyClassInverses[g];
307     mod = OptionValue[Modulus];
308     (* we calculate the sum *)
309     ret = Sum[sizes[[i]]*a[[i]]*b[[inv[[i]]]], {i, Length[sizes]}];
310     (* we have to divide it by GroupOrder[g] *)
311     If[mod != 0, Mod[ret*PowerMod[GroupOrder[g], -1, mod], mod], ret/
            GroupOrder[g]]
312   ]
313
314   (* we search for the smallest prime with e|p-1 and p > 2*sqrt(|G|) *)
315   GroupDixonPrime[g_?GroupQ] := GroupDixonPrime[g] = Module[{p, e},
316     (* e is the exponent *)
317     e = GroupExponent[g];
318     (* p is the first number that e|p-1 and p > 2*sqrt(|G|) *)
319     p = (Floor[(2*Floor[Sqrt[GroupOrder[g]]]-1)/e]+1)*e+1;
320     (* while p is not a prime, we increase it by e *)
321     While [!PrimeQ[p], p = p + e];
322     p
323   ]
```

```
(* private function: we try to split V into direct sum of smaller
    subspaces based on GroupMTTable[g, i] *)
GroupCharacterTableSplit[g_, i_, V_] := Module[{r, p, x, Mp, bp, id, j,
    s, iinv, A, c, im, eigenvalues, lin},
  s = Length[V];
  (* if V is 1-dimensional, we cannot split it *)
  If[s == 1, Return[{V}]];
  r = GroupNumOfConjugacyClasses[g];
  p = GroupDixonPrime[g];
  iinv = GroupConjugacyClassInverses[g][[i]];
  (* if all basevectors but the first has 0 at position iinv, then we
      cannot split *)
  If[Count[Map[#[[iinv]]&, Rest[V]], 0] == Length[V]-1, Return[{V}]];
  (* we compute the place of leading non-zero elements of the
      basevectors *)
  c = Map[Position[#, Except[0], 1, Heads -> False][[1, 1]]&, V];
  (* and the basevectors at c places *)
  bp = Table[Map[V[[j, #]]&, c], {j, s}];
  (* we need these rows of MT *)
  Mp = Mod[Map[GroupMTTableRow[g, i, #]&, c], p];
  (* we construct an A matrix such that MT.V^T = V^T.A where MT is
      GroupMTTable[g, i] *)
  A = Transpose[Map[Function[{b},
    (* first we compute the image of the basevector, but only the
        values at the c places *)
    im = Mod[b.Transpose[Mp], p];
    (* next we want to have im as linear combination of basevectors *)
    lin = {};
    (* we do something like Gauss-elimination to get the coefficients (
        it works because V is a row-reduced base) *)
    Do[
      lin = Append[lin, im[[j]]];
      im = Mod[im-im[[j]]*bp[[j]], p]
    , {j, s}];
    lin
  ], V]];
  (* we find A's eigenvalues *)
  eigenvalues = Map[#[[1,2]]&, Union[Solve[CharacteristicPolynomial[A,
      x] == 0, x, Modulus -> p]]];
  (* for each eigenvalue we find its eigenspace, and calculate its
      generators in the normal coordinate system *)
  id = IdentityMatrix[s];
```

```
358     Select[Map[RowReduce[NullSpace[A - #*id, Modulus -> p].V, Modulus ->
            p]&, eigenvalues], (Length[#] > 0)&]
359 ]
360
361 (* we determine the character table over F_p by figuring out the common
        eigenvectors of the MT matrices *)
362 GroupCharacterTableOverFiniteField[g_?GroupQ] :=
      GroupCharacterTableOverFiniteField[g] = Module[{subspaces, r, i, p
      },
363   r = GroupNumOfConjugacyClasses[g];
364   p = GroupDixonPrime[g];
365   (* initially we have the whole vectorspace as the single subspace *)
366   subspaces = {IdentityMatrix[r]};
367   (* we split subspaces into direct sums until each is 1-dimensional *)
368   Do[
369     subspaces = Apply[Union, Map[GroupCharacterTableSplit[g, i, #]&,
          subspaces]]
370   , {i, 2, r}];
371   (* we calculate the characters from the eigenvectors and then sort
        them *)
372   Sort[Map[
373     (* we have to multiply each eigenvector with a constant such that
          their norms becomes 1 *)
374     Mod[#*PowerMod[GroupCharacterScalarProduct[g, #, #, Modulus -> p],
          -1/2, p], p]&,
375     Table[subspaces[[i, 1]], {i, r}]
376   ]]
377 ]
378
379 (* we use the finite table to compute the normal one *)
380 GroupCharacterTable[g_?GroupQ] := GroupCharacterTable[g] = Module[{e,
      einv, r, p, t, s, i, j, k, l, repr, fin, reprl, inv, skip, row},
381   r = GroupNumOfConjugacyClasses[g];
382   p = GroupDixonPrime[g];
383   e = GroupExponent[g];
384   repr = GroupConjugacyClassRepresentatives[g];
385   inv = GroupConjugacyClassInverses[g];
386   (* we compute the finite version first *)
387   fin = GroupCharacterTableOverFiniteField[g];
388   (* we can skip computation of those columns whose inverses' column is
        already computed, because we can conjugate those *)
389   skip = Table[inv[[i]] < i, {i, r}];
390   (* t is a complex e-th root of unity *)
391   t = (-1)^(2/e);
```

```
392     (* s is an element which has order e in F_p *)
393     s = PowerMod[PrimitiveRoot[p], (p-1)/e, p];
394     (* we precompute some expressions that will be used a lot *)
395     reprl = Table[GroupConjugacyClassNum[g, repr[[j]]^k], {j, r}, {k, e
            }];
396     einv = PowerMod[e, -1, p];
397     (* we finally compute the character table, and then try to simplify
            the result for at most 10 seconds *)
398     FullSimplify[Table[
399       row = {};
400       Do[
401         (* we write the elements of the character table as a polynomial
                of t, row contains the coefficients *)
402         (* we can compute them with this sum (or we reverse the
                coefficients of the inverse column for conjugation) *)
403         row = Append[row, If[skip[[j]], RotateLeft[Reverse[row[[inv[[j
                ]]]]]], Table[Mod[einv*Sum[fin[[i, reprl[[j, l]]]]*PowerMod[s
                , -k*l, p], {l, e}], p], {k, e}]]]
404       , {j, r}];
405       (* now we sum it *)
406       Table[Sum[row[[j, k]]*t^k, {k, e}], {j, r}]
407     , {i, r}], TimeConstraint -> 10]
408 ]
409
410 End[]
411 EndPackage[]
```

Hivatkozások

[BS84] L. Babai and E. Szemerédi. On the complexity of matrix group problems i. In *25th Ann. Symp. on Found. of Comp. Sci.*, pages 229–240, 1984.

[Bur11] W. Burnside. *Theory of Groups of Finite Order*. Cambridge University Press. Reprinted by Dover 1955, New York, 2nd edition, 1911.

[Con90] S. B. Conlon. Calculating characters of p-groups. *J. Symb. Comput.*, 9(5/6):535–550, 1990.

[CZ81] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. of Comput.*, 36(154):587–592, 1981.

[Dix67] J. D. Dixon. High speed computation of group characters. *Numerische Mathematik*, 10(5):446–450, 1967.

[HEO05] D. F. Holt, B. Eick, and E. A. O'Brien. *A Handbook of Computational Group Theory*. Chapman and Hall, 2005.

[Jer95] M. Jerrum. Computational Pólya theory. In *In Surveys in combinatorics*, pages 103–118. University Press, 1995.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[Luk93] E. M. Luks. Permutation groups and polynomial-time computations. In *Groups and computation*, volume 11 of *DIMACS*, pages 139–175. Amer. Math. Soc., 1993.

[Mur03] S. H. Murray. The Schreier-Sims algorithm, November 2003.

[Nag12] G. G. Nagy. GroupExt - Group Theory Extensions for Mathematica 8. `https://github.com/ngg/groupext`, 2012.

[Sch90] G. J. A. Schneider. Dixon's character table algorithm revisited. *J. Symb. Comput.*, 9(5/6):601–606, 1990.

[Ser03] Á. Seress. *Permutation Group Algorithms*. Cambridge University Press, 2003.

[Sim70] C. C. Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183, 1970.

[Sla86] M. C. Slattery. Computing character degrees in p-groups. *J. Symb. Comput.*, 2(1):51–58, 1986.

[Ung06] W. R. Unger. Computing the character table of a finite group. *J. Symb. Comput.*, 41(8):847–862, 2006.