NYILATKOZAT

Név: Rózsás Tamás

ELTE Természettudományi Kar, szak: matematika Bsc, alkalmazott matematikus

NEPTUN azonosító: Q6C430

Szakdolgozat címe: PERMUTÁCIÓCSOPORT ALGORITMUSOK

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2020.05.30

Rózsás Tamás

a hallgató aláírása

EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

Permutációcsoport Algoritmusok

Rózsás Tamás
BSc szakdolgozat

Témavezető:
Halasi Zoltán
Algebra és Számelmélet tanszék

Budapest,2020

Köszönetnyilvánítás

Hálás köszönetemet szeretném kifejezni Halasi Zoltánnak, akinek segítő munkája nélkül a szakdolgozat nem készülhetett volna el. Köszönöm a téma felvetését, a szakirodalmi ajánlásokat, a hasznos formai és tartalmi tanácsokat és minden egyéb támogatást.

Köszönöm családomnak és barátaimnak, akik mindig mellettem álltak és támogattak, egyetemen belül és kívül is, és külön hálás vagyok a barátnőmnek, aki segített átnézni a dolgozatot és kijavítani a hibákat. Köszönet illeti középiskolai tanáraimat, akik munkája és iránymutatása is hozzájárult mindehhez. Végül köszönöm kollégáimnak a támogatást és a biztató szavakat.

Tartalomjegyzék

Bevezető

A csoportelmélet első eredményeinek általában Cauchy és Galois 1846-os publikációinak anyagát tekintik. Annak ellenére, hogy az algoritmikus csoportelmélet főleg a 20. század közepe és vége felé kezdett el egyre elismertebb és fontosabb szerepet betölteni a témakörben, fontos eredmények már a század első felében is születtek. A permutációcsoportok fogalma hamar megszületett, majd annak segítségével Galois bizonyítani tudta tételét az algebrai egyenletek gyökképlettel való megoldásáról. Innen eredeztethető a csoportok absztrakt fogalma. A csoportelmélet megalapozásában Arthur Caleynek szerepét érdemes még kiemelni, hiszen tőle származik a csoportok ma is használt definíciója, továbbá bizonyította reprezentációs tételét, mely szerint minden csoport izomorf egy permutációcsoporttal. Ezekre építkezve számos kiváló matematikus segítségével juthattunk el Charles Simshez, akinek 1970-ben megjelent cikkében[3], a modern algoritmikus csoportelméletben megkerülhetetlen fogalmak, illetve algoritmusok kerültek bemutatásra.[12]

A szakdolgozat célja, hogy Seress Ákos művének[1] segítségével betekintésként szolgáljon a csoportelméletbeli algoritmusok és ezen belül is a permutációcsoport algoritmusok alapjaiba. Továbbá ezekre építve bemutassa Eugene M. Luks[2] 1982-ben megjelent cikkét, melyben bizonyítja, hogy a korlátos fokszámú gráfok izomorfiája polinomidőben eldönthető. Ez az eredmény szolgált kiindulópontként Babai László 2015-ben publikált eredményéhez[6], mely kimondja, hogy a gráf izomorfizmus probléma kvázipolinomiális időben eldönthető.

A dolgozatban vizsgált algoritmusokról, azoknak a nem determinisztikus változatairól, illetve véletlenszerűsítéssel való javításaikról bővebben Seress Ákos[1] könyvében, Eugene Luks[2] cikkében vagy egyéb, a szövegben megjelölt művekben lehet olvasni.

1. Előzetes áttekintés

A bevezetőben megfogalmazottak szerint gráfok közötti izomorfiákat szeretnénk vizsgálni, a csoportelmélet segítségével. Először is nézzük meg, hogyan vezethetjük vissza a kérdést algebrai problémára.

Legyen X_1, X_2 két összefüggő gráf, melyek izomorfiáját szeretnénk eldönteni. Elegendő összefüggő gráfokat vizsgálni, hiszen ha feltesszük, hogy X_1, X_2 nem összefüggőek, és a különálló gráf komponensek száma m, akkor a probléma visszavezethető legfeljebb m^2 darab összefüggő gráfpár izomorfiájának vizsgálatára. Legyen tehát $X = X_1 \cup X_2$ a diszjunkt uniója a két gráfnak. Ezek a gráfok akkor és csak akkor izomorfok egymással, ha létezik X-nek olyan automorfizmusa, melyre teljesül, hogy a két komponense teljes egészében megcserélődik.

Abban az esetben ha ismernénk X automorfizmus csoportjának elemeit, elegendő lenne ellenőrizni, hogy létezik-e ilyen tulajdonságú elem. Belátható, hogy amennyiben létezik ilyen $\sigma \in \mathrm{Aut}(X)$, akkor $\mathrm{Aut}(X)$ tetszőleges generátorrendszerében is van legalább egy olyan elem, amelyre igaz az, hogy megcseréli a két komponenst. Ezzel jelentősen csökkenthetjük a vizsgálandó automorfizmusok számát, hiszen elegendő $\mathrm{Aut}(X)$ egy tetszőleges generátorrendszerét előállítani.

Ez a feladat tovább redukálható más, elemibb problémákra, de ebben a dolgozatban Luks[2] cikke alapján, a szín automorfizmus problémára vezetjük vissza.

Először tehát tekintsük a szín automorfizmus problémát.

1.0.1. Probléma. *Legyen A egy tetszőleges színezett halmaz, továbbá legyen adott egy generátorrendszere egy $G \leqslant \mathrm{Sym}(A)$ csoportnak. Feladat, hogy meghatározzuk G azon részcsoportjának egy generátorrendszerét, amelynek elemei pontosan a színtartó leképezések.*

A feladat, hogy $\mathrm{Aut}(X)$ egy generátorrendszerét előállíthassuk a következőképpen vezethető vissza a szín automorfizmus problémára: Legyen az A halmaz olyan, amely egy n csúcsú gráf pontjaiból készített rendezetlen párokat tartalmazza. Színezzük két színnel az éleket, illetve a nem éleket. Definiáljuk G-t mint egy, a gráf pontjain ható permutációcsoportot, amelyre teljesül az is, hogy $G \geqslant \mathrm{Aut}(A)$. Ebben a speciális esetben a színtartó részcsoport nem más, mint a gráf automorfizmus csoportja. Alapvetően $G = S_n$ minden esetben megfelelő választás, de ez nem szükséges feltétel. Különböző típusú gráfok esetén többféle visszavezetés is lehetséges, melyekhez más $G \leqslant S_n$ csoport szükséges. Például abban az esetben, ha az X gráfban minden csúcs foka legfeljebb k, akkor $\mathrm{Aut}(X)$ minden kompozíciófaktora izomorf S_k egy részcsoportjával. Ezt a tényt Luks is kihasználja annak érdekében, hogy az algoritmusa polinomiális időben fusson le korlátos fokú gráfok esetén.

1.1. Primitív és imprimitív csoportok. A dolgozatban az Algebra 3 tantárgy keretein belül tárgyalt alapvető jelöléseken és fogalmakon túl szükség lesz még további definíciókra is.

A csoporthatás, orbit-stabilizátor tétel, illetve a hűséges vagy a tranzitív hatás fogalmát ismertnek tekintjük, de a továbbiakban szükség lesz még a következő ismeretekre is.

Ezentúl a dolgozatban Ω mindig egy véges halmazt jelöl, valamint G egy véges csoportot, amely hat az Ω halmazon. Továbbá ezen a fejezeten belül feltesszük azt is, hogy G hatása Ω-n tranzitív.

1.1.1. Definíció. *Egy $g \in G$ elem tartójának nevezzük azt a részhalmazát Ω-nak, amely azon Ω-beli elemekből áll, amelyeket g mozgat.*

Formálisan, $supp(g) = \{\omega \in \Omega \mid \omega^g \neq \omega\}$.

Például egy permutáció tartója könnyen megállapítható, hiszen ez nem más mint a komplementere a fixpontok halmazának. A ciklusfelbontásos alakban, minden elem amely egy egynél hosszabb ciklus része, eleme a tartónak. A leképezéses alakban pedig azon elemek, amelyek nem önmagukba képeznek, alkotják a tartót.

1.1.2. Definíció. *Legyen $B \subseteq \Omega$ nem üres részhalmaz. Ekkor B-t G-blokknak nevezzük, amennyiben teljesül, hogy tetszőleges $\sigma, \tau \in G$-re. $\sigma(B) = \tau(B)$ vagy $\sigma(B) \cap \tau(B) = \varnothing$. Azokat az eseteket, hogy $|B| = 1$, vagy $B = \Omega$, triviális blokkoknak nevezzük. A B blokkra azt mondjuk, hogy minimális, ha nem triviális és nincs olyan legalább kételemű valódi részhalmaza, amely maga is G-blokk, illetve maximális akkor, ha $B \neq \Omega$ és az egyetlen B-t tartalmazó blokk az maga Ω.*

Szemléletes példaként tekintsünk egy 2x2-es Rubik kockát. Az Ω halmaz álljon a kocka kis lapjaiból, tehát Ω számossága $6 \cdot 4 = 24$. Legyen $G \leqslant \mathrm{Sym}(\Omega)$ az a csoport, amelyet a 6 lehetséges forgatásunkkal generálunk. Akár egyszerű próbálkozás útján is belátható, hogy ez a hatás tranzitív, hiszen tetszőleges kis kockát el tudunk juttatni bármely másik kis kockába, illetve egy kis lapkát könnyen átforgathatunk ugyanabban a kis kockában egy másik pozícióba. Láthatjuk tehát, hogy a hatás tranzitív, hiszen csupán egy orbit létezik. Ekkor látszik, hogy bármely kis kocka színezett lapkái alkotnak egy G-blokkot, hiszen két tetszőleges forgatássorozat esetén, vagy különböző pozícióba jutnak el, ekkor tehát a képhalmazok metszete az üres halmaz, vagy egy pozícióba esnek, ekkor pedig a képhalmazok megegyeznek.

1.1.3. Definíció. *Legyen B egy G-blokk. Ekkor a $\{\sigma(B) \mid \sigma \in G\}$ halmazrendszert a G, B-t tartalmazó blokkrendszerének nevezzük Ω-ban. A blokkrendszert minimálisnak nevezzük, ha minden blokk maximális. Hasonlóan maximális akkor, ha minden blokk minimális.*

Ezen felül abból, hogy feltettük, hogy G hatása Ω-n tranzitív, nyilvánvalóan következik, hogy G egy blokkrendszer blokkjain is tranzitívan hat.

Az előző példát folytatva, látszik, hogy az összes kis kocka együttesen blokkrendszert alkot G-ben. Az is belátható, hogy a blokkok minimálisak, tehát a blokkrendszer maximális. Ehhez azt kell észrevennünk, hogy abban az esetben, hogy ha adott egy $G \leqslant \mathrm{Sym}(\Omega)$ csoportunk, amelynek blokkrendszere $B_1, ...B_k$, akkor az egy egyenlő elemszámú partíciót is alkot Ω-ban, $k|B_1| = |\Omega|$. Következésképpen, egy blokk mérete osztója kell legyen a permutációcsoport fokának. Továbbá ha $B' \subset B \subset \Omega$ is G-blokkok, akkor $|B'| \mid |B|$-nek is teljesülnie kell. Ebből következik, hogy mivel a 3 prímszám ezért minden kis kocka egy minimális blokkot határoz meg. Ebből pedig következik, hogy a blokkrendszerünk maximális.

Ugyanakkor ezek a blokkok maximálisak is, hiszen a tartalmazó blokkok csupán a 3 elemű blokkokból állhatnak össze, de az egész Rubik-kockán kívül nem tudunk mondani olyan másik, legalább kételemű partícióját a kis kockáknak, amely kielégítené a blokkra vonatkozó definíciót. Ilyen módon tehát a blokkok maximálisak és blokkrendszerünk minimális is.

Ebből az is következik, hogy ez az egyetlen nem triviális blokkrendszere létezik a 2x2-es Rubik kocka csoportjának.

1.1.4. Definíció. *Tegyük fel, hogy az Ω halmazban nincsenek 1-nél nagyobb elemszámú G-blokkok. Ekkor azt mondjuk, hogy G primitíven hat Ω-n. Ellenkező esetben a hatást imprimitívnek szokás nevezni. Ha G egy minimális blokkrendszeren hat, akkor az a hatás primitív.*

Természetesen látszik, hogy a Rubik kockás példánkban a blokkok 3 eleműek, tehát ez nem egy primitív hatás. Ugyanakkor a blokkokon már primitíven hat, hiszen az előbb beláttuk, hogy a blokkrendszer minimális.

2. Alapvető csoport algoritmusok

Ennek a fejezetnek a célja, hogy néhány elemi, polinomiális futásidejű permutációcsoport algoritmust bemutassunk. Ezek az algoritmusok általános csoporthatásokra is működnek, ennek ellenére jelenleg csupán arra az esetre szorítkozunk, amikor a G csoport, amely hat valamilyen Ω halmazon, egy permutációcsoport, tehát $G \leqslant \mathrm{Sym}(\Omega)$.

A permutációcsoportokat az algoritmusokban inputként és outputként is mindig egy generátorrendszerével adjuk meg. Erre azért van szükség, mert $|\Omega| = n$ esetén $n!$ darab csoportelemet kellene tárolnunk, ami már nem túl nagy n-re is megvalósíthatatlan, hiszen egy általános permutáció tárolásához körülbelül $n \log_2(n)$ bit szükséges. Ezzel szemben egy S generátorrendszer megadásához csupán $|S|\ n\ \log_2(n)$ nagyságrendű tárhely már elegendő. Ezt a konvenciót az is elősegíti, hogy a gyakorlatban előforduló esetekben $|S|$ általában kicsi. Ebből kifolyólag

egy algoritmust, amely permutációcsoportokkal dolgozik, akkor nevezhetünk polinomiális futásidejűnek, ha valamely k pozitív egészre az algoritmus legfeljebb $O(|S|^k n^k)$ lépésben leáll.

2.1. Orbit kiszámító algoritmus. A továbbiakban sűrűn előfordulhat, hogy szeretnénk meghatározni egy G-orbitot. Ebben az alfejezetben azt a problémát szeretnénk áttekinteni, hogy miként találhatunk megfelelő nagyságrendű futásidőben orbitokat.

Szeretnénk tehát kiszámítani valamely $\alpha \in \Omega$-ra, az $\alpha^G := \{\alpha^g | g \in G\}$ orbitot. A G csoportot egy S generátorrendszerével adjuk meg, amely valamilyen részhalmaza Ω permutációinak. Ehhez elegendő, ha adott $\beta \in \Omega$ és $s \in S$-re ki tudjuk számítani a β^s képet, illetve, hogy Ω két eleméről el tudjuk dönteni, hogy megegyeznek-e. Előfordulhat olyan eset, hogy az utóbbi nem magától értetődő. Példaként vizsgáljuk azt az esetet, hogy G konjugálással hat az $\Omega = \{H \mid H \leqslant G\}$ halmazon. Ekkor ha $H \in \Omega$ egy generátorrendszerével van megadva, akkor a $H^{s_1} = H^{s_2}$ egyenlőség eldöntése tulajdonképpen azt a kérdést veti fel, hogy permutációk két halmaza ugyanazt a csoportot generálja-e.

Az α^G orbit kiszámítását visszavezethetjük egy gráfokon végzett algoritmusra, nevezetesen arra, amellyel meghatározhatjuk gráfok összefüggő komponenseit.

2.1.1. Algoritmus. *Definiáljunk Ω elemein egy $D(\Omega, S)$ irányított gráfot úgy, hogy egy $\beta \in \Omega$-ból akkor megy él $\gamma \in \Omega$-ba, ha létezik olyan $s \in S$ generátor elem amelyre igaz, hogy $\beta^s = \gamma$.*

Mivel G véges csoport, az is igaz, hogy $D(\Omega, S)$ komponensei erősen összefüggőek, hiszen ilyen s létezésekor valamely k-ra $\gamma^{s^k} = \beta$ is teljesül.

Tegyük fel, hogy $G = \langle S \rangle$, ekkor az α^G orbit az a legkisebb X részhalmaza Ω-nak, amelyre $\alpha \in X$ és mely zárt az S elemeire nézve. Az algoritmus során az α^G orbitot végeredményben a $D(\Omega, S)$ gráf egy szélességi bejárásával kapjuk meg, ha a bejárást az α-ból indítjuk. Rekurzívan definiáljuk a csúcsok részhalmazainak egy $L_0, L_1, L_2, ..., L_{m-1}$ sorozatát, mely a továbbiakban a bejárás szintjeit fogja jelölni.

- $L_0 = \{\alpha\}$
- *Minden további i-edik szinten Ω azon elemei találhatóak, melyek nincsenek benne $L_0 \cup L_1 \cup ... \cup L_{i-1}$-ben és végpontjai azoknak az éleknek, amelyek kezdőpontjai az előző szinten szerepeltek.*

Akkor állítjuk meg majd az algoritmust, ha valamelyik szintre az üres halmazt kapjuk, vagyis $L_m = \varnothing$ valamely $m \in \mathbb{N}$. Formálisan

$$L_0 := \{\alpha\}, L_m = \varnothing$$

$$L_i := \{\gamma \in \Omega \mid (\exists \beta \in L_{i-1})(\overrightarrow{(\beta, \gamma)} \in \overrightarrow{E}))\} \setminus \bigcup_{j<i} L_j$$

Ekkor az $\bigcup_{j<m} L_j$ *összefüggő komponens lesz az* α^G *orbit.*

A következő pszeudokód szemlélteti az orbit kiszámító algoritmust. A bemenet tehát az S generátorrendszer, illetve az α elem, a kimenet pedig az *orbit* tömb, mely az α^G elemeit tartalmazza.

Pszeudokód - Orbit kiszámító algoritmus

```
1:  procedure ORBIT KISZÁMÍTÁS(α, S)
2:      orbit ← α
3:      generátorok ← S
4:      i ← 1
5:      while i ⩽ hossz(orbit) do
6:          for s ∈ S do
7:              if orbit[i]^s ∉ orbit then
8:                  orbit ← orbit + orbit[i]^s
9:          i ← i + 1
10:     return orbit
```

2.1.2. Tétel. *Ha a* $G = \langle S \rangle$ *csoport hat az* Ω *halmazon, akkor az* $\alpha \in \Omega$ *orbitját* $O(|\alpha^G||S|)$ *kép kiszámításával megkaphatjuk. Továbbá a futásidőről megállapítható, hogy*

- *ha ismert egy olyan* $\Delta \supseteq \alpha^G$ *tartalmazó halmaz, mely számára van elég szabad memóriaterület, akkor az össz futási időt dominálja a képek kiszámítása*
- *ha nincs ilyen tartalmazó halmaz, akkor az időigény* $O(|\alpha^G|^2|S|)$ *darab pár összehasonlításához szükséges idővel nő*

Bizonyítás. A tétel első felét abból az észrevételből kapjuk, hogy az összefüggő komponens és az L_i halmazok kiszámításához minden $\beta \in \alpha^G$-re ki kell számolnunk a β^s, $s \in S$ képeket, majd a második fele a tételnek arra utal, hogy ezeket az L_i-ket egy U tömbben szeretnénk tárolni, és az új képeket az U tömb elemeivel össze kell hasonlítanunk, hogy előfordultak-e már az algoritmus során.

Az első esetben foglaljunk egy $|\Delta|$ méretű T segédtömböt. Ebben a tömbben azt fogjuk jelölni, hogy az adott elem előfordult-e már U-ban. Amikor új elemet akarunk hozzáadni U-hoz, akkor csupán megvizsgáljuk, hogy az éppen kiszámított elem már meg lett-e jelölve T-ben, avagy sem.

A második pontban jobb híján az újonnan kiszámolt elemet az összes, már megtalált elemmel össze kell hasonlítanunk, amely $O(|\alpha^G|^2|S|)$ összehasonlításhoz szükséges időtöbbletet eredményez.

□

A tételnek valójában van egy harmadik része is, amely azt mondja ki, hogy amennyiben nincsen megfelelő tartalmazó halmaz, de van egy teljes rendezés Ω-n, akkor az előbb megállapított többletköltség csupán $O(|\alpha^G|log^2|\alpha^G||S|)$ pár összehasonlítására csökken. Tekintve, hogy permutációcsoportok esetén az alaphalmaz, amelyen hatunk, általában nem túl nagy, továbbá a dolgozat során hosszabb távon a futásidőre szeretnénk polinomiális felső becslést adni, ezért sok esetben alkalmazhatjuk a tételben szereplő első lehetőséget. Ennek ellenére a második opció sem fogja elrontani a polinomiális futásidőt, tehát ez sem okozhat gondot. Az első esetben Δ halmazt általában választhatjuk Ω-nak. Ekkor ez a $|\Delta|$ hosszú tömb Ω elemeit reprezentálja és azt jelöli, hogy melyik elem került már kiszámításra az algoritmus során. További képek előállítása után pedig elegendő ellenőrizni, hogy a tömbben meg van-e már jelölve a megfelelő elem.

Előfordulhat, hogy nem csak azt szeretnénk tudni, hogy valamely $\gamma \in \Omega$ benne van-e az α^G orbitban, hanem szeretnénk megkapni egy olyan $g \in G$ elemet is, melyre igaz, hogy $\alpha^g = \gamma$. Az algoritmus amely segítségével egy ilyen g elemet megkaphatunk, az orbit kiszámító eljárás egy módosított változata.

2.1.3. Algoritmus. *A* 2.1.1*-es Algoritmus csupán úgy változik, hogy karban kell tartanunk még egy V tömböt, amely a következő tulajdonsággal rendelkezik. Minden $i \leqslant |U|$-ra teljesül, hogy ha $U[i] = \beta$ és $V[i] = g \in G$, akkor $\alpha^g = \beta$. Egy ilyen V tömböt az U tömbbel egyidejűleg tudunk előállítani, ha mindig, amikor az α^G egy új elemét megkapjuk β^s alakban, ahol $\beta = U[i]$ és $s \in S$, akkor az U végéhez fűzzük β^s-et, illetve V-hez pedig $V[i]s$-et.*

Ez plusz költséggel jár memóriában, hiszen a V-ben tárolt permutációk memóriaigénye $|\alpha^G||\Omega| \log |\Omega|$, továbbá a V elemeinek meghatározásához szükséges permutáció szorzatok kiszámítása növeli a futásidőt is. Ezt megkerülve, ha nem túl sok elemre vagyunk kíváncsiak, a gyakorlatban mutatókat használunk. Egy új β^s elem U-ba történő felvételekor $V[end + 1]$-be az s generátorelem mutatóját tesszük. Ez azért lehet hasznos, mert sok esetben nincs szükség minden $\beta \in \alpha^G$-hez meghatároznunk a megfelelő $g \in G$ elemet, amelyre $\beta = \alpha^g$. Elegendő csak speciális β-khoz megtalálni azokat, ezáltal a generátorok szorzásait csupán akkor végezzük el, ha ténylegesen szükséges. Ezt a következő módon tehetjük meg. A kiválasztott β elemünket megkeressük U-ban, legyen ennek a helye i. Ekkor kiszámoljuk, $U[i]V[i]^{-1}$-t, majd i-t csökkentve megkeressük, hogy ez az elem hol található U-ban, ahol már j. indexre számoljuk ki az előző szorzatot. Addig folytatjuk amíg a tömb elejére, α-hoz nem érünk. Ezáltal megtaláljuk azt a generátor sorozatot amely előállítja a keresett β-t.

Érdemes megjegyezni azt is, hogy nem csupán egy kezdeti $\alpha \in \Omega$-ra alkalmazhatóak a fentebb említett algoritmusok, hanem amennyiben

U-t A-ként inicializáljuk, akkor tetszőleges $A \subseteq \Omega$-ra is kiszámítható $A^G := \{\alpha^g \mid \alpha \in A,\, g \in G\}$.

2.2. Algebrai struktúrák lezártja. Ebben a fejezetben olyan problémákat vizsgálunk, amelyek során egy G csoporttal hatunk egy olyan Ω-n, amelyen adott valamilyen algebrai struktúra. Ekkor az $A \subseteq \Omega$ halmaz G-lezártjának nevezzük Ω-nak azt a tartalmazásra nézve legszűkebb részhalmazát, amely tartalmazza A-t és zárt a G hatására, illetve az Ω-n belüli műveletekre is. Jelölése $\langle A^G \rangle$.

Az orbit-kereső algoritmust egy kis módosítással használhatjuk arra is, hogy halmazok G-lezártját meghatározzuk. Ezen módosított algoritmus akkor működik, ha feltesszük, hogy tartalmazási vizsgálatokat végre tudunk hajtani a már meglévő alstruktúrákban, azaz $T \subseteq \Omega$ és $\omega \in \Omega$ esetén, rendelkezésünkre áll egy jó algoritmus $\omega \overset{?}{\in} \langle T \rangle$ eldöntésére. Általában ilyen esetekben is csak egy generátorrendszerét keressük a lezártnak, nem feltétlen szükséges felsorolni minden elemét. Ezzel nagy elemszámú lezártakat is le tudunk írni. Abban a speciális esetben, amikor $G = \Omega$ hat saját magán a konjugálással, ezt az algoritmust használhatjuk normális lezárt meghatározására is. Ekkor egy $A \subset G$-ből kiindulva, az algoritmus a legkisebb, A-t tartalmazó normálosztó egy generátorrendszerét adja meg.

2.2.1. Algoritmus. *Az algoritmus inputja egy $S \subseteq G$ generátorrendszer, illetve egy $A \subseteq \Omega$ részhalmaz. Az $\langle A^G \rangle$ halmaz generátorait egy T tömbben tároljuk és A-ként inicializáljuk. Egy általános lépésben minden tömbben lévő $t \in T$ elemre kiszámoljuk a t^s, $s \in S$ képeket, majd hozzáadjuk a tömbhöz azokat a t^s-eket melyek nincsenek benne a $\langle T \rangle$-ben. Ilyen módon minden új elemet, közvetve vagy közvetlenül de valamilyen a^g, $a \in A$, $g \in G$ alakban állítunk elő, tehát a tömb által generált részstruktúra megegyezik $\langle A^G \rangle$-vel. Az időkritikus része az algoritmusnak általában az új elemek tartalmazásának eldöntése, ugyanis a legtöbb alkalmazásban Ω maga is egy csoport, tehát a tömbhöz hozzáadandó elemek száma legfeljebb $log|\langle A^G \rangle|$.*

Pszeudokód - Algebrai struktúrák lezártja

```
1: procedure LEZÁRT KISZÁMÍTÁS(A, S)
2:     T ← A
3:     generátorok ← S
4:     i ← 1
5:     while i ⩽ hossz(T) do
6:         for s ∈ S do
7:             if T[i]^s ∉ ⟨T⟩ then
8:                 T ← T + T[i]^s
9:         i ← i + 1
```

10: **return** T

Ez az algoritmus segíthet további problémák eldöntésében is. Hasonló módon mind a kommutátor lánca, mind az alsó centrális lánca kiszámítható egy csoportnak. Továbbá meghatározható, hogy egy részcsoport szubnormális-e. Ennek ellenére, ezeknek a feladatoknak a nehézsége erősen függ attól, hogy tudunk-e tartalmazási kérdéseket megválaszolni a futás folyamán.

3. Bázisok és erős generátorrendszerek

Ezen fejezet fő célja a következő lemma bizonyítása, mely fontos részét képezi Luks[2] algoritmusának is.

3.0.1. Lemma (Furst-Hopcroft-Luks)**.** [9] *Legyen $G \leqslant S_n$, továbbá adott egy $S \subseteq G$ generátorrendszer, ekkor polinomidőben eldönthető,*

- *G rendjének meghatározása*
- *tetszőleges $g \in \mathrm{Sym}(\Omega)$-ra annak eldöntése, hogy eleme-e a $G = \langle S \rangle$ csoportnak*
- *G minden olyan részcsoportjához található generátorrendszer, amelyre igaz, hogy polinomiálisan korlátolt indexe van G-ben és amihez van polinomidejű tartalmazási teszt*

Ennek a problémának a vizsgálatában nyújtanak segítséget Charles C. Sims[3][4][5] publikációi, melyekben olyan fogalmakat határoz meg, amelyeket még napjainkban is használnak, továbbá alapul szolgálnak a legtöbb permutációcsoportokkal kapcsolatos kérdés algoritmikus megválaszolásának. Ebben a fejezetben bemutatásra kerül a bázis, illetve az erős generátorrendszer fogalma és ezek jelentősége. Ezen felül a legvégén az úgynevezett Schreier-Sims algoritmust vizsgáljuk, amely polinomiális időben, tetszőleges csoporthoz előállít egy erős generátorrendszert.

3.1. Bázis és erős generátorrendszer. A továbbiakban feltehető, hogy az alaphalmazunk $\Omega = \{1, 2, ..., n\}$ és $G \leqslant \mathrm{Sym}(\Omega)$ permutációcsoport Ω-n.

3.1.1. Definíció. *A G csoport bázisa alatt egy olyan $B = (\beta_1, ..., \beta_m)$, Ω-beli sorozatot értünk, amelyre teljesül, hogy ha egy $g \in G$ pontonként fixen hagyja B-t, akkor g csakis az egység elem lehet.*

A B bázis meghatároz egy

$$G = G^{[1]} \geqslant G^{[2]} \geqslant ... \geqslant G^{[m]} \geqslant G^{[m+1]} = 1,$$

részcsoport láncot, ahol $G^{[i]} := G_{(\beta_1,...,\beta_{i-1})}$ a pontonkénti stabilizátora a $\{\beta_1, ..., \beta_{i-1}\}$ halmaznak. Más szóval, $G^{[i]}$ fixen tartja B első $i-1$

elemét. Ezt a láncot a B bázishoz tartozó, vagy B bázis által meghatározott *stabilizátor lánc*-nak nevezzük.

Amennyiben az is teljesül, hogy

$$G = G^{[1]} > G^{[2]} > ... > G^{[m]} > G^{[m+1]} = 1,$$

vagyis $G^{[i+1]}$ valódi részcsoportja $G^{[i]}$-nek, minden $1 \leqslant i \leqslant m$-re, akkor B-t *irredundáns* bázisnak nevezzük.

Tekintsünk erre is egy példát. Vizsgáljuk meg ismét a 2x2-es Rubik kockánkat. Válasszunk ki tetszőleges számú lapkát úgy, hogy már ne tudjunk olyan forgatást vagy forgatássorozatot találni, hogy bármelyik is elmozduljon. Ekkor ezek a lapkák a csoportunk egy bázisát fogják alkotni. Triviális példaként válasszuk ki az összes lapkát, ez biztosan bázis lesz. Ebből még számos lapkát el kell hagynunk ahhoz, hogy a megmaradó lapkák irredundáns bázisát alkossák a Rubik-kocka csoportjának. Például, amennyiben minden kis kockáról kiválasztunk egy-egy lapkát, az egy nyolc elemű bázist alkot. Valójában még ez a bázis is redundáns. Belátható, hogy még egy lapkát elhagyva ezek közül kapjuk a 2x2-es Rubik-kocka csoportjának egy hételemű irredundáns bázisát.

Ebből is látszik, hogy G csoportnak több különböző bázisa is lehet, illetve a lineáris algebra megszokott fogalmával ellentétben ezek különböző méretűek is lehetnek. Az irredundáns bázisoknak egyik előnye az, hogy méretük becsülhető:

$$|G^{[i]} : G^{[i+1]}| \geqslant 2,$$

$$|G^{[i]} : G^{[i+1]}| = |\beta_i^{G^{[i]}}| \leqslant n, \ \forall i \in [1, m],$$

hiszen $G^{[i+1]}$ szerint vett bal oldali mellékosztályok $G^{[i]}$-ben éppen a $\beta_i^{G^{[i]}}$ orbit elemeinek felelnek meg. Ekkor Lagrange tételét többször is alkalmazva kapjuk, hogy

$$|G| = \prod_{i=1}^{m} |G^{[i]} : G^{[i+1]}|.$$

Felhasználva az alsó és felső becslést kapjuk, hogy $2^{|B|} \leqslant |G| \leqslant n^{|B|}$, melyből már adódik, hogy

$$\frac{log|G|}{logn} \leqslant |B| \leqslant log|G|.$$

Ez az összefüggés garantálja, hogy egyetlen irredundáns bázis sem lehet túl nagy. A következő példa azt mutatja, hogy az irredundáns bázisok elemszámára adható becslések általában nem javíthatók.

3.1.2. Feladat. *Seress Ákos*[1] *4.2 feladat. Adjunk meg egy olyan $G \leqslant S_n$ csoportot és olyan $B_1, B_2 \subset \{1, 2, ..., n\}$ irredundáns bázisait G-nek, melyekre $|B_1| = \lfloor log|G| \rfloor$ és $|B_2| = \lceil \frac{log|G|}{logn} \rceil$ teljesül.*

Megoldás. A feladat kitűzésében segítségként volt adva, hogy G-t célszerű ciklikus csoportként keresni. Ezt a segítséget kihasználva tehát egy olyan $G = \langle\sigma\rangle \leqslant S_n$ ciklikus csoportot adunk meg, mely rendelkezik a kívánt tulajdonsággal. Vizsgáljunk először egy meglehetősen egyszerű elemet, hogy jobban megérthessük, milyen tulajdonságot kellene kihasználnunk. Legyen $\sigma \in S_n$ egy olyan elem, melynek ciklusfelbontásában csupán egy nem triviális ciklus van. Láthatjuk, hogy a ciklusból tetszőleges elemet választva bázisként, azonnal fixáljuk a többit is, hiszen önmagával szorozgatva, vagy mindegyik mozog, vagy egyik sem. Ennek két fontos következménye van a cél szempontjából.

(1) Tetszőleges elem választása már irredundáns bázis
(2) Csupán egyféle irredundáns bázisunk létezik

Megjegyzésként elmondható, hogy triviális példaként már egy ilyen elem is megfelelő lehet, hiszen tekintsük egy olyan permutációt melynek ciklusfelbontása $(k\ l)$. Ekkor $n \geqslant 2$, és $|G|$ pedig egy kételemű részcsoportja S_n-nek. Legyen $B_1 = k$ és $B_2 = l$. Ekkor $\lfloor \log |G| \rfloor = 1$, illetve $\lceil \frac{\log |G|}{\log n} \rceil = 1$, tehát készen vagyunk. Ennek ellenére olyan példát is tudunk mutatni, amelyben $|G|$ és n is tetszőlegesen nagy lehet. Azt már megállapítottuk, hogy erre a célra több ciklussal rendelkező permutációt kell keresnünk. A konstrukció azzal a tulajdonsággal is rendelkezni fog, hogy a megfelelő ciklusból választva egy elemet, irredundáns bázist kapunk. Tegyük fel először, hogy σ két ciklussal rendelkezik. Amennyiben ezek hossza egyenlő, nem jutunk előrébb, hiszen egy elem már bázis. Különböző hosszok esetén két lehetőségünk van, nevezetesen, hogy a ciklushosszok közül valamelyik osztja-e a másikat, vagy sem. Amennyiben egyik sem osztja a másikat elveszítjük azt a kívánt tulajdonságot, hogy létezik egyelemű bázis, tehát most ne ezekkel foglalkozzunk. Oszthatóság esetén azonban észrevehetjük, hogy a hosszabb ciklusból választva egy elemet már bázist kapunk, ugyanakkor a rövidebb választása esetén, a hosszabb ciklus még nem "tűnik" el, a megmaradt elemek közül tudunk még választani egy elemet, amivel együtt egy kételemű irredundáns bázist kapunk. Ezután tetszőleges k pozitív egészre előállíthatunk egy $G_k = \langle\sigma_k\rangle \leqslant S_n$ csoportot, melyre $o(\sigma_k) = 2^k$ és $n = 2^{k+1} - 1$. Ehhez legyenek $C_i = (2^i, 2^i + 1, ..., 2^{i+1} - 1)$ ciklusok, minden $1 \leqslant i \leqslant k$-ra, és legyen $\sigma_k = C_1 C_2 ... C_k$. Ekkor $B_1 = \{2^1, ..., 2^k\}$ és $B_2 = \{2^k\}$ egyaránt irredundáns bázisa G-nek, továbbá $|B_1| = k = \lfloor \log |G| \rfloor$ és $|B_2| = 1 = \lceil \frac{\log |G|}{\log n} \rceil$ is teljesül.

3.1.3. Definíció. *Legyen $G \leqslant \mathrm{Sym}(\Omega)$, legyen $B = (\beta_1, ..., \beta_m) \subseteq \Omega$ bázisa G-nek, illetve $G^{[i]}$, a B-hez tartozó stabilizátor lánca G-nek. Tekintsük ekkor G egy S generátorrendszerét. Abban az esetben, ha S-re teljesül, hogy*

$$\left\langle S \cap G^{[i]} \right\rangle = G^{[i]},\ \forall\ 1 \leqslant i \leqslant m + 1$$

akkor az S-et a G erős generátorrendszerének hívjuk (a B bázisra nézve).

Ezen generátorrendszerek haszna abban rejlik, hogy a G csoport B bázisára nézve könnyen tudjuk számolni a $\beta^{G^{[i]}}$, úgynevezett fundamentális orbitokat. Ezeknek a segítségével pedig azonnal adódik G számossága. Továbbá könnyen megkaphatunk egy, tetszőleges $G^{[i]} : G^{[i+1]}$-hez tartozó R_i transzverzálist. Hogyan tudjuk ezeket megtenni?

Tegyük fel, hogy adott egy G csoportunk, ennek egy $B = (\beta_1, ..., \beta_m)$ bázisa, illetve egy S erős generátorrendszere a B bázisra nézve. Ekkor B bázis által meghatározott stabilizátor láncában szereplő minden $G^{[i]}$ csoporthoz meg tudjuk adni annak S_i generátorrendszerét a következő módon. $S_i := \{s \in S \mid s(\beta_j) = (\beta_j)\ \forall j < i\}$. A $\beta_i^{G^{[i]}}$ $1 \leqslant i \leqslant m$ fundamentális orbitokat ezek után könnyen meghatározhatjuk a 2.1.1-es orbit kiszámító algoritmus segítségével. Felhasználva az orbit-stabilizátor tételt, mely szerint $|\beta_i^{G^{[i]}}| = |G^{[i]} : G^{[i+1]}|$, megkaphatjuk,

$$|G| = \prod_{i=1}^{m} |G^{[i]} : G^{[i+1]}| = \prod_{i=1}^{m} |\beta_i^{G^{[i]}}|$$

a G csoport rendjét. Ezzel visszavezettük a 3.0.1-es Lemma első pontját erős generátorrendszer keresésének problémájára, amely feladat megoldása a fejezet fő célja.

Ezen felül a 2.1.3-as Algoritmust segítségül hívva, felépíthetünk egy bal oldali mellékosztály reprezentáns rendszert G-ben.

3.2. Schreier fa adatszerkezet. Legyen $G \leqslant \mathrm{Sym}(\Omega)$ csoport, annak egy B bázisa, illetve egy S erős generátorrendszere G-nek a B bázisra nézve. Legyen ekkor a B által meghatározott stabilizátor lánc i. elemének generátorrendszere $S_i = S \cap G^{[i]}$. Ekkor elkészíthetjük G-nek az úgynevezett *Schreier fa adatszerkezetét*. Az adatstruktúra tulajdonképpen egy sorozat, ami az összes β_i, $1 \leqslant i \leqslant m$ báziselemhez tartozó (S_i, T_i) *Schreier-fáknak* nevezett párokból áll. Minden i-re, T_i egy olyan irányított fát jelöl, melynek csúcshalmaza a $\beta_i^{G^{[i]}}$ fundamentális orbit, a gyökere pedig β_i báziselem. Minden éle a gyökér felé irányul és kap egy címkét az S_i halmaz elemei közül. Ennek a címkének teljesítenie kell azt a tulajdonságot, hogy ha egy tetszőleges él kezdő és végpontja rendre γ, δ, akkor az élen az a h szerepelhet, melyre igaz, hogy $\gamma^h = \delta$. Ez azt jelenti, hogy a fa tetszőleges pontjából indulva, ha a címkén lévő permutációkat összeszorozzuk, azt az permutációt kapjuk, amely a kiinduló pontot a β_i báziselembe viszi. Ezzel megkapjuk az inverzeit egy $G^{[i]}$-ben $G^{[i+1]}$ szerint vett mellékosztály reprezentáns halmaznak.

Ezzel a reprezentációval csökkenthetjük a memóriaigényt, hiszen $O(n)$ tárhely szükséges ahhoz, hogy T_i fákat tárolhassuk. Minden i-re egy n hosszú tömb elegendő. Továbbá az S_i-beli permutációkat $O(|S_i|n)$ nagyságrendben tudjuk megőrizni.

3.3. Szitáló eljárás. Ez az eljárás akár tekinthető a permutáció csoportok Gauss eliminálásának. Az algoritmus során tetszőleges $g \in G$ csoportelemet felbonthatunk a következő alakban,

$$g = r_m r_{m-1} ... r_1, r_i \in R_i$$

ahol R_i a fentebb említett $G^{[i]} : G^{[i+1]}$ mellékosztály reprezentáns rendszer. Ez a felbontás ráadásul Lagrange tétele szerint egyértelmű.

3.3.1. Algoritmus. *Amennyiben egy $g \in G$ elemet szeretnénk a fenti szorzatként felírni, meg kell keresnünk azt az egyértelmű $r_1 \in R_1$ mellékosztály reprezentánst, melyre teljesül, hogy $\beta_1^g = \beta_1^{r_1}$. Ekkor legyen $g_2 = gr_1^{-1} \in G^{[2]}$, melyre megkeressük $r_2 \in R_2$ reprezentáns elemet úgy, hogy $\beta_2^{g_2} = \beta_2^{r_2}$, és így tovább. Az m-edik lépés után, a $g_{m+1} = gr_1^{-1}...r_m^{-1}$ már a B bázis minden elemét fixen hagyja, tehát $g_{m+1} = 1$.*

Pszeudokód - Szitálás

```
1: procedure SZITÁLÁS(g, B, {R_1, R_2, ..., R_n})
2:     g_x ← g
3:     felbontás ← 1
4:     for i ∈ {1, ..., m} do
5:         r_i ← REPREZENTÁNS ELEM KIVÁLASZTÁSA(β, g_x, R_i)
6:         g_x ← g_x r_i^{-1}
7:         felbontás ← r_i · felbontás
8:     return felbontás
9: procedure REPREZENTÁNS ELEM KIVÁLASZTÁSA(β, g, R)
10:    for r ∈ R do
11:        if β^r = β^g then return r
```

3.3.2. Következmény. *Adott $G \subseteq \mathrm{Sym}(\Omega)$ és tetszőleges $h \in \Omega$ esetén alkalmazzuk a szitáló eljárást, próbáljuk meg felbontani h-t. Ekkor, ha a szorzatra bontás sikerül, abból következik, hogy $h \in G$. Ellenkező esetben két dolog következhet be, amelyek következménye az, hogy $h \notin G$. Előfordulhat, hogy $h_{m+1} := hr_1^{-1}r_2^{-1}...r_{m-1}^{-1}r_m^{-1} \neq 1$, illetve az, hogy $h_i := hr_1^{-1}r_2^{-1}...r_{i-1}^{-1}$ már nem tartja benn β_i-t a $\beta_i^{G^{[i]}}$ orbitban, valamely $i \leqslant m$-re.*

Az algoritmus tehát alkalmas tartalmazási kérdések eldöntésére is, vagyis a 3.0.1-es Lemma második pontját is visszavezettük erős generátorrendszer keresésének problémájára, cserébe nem a legoptimálisabb memória használatban és futási időben. A transzverzálisok számolásához és tárolásához egyaránt $O(n^2)$ idő és tárhely szükséges.

Ennek más gyakorlati alkalmazása is lehet. Amennyiben ismerjük G-nek egy bázisát és egy hozzá tartozó erős generátorrendszerét, továbbá képesek vagyunk egyenletes eloszlással, véletlen elemet választani egy listából, akkor G-nek is ki tudjuk választani egy véletlen elemét úgy, hogy minden R_i-ből véletlenszerűen egy elemet választunk, majd azokat összeszorozzuk.

3.4. Schreier-Sims algoritmus. A Schreier-Sims algoritmus egy hatékony módszert biztosít erős generátorrendszerek előállítására. Ennek előfeltételeit, hátterét és lépéseit vesszük sorra. A következő lemma O. Schreiertől származik.

3.4.1. Lemma. *Legyen $H \leqslant G = \langle S \rangle$, továbbá, R egy olyan H szerint vett jobb oldali mellékosztály reprezentáns rendszer G-ben, melyben az egységelem is szerepel. Ekkor H-t generálja a következő T halmaz.*

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

ahol tetszőleges $g \in G$-re $\overline{g} \in R$ az az elem, melyre $Hg = H\overline{g}$ teljesül. Ekkor T elemeit H-hoz tartozó Schreier generátoroknak nevezzük.

Bizonyítás. A definícióból adódik, hogy T elemei mind benne vannak H-ban. Elég tehát annyit belátni, hogy $T \cup T^{-1}$ valóban generálja H-t. Megállapítható, hogy $T^{-1} = \{rs(\overline{rs})^{-1} \mid r \in R, s \in S^{-1}\}$. Tekintsünk ekkor egy tetszőleges $h \in H$ elemet, amelyet felírhatunk, $h = s_1 s_2 ... s_k$ alakban, úgy, hogy minden $s_i \in S \cup S^{-1}$-ből való, hiszen $H \leqslant G$.

Ahhoz, hogy belássuk, hogy $T \cup T^{-1}$ valóban generálja H-t, rekurzívan definiáljuk a $t_j \in T \cup T^{-1}$ sorozatot és ehhez az $r_{j+1} \in R$ elemeket, amelyek teljesítik a $h = t_1 t_2 ... t_j r_{j+1} s_{j+1} ... s_k$ egyenletet $0 \leqslant j \leqslant k$ esetén. A $j = 0$ értékre az $r_1 = 1 \in R$ választással teljesül a feltétel. Tegyük fel, hogy valamely $j < k$-ra már találtunk megfelelő $t_1, ..., t_j \in T \cup T^{-1}$ és $r_{j+1} \in R$ elemeket melyekre

$$h = t_1 ... t_j r_{j+1} s_{j+1} ... s_k$$

teljesül. Ekkor a következő választásokkal

- $t_{j+1} := r_{j+1} s_{j+1} (\overline{r_{j+1} s_{j+1}})^{-1} \in T \cup T^{-1}$
- $r_{j+2} := \overline{r_{j+1} s_{j+1}} \in R$

kapjuk, hogy $t_{j+1} r_{j+2} = r_{j+1} s_{j+1}$. Emiatt pedig,

$$h = t_1 ... t_j (r_{j+1} s_{j+1}) s_{j+2} ... s_k = t_1 ... t_j (t_{j+1} r_{j+2}) s_{j+2} ... s_k$$

szintén teljesül. Végezetül, $j = k$-ra $h = t_1 ... t_k r_{k+1}$ adódik. Mivel $h \in H$ és $t_1 t_2 ... t_k \in \langle T \rangle \leqslant H$, ezért $r_{k+1} \in H \cap R = \{1\}$. Tehát $h \in \langle T \rangle$ is teljesül. □

3.4.2. Lemma. [4] *Legyen $B = \{\beta_1, ..., \beta_k\} \subseteq \Omega$, illetve $G \leqslant \mathrm{Sym}(\Omega)$. Tegyük fel, hogy az $S_j \subseteq G$ halmazokra teljesülnek a következő feltételek minden $1 \leqslant j \leqslant k$-ra.*

- $S_j \subseteq G_{(\beta_1,...,\beta_{j-1})}$;
- $\langle S_j \rangle \geqslant \langle S_{j+1} \rangle$;
- $G = \langle S_1 \rangle$ *és* $S_{k+1} = \varnothing$;
- $\langle S_j \rangle_{\beta_j} = \langle S_{j+1} \rangle$

Ekkor B bázisa G-nek, továbbá $S = \bigcup_{1 \leqslant j \leqslant k} S_j$ erős generátorrendszere G-nek B bázisra nézve.

Bizonyítás. A lemmát Ω elemszámára tekintett indukcióval bizonyíthatjuk. Legyen tehát $\Omega^* = \Omega \backslash \{\beta_1\}$. Ekkor G_{β_1}-nak természetes módon megfeleltethetünk egy $G^* \leqslant \mathrm{Sym}(\Omega^*)$ csoportot a következő módon. $G^* := \langle S_2 \rangle = G_{\beta_1}$. Ezen felül $B^* = B \backslash \{\beta_1\}$-re, és az $S_2, ..., S_{k+1}$-re teljesülnek a tétel feltételei, tehát az indukciós feltevés szerint G^*-nak bázisa B^*, továbbá S^* erős generátorrendszere G^*-nak, B^* bázisra nézve. Vezessük be a következő jelölést $G^{[i]} := G_{(\beta_1,...,\beta_{i-1})}$. Végül pedig ellenőriznünk, hogy az erős generátorrendszer kritériuma, vagyis hogy $G^{[i]} = \langle S \cap G^{[i]} \rangle$ teljesül-e minden i-re, ha $1 \leqslant i \leqslant k+1$. Az $i = 1$ esetre triviálisan teljesül, hiszen a tétel feltételei között adott, hogy $\langle S_1 \rangle = G$, amely halmazt tartalmazza S. Ezen felül az $i = 2$ esetre is teljesül, és azt kapjuk, hogy $G_{\beta_1} = \langle S_2 \rangle \leqslant \langle S \cap G_{\beta_1} \rangle$, amely tartalmazásnak a fordítottja is igaz. Minden további $i > 2$-re úgy kapjuk az állítást, hogy az indukciós feltevés értelmében, $S^* \cap G^{[i]}$ generálja $\langle S_2 \rangle_{(\beta_2,...\beta_{i-1})}$-t, tehát $G^{[i]} \geqslant \langle S \cap G^{[i]} \rangle \geqslant \langle S^* \cap G^{[i]} \rangle = \langle S_2 \rangle_{(\beta_2,...\beta_{i-1})} = (G_{\beta_1})_{(\beta_2,...\beta_{i-1})} = G^{[i]}$. □

Most már minden eszközünk megvan ahhoz, hogy egy T generátorrendszerével megadott $G \leqslant \mathrm{Sym}(\Omega)$ permutációcsoportnak algoritmikusan előállíthassuk egy bázisát és egy erős generátorrendszerét.

3.4.3. Algoritmus. *Legyen $G \leqslant \mathrm{Sym}(\Omega)$ permutációcsoport a $T \subseteq G$ generátorrendszerével megadva. Az algoritmus során számon tartunk három listát. Az elsőben B-ben, a $\beta_1, \beta_2, ...$ báziselemeket tároljuk. A másodikban, S-ben, a végeredményül adódó erős generátorrendszer elemeit gyűjtjük. Az erős generátorrendszert $S = \bigcup S_j$ alakban keressük, ahol $S_1, S_2, ...$ halmazokat rekurzívan definiáljuk. A célunk, hogy a halmazok teljesítsék a* 3.4.2*-es Lemma összes feltételét. Az utolsó tömbben, $R = \bigcup R_j$-ben pedig minden j-re tárolunk egy-egy $\langle S_j \rangle_{\beta_j}$ szerint vett R_j jobb oldali mellékosztály reprezentáns rendszert $\langle S_j \rangle$-ben. Továbbá ismét használjuk a korábban megismert $G^{[j]} := G_{(\beta_1,...,\beta_{j-1})}$ jelölést, amely B első $j-1$ elemének stabilizátorát jelöli G-ben.*

Az algoritmusunkat a következőképpen inicializáljuk:

- *B első eleme legyen egy tetszőleges β_1 elem, amelyre igaz, hogy $\exists t \in T$, $\beta_1^t \neq \beta_1$, tehát létezik olyan generátor elem, amely mozgatja azt*
- *$S_1 := T$, ahol $\langle T \rangle = G$*
- *Számítsuk ki egy G_{β_1} szerint vett R_1 jobb oldali mellékosztály reprezentánsrendszert a már korábban megismert módon, a* 2.1.3-*as Algoritmus segítségével*

Ahhoz, hogy az algoritmus egy általános lépését vizsgáljuk, feltesszük, hogy a futás során már előállt egy $B = (\beta_1, ..., \beta_t)$ báziselemek tömbje, illetve G részhalmazainak egy $S_1 = T, S_2, ..., S_t, S_{t+1} = \varnothing$ sorozata, amelyekre igaz, hogy $S_j \subseteq G^{[j]}$ továbbá $\langle S_j \rangle \geqslant \langle S_{j+1} \rangle$ minden j-re. Az algoritmus futása akkor ér véget, ha $\langle S_j \rangle_{\beta_j} = \langle S_{j+1} \rangle$ teljesül minden j-re, hiszen ekkor a 3.4.2-*es Lemma minden feltétele teljesül, tehát elértük a célunkat.*

A $\langle S_j \rangle_{\beta_j} = \langle S_{j+1} \rangle$ egyenlőségeket a következő módon ellenőrizhetjük. A $j = t$-től csökkenő indexeléssel haladunk. Tegyük fel, hogy ezt már megtettük egészen l-ig. Ekkor $l-1$-re a következőket tesszük. Ismét a 3.4.2-*es Lemmát alkalmazzuk, mely szerint $\bigcup_{j=l}^{t} S_j$ erős generátorrendszere az $\langle S_l \rangle$ csoportnak a $(\beta_l, ..., \beta_t)$ bázisra nézve. Ekkor* 2.1.3-*as Algoritmus segítségével megkonstruálhatunk egy $\langle S_{l-1} \rangle_{\beta_{l-1}}$ szerint vett R_{l-1} reprezentánsrendszert S_{l-1}-ben. Ennek segítségével kiszámíthatjuk az $X = \{rs\overline{rs}^{-1} \mid r \in R_{l-1},\ s \in S_{l-1}\}$ Schreier-generátorait az $\langle S_{l-1} \rangle_{\beta_{l-1}}$ csoportnak. Egyenként minden $x \in X$ Schreier-generátorra ellenőrizzük, hogy eleme-e $\langle S_m \rangle$-nek. Itt a* 3.3-*as Fejezetben leírtakhoz folyamodhatunk segítségért, hiszen az előbb állapítottuk meg, hogy $\langle S_l \rangle$-nek $\bigcup_{j=l}^{t} S_j$ erős generátorrendszere. Az ellenőrzés során két eset lehetséges.*

- *$\forall x \in X$-re $x \in \langle S_l \rangle$, ekkor az algoritmust az $l-2$ eset ellenőrzésével folytatjuk, vagy leáll, ha $l - 1 = 1$ volt. Ekkor a kimenet nem más, mint $S = \bigcup_j S_j$*
- *$\exists x \in X$, melyre $x \notin \langle S_l \rangle$*

A második esetben a szitálás során adódó $h = xr_l^{-1}r_{l+1}^{-1}...r_t^{-1} \in G^{[l]}$ elemet hozzáadjuk az S_l halmazhoz. Ebben az esetben újra kell számítanunk az R_l reprezentánsrendszert is, majd ismét ellenőriznünk, hogy $\langle S_l \rangle_{\beta_l} = \langle S_{l+1} \rangle$. Amennyiben $t = l$ esetben találunk ilyen elemet az azt jelenti, hogy a B tömbben tárolt β_k elemek még nem alkotnak bázist. Ekkor B-hez egy olyan $\beta_{t+1} \in \Omega$ elemet fűzünk, amelyre igaz, hogy h nem hagyja fixen, továbbá S_{t+2}-t, mint üres halmazt definiáljuk.

Pszeudokód - Schreier-Sims algoritmus

1: **procedure** SCHREIER-SIMS(T)
2: $S[1] \leftarrow T$
3: $B \leftarrow \beta_1$, amire $\exists t \in T, \beta_1^t \neq B_1$

4: $R[1] \leftarrow R_{\beta_1}$
5: $i \leftarrow 1$
6: **while** $i \geqslant 0$ **do**
7: *Schreier-generátorok szitálása*
8: **if** $\forall$ *Schreier-generátor* $\in \langle S_{i+1} \rangle$ **then**
9: $i \leftarrow i - 1$
10: **else**
11: $S[i+1] \leftarrow S[i+1] + h$ *(Schreier-generátor szitáltja)*
12: **if** hossz$(B) = i$ **then**
13: $B[i+1] \leftarrow \beta_{i+1}$, amelyre igaz, hogy $\beta_{i+1} \neq \beta_{i+1}^h$
14: $R[i+1]$ *transzverzális újraszámítása*
15: $i \leftarrow i + 1$
16: **return** $\bigcup S[i]$

3.4.4. Tétel. *Tetszőleges $G = \langle T \rangle$-nek, determinisztikus algoritmus segítségével, vagy $O(n^2 log^3|G| + |T|n^2 log|G|)$ idő- és $O(n^2 log|G| + |T|n)$ memóriaigénnyel, vagy $O(n^3 log^3|G| + |T|n^3 log|G|)$ idő- és $O(nlog^2|G| + |T|n)$ memóriaigénnyel állítható elő egy erős generátorrendszere.*

Bizonyítás. Az algoritmus futásidő- és memóriaköltsége a következő három részből áll,

(1) Minden i-re a $\beta_i^{\langle S_i \rangle}$ orbit meghatározása;
(2) Minden i-re az $\langle S_i \rangle_{\beta_i}$ szerint vett R_i reprezentáns rendszer meghatározása $\langle S_i \rangle$-ben;
(3) Minden i-re az S_i-ből és R_i-ből képzett Schreier-generátorok szitálása.

Először is vizsgáljuk meg, hogy mennyi β_i báziselemünk lehet egyáltalán. A B bázisról elmondható, hogy irredundáns bázis, hiszen csupán $h \in G^{[t]}$ által nem fixált elemet fűzhetünk hozzá. Ez azt jelenti, hogy a 3.1-es Fejezetben látott becslést alkalmazhatjuk, mely szerint $k = |B| \leqslant \log|G|$. Tárolnunk kell továbbá az S_i generátorrendszereket is, melyeknek a költsége

$$|S| = |S_1| + ... + |S_k| \leqslant O(|T| + \min(\log^2|G|, n \log|G|))$$

nagyságrenddel becsülhető felülről, ahol $n = |\Omega|$. Ezt úgy kapjuk, hogy észrevesszük, hogy minden j-re S_j-t egyesével növeljük, ráadásul minden ilyen növeléskor az általa generált csoport mérete is nő. Ez azt jelenti, hogy legfeljebb akkorák lehetnek, mint a leghosszabb szigorúan növekvő részcsoportlánc hossza G-ben. Ennek értéke nem lehet több, mint $\min(\log|G|, 3n/2)$[11]. Tehát alkalmazva ezt, továbbá használva a $|B|$ becslését és azt, hogy $S_1 = T$, megkapjuk a fenti becslést.

Térjünk most rá az (1)-es pont becslésére. A 2.1.2-es Tétel első esetét alkalmazva kapjuk, hogy a $\beta_i^{\langle S_i \rangle}$ orbitok összes memória költsége

$O(n|B|) \leqslant O(n \log |G|)$. Az időigénynél segítséget nyújt az a tény, hogy mivel a halmaz csupán növekszik ezért a megtalált eredményeket nem kell sosem eldobni, mindig csak az új elemeket kell vizsgálni. Ez azt is jelenti, hogy minden β_i orbit meghatározásához $O(n|S_i|)$ szükséges. Ebből azt kapjuk, hogy az összesített időigény $O(|B|log|G|n + |T|n) = O(n \log^2 |G| + n|T|)$.

A hátralevő pontok (2) és (3) esetén már két esetet kell megkülönböztetnünk attól függően, hogy milyen módon kezeljük az R_i reprezentánselem rendszereket. Az első esetben feltételezzük, hogy van elég memória ahhoz, hogy az $\bigcup_i R_i$ összes elemét tároljuk. A másodikban a 3.2-es Fejezetben megismert módon Schreier-fákban tároljuk azokat.

Nézzük tehát a (2)-es pontot. Az első esetben tudjuk, hogy $|R_i| \leqslant n$ minden i-re. Összesítve minden i-re ez pontosan $O(n \log |G|)$ darab mellékosztály reprezentánst jelent, amelyet tárolni $O(n^2 \log |G|)$ költséggel tudunk. Az időigény az orbit számítás során adódó permutáció szorzásokból adódik. Egy R_i összes elemének kiszámításához elegendő n darab ilyen műveletet végezni, amely egyenként $O(n)$ időt igényel, tehát összesen $O(n^2|B|) \leqslant O(n^2 \log |G|)$. Schreier-fákat alkalmazva egy R_i halmazhoz csupán $O(n)$ memória is elég, tehát összesen $O(n \log |G|)$ tárhelyre van szükségünk. A kiszámításuk során nem végzünk permutáció-szorzásokat, tehát csupán a Schreier-fába való beillesztésük időigényét kell számolnunk, ami $O(n \log |G|)$.

Végezetül térjünk rá az utolsó pontra, az $s \in S_i$-k és $r \in R_i$-k által meghatározott $rs(\overline{rs})^{-1}$ Schreier-generátorok szitálására. Fontos megjegyezni, hogy már ellenőrzött Schreier-generátorokat nem kell újra összeállítani és szitálni, tehát ez legfeljebb

$$O(\sum_i |S_i||R_i|) = O(\sum_{i>1} \log |G|n + |T|n) = O(n \log^2 |G| + n|T|)$$

hívást jelent. Az első esetben a hívásokban egyenként legfeljebb $|B| \leqslant log|G|$ permutáció szorzást jelent amely már az említett $O(n)$ költséggel a $O(n^2 \log^3 |G| + n^2|T| \log |G|)$ futásidőigényt eredményezi. Ezzel szemben Schreier-fákat használva R_i elemeit nem tároltuk közvetlenül, tehát ezek kiszámítását is meg kell tennünk. Ezek az elemek S_i elemeinek $O(n)$ hosszúságú szorzatokként állnak elő a következő módon $x \to h = xr_m^{-1}r_{m+1}^{-1}...r_t^{-1}$. Ez $O(n \log |G|)$ darab elem kiszámítását eredményezi minden S_i-t összegezve. Ezt szorozva a permutáció szorzás költségével, továbbá a hívások számával kapjuk az $O(n^3 \log^3 |G| + |T|n^3 \log |G|)$ futásidő becslést.

Összegezve tehát a pontokat az első esetben:
Memóriaigény:

$$\begin{aligned} O(n \log |G|) + O(n^2 \log |G| + |T|n) + O(n^2 \log |G|) \\ = O(n^2 \log |G| + |T|n) \end{aligned}$$

Időigény:

$$O(n\log^2|G| + n|T|) + O(n^2\log|G|) + O(n^2\log^3|G| + |T|n^2\log|G|)$$
$$= O(n^2\log^3|G| + |T|n^2\log|G|)$$

Ezzel szemben a második esetben:
Memóriaigény:

$$O(n\log|G|) + O(n\log^2|G| + |T|n) + O(n\log|G|)$$
$$= O(n\log^2|G| + |T|n)$$

Időigény:

$$O(n\log^2|G| + n|T|) + O(n\log|G|) + O(n^3\log^3|G| + |T|n^3\log|G|)$$
$$= O(n^3\log^3|G| + |T|n^3\log|G|)$$

□

A Schreier-Sims algoritmussal most már minden építőkockánk megvan ahhoz, hogy a 3.0.1-es Lemma harmadik pontját is igazoljuk. A kiinduló állítás tehát az, hogy $G \leqslant \mathrm{Sym}(\Omega)$ csoport egy H részcsoportjához tudunk polinomiális időben találni egy T generátorrendszert, amennyiben H-nak létezik polinomiális idejű tartalmazási tesztje, illetve a $|G:H|$ indexe $n = |\Omega|$ egy polinomjával korlátozható.

Bizonyítás. Ahhoz, hogy ezt belássuk tegyük fel, hogy a Schreier-Sims algoritmussal már kiszámítottuk G-nek egy $B = (\beta_1, ..., \beta_k)$ bázisát és erre a bázisra nézve egy S erős generátorrendszerét. Az algoritmus során H generátorrendszere mellett előállítunk egy H szerint vett X mellékosztály reprezentáns rendszert is G-ben. Legyen $G^{[i]} = G_{\beta_1,...,\beta_{i-1}}$ és $S_i = S \cap G^{[i]}$ a megszokott módon definiálva. Tudjuk továbbá, hogy $G^{[i]} = \langle S_i \rangle$ minden $1 \leqslant i \leqslant k$ esetén. Ekkor definiáljuk a következő

$$G \geqslant H = H_1 \geqslant H_2 \geqslant ... \geqslant H_k = 1$$

részcsoportláncot, ahol minden i-re, $H_i = H \cap G^{[i]} = H_{\beta_1,...,\beta_{i-1}}$. Észrevehetjük, hogy az összes H_i-re teljesülnek a kiinduló feltételeink, hiszen G_i-nek erős generátorrendszere S_i, valamint egy $g \in G$ elem eldöntése, hogy H_i-ben van-e, arra módosítható, hogy $g \stackrel{?}{\in} H$, illetve, hogy stabilizálja-e a $\beta_1, ..., \beta_{i-1}$ pontokat. Végezetül a polinomiális index is teljesül, hiszen $|G_i : H_i| \leqslant |G_1 : H_1|$. Tudjuk továbbá, hogy $H_k = \langle \varnothing \rangle$. Így tehát a $H = H_1, ..., H_k$ egy-egy generátorrendszerét rekurzívan elő tudjuk állítani.

Ezek segítségével feltehetjük, hogy valamely i-re már meghatároztunk egy $T_i \subseteq H_i$ generátorrendszert, továbbá egy ehhez tartozó X_i mellékosztály reprezentáns rendszert is. Segítségül hívjuk a Schreier-Sims algoritmus során kiszámolt G_i szerint vett R_{i-1} mellékosztály reprezentáns rendszert G_{i-1}-ben. Ekkor az $X_iR_{i-1} = \{xr \mid x \in X_i,\ r \in R_{i-1}\}$ halmaz egy H_i szerinti mellékosztály reprezentáns rendszert ad

G_{i-1}-ben. A H_i részcsoport tartalmazás tesztjének segítségével meghatározhatjuk az X_{i-1}, H_{i-1}-ben vett H_i szerinti mellékosztály reprezentáns elemeit, hiszen $X_{i-1} = X_i R_{i-1} \cap H_{i-1}$. Végezetül $T_i \cup X_{i-1}$ generátorrendszere H_{i-1}-nek. □

A 3-as Fejezetet a következő két feladatmegoldással zárjuk.

3.4.5. Feladat. *Állapítsuk meg a 3x3-as Rubik kocka legkisebb elemszámú bázisának méretét*

Megoldás. Tekintsük a 3x3-as Rubik kocka szerkezetét. 20 darab "mozgó" kisebb kockából áll, amelyből 8 darab sarokkocka, illetve 12 darab élkocka. Könnyen belátható, hogy ha egy kis kockának az egyik oldalát fixáljuk, akkor azzal a többi oldala is rögzül. Ez azt jelenti, hogy minden kis kockáról választva egy oldalt már garantáltan fixáltuk a kockát, tehát kapunk egy bázist. Vajon ez a legkisebb bázis? Szükséges mindegyik kis kockát külön rögzíteni?

A végleges megoldásban nagy segítséget nyújtott a Schreier-Sims implementáció (Függelék A), hiszen a Rubik kockáról nem sok előzetes tapasztalatom volt. Először is megmutatta, hogy elő lehet állítani 18 elemű bázist, melyből 11 élkockáról való, 7 pedig sarokkockáról. Felmerült tehát a kérdés, hogy ennyi valóban szükséges-e. Ennek ellenőrzésére kicsit módosítottam az algoritmust, hogy próbáljon meg sarokelemből legfeljebb 6-ot választani, illetve egy másik futásnál pedig az oldalelemek számát korlátoztam 10-re. Mindkettő esetben hibába futott az algoritmus, nem tudott előállítani bázist és erős generátorrendszert, de sikeresen adott egy olyan szitáltat amely megmutatta, hogy adott két élkocka esetén van olyan eleme a Rubik kocka csoportjának, amely a két kockát helyben hagyja, de az oldalaikat megcseréli, illetve két megmaradó sarokkocka esetén, azokat is helyben hagyja de ellenkező irányba forgatja őket. Ebből következik, hogy ahhoz, hogy a sarokkockákat fixáljuk legalább 7-et ki kell választanunk közülük, míg az élkockák esetében ez a szám 11. Ebből azt kapjuk, hogy legalább 18 eleműnek kell lennie egy bázisnak, amelyre már kaptunk is példát.

3.4.6. Feladat. *Seress Ákos*[1] *4.7 feladat (Sims*[4]*)*
Készítsünk egy algoritmust, mely egy meglévő erős generátorrendszerből kiindulva kiszámít egy olyan erős generátorrendszert, amelynek számossága legfeljebb log$|G|$, végül vizsgáljuk meg a komplexitását.

Megoldás. Segítségként meg volt adva, hogy a meglévő erős generátorrendszert célszerű sorba rendezni, majd megvizsgálni, hogy generálja-e egy addig tekintett szűkebb részhalmaz.

Legyen tehát S egy erős generátorrendszere G-nek, a B bázisra nézve, azaz $S_i := S \cap G^{[i]}$ generálja $G^{[i]}$-t minden i-re. Keressük azt a $T \subseteq S$-t, melyre $|T| \leqslant \log |G|$. Tudjuk, hogy X tetszőleges generátorrendszer esetén ki tudunk választani $\log |G|$ elemű részhalmazát X-nek, amely még mindig generálja G-t. Ezt úgy tudjuk megtenni, hogy tetszőleges

sorrendben végigmegyünk az elemeken és kidobjuk azokat amelyek benne vannak a korábbi elemek által generált részcsoportban. Alkalmazva ezt S-re azt is biztosítanunk kell, hogy miközben eldobunk egyes elemeket, T-nek fenn kell tartani azt a tulajdonságát, hogy $\langle T_k \rangle = \langle S_k \rangle$, ahol $T_k := T \cap S_k$. Legyen $m = |B|$, ekkor az előző tulajdonságot úgy tudjuk garantálni, hogy S elemeit sorba rendezzük úgy, hogy S_m elemei jöjjenek először, majd S_{m-1} elemei, utána S_{m-2} elemei, és így tovább. Ezt a felsorolást megtehetjük a mozgatott báziselemek alapján, ahogy korábban láttuk. Ez $O(|S|)$ időben megoldható, mert valójában csak halmazokba rendezünk, az azon belüli sorrend nem számít. Ekkor T_m részt kiválaszthatunk S_m-ből úgy, hogy $\langle T_m \rangle = \langle S_m \rangle$. Ebben a T_m-ben nincs egy olyan elem sem, amely benne lenne a többi elem által generált részcsoportban. Ezután $S_m \backslash S_{m-1}$ elemek közül veszünk hozzá újakat, amivel megkapjuk T_{m-1}-et. Ezt folytatva eljutunk T-hez, melyre $T \cap S_k$ és $\langle T_k \rangle = \langle S_k \rangle = G^{[k]}$ minden k-ra, tehát T is erős generátorrendszer. Az elemek elhagyását a következő módon tehetjük meg. Tegyük fel, hogy már megtaláltuk $T_k, T_{k+1}, ..., T_m$ halmazokat, hogy minden $i \geqslant k$-ra teljesül, hogy $\langle T_i \rangle = \langle S_i \rangle = G^{[i]}$. Ekkor $G^{[k-1]} = \langle S_{k-1} \rangle = \langle T_k, S_{k-1} \backslash S_k \rangle$. Ekkor tetszőleges I részhalmazát választva $S_{k-1} \backslash S_k$-nak, $\{T_k \cup I\}$ erős generátorrendszere $\langle T_k \cup I \rangle$-nek. Ez azt jelenti, hogy $S_{k-1} \backslash S_k$-n tetszőleges sorrendben haladva egy új elemek ellenőrzésekor alkalmazhatjuk a szitáló algoritmust, továbbá, ha az elemet hozzá kell adnunk a T_{k-1} halmazhoz, akkor újra kell számítanunk a mellékosztály reprezentáns rendszerünket is. Ez tehát összesen legfeljebb $|S|$ darab szitálást jelent, ezen felül minden i-re ki kell számolnunk a mellékosztály reprezentáns rendszereket. Ezeket a 3.4.4-es Tétel bizonyítása alapján, $O(|S|n|B|)$, illetve $O(|B|n^2)$ futásidővel tehetjük meg. Tehát összesen azt kapjuk, hogy $O(|S|n^2|B|)$, amely irredundáns bázis esetében $O(|S|n^2 \log |G|)$ időt jelent.

4. Blokkok, blokkrendszerek

Ahhoz, hogy a dolgozat elején felvetett gráfizomorfiát vizsgáljuk, Luks[2] megfogalmaz két lemmát is.

4.0.1. Lemma. *Legyen $G \leqslant S_n$ egy S generátorrendszerével megadva, valamint egy G-orbitot, B-t. Ekkor polinomidőben megtalálhatunk egy minimális G-blokk rendszert B-ben.*

A következő lemma szerint pedig gyorsan meghatározhatjuk az előző lemma alapján megtalálható blokkrendszer stabilizátorát is.

4.0.2. Lemma. *Tekintsük egy S generátorrendszerét a $G \leqslant S_n$ csoportnak, illetve egy G-orbitot, B-t. Legyen továbbá $B_1, B_2, ..., B_k$ egy G-blokk rendszer B-ben és $H \leqslant G$ ezen blokkok stabilizátorainak metszete. Ekkor H egy generátorrendszere polinomidőben megtalálható.*

A továbbiakban azt vizsgáljuk meg, hogyan is tudunk ilyen blokkrendszereket előállítani.

4.1. Adott részhalmazt tartalmazó legkisebb blokk. Egy tranzitív csoport blokkjait meghatározó algoritmusok fontos szerepet töltenek be a permutációcsoport algoritmusai között. Sok más feladathoz fel lehet használni, hogy a problémát visszavezesse primitív permutációcsoportokra. M. D. Atkinson publikált először ilyen eljárást 1975-ben[7]. Különlegességéhez az is hozzátartozik, hogy egyike azon kevés algoritmusoknak, amelyhez nem szükséges egy erős generátorrendszer. Ennek hátterében az áll, hogy a blokkok kiszámítását, meg lehet oldani egy költségesebb orbit számítási feladatként.

Egy közel lineáris időben futó blokkokat számító algoritmust részletesen, implementációval Seress Ákos[1] könyvének 5.5.1-es fejezetében lehet olvasni. A következő alfejezetben egy olyan algoritmust vizsgálunk[8], mely egy részhalmazt tartalmazó legkisebb blokkot számít ki.

Adott $G = \langle S \rangle \geqslant \mathrm{Sym}(\Omega)$, $\Omega = [1, n]$ tranzitív csoport, és egy $\Delta \subseteq \Omega$. Szeretnénk egy olyan Σ blokkrendszerét G-nek, melyre Δ elemei ugyanabba a blokkba tartoznak.

4.1.1. Algoritmus. *Az algoritmus futása során szükség lesz arra, hogy számon tartsuk Ω-nak egy Π partícióját. Kezdetben Π olyan alakú, hogy benne van Δ továbbá $|\Omega| - |\Delta|$ darab egyelemű részhalmaz. Az algoritmus során, a Π-beli halmazokat fogjuk egybevonni, úgy, hogy végig megtartjuk azt a tulajdonságot, hogy a pontok, amelyek egy részhalmazban vannak Π-ben, azok a Σ-val jelölt G-blokkrendszerben egy blokkban lesznek.*

Első lépésként megállapíthatjuk, hogy amennyiben $\alpha, \beta \in \Omega$ ugyanabban a Π-beli halmazban vannak, akkor tetszőleges $s \in S$-re vett képüknek is ugyanazon halmazba kell esniük. Következményképpen $A_1, A_2 \in \Pi$ halmazokat $A_1 \cup A_2$ halmazra cseréljük a felosztásban, ha valamely ugyanabba a halmazba eső $\alpha, \beta \in \Omega$-ra és $s \in S$-re igaz, hogy $\alpha^s \in A_1$ de $\beta^s \in A_2$. Az algoritmust addig futtatjuk, míg $\forall s \in S$ és $\forall A \in \Pi$-re $\exists B \in \Pi$, amire $A^s = B$, amely feltétel teljesülésekor $\Sigma := \Pi$.

Pszeudokód - Adott részhalmazt tartalmazó legkisebb blokkrendszer

```
1: procedure LEGKISEBB BLOKK(S, Δ)
2:     Π ← Δ ∪ {α}, ∀α ∈ Ω\Δ
3:     while ∃A ∈ Π, ∃s ∈ S, melyre ∄B ∈ Π, hogy A^s = B do
4:         if ∃α, β ∈ A és s ∈ S, melyre α^s ∈ A_j, de β^s ∈ A_k then
5:             A_j ← A_j ∪ A_k
6:     return Π
```

Az algoritmus hatékony végrehajtásához, egy különleges, úgynevezett *Unió-Keresés adat struktúrát* használunk. Ebben fogjuk tárolni a Π-beli halmazokat. Minden A halmazt egy T_A gyökeres fában tárolunk, melynek csúcsait A elemeinek feleltetjük meg, míg a gyökér az egész halmazra nézve is hordoz információt. Minden csúcs három cellából áll, melyekből csupán kettőt használunk. Az első tartalmazza a tényleges elemet, a második a csúcs szülőjére mutat. A harmadik cellát csak a gyökérnél használjuk amely $|A|$-t tárolja. A gyökér második, szülőre mutató cellája önmagára mutat. Ezen az adat struktúrán kétfajta műveletet definiálunk. Egy Kereső és egy Unió műveletet. Legyen l, T_A-ban a gyökér és α-t összekötő út hossza, ekkor a keresést $\Theta(l)$ időben tudjuk lefuttatni. A kereső művelet egy $\alpha \in \Omega$-ból kiindulva megtalálja azon T_A fa gyökerét, melyre $\alpha \in A$, úgy, hogy a csúcsok 2. cellájában levő mutatókon végiglépkedve követjük az utat α-ból az őt tartalmazó T_A gyökeréig. Sokszor megelőző jelleggel még egyszer végigfutunk α-tól a gyökérig és minden csúcs 2. cellájára rögtön a már ismert gyökeret írjuk, hogy később megkönnyítsük a keresést (összeomlasztjuk a fát). Ez nem változtatja meg az előbb említett futási idő nagyságrendjét.

Az Unió egy kétváltozós művelet, amely az $A, B \in \Pi$ halmazokat cseréli le az uniójukra. Megkeressük T_A, T_B gyökereit és a cellákat úgy módosítjuk, hogy az egyikük gyökerét mostantól a másik gyerekének tekintjük, tehát az egyik halmaz gyökerének második cellája a másik halmaz gyökerére fog mutatni, ezzel a két halmazt összevonjuk, az újonnan létrejövő nagyobb $T_{A\cup B}$ fa gyökerének 3. cellájába pedig $|A| + |B|$-ot írunk. Általában érdemes az úgynevezett súlyozott unió szabályt alkalmazni, vagyis $|A| \geqslant |B|$ esetén az új szülő gyökér legyen T_A gyökere. Az Unió művelet költsége tekinthető konstansnak.

4.1.2. Tétel. *Legyen $G = \langle S \rangle \leqslant \mathrm{Sym}(\Omega)$, $\Omega = [1, n]$ tranzitív csoport és $\Delta \subseteq \Omega$. Ekkor azt a legkisebb blokkját G-nek, amely tartalmazza Δ-t, $2|S|n$ darab Keresés és kevesebb, mint n darab Unió művelet használatával ki tudjuk számítani.*

Bizonyítás. Kezdjük el Π már fentebb említett felosztásával, és hozzuk Δ-t gyökeres fa alakúra, úgy, hogy tetszőleges elemét kijelöljük gyökérnek és minden más eleme ennek lesz a gyereke, így kapunk egy egy magasságú fát. Minden más részhalmaz egy csak gyökérből álló faként fogható fel. Ahhoz, hogy két halmazt összevonjunk szükségünk van arra, hogy két elemet össze tudjunk hasonlítani, hogy képeik minden $s \in S$-re ugyanazon halmazba esnek-e, ezért az algoritmus futása során szükségünk van egy L listára amely tartalmazza az összehasonlítandó párokat. Kezdetben Δ gyökerét hasonlítjuk össze Δ minden másik elemével. Az algoritmus addig fut, amíg ez a lista ki nem ürül. Minden listaelemet megvizsgálunk a következőképpen. Legyen $\{\alpha, \beta\}$ a sorra kerülő elem. Ekkor a feltétel szerint meg kell vizsgálnunk minden

$s \in S$ -re, hogy α^s, β^s egy halmazba esnek-e, így mindkettőre futtatunk egy keresést. Ha az α-ból, illetve β-ból induló keresések ugyanazt a gyökeret szolgáltatják, elhagyjuk $\{\alpha, \beta\}$-t és folytatjuk L feldolgozását. Ha különböznek, akkor futtatunk $A, B \in \Pi$-re egy Uniót és a listához adjuk azt az $\{r_A, r_B\}$ párt, ahol r_A és r_B, a T_A és T_B fák gyökereinek megfelelő Ω-beli elemek.

Vizsgáljuk meg $A_1 \subsetneq A_2 \subsetneq ... \subsetneq A_m$, $A_m \in \Pi$ halmazsorozatot amely valamely $\alpha \in \Omega$-t tartalmazó, az algoritmus során bővülő halmazok láncolata. Vegyük ezeknek a halmazokhoz tartozó fáknak az $(r_{A_1}, ..., r_{A_m})$ gyökérsorozatát. Ebben a sorozatban, ha két egymást követő elem nem egyezik meg, példaként legyen $r_{A_i} \neq r_{A_{i+1}}$, az azt jelenti, hogy a fa gyökere megváltozott, tehát Unió műveletet hajtottak végre a halmazon és egy másikon. Ennek következményeképpen, az $\{r_{A_i}, r_{A_{i+1}}\}$ pár belekerül az L listába. Továbbá $\{r_{A_1}, \alpha\}$ is szerepel a listában, ha $r_{A_1} \neq \alpha$, hiszen ez azt jelenti, hogy a kezdeti gyökér mellett a halmaznak van egyéb eleme is, ami a Π inicializálása szerint csak Δ-ra igaz és L listát pont az ilyen típusú párokkal inicializáljuk. Ezek miatt kijelenthető, hogy $\forall s \in S$-re α^s és $r^s_{A_m}$ egy halmazba esnek. A Π összes halmazára ugyanezt elmondhatjuk, tehát az algoritmus leállása után kapott Π halmazait valóban permutálja G.

Ennek értelmében számoljuk össze, hányszor futtattuk le a műveleteket. Eleinte Π halmazainak száma $|\Omega| - |\Delta| + 1$ volt, tehát legfeljebb $|\Omega| - |\Delta| \leqslant n - 1$ Uniót futtathatunk le, illetve az algoritmus szerint minden uniózásnál hozzáadunk egy elemet az L listánkhoz tehát L legfeljebb $|\Omega| - |\Delta|$ elemmel bővülhet. Az L lista ezen felül $|\Delta| - 1$ elemmel bír kezdetben, tehát $|L|$ nem lehet több $n - 1$-nél.

□

Az algoritmus használható arra is, hogy elemszámra tekintve maximális G-blokkokat találjunk. Ezt úgy tehetjük meg, hogy rögzítünk egy $\omega \in \Omega$ elemet, majd minden $\zeta \in \Omega$, $\zeta \neq \omega$ elemre megkeressük a legkisebb $\{\zeta, \omega\}$-t tartalmazó G-blokkot. Amennyiben a hatás nem primitív mindenképp találunk egy nem-triviális G-blokkrendszert ilyen módon. Ezután folytassuk a folyamatot G indukált hatásával ezen a blokkrendszeren, ahol ugyanezeket a lépéseket tehetjük, amíg a vizsgált hatás nem lesz primitív, ekkor a blokkrendszerünk minimális. Ezzel a 4.0.1-es Lemmát beláttuk.

A 4.0.2-es Lemmához nincs másra szükségünk csupán segítségül kell hívnunk a 3.0.1-es Lemmát, melynek harmadik pontját alkalmazva készen vagyunk. Legyen $B_1, .., B_k$ egy G-blokkrendszer, és $G \geqslant G_1 \geqslant G_2 \geqslant ... \geqslant G_k$ részcsoportok csökkenő lánca, ahol G_i a $B_1, ..., B_i$ blokkok mindegyikét stabilizáló elemek halmaza. Ekkor kiszámíthatjuk $G_1, G_2, ...$ egy-egy generátorrendszerét polinomidőben, hiszen minden i-re, a $g \overset{?}{\in} G_i$ kérdést, a $g(B_i) = B_i$ ellenőrzésével könnyen eldönthetjük. Ráadásul $|G_i : G_{i+1}| \leqslant$ blokkok száma $\leqslant n$.

5. Korlátos fokszámú gráfok izomorfiája

Ebben a fejezetben Luks[2] cikkének eredményét tárgyaljuk. Ahogy az eredeti publikációban is szerepel, az alapötletet trivalens gráfokon ismertetjük először, majd az ebben szerzett tapasztalatokat, észrevételeket általánosítjuk, hogy végül elérkezzünk a célhoz, és megadjunk egy polinomiális futásidejű algoritmust, mely két tetszőleges korlátos fokszámú gráfról eldönti, hogy izomorfok-e. Az első alfejezetben részletesen vizsgáljuk a trivalens eset meggondolásait, de a másodikban az általános esetet csupán vázlatosan ismertetjük.

5.1. A trivalens eset. Az alfejezetben olyan gráfok izomorfiáját szeretnénk eldönteni, melyekről tudjuk, hogy pontjaik foka legfeljebb 3. Első lépésként vizsgáljuk meg, hogy hogyan tudjuk a problémát lefordítani a csoportelmélet nyelvére.

5.1.1. Állítás. *Trivalens gráfok izomorfiájának eldöntése polinomidőben visszavezethető arra, hogy egy X összefüggő trivalens gráfnak, egy kitüntetett e éléhez tartozó $\mathrm{Aut}_e(X)$ automorfizmus csoport egy generátorrendszerét meghatározzuk.*

Bizonyítás. Tegyük fel, hogy van egy eljárásunk, amely tetszőleges X trivalens gráfra és annak egy e élére kiszámítja $\mathrm{Aut}_e(X)$ egy generátorrendszerét.

Tekintsünk két trivalens gráfot X_1-et és X_2-t. Feltehető, hogy X_1 és X_2 összefüggőek. Ekkor az élek segítségével megpróbáljuk meghatározni, hogy izomorfok-e. Rögzítsünk tehát egy $e_1 \in E(X_1)$ élt az első gráfból, majd vizsgáljuk meg X_2 összes $e_2 \in E(X_2)$ élére, hogy létezik-e e_1-et, e_2-re leképező izomorfizmus. Az előzetes áttekintésben megfogalmazott eljáráshoz hasonlóan, ezt úgy tudjuk megtenni, hogy készítünk egy közös gráfot a két gráf diszjunkt uniójának kiegészítésével. Jelen esetben a két gráfot összefüggővé szeretnénk tenni, úgy, hogy a trivalens tulajdonság is megmaradjon. Induljunk ki X_1 és X_2 diszjunkt uniójából, majd osszuk ketté e_1-et, illetve e_2-t egy-egy új csúccsal. Legyenek ezek a csúcsok v_1 és v_2. Ezt a két pontot kössük össze egy e éllel. Jelöljük az így kialakuló gráfot X-szel, amelyről azt is tudjuk, hogy összefüggő és trivalens gráf. A két eredeti gráf között tehát akkor és csak akkor létezik izomorfizmus, amely e_1-et e_2-be képezi, ha $\mathrm{Aut}_e(X)$-nek létezik olyan eleme, amely megcseréli v_1-et és v_2-t. Továbbá igaz az is, hogy ha létezik ilyen automorfizmus, akkor tetszőleges generátorrendszerben lesz olyan elem amely teljesíti ezt kritériumot.

A feltevésünk szerint előállíthatjuk $\mathrm{Aut}_e(X)$ egy generátorrendszerét. Az előállított generátorrendszert megvizsgálva, hogy létezik-e eleme, amely felcseréli v_1-et és v_2-t, eldönthetjük, hogy létezik-e olyan $X_1 \to X_2$ izomorfizmus, mely e_1-et e_2-be viszi. Az eljárást X_2 összes élére elvégezve eldönthetjük, hogy X_1 és X_2 izomorfok-e. Összesen $|E(X_2)| = O(|X_2^2|)$ alkalommal kell meghatároznunk egy-egy X gráfra az $\mathrm{Aut}_e(X)$

egy generátorrendszerét, ami polinomiális idejű visszavezetést jelent. □

A következő lépés tehát nem más, mint hogy egy tetszőleges X összefüggő és trivalens gráf kiválasztott e élére meg tudjuk határozni $\mathrm{Aut}_e(X)$-et. Ezt a következő módon tesszük. Definiáljuk részgráfok egy

$$X_1 \subseteq X_2 \subseteq ... \subseteq X_n = X$$

sorozatát. Minden r pozitív egészre legyen X_r, az e-t tartalmazó legfeljebb r hosszú utak uniója. Ekkor X_1 csupán e-t és a két végpontját tartalmazza, valamint $X_n = X$ mindenképp teljesül az $n = |V(X)|$ választással. Látszik, hogy $Aut_e(X)$ halmazként fixen hagyja mindegyik X_r-et. Az X_r gráfok e-t fixen hagyó automorfizmusainak csoportjait jelöljük $\mathrm{Aut}_e(X_r)$-el. Ezek a csoportok összekapcsolhatóak a következő homomorfizmusokkal,

$$\pi_r : \mathrm{Aut}_e(X_{r+1}) \to \mathrm{Aut}_e(X_r)$$

ahol a π_r indukált homomorfizmust úgy definiáljuk, hogy tetszőleges $\sigma \in \mathrm{Aut}_e(X_{r+1})$-re, legyen $\pi_r(\sigma)$, a σ megszorítása az $X_r \subset X_{r+1}$ halmazra.

Felismerhető, hogy X_1 automorfizmus csoportját könnyen megadhatjuk, tehát a kisebb részgráfok felől szeretnénk a nagyobb felé haladni, ezért a probléma során $\mathrm{Aut}_e(X_r)$ generátorrendszerének ismeretét feltételezzük és ebből szeretnénk meghatározni $\mathrm{Aut}_e(X_{r+1})$ generátorrendszerét. Vezessük be a következő jelöléseket. Legyen $K_r := \ker(\pi_r)$, a π_r magja, valamint $H_r := \mathrm{Im}(\pi_r)$, a π_r képe. A célunk az, hogy meghatározzuk egy-egy generátorrendszerét K_r-nek és H_r-nek. Amennyiben rendelkezünk már K_r-nek egy S_r generátorrendszerével, továbbá találunk egy $T_r \subseteq X_{r+1}$ halmazt, amelyre teljesül, hogy $\pi_r(T_r)$ generátorrendszere H_r-nek, akkor $S_r \cup T_r$ generálja $\mathrm{Aut}_e(X_{r+1})$-et. Egy ilyen T_r halmazt könnyen megkaphatunk a H_r generátorainak egy-egy $\mathrm{Aut}_e(X_{r+1})$-beli elemmé való kiterjesztésével.

Ahhoz, hogy ezeket a halmazokat meghatározhassuk, jobban meg kell vizsgálnunk két részgráf "határát", vagyis $V(X_{r+1})\backslash V(X_r)$-t. Trivalens gráfok lévén egy $V(X_{r+1})\backslash V(X_r)$-beli pont $V(X_r)$-beli szomszédjainak a száma legalább egy, de legfeljebb három lehet. Csoportosítsuk most a $V(X_{r+1})\backslash V(X_r)$ halmaz pontjait a következőképpen. Két pontot *ikreknek* nevezünk akkor, ha a $V(X_r)$-beli szomszédaik megegyeznek. Hármas ikrek a fokszám korlátozás miatt nem létezhetnek. Észrevehetjük, hogy egy $\sigma_1 \in \mathrm{Aut}_e(X_{r+1})$-beli elem az azonos számú $V(X_r)$-beli szomszédokkal rendelkező pontokat legfeljebb egymás között cserélgetheti. Továbbá egy $\sigma_2 \in K_r$ permutáció, tudván, hogy X_r minden elemét helyben kell, hogy hagyja, legfeljebb az ikreket cserélgetheti egymás között. Az is világos, hogy X_{r+1} csúcsainak minden olyan permutációja, mely X_r minden csúcsát helyben hagyja,

és minden $V(X_{r+1})\backslash V(X_r)$-beli ikerpárt halmazként helyben hagy, eleme K_r-nek, így K_r-et generálják az egy-egy ikerpárt felcserélő transzpozíciók. Írjuk le ezeket az észrevételeket formálisan is. A szomszédsági kapcsolathoz először is vezessük be a következő fogalmakat, jelöléseket.

A szomszédsági kapcsolatot a következő függvénnyel írhatjuk le.

$$f_r : V(X_{r+1})\backslash V(X_r) \to A_r$$

$$f_r(v) = \{w \in V(X_r) \mid (v, w) \in E(X)\}$$

ahol A_r az a halmaz amely $V(X_r)$, egy-, két-, illetve háromelemű részhalmazait foglalja magába. A $v_1 \neq v_2 \in V(X_{r+1})\backslash V(X_r)$ pontok pontosan akkor ikrek, ha $f_r(v_1) = f_r(v_2)$. Továbbá egy tetszőleges $\sigma_1 \in \mathrm{Aut}_e(X_{r+1})$ esetén teljesül, hogy $f_r(\sigma_1(v_1)) = \sigma_1(f_r(v_1))$.

Ekkor tehát, $v \in V(X_{r+1})\backslash V(X_r)$ és $\sigma \in K_r$ esetén, tudván, hogy σ fixálja X_r minden elemét, $f_r(v) = f_r(\sigma(v))$, tehát vagy $v = \sigma(v)$ igaz, vagy v és $\sigma(v)$ ikrek.

Annak segítségével, hogy $|\mathrm{Aut}_e(X_{r+1})| = |H_r| \cdot |K_r|$ indukcióval kaphatjuk Tutte következő állítását.

5.1.2. Állítás. *Minden r-re, $\mathrm{Aut}_e(X_r)$ egy 2-csoport.*

Most, hogy K_r egy generátorrendszerét már megtaláltuk, próbáljuk meg megtalálni H_r egy generátorrendszerét is. Induljunk ki megint abból a tényből, hogy $\sigma \in Aut_e(X_{r+1})$ és $v \in V(X_{r+1})\backslash V(X_r)$ esetén $f_r(\sigma(v)) = \sigma(f_r(v))$. Ebből az is következik, hogy minden $\sigma \in H_r$ stabilizálja A_r azon a elemeinek halmazát, melyekre $a = f_r(v)$ egyetlen $v \in V(X_{r+1})\backslash V(X_r)$-beli elemre teljesül. Hasonlóan, stabilizálja A_r azon a elemeinek halmazát, melyekre $a = f_r(v) = f_r(w)$ teljesül egy v, w ikerpárra. Ezen felül stabilizálja azokat az éleket, amelyek $V(X_r)$-beli pontokat kötnek össze, de csak X_{r+1}-ben jelennek meg. Ez a három halmaz a következő,

$$A_{r,1} = \{a \in A_r \mid \exists! v \in V(X_{r+1})\backslash V(X_r),\ a = f_r(v)\} \tag{1}$$

$$A_{r,2} = \{a \in A_r \mid a = f_r(v_1) = f_r(v_2), v_1 \neq v_2\} \tag{2}$$

$$A'_r = \{\{w_1, w_2\} \in A_r \mid (w_1, w_2) \in E(X_{r+1})\backslash E(X_r)\} \tag{3}$$

Ezzel a megállapítással megágyaztunk a következő állításnak.

5.1.3. Állítás. *A H_r pontosan azon $\sigma \in Aut_e(X_r)$ elemek halmaza, amelyek stabilizálják az $A_{r,1}, A_{r,2}, A'_r$ halmazokat.*

Bizonyítás. Azt szeretnénk belátni, hogy egy tetszőleges $\sigma \in \mathrm{Aut}_e(X_r)$ kiegészíthető egy $\mathrm{Aut}_e(X_{r+1})$-beli elemmé abban az esetben, ha valóban stabilizálja az $A_{r,1}$, $A_{r,2}$ A'_r halmazokat. Tekintsük sorra, hogy mi a teendőnk az egyes halmazok esetében.

- Egy v pontot, melyre $f_r(v) \in A_{r,1}$, hozzárendeljük a $\sigma(f_r(v))$ egyetlen $V(X_{r+1})\backslash V(X_r)$-beli szomszédjához, hiszen $\sigma(f_r(v))$ is eleme $A_{r,1}$-nek
- Egy v, v' ikerpár elemeit, melyekre $f_r(v) = f_r(v') \in A_{r,2}$, hozzárendeljük $\sigma(f_r(v))$, $V(X_{r+1})\backslash V(X_r)$-beli szomszédjaihoz, valamilyen sorrendben, hiszen $\sigma(f_r(v))$ is eleme $A_{r,2}$-nek
- A'_r esetén nincs teendőnk, hiszen a stabilizálási feltevésünk miatt automatikusan teljesül a megfelelő leképezés az "új" élek és "régi" pontok között

□

A H_r egy generátorrendszerének kiszámítása még mindig nem megoldott, de az eddigi észrevételekkel már vissza tudjuk vezetni a következő problémára.

5.1.4. Probléma. *Legyen A egy színezett halmaz. Szeretnénk egy algoritmust, amelynek bemenete S egy $G \leqslant \mathrm{Sym}(A)$ 2-csoport generátorrendszere. Kimenetként pedig megkapjuk $H \leqslant G$ generátorrendszerét, ahol H a G-beli színtartó leképezések részcsoportja.*

Folytassuk a visszavezetést. A színezett halmazunk természetesen az A_r halmaz lesz. Legyen $A_{r,0} = A_r\backslash(A_{r,1} \cup A_{r,2})$. Ekkor meghatározhatunk 6 diszjunkt részhalmazt A_r-ben.

$$A_{r,0} \cap A'_r,\ A_{r,1} \cap A'_r,\ A_{r,2} \cap A'_r,\ A_{r,0}\backslash A'_r,\ A_{r,1}\backslash A'_r,\ A_{r,2}\backslash A'_r$$

Ezeket a halmazokat különböző színekkel színezve, a H_r megegyezik ezen A színezett halmaz $\mathrm{Aut}_e(X_r)$ színtartó elemeinek halmazával.

A továbbiakban a célunk, hogy a csoport hatását az A halmazon rekurzívan feldaraboljuk kisebb problémákra orbitok, vagy tranzitív esetben blokkok mentén, így állítva elő végeredményben a kimenetet. Ehhez az előbbi problémát még tovább kell általánosítanunk. Vezessük be a következő jelöléseket.

(1) Legyen A egy n elemű színezett halmaz. Az $a,\ b \in A$ esetén jelöljük $a \sim b$-vel azt, hogy a színe megegyezik b színével.
(2) Legyen $B \subseteq A$ továbbá $K \subseteq \mathrm{Sym}(A)$. Ekkor legyen $\mathcal{C}_B(K) := \{\sigma \in K \mid \forall b \in B,\ \sigma(b) \sim b\}$

A $\mathcal{C}_B(K)$ halmaznak azonnal látszik a következő két tulajdonsága.

- $\mathcal{C}_B(K \cup K') = \mathcal{C}_B(K) \cup \mathcal{C}_B(K')$
- $\mathcal{C}_{B\cup B'}(K) = \mathcal{C}_{B'}(\mathcal{C}_B(K))$

Ezek segítségével felírhatjuk a probléma következő általánosítását.

5.1.5. Probléma. *Legyen A egy színezett halmaz. Keresendő egy olyan algoritmus, amely bemenetként várja egy $G \leqslant \mathrm{Sym}(A)$ 2-részcsoport egy S generátorrendszerét, egy G-stabil B részhalmazát A-nak, továbbá egy tetszőleges $\sigma \in \mathrm{Sym}(A)$ permutációt. Kimenetként pedig a $\mathcal{C}_B(\sigma G)$ halmazt várjuk.*

Az eredeti 5.1.4. Problémát $B = A$, $\sigma = 1$ választással kapjuk vissza. A feladatot megoldó algoritmustól elvárjuk, hogy bemenetként elfogadja egy csoport generátorrendszerét, illetve egy reprezentáns elemet, vagyis a bemenet lehessen mellékosztály is. Hasonlóan a kimenettől is megköveteljük ezt. A következő lemma biztosítja, hogy $\mathcal{C}_B(\sigma G)$ tényleg megadható ilyen módon.

5.1.6. Lemma. *A $\mathcal{C}_B(\sigma G)$ vagy az üres halmaz, vagy egy bal oldali mellékosztálya $\mathcal{C}_B(G)$-nek.*

Bizonyítás. Megállapítható, hogy $\mathcal{C}_B(G)$ egy részcsoportja G-nek. Ez abból következik, hogy a problémafelvetésben feltettük, hogy B egy G-stabil halmaz. Ekkor tetszőleges $\sigma_0 \in \mathcal{C}_B(\sigma G)$-ra fennáll, hogy $\sigma G = \sigma_0 G$, hiszen ha $\sigma_0 \in \mathcal{C}_B(\sigma G) \subset \sigma G$, akkor $\sigma_0 \in \sigma_0 G \cap \sigma G$, tehát $\sigma_0 G = \sigma G$. Továbbá B minden elemére igaz lesz, hogy ha egy g csoportelemmel hatunk rá, akkor B-ben marad, vagyis $g(b) \in B$ minden $g \in G$-re és $b \in B$-re teljesül. Ezekből a megállapításokból következik, hogy $\sigma_0 g(b) \sim g(b)$. Ebből pedig a $\sigma_0 g \in \mathcal{C}_B(\sigma_0 G)$ akkor és csak akkor teljesül, ha $g \in \mathcal{C}_B(G)$, tehát $\mathcal{C}_B(\sigma_0 G) = \sigma_0 \mathcal{C}_B(G)$. □

Ezek alapján bemenetként egy $B \subseteq A$ színezett részhalmazt, egy $G \leqslant \mathrm{Sym}(A)$ 2-részcsoport egy generátorrendszerét illetve egy $\sigma \in \mathrm{Sym}(A)$ elemet kapunk. Ekkor két eset lehetséges.

- $B = B_1 \cup B_2$ két G-stabil részhalmaz uniója
- G tranzitívan hat B-n, ekkor a $B = B_1 \cup B_2$, két G-blokkra osztható

Az első, intranzitív esetben nincs más dolgunk mint rekurzívan tovább haladni a két részhalmazon, hiszen $\mathcal{C}_B(\sigma G) = \mathcal{C}_{B_1}(\mathcal{C}_{B_2}(\sigma G))$.

Tranzitív esetben szükségünk lesz még a következő lemmára.

5.1.7. Lemma. *Legyen $P \leqslant Sym(A)$ tranzitív p-részcsoport és $|A| > 1$. Ekkor P-blokkok egy tetszőleges minimális rendszere pontosan p darab blokkból áll. Ezen felül arra a $P' \leqslant P$ részcsoportra, amely stabilizálja egy minimális blokkrendszer elemeit, igaz, hogy $|P : P'| = p$.*

Bizonyítás. [10] A P/P' faktorcsoport egy primitív p-csoportként hat a blokkokon. Ebből következik, hogy P/P' rendje egyenlő a blokkok számával, p-vel. □

Azonban ebben az esetben nem tudjuk közvetlenül kiszámítani a $\mathcal{C}_{B_1}(\sigma G)$ halmazokat, hiszen a tranzitivitás miatt sem B_1, sem B_2 nem G-stabil. Ekkor segítségül hívjuk a 4. Fejezet eredményeit, melyek szerint egy $G \leqslant S_n$ és egy B-vel jelölt G-orbit esetén polinomiális időben megtalálható egy B-beli G-blokkrendszer H stabilizátorának egy generátorrendszere.

Ekkor $G = H \cup \tau H$ tetszőleges $\tau \in G$ elemre, amely felcseréli B_1-et és B_2-t (sőt τ választható a G generátorrendszeréből). Továbbá a $\mathcal{C}_B(K)$

tulajdonságait alkalmazva,

$$\mathcal{C}_B(\sigma G) = \mathcal{C}_B(\sigma H) \cup \mathcal{C}_B(\sigma\tau H) = \mathcal{C}_{B_1}(\mathcal{C}_{B_2}(\sigma H)) \cup \mathcal{C}_{B_1}(\mathcal{C}_{B_2}(\sigma\tau H))$$

Az 5.1.6-os Lemma értelmében, ha a $\mathcal{C}_B(\sigma H)$ és $\mathcal{C}_B(\sigma\tau H)$ mindketten nem üres halmazok, akkor ezek egy-egy $\mathcal{C}_B(H)$ szerinti mellékosztályok, és az uniójuk egy $\mathcal{C}_B(G)$ szerinti mellékosztály. Ekkor alkalmas g_1, g_2 elemekre,

$$\mathcal{C}_B(\sigma H) = g_1\mathcal{C}_B(H), \quad \mathcal{C}_B(\sigma g H) = g_2\mathcal{C}_B(H)$$

$$\mathcal{C}_B(\sigma G) = g_1\langle \mathcal{C}_B(H), g_1^{-1}g_2\rangle$$

Nem maradt más hátra, mint összegezni az algoritmust.

5.1.8. Algoritmus. *Bemenetként tehát kapunk egy $B \subseteq A$ színezett részhalmazt, egy $G \leqslant \mathrm{Sym}(A)$, 2-részcsoportot, illetve egy $\sigma \in G$ elemet. Amennyiben B két G-stabil halmaz, $B = B_1 \cup B_2$ uniója, a megoldást rekurzívan a két részhalmazon keressük tovább, melynek a metszete lesz majd a kimenet. Amennyiben G hatása B-n tranzitív, négy rekurzív hívást hajtunk végre az orbit két G-blokkja, illetve az ezeket stabilizáló H részcsoport segítségével.*

Amennyiben B halmaznak már csak egy eleme van, felhasználjuk azt, hogy tudjuk, $B = \{b\}$ és $GB = B$, ebből azt kapjuk, hogy

- $\mathcal{C}_B(\sigma G) = \sigma G$, *amennyiben* $\sigma(b) \sim b$
- $\mathcal{C}_B(\sigma G) = \varnothing$, *amennyiben* $\sigma(b) \nsim b$

5.2. Korlátos fokszámú eset. Az alfejezetben a részletek mellőzésével azokat a megfontolásokat vesszük sorra, amelyek ahhoz szükségesek, hogy a fenti gondolatmenetet korlátos t fokszámú gráfokra is elmondhassuk. Első lépésként ebben az esetben is a gráfokat vizsgáljuk.

A trivalens esethez hasonlóan, itt is egy kiemelt e élt fixen hagyó automorfizmusokat fogjuk tekinteni. Hasonlóan definiáljuk az X_r gráfsorozatot is, tehát ebben az esetben is a

$$\pi_r : \mathrm{Aut}_e(X_{r+1}) \to \mathrm{Aut}_e(X_r)$$

homomorfizmusok magját, illetve képét keressük.

Az A halmazunk, amelyben korábban az egy-, két-, illetve háromelemű részhalmazait gyűjtöttük $V(X_r)$-nek, most $V(X_r)$ összes, legfeljebb t elemű részhalmazát magába foglalja. Ezután hasonlóan definiáljuk az

$$f_r : V(X_{r+1})\backslash V(X_r) \to A_r$$

szomszédsági függvényt. Ekkor a K_r mag izomorf a

$$\bigtimes_{a\in A_r} \mathrm{Sym}(f_r^{-1}(a))$$

csoporttal, mivel a trivalens eset alapján, most is a "testvérek" közötti cserélgetések hagyják fixen $V(X_r)$ minden pontját.

Az $A_{r,s}$ halmazokat hasonlóan definiáljuk, mint a trivalens esetben.

$$A_{r,s} = \{a \in A_r \mid |f_r^{-1}| = s,\ 1 \leqslant s \leqslant t-1\}$$

, illetve

$$A_{r,0} = A_r \backslash (\bigcup_{1 \leqslant i \leqslant t-1} A_{r,i})$$

A $V(X_r)$ pontok között futó új éleket ismét A'_r-rel jelöljük. Ez ismét meghatároz $2t$ darab halmazt, az $A_{r,i} \cap A'_r$ és az $A_{r,i} \backslash A'_r$ halmazokat.

Az észrevételt, miszerint $t = 3$ esetben az automorfizmus csoportok 2-csoportok voltak, már nem tudjuk alkalmazni, helyette más tulajdonságot szeretnénk kihasználni.

5.2.1. Definíció. *Minden $i \geqslant 2$-re, legyen Γ_i olyan G csoportok osztálya, mely kompozíció faktorai mind részcsoportjai S_i-nek.*

A következő három lemma segítségével belátható, hogy a π_r homomorfizmusok magjai mind elemei Γ_{t-1}-nek.

5.2.2. Lemma. *Legyen $N \lhd G$, ekkor G akkor és csak akkor eleme Γ_k-nak, ha N és G/N is az.*

5.2.3. Lemma. *S_k részcsoportjai mind elemei Γ_k-nak.*

5.2.4. Lemma. *Amennyiben $G \in \Gamma_k$, akkor G tetszőleges részcsoportja is eleme Γ_k-nak.*

A kiinduló állításunkat a $Sym(f_r^{-1}(a)) = S_j$ valamely $j \leqslant t-1$-re észrevételből kapjuk. Ebből indukcióval következik, hogy $\mathrm{Aut}_e(X_r) \in \Gamma_{t-1}$.

Vizsgáljuk tehát a következő problémát: Γ_k-beli csoportok szín automorfizmus algoritmusa. A színezett halmazok esetén ugyanúgy jelöljük a " $\sim$ " relációt, valamint a $\mathcal{C}_B(\sigma G)$ halmazt, mint a trivalens esetben.

Így a probléma nem lesz más mint,

5.2.5. Probléma. *Adott bemenetként egy A színezett halmaz, egy $G \leqslant \mathrm{Sym}(A)$ permutációcsoport egy S generátorrendszerével megadva, egy G-invariáns $B \subseteq A$ részhalmaz és egy $\sigma \in \mathrm{Sym}(A)$ permutáció. Feltéve, hogy $G \in \Gamma_k$, keressünk polinomiális futásidejű algoritmust $\mathcal{C}_B(\sigma G)$ meghatározására. A 5.1.6-os Lemmát is figyelembe véve kimenetként $\mathcal{C}_B(G)$ egy generátorrendszerét és egy $\sigma_0 \in \mathcal{C}_B(\sigma G)$ elemet várunk, vagy az üres halmazt.*

Ismét két esetet kell megkülönböztetni. Az elsőben B diszjunkt uniója két G-stabil részhalmaznak, B_1 és B_2-nek, ekkor

$$\mathcal{C}_B(\sigma G) = \mathcal{C}_{B_2}\mathcal{C}_{B_1}(\sigma G)$$

Ellenkező esetben B egy minimális G-blokk rendszerre bontható.

$$B = B_1 \cup B_2 \cup .. \cup B_m$$

Sajnos egy ilyen felbontást nem tudunk könnyen meghatározni. Helyette inkább egy olyan P részcsoportot keressünk, melyre $|G : P| \leqslant m^c$ teljesül, illetve P hatása B-n, egy p-részcsoportját indukálja $\mathrm{Sym}(B)$-nek.

Ezt a következő állítások teszik lehetővé.

5.2.6. Állítás. *Minden j pozitív egészhez létezik olyan $c = c(j)$ csak j-től függő konstans, hogy ha G egy primitív részcsoportja S_n-nek és eleme Γ_j-nek, akkor valamilyen p prímre, G-nek létezik olyan Sylow p-részcsoportja, amelynek indexe kisebb, mint n^c.*

Felmerül a kérdés, hogyan is találhatunk ilyen részcsoportot tetszőleges csoportban, de szerencsére ez megoldható polinom idejű algoritmus segítségével.

5.2.7. Állítás. *Tetszőleges fix c-re létezik olyan polinomidejű algoritmus, amely $G \leqslant S_n$ csoportnak meghatározza egy Sylow p-részcsoportját, amennyiben G-re teljesül az, hogy $|G| = p^s m$, ahol $m \leqslant n^c$.*

A 5.2.6 Állítás segítségével belátható az is, hogy,

5.2.8. Állítás. *Legyen $G \leqslant Sym(A)$ és legyen $G \in \Gamma_k$. Ekkor valamilyen B, G-orbit esetén polinomális időben találhatunk egy minimális G-blokkrendszert B-ben, illetve egy olyan P részcsoportját G-nek, amely B-n p-csoportként hat és az indexe G-ben legfeljebb $m^{c(k)}$.*

Felírva G-t a P szerinti bal oldali mellékosztályainak uniójaként és alkalmazva a $\mathcal{C}_B(K)$ korábban ismertetett tulajdonságait, azt kapjuk, hogy,

$$G = \bigcup_i \tau_i P$$

$$\mathcal{C}_B(\sigma G) = \bigcup_i \mathcal{C}_B(\sigma \tau_i P).$$

Ezzel a $\mathcal{C}_B(\sigma G)$ mellékosztály meghatározásának feladatát, a $\mathcal{C}_B(\sigma \tau_i P)$ mellékosztályok meghatározására redukáltuk. A $|G : P|$-re vonatkozó korlát miatt, ha találunk polinomiális futásidejű algoritmust, akkor az eredeti problémára is polinomiális idejű eljárást kapunk, minden i-re futtatva. Az algoritmust kétféleképpen bonthatjuk kisebb részekre. Legyen $\mathcal{B} = \{B_1, B_2, ..., B_m\}$, az 5.2.8-as Állításnak megfelelően megtalált minimális G-blokkrendszer B-ben. Tudjuk, hogy P egy p-csoportként hat $\mathcal{B}$-n. A továbbiakban P hatását nem B-n, hanem $\mathcal{B}$-n vizsgáljuk. Tehát amennyiben $\mathcal{B}$ diszjunkt uniója a P-stabil $\mathcal{B}_1$ és $\mathcal{B}_2$ halmazoknak, akkor ezeken dolgozunk tovább egyenként. Ellenkező esetben megkeressük $\mathcal{B}$-nek egy minimális P-blokk rendszerét, amelynek pontosan p blokkja lesz, továbbá ezen blokkok P-beli stabilizátorainak indexe P-ben is p lesz. Így az eredeti probléma legrosszabb esetben is csak p^2 problémára esik szét. Ilyen módon haladva tovább egyre csökkenő

méretű részhalmazokon, előbb-utóbb eljutunk odáig, hogy valamelyik eredeti B_i-hez érkezünk. Erről azonban tudjuk, hogy $|B_i| = |B|/m$. Tehát az eredeti problémát legfeljebb m^2 számú problémára bontjuk, melyeknek mérete $|B|/m$, tehát az eredeti probléma legrosszabb esetben, $m^c \cdot m^2$ részfeladatra bontható. Minden ilyen bontás n-ben polinomiális, tehát az egész algoritmus is az marad.

Végül felírjuk a két algoritmust formálisan is

5.2.9. Algoritmus. $\mathcal{C}_B(\sigma G)$ *kiszámítása*

Input: $G \subseteq Sym(A)$ *egy generátorrendszere,* A *színezett halmaz,* $\sigma \in Sym(A)$ *és egy* G*-stabil* B *részhalmaz.*

Output: Egy $\sigma_0 \in \mathcal{C}_B(\sigma G)$ *és* $\mathcal{C}_B(G)$ *egy generátorrendszere. (A továbbiakban az egyszerűség kedvéért ehelyett* $\mathcal{C}_B(\sigma G)$*-t írunk.)*

- *Ha* $B = \{b\}$

$$\begin{cases} \mathcal{C}_B(\sigma G) = \varnothing & ha\ \sigma(b) \not\sim b \\ \mathcal{C}_B(\sigma G) = \sigma G & ha\ \sigma(b) \sim b \end{cases}$$

- *Ha* B *két* G*-stabil halmaz diszjunkt uniója*

$$Legyen\ \mathcal{C}_B(\sigma G) = \mathcal{C}_{B_2}\mathcal{C}_{B_1}(\sigma G)$$

- *Ha* G *tranzitív* B*-n és* $|B| > 1$*, akkor az* 5.2.8*-as Állításnak megfelelően keresünk egy* $\mathcal{B} = \{B_1, B_2, ..., B_m\}$ *minimális* G*-blokkrendszert, és egy* $P \leqslant G$ *részcsoport egy generátorrendszerét, amelyre teljesül, hogy* P *által* $\mathcal{B}$*-n indukált permutációcsoport egy* p*-csoport, illetve teljesül az is, hogy* $|G : P| \leqslant m^{c(k)}$*. Keresünk továbbá a* P *szerint vett* τ_i *mellékosztály reprezentánsok egy rendszerét* G*-ben. Ekkor,*

$$\mathcal{C}(\sigma G) = \bigcup_i \mathcal{C}_{\mathcal{B}}(\sigma \tau_i P)$$

5.2.10. Algoritmus. $\mathcal{C}_{\mathcal{B}}(\sigma P)$ *kiszámítása*

Input: $P \subseteq Sym(A)$ *egy generátorrendszere,* A *színezett halmaz,* $\sigma \in Sym(A)$ *és egy* P*-stabil* $\mathcal{B}$ *halmazrendszere* A*-nak, amelyre* P p*-csoportként hat. Output:* $\mathcal{C}_{\mathcal{B}}(\sigma P)$*.*

- *Ha* $\mathcal{B} = \{B_0\}$*, legyen* $\mathcal{C}_{\mathcal{B}}(\sigma P) = \mathcal{C}_{B_0}(\sigma P)$
- *Ha a* $\mathcal{B}$ *diszjunkt uniója a* P*-stabil* $\mathcal{B}_1$ *és* $\mathcal{B}_2$ *halmazrendszereknek,*

$$\mathcal{C}_{\mathcal{B}}(\sigma P) = \mathcal{C}_{\mathcal{B}_2}\mathcal{C}_{\mathcal{B}_1}(\sigma P)$$

- *Ha* P *tranzitív* $\mathcal{B}$*-n és* $|\mathcal{B}| > 1$*, keressünk egy minimális* P*-blokkrendszert* $\mathcal{B}$*-ben*

$$\mathcal{B} = \mathcal{B}_1 \cup ... \cup \mathcal{B}_p$$

Megkeressük a $\mathcal{B}_i$ blokkok közös P' stabilizátorát. Szedjük szét P-t,

$$P = \bigcup_{i=1}^{p} \tau_i P'.$$

Ekkor

$$\mathcal{C}_{\mathcal{B}}(\sigma P) = \bigcup_{i=1}^{p} \mathcal{C}_{\mathcal{B}}(\sigma \tau_i P').$$

Az algoritmusok valójában együttesen alkotnak egy rekurzív eljárást $\mathcal{C}_B(\sigma G)$ kiszámítására. A futás az 5.2.9-es Algoritmussal indul, amely egy esetben hívhatja önmagát kisebb B_1 és B_2 halmazokra sorban, vagy megállíthatja a rekurziót. Tranzitív hatás esetén azonban kiszámít egy $\mathcal{B}$ minimális B-beli G-blokkrendszert, illetve a $\mathcal{B}$-n, p-csoportként ható $P \leqslant G$ kis indexű részcsoportot, továbbá a P-hez tartozó $\{\tau_i\}$ mellékosztály reprezentáns rendszert. Ez több hívást eredményez a 5.2.10-es Algoritmusba, a $\sigma' := \sigma\tau_i$, $\mathcal{B}$, P bemenetekkel. Ez az algoritmus alapvetően önmagát hívja, vagy kisebb csoportokra, vagy kisebb halmazrendszerekre, azonban egyelemű $\mathcal{B}$ esetén ismét az 5.2.9-es Algoritmust hívjuk, B_0, σ, P paraméterekkel.

Hivatkozások

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

[2] Eugene M. Luks *Isomorphism of bounded valence can be tested in polynomial time* J. Comp. Syst. Sci., 25:42-65., 1982.

[3] C. C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra,11* pages 169-183, Pergamon Press, Oxford, 1970.

[4] C. C. Sims. Computation with permutation groups. In *Proc. Second Symposium on Symbolic and Algebraic Manipulation,* pages 23-28, ACM Press, New York. 1971.

[5] C. C. Sims. "Some Group-Theoretic Algorithms," Lecture Notes in Math. No. 697. Springer-Verlag, Berlin, pp, 108-124, 1978.

[6] L. Babai. Graph Isomorphism in Quasipolynomial Time, arXiv.org, arXiv:1512.03547 version 1 [v1]

[7] M. Atkinson. An algorithm for finding the blocks of a permutation group. *Math. Comp.,* 29:911-913. 1975.

[8] M. Atkinson,R. Hassan,M. Thorne. Group Theory on a micro-computer. In Atkinson, M. editor, *Computational Group Theory,* pages 275-280, Academic Press, London, 1984.

[9] M. Furst, J. Hopcroft, E. Luks. A polynomial-time algorithms for permutation group. *in* "21st IEEE Symp. on Foundations of Comp. Sci. (1980)," pp. 36.41.

[10] M. Hall, "The Theory of Groups," Macmillan Co., New York, 1959

[11] P.J. Cameron, R. Solomon, A. Turull. Chains of subgroups in symmetric groups. *J. Algebra*, 127:340-352. (1989)

[12] https://hu.wikipedia.org/wiki/Csoportelmélet#Történet

függelék A. Schreier-Sims implementáció

A függelékben egy egyszerű Schreier-Sims algoritmus található 3.7-es Python verzióban implementálva. A fejezet végén példakód is található.

```
# ######################################################################
# ############# Class definition for permutation groups ##############
# ######################################################################
class PermutationGroup:
    # constructor method, accepts list of permutations, sets it as generators
    # also sets the set to act on to the identity element
    def __init__(self, perm_list):
        if not isinstance(perm_list, list):
            raise ValueError("Input is not a list")
        if any(not isinstance(elem, Permutation) for elem in perm_list):
            raise ValueError("Input is not a list of permutations")
        self.generators = perm_list
        self.acting_on_set = list(range(1, len(perm_list[0])+1))
        self.subgroup_chain = SubgroupChain(self)

    # printing the class shall be shown as a list of lists (the generators)
    def __str__(self):
        return str([generator.symbols for generator in self.generators])

    # get identity element
    def get_identity(self):
        return Permutation(list(range(1, len(self.generators[0])+1)))

    # define the set to act on
    def acting_on(self, input_set):
        if len(input_set) != len(self.generators[1]):
            raise ValueError("Input is not with right length")
        if not isinstance(input_set, list):
            raise ValueError("Input has wrong class")
        self.acting_on_set = input_set

    # calculate orbit of element(s) in the list
    def get_orbit(self, input_list):
        i = 0
        orbit = input_list
        elements = [self.get_identity()]*len(input_list)
        while i + 1 <= len(input_list):
            for gen in self.generators:
                image = gen.get_image(orbit[i])
                if image not in orbit:
                    orbit.append(image)
                    elements.append(elements[i]*gen)
            i += 1
        return orbit, elements

    def is_element(self, permutation_in):
        if not self.subgroup_chain.structure_valid:
            self.get_base_and_strong_generating_set()
        permutation = permutation_in
        for i in range(len(self.subgroup_chain.base)):
            transversal_element_found = False
            for j in range(len(self.subgroup_chain.transversals[i])):
```

```
                if permutation.get_image(self.subgroup_chain.base[i]) == \
                        self.subgroup_chain.transversals[i][j]. \
                                get_image(self.subgroup_chain.base[i]):
                    permutation = permutation * \
                                  self.subgroup_chain.\
                                        transversals[i][j].inverse()
                    transversal_element_found = True
                    break
            if not transversal_element_found:
                return False, permutation
        if permutation == self.get_identity():
            return True, permutation
        else:
            return False, permutation

    def get_cardinality(self):
        if not self.subgroup_chain.structure_valid:
            self.get_base_and_strong_generating_set()
        cardinality = 1
        for tvers in self.subgroup_chain.transversals:
            cardinality *= len(tvers)
        return cardinality

    def get_base_and_strong_generating_set(self):
        def sift_schreier_generators(generators, transversals, base):
            def get_rep_elem(perm, transversal_list, base_pt):
                for transversal_elem in transversal_list:
                    if transversal_elem.get_image(base_pt) == \
                                            perm.get_image(base_pt):
                        return transversal_elem.inverse()
            subgroup = PermutationGroup(generators)
            subgroup.subgroup_chain.transversals = \
                                    transversals[1:len(transversals)]
            subgroup.subgroup_chain.base = base[1:len(base)]
            subgroup.subgroup_chain.structure_valid = True
            for gen in generators:
                for tvers in transversals[0]:
                    elem_f, siftee = subgroup.is_element(
                                     tvers*gen* \
                                     get_rep_elem(tvers*gen,
                                                  transversals[0],
                                                  base[0])
                                     )
                    if not elem_f:
                        return siftee
            return self.get_identity()
        # algorithm init
        self.subgroup_chain.reset(self)
        self.subgroup_chain.base.append(
                           (self.generators[0].get_cycles()).cycles[0][0])
        orbit, elements = self.get_orbit([self.subgroup_chain.base[0]])
        self.subgroup_chain.transversals.append(elements)
        i = 1
        while i > 0:
            sifted_element = sift_schreier_generators(
                             self.subgroup_chain.generator_chain[i-1],
                             self.subgroup_chain.
```

```
                                transversals[(i - 1):
                                        len(self.subgroup_chain.transversals)],
                            self.subgroup_chain.base[i-1:
                                        len(self.subgroup_chain.base)])
            if sifted_element == self.get_identity():
                i -= 1
            else:
                if len(self.subgroup_chain.generator_chain) == i:
                    self.subgroup_chain.generator_chain.append([sifted_element])
                else:
                    self.subgroup_chain.generator_chain[i].append(sifted_element)
                if len(self.subgroup_chain.base) == i:
                    cycles_notation_of_siftee = sifted_element.get_cycles()
                    new_base_found = False
                    for cycle in cycles_notation_of_siftee.cycles:
                        for elem_moved in cycle:
                            if elem_moved not in self.subgroup_chain.base:
                                self.subgroup_chain.base.append(elem_moved)
                                new_base_found = True
                                break
                        if new_base_found:
                            break
                    # recomputing transversals
                    temp_grp = PermutationGroup(self.subgroup_chain.generator_chain[i])
                    orb, elems = temp_grp.get_orbit([self.subgroup_chain.base[i]])
                    if len(self.subgroup_chain.transversals) == i:
                        self.subgroup_chain.transversals.append(elems)
                    else:
                        self.subgroup_chain.transversals[i] = elems

                i += 1
        sgs = []
        for subgroup_gen in self.subgroup_chain.generator_chain:
            for gen_elem in subgroup_gen:
                if gen_elem not in sgs:
                    sgs.append(gen_elem)
        self.subgroup_chain.strong_generating_set = sgs
        self.subgroup_chain.structure_valid = True
        return sgs

    def print_group_elements(self):
        elements = self.group_elements()
        print(str([element.symbols for element in elements]))

    # very naive implementation for getting elements, extremely slow
    # only left here for occasional debug purposes later
    def group_elements(self):
        elements = self.generators.copy()
        i = 0
        while i < len(elements):
            for j in range(len(self.generators)):
                if elements[i] * self.generators[j] not in elements:
                    elements.append(elements[i] * self.generators[j])
                if self.generators[j] * elements[i] not in elements:
                    elements.append(self.generators[j] * elements[i])
            i += 1
        return elements
```

```
# ######################################################################
# ################ Class definition for permutations ################
# ######################################################################
class Permutation:
    # constructor for the permutations
    # the input is required to be a list consisting
    # of the integers from 1 to the length of the list
    def __init__(self, perm_in):
        if isinstance(perm_in, CycleNotation):
            perm_in = self.cycles_to_map(perm_in)
        else:
            # casting data type to list
            if not isinstance(perm_in, list):
                perm_in = list(perm_in)
            # throw error if the content of the list is
            # different as it is expected
            for i in range(len(perm_in)):
                if i+1 not in perm_in:
                    raise ValueError("Wrong input")
        self.symbols = perm_in

    # printing the value shall be shown as a list
    def __str__(self):
        return str(self.symbols)

    # overloading len() operation
    def __len__(self):
        return len(self.symbols)

    # overloading "==" operator to test equality of permutations
    def __eq__(self, other):
        self._operator_input_check(other)
        return self.symbols == other.symbols

    # overloading != operator -> "not =="
    def __ne__(self, other):
        return not self == other

    # overloading the '*' operator to allow two
    # variables of permutation class to be
    # "multiplied"
    def __mul__(self, other):
        self._operator_input_check(other)
        return Permutation([other.symbols[i - 1] for i in self.symbols])

    # overloading the '**' operator to allow
    # permutations to be raised to the nth power
    def __pow__(self, pwr):
        # checking for data type
        if not isinstance(pwr, int):
            raise ValueError("Power is not an integer")
        # checking for data validity
        # power is -1 -> return inverse
        if pwr == -1:
            return self.inverse()
```

```
    # power is 0 return identity
    if pwr == 0:
        return Permutation(list(range(1, len(self)+1)))
    # power is 1 -> return self
    if pwr == 1:
        return self
    # power is higher -> multiply it accordingly

    temp_perm = self
    for i in range(pwr-1):
        temp_perm = temp_perm * self
    return temp_perm

# get order of permutation
def order(self):
    # least common multiple of a list
    def lcm(a):
        if len(a) == 0:
            return 1
        elif len(a) == 1:
            return a[0]
        elif len(a) == 2:
            n = int(a[0])
            m = int(a[1])
            if n > m:
                c = n
            else:
                c = m
            while c % n != 0 or c % m != 0:
                c += max(n, m)
            return c
        else:
            return lcm(a[0:-1])
    # calculate the length of the cycles
    cycles_len = [len(cycle) for cycle in self.get_cycles().cycles]
    # return the lcm of the cycle length
    return lcm(cycles_len)

# get inverse of permutation
def inverse(self):
    # temporary list
    temp_list = [0] * len(self)
    # assembling the inverse
    for i in range(len(temp_list)):
        temp_list[self.symbols[i] - 1] = i + 1
    return Permutation(temp_list)

# getting the length of the permutation
def length(self):
    return len(self.symbols)

# check input for operators
def _operator_input_check(self, other):
    # checking for data type
    if not isinstance(other, Permutation):
        raise ValueError("Data type mismatch")
    # checking for permutation length
```

```
        if other.length() != len(self):
            raise ValueError("symbols length mismatch")

    # get cyclic form of permutation
    def get_cycles(self):
        cycles = []
        for i in range(len(self)):
            if not any(self.symbols[i] in cycle for cycle in cycles) \
                    and self.symbols[i] != i+1:
                cycle = [i+1, self.symbols[i]]
                while self.symbols[cycle[-1]-1] != cycle[0]:
                    cycle.append(self.symbols[cycle[-1]-1])
                cycles.append(cycle)
        return CycleNotation(cycles, len(self))

    @ staticmethod
    def cycles_to_map(cycles_in):
        perm = list(range(1, cycles_in.domain_size + 1))
        for i in range(len(cycles_in.cycles)):
            for j in range(len(cycles_in.cycles[i])):
                if (j+1) < len(cycles_in.cycles[i]):
                    perm[cycles_in.cycles[i][j]-1] = cycles_in.cycles[i][j+1]
                else:
                    perm[cycles_in.cycles[i][j]-1] = cycles_in.cycles[i][0]
        return perm

    def get_image(self, point):
        return self.symbols[point-1]


# ######################################################################
# ################### Class definition for cycles ####################
# ######################################################################
class CycleNotation:
    def __init__(self, cycles_in, domain_size_in=0):
        def sort_length(var_in):
            return len(var_in)
        if not isinstance(cycles_in, list) and \
                any([not isinstance(cycle, list) for cycle in cycles_in]):
            raise ValueError("Input cycles are not list of lists")
        elif not isinstance(domain_size_in, int):
            raise ValueError("Input domain size is not an integer")
        elif domain_size_in < 0:
            raise ValueError("Input domain size is not positive")
        self.cycles = cycles_in
        self.cycles.sort(key=sort_length, reverse=True)
        if domain_size_in == 0:
            self.domain_size = max([max(cycle) for cycle in cycles_in])
        # elif domain_size_in < max([max(cycle) for cycle in cycles_in]):
        #     raise ValueError("Wrong input for permutation domain size")
        else:
            self.domain_size = domain_size_in

    def __str__(self):
            return str(self.cycles) + "\n" + str(self.domain_size)
```

```
# ######################################################################
# ################ Class definition for subgroup chain ################
# ######################################################################

class SubgroupChain:
    def __init__(self, permutation_group):
        if not isinstance(permutation_group, PermutationGroup):
            raise ValueError("Input is not of class PermutationGroup")
        self.generator_chain = [permutation_group.generators]
        self.transversals = []
        self.base = []
        self.strong_generating_set = []
        self.structure_valid = False

    def reset(self, permutation_group):
        self.__init__(permutation_group)


# ################################
# ### Example for Rubik's cube ###
# ################################

# intended as a separate file
from PermutationGroup import *

right = [[3, 12, 21, 30], [6, 15, 24, 33],
         [9, 18, 27, 36], [46, 52, 54, 48], [49, 53, 51, 47]]
left = [[1, 10, 19, 28], [4, 13, 22, 31],
        [7, 16, 25, 34], [39, 45, 43, 37], [42, 44, 40, 38]]
back = [[1, 46, 27, 37], [2, 47, 26, 38],
        [3, 48, 25, 39], [34, 36, 30, 28], [35, 33, 29, 31]]
front = [[7, 52, 21, 43], [8, 53, 20, 44],
         [9, 54, 19, 45], [10, 12, 18, 16], [11, 15, 17, 13]]
top = [[10, 52, 36, 39], [11, 49, 35, 42],
       [12, 46, 34, 45], [7,  9,  3,  1], [8,  6,  2,  4]]
bottom = [[16, 54, 30, 37], [17, 51, 29, 40],
          [18, 48, 28, 43], [19, 21, 27, 25], [20, 24, 26, 22]]

right = Permutation(CycleNotation(right, 54))
left = Permutation(CycleNotation(left, 54))
back = Permutation(CycleNotation(back, 54))
front = Permutation(CycleNotation(front, 54))
top = Permutation(CycleNotation(top, 54))
bottom = Permutation(CycleNotation(bottom, 54))

rubik = PermutationGroup([right, left, back, front, top, bottom])

print("Cardinality:")
print(rubik.get_cardinality())
print("Lenght of base:")
print(len(rubik.subgroup_chain.base))
print("Base of the Rubik cube's group")
print(rubik.subgroup_chain.base)


# ###############
# ### Output ###
```

```
# ###############
Cardinality:
43252003274489856000
Length of base:
18
A base of the Rubik cube's group
[3, 1, 4, 2, 8, 7, 24, 26, 9, 31, 15, 22, 13, 18, 16, 25, 6, 17]
```