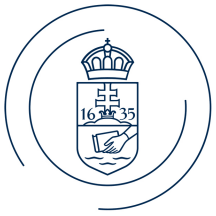
Eötvös Loránd Tudományegyetem
Természettudományi Kar

Gráfok izomorfizmus-problémája algebrai eszközökkel

BSc Szakdolgozat

Írta:
Dávid Balázs Márton
Matematika BSc

Témavezető:
Somlai Gábor
Egyetemi Adjunktus



ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Budapest, 2024

Tartalomjegyzék

1. Bevezető **5**

2. Gráfelmélet **6**
2.1. Gráfelméleti fogalmak 6
2.2. Gráf izomorfia 7
2.3. Gráf izomorfia bonyolultsága 8
2.4. Izomorfia-teljes 9
2.5. NP általánosítása 10

3. Cayley gráfok izomorfiája **11**
3.1. Csoportelmélet 11
3.2. Cayley gráfok 12
3.3. Cayley gráfok izomorfia problémája 13
3.4. Cayley izomorfia $K_{m,k}$ gráfokon 15

4. CI-gráfok **16**
4.1. Kritérium CI-gráfokra 18

5. Normál Cayley gráfok **20**

6. A.Ádám sejtése $n = p$ esetén **26**
6.1. Elégséges feltétel és a sejtés egy más formája 26
6.2. A sejtés bizonyítása 27

7. CI-ellenörző **29**
7.1. group.py felépítése és működése 29
7.1.1. __init__() 30
7.1.2. get_base_generator() 30
7.1.3. execute_inner_product() 31
7.1.4. generate_cayley_graph() 31
7.1.5. _is_homomorphism() 31
7.1.6. _make_automorphism_group() 31
7.1.7. _generate_potential_isomorph_graphs() 31
7.1.8. _make_isomorph_graphs() 32
7.1.9. check_ci_property() 32
7.1.10. nx.isomorphism.DiGraphMatcher() 33
7.2. products.py felépítése és működése 34

7.3. group_products.py felépítése és működése 34
7.4. ultis.py 35
7.5. CI_property_tester.py felépítése és működése 35
7.6. Példák a kód felhasználására 36
7.6.1. Véletlenszerű részhalmazra CI vizsgálat 36
7.6.2. Egy csoport összes Cayley gráfjának a CI vizsgálata . . 36
7.6.3. Gráfszorzatra a CI vizsgálata 37

8. Függelék **40**
8.1. group.py 40
8.2. products.py 43
8.3. group_products.py 44
8.4. utils.py 46
8.5. CI_property_tester.py 46

Köszönetnyilvánítás

Szeretném megköszönni konzulensemnek, Somlai Gábornak a sok munkáját, hogy egy számomra eddig ismeretlen témakörbe beletekintést nyerhettem. Illetve hálás vagyok, hogy bármikor rendelkezésre állt, és választ adott a felmerülő kérdéseimre.

1. Bevezető

Az életben könnyen felmerülhet a kérdés, hogy két dolog hasonló-e. Így a gráfelméletben is adódik, hogy vizsgáljuk ezt a problémát.

Ahhoz, hogy eldöntsük két gráfról, hogy izomorfak, vagy kell mutatnunk ellenpéldát, hogy miért nem lehetnek izomorfak, vagy kell mutatnunk egy leképezést, ami az egyik gráfot a másikba viszi. Az utóbbi ellenőrzése egy könnyű feladat, így a probléma NP-beli. Viszont arról, hogy P-beli vagy NP-teljes lenne, csak sejtéseink vannak. A bonyolultságának a vizsgálatát viszont megtehetjük több módon is. Például feltehetjük, hogy a saját bonyolultsági osztálya, vagy vizsgálhatjuk $IP(2)$-be tartozását.

A gráfizomorfia problémát vizsgálhatjuk algebrai eszközökkel is. Az utóbbira egy hasznos eszköz a Cayley gráf fogalma, ami egy csoportból és részhalmazából képez gráfot.

A szakdolgozatom célja a Cayley gráfok tulajdonságainak, és rajtuk keresztül a gráf izomorfia vizsgálata. Így egy gráfelméleti betekintés után, belátjuk pár közvetlen következményét a Cayley gráf definíciójának. Majd rátérünk két Cayley gráf izomorfiájának a problémájára, illetve a Cayley reprezentáció fogalmára, amire példát is adunk. Megvizsgáljuk a Cayley gráfok egy csoportosítását, és az automorfizmus csoporton keresztül bevezetjük az $Aut(G, S)$ csoportot. Kimondunk egy fontos tételt, ami a CI-gráf eldöntési problémáját megegyezteti egy konjugálási tulajdonsággal. Bevezetjük a normál Cayley gráf fogalmát és két példán keresztül vizsgáljuk a normális részcsoport fogalmát. Megnézzük az egyik első CI csoportos eredmények egyikét, végül pedig bemutatunk egy CI ellenörző programot.

2. Gráfelmélet

A Cayley gráfok témaköre a csoportelmélet és a gráfelmélet együttes eredménye, így az alábbi fejezetben először a gráfelmélet ehhez szükséges alapvető fogalmait fogjuk bevezetni. A fejezet további részében kitérünk a gráfizomorfia témakörére, majd vizsgáljuk a gráfizomorfia nehézségét.

2.1. Gráfelméleti fogalmak

A következőkben felsorolunk pár alapvető definíciót, amiket később használni fogunk.

2.1.1. Definíció. *Legyen E és V halmazok, ahol V tartalmazza a csúcsokat. Továbbá értelmezzünk egy $\varphi : E \to \{(v, v') | v, v' \in V\}$ leképezést. Ekkor a $G = (\varphi, E, V)$ hármast irányítatlan gráfnak nevezzük. Ha különbséget teszünk (v, v') és (v', v) között, akkor irányított gráfról beszélünk. Ha legfeljebb egy él fut v és v' között, akkor egyszerű gráfról beszélünk*

2.1.2. Definíció. *Adott egy $G = (\varphi, E, V)$ irányítatlan egyszerű gráf, ha minden $v \neq v'$-re teljesül, hogy létezik pontosan egy (v, v') él, akkor teljes gráfról beszélünk.*

2.1.3. Definíció. *Adott egy $G = (\varphi, E, V)$ gráf és egy $v \in V$ csúcsa. Ha egy él (v, v), tehát v-ből v-be fut, akkor azt az élet hurok élnek nevezzük.*

2.1.4. Definíció. *Adott egy $G = (\varphi, E, V)$ gráf, egy $v_0, e_1, v_1, \ldots, v_n$ sorozatot sétának nevezünk, ha minden $0 \leq i \leq n$-re teljesül, hogy $\varphi(e_i) = (v_{i-1}, v_i)$.*

2.1.5. Definíció. *Egy G gráf összefüggő, ha minden csúcsából minden csúcsába létezik séta.*

2.1.6. Definíció. *Adott egy $G = (\varphi, E, V)$ irányított gráf és egy $v \in V$ csúcsa. Ekkor a v csúcsból induló (v, v') élek számát kifoknak nevezzük, még a csúcsba futó (v', v) élek számát pedig befoknak. Irányítatlan gráf esetében a két érték megegyezik.*

2.1.7. Definíció. *Egy $G = (V, E)$ gráfot, k-részes gráfnak nevezünk, ha a V csúcsok halmaza feloszható k partícióra, hogy minden $v, v' \in K_i$-re teljesül, hogy $(v, v') \notin E$. Tehát semelyik partíción belül nem fut él két csúcs között. Ha minden más csúcs között fut pontosan egy él, akkor azt teljes k-részes gráfnak hívjuk.*

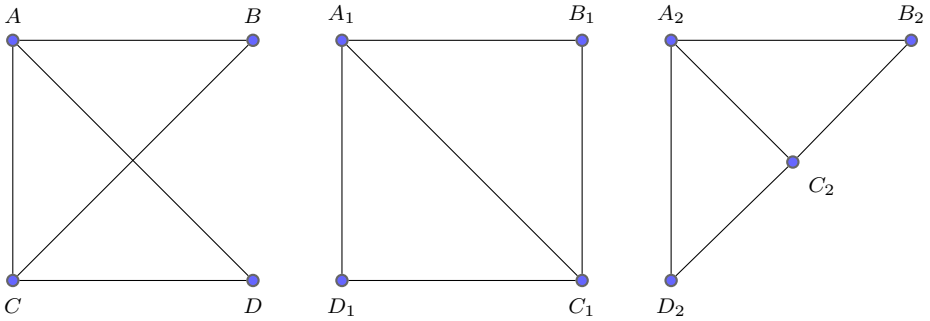
2.2. Gráf izomorfia

Az alábbi alfejezben beletekintünk a gráfok izomorfia problémájába, de ehhez először szükségünk lesz a gráf izomorfia definíciójára:

2.2.1. Definíció. *Legyen $G = (\varphi, E, V)$ és $G' = (\varphi', E', V')$ gráfok. A két gráf izomorf, ha $\exists f : V \to V'$ bijekció, hogy $\forall a, b \in V$, $(a, b) \in E$ pontosan akkor, ha $(f(a), f(b)) \in E'$.*

Elsőre egyszerűnek tűnhet a probléma, hiszen "kis" gráfokra könnyen meg tudjuk határoznia bijekciót. Példának vegyük az alábbi gráfokat:

*P*élda:



Sorban G, G_1 és G_2 gráfok.

Itt akár az összes lehetőség végigpróbálására is van lehetőségünk. Így találhatunk egy $g : V \to V_1$ bijekciót, ami az első gráf V csúcsait a második gráf V_1 csúcsaira képzi. A próbálgatás alatt szintén feltűnhet, hogy nem csak 1 bijekciót találhatunk. A második gráfban a B_1 és D_1, illetve az A_1 és C_1 csúcsokat felcserélve más-más bijekciókat kapunk. Tehát két gráf között több bijekció is lehet.

Nagy gráfok esetén, ha kevés az izomorfia két gráf között, nem feltétlenül hagyatkozhatunk a próbálgatásra, így felmerül a kérdés, hogy van-e valamilyen "gyors" algoritmus a gráf izomorfia problémára. A válasz az, hogy általános esetben azt se tudjuk a problémáról, hogy P-beli vagy NP-teljes lenne-e, csak annyit hogy NP-beli. Tehát az általános eset helyett, valamilyen megszorítások mellett fogjuk vizsgálni az algoritmusokat.

2.3. Gráf izomorfia bonyolultsága

Ahogy korábban is említettük, a gráf izomorfia probléma P-beliségéről, illetve NP-teljességéről nem tudunk. A témakörbe való elmélyüléshez szükségünk lesz P és NP definícióra, amiket a Turing-gépből vezetünk le.

P definiálása Turing-gépen keresztül

2.3.1. Definíció. *Turing-gépnek nevezzük a* $T = (k, \Sigma, \Gamma, \alpha, \beta, \gamma)$ *hatost, ahol:*

- $k \in \mathbb{N}$*, a szalagok száma*
- $3 \leq |\Sigma| < \infty, \star \in \Sigma$*, egy ábécé*
- $|\Gamma| < \infty, START, STOP \in \Gamma$*, az állapothalmaz*
- $\alpha : \Sigma^k \times \Gamma \to \Gamma$*, az új állapot átmenetfüggvénye*
- $\beta : \Sigma^k \times \Gamma \to \Sigma^k$*, az írás átmenetfüggvénye*
- $\gamma : \Sigma^k \times \Gamma \to \{-1, 0, \}^k$*, a fej mozgás átmenetfüggvénye.*

2.3.2. Definíció. *L-et egy* Σ *feletti nyelvnek nevezzük, ha* $L \subseteq \Sigma_0^*$*, ahol* Σ^* *a* Σ *betűiből képzett véges sorozatok halmaza és* $\Sigma_0 = \Sigma \setminus \{\star\}$*.*

2.3.3. Definíció. $DTIME(f(n))$ *jelöli azon L nyelvek osztályát, ahol minden L-hez létezik T Turing-gép mely felismeri L-et, és a* $w \in L$ *feladatot* $f(|w|)$ *lépésben megválaszolja.*

2.3.4. Definíció. $P = \bigcup_{i=1}^{\infty} DTIME(n^i)$

Tehát a P-be egy polinomiális idejű algoritmussal (Turing-géppel) kiszámolható problémák tartoznak.

NP definiálása Turing-gépen keresztül

2.3.5. Definíció. *Nemdeterminisztikus Turing-gépnek nevezzük a* $T = (k, \Sigma, \Gamma, \Phi)$ *négyest, ahol:*

- $k \in \mathbb{N}$*, a szalagok száma*
- $3 \leq |\Sigma| < \infty, \star \in \Sigma$*, egy ábécé*
- $|\Gamma| < \infty, START, STOP \in \Gamma$*, az állapothalmaz*

- $\Phi \subset (\Sigma^k \times \Gamma) \times (\Sigma^k \times \Gamma \times \{-1, 0, 1\}^k)$, *az átmenetreláció.*

2.3.6. Definíció. *$NTIME(f(n))$ jelöli azon L nyelvek osztályát, melyekhez létezik egy T Nemdeterminisztikus Turing-gép, amely L-et felismeri $f(|w|)$ lépésben, minden mást elutasít.*

2.3.7. Definíció. $NP = \bigcup_{i=1}^{\infty} NTIME(n^i)$

Tehát NP-be a polinomiális időben leellenőrizhető problémák tartoznak.

A matematika egy nagy kérdése, hogy $P = NP$ igaz-e, de ebbe nem fogunk belemélyedni itt. A gráfizomorfia probléma bonyolultságának a P-be tartozása, illetve NP-teljességének vizsgálata helyett két másik főbb irányzatot vizsgálunk meg. [6]

2.4. Izomorfia-teljes

Az első ötlet a gráfizomorfia bonyolultságának a vizsgálatára, hogy a saját bonyolultási osztályaként kezeljük. Az ötlet legnagyobb előnye, hogy így tudjuk vizsgálni, mely más feladatok tartoznak a bonyolultsági osztályába.

Az ötlet megszületése óta több problémáról is sikerült belátni, hogy polionimálisan egyenlőek a gráf izomorfia problémával. Egy ehhez kapcsolódó érdekes eredmény Mathon [12] nevéhez kötődik, aki bemutatta, hogy két gráf közötti izomorfiák száma Izomorfia-teljes. Az eredmény érdekessége onnan adódik, hogy az eddigi megfigyelések alapján az NP-teljes problémák megszámlálási feladata sokkal bonyolultabbnak tűnik, mint az eldöntési feladatuk.

Szintén Mathon nevéhez kapcsolódik egy gráf automorfizmusról belátott eredmény is, még pedig hogy a nem triviális gráf automorfizmusok megszámlálási feladata egyenértékű a gráfizomorfia feladattal.

Sok más eredmény is született a témakörben, de az alábbiakban már csak egyet emelünk ki. A probléma különlegessége, hogy az alábbi nem egy gráfelméleti probléma.

Vegyünk két term-et, melyeknek lehetnek asszociatív és/vagy kommutatív függvényei, illetve lehetnek kommutatív változó-kötő operátorjai. A két term ekvivalenciájára vonatkozó eldöntési problémáról Basin [2] belátta, hogy Izomorfia-teljes.

2.5. NP általánosítása

A második ötlet, hogy az eddig használt NP deffiníció helyett, vagyis hogy polinomiális időben ellenőrizhető problémák osztálya, egy új megközelítést használjunk.

Tegyük fel, hogy van egy kellően erős gépünk, ami meg tudja oldani a feladatunkat. Ekkor ez a gép ad egy bizonyítékot is, amit leellenőrzünk polinomiális időben egy másik géppel.

Az utóbbi gondolatmenet ismerős lehet, hiszen itt az ellenörző gépünk az NP osztályra utal. Ha pedig nem szükséges kommunikáció a két gép között, akkor pedig a P osztály juthat eszünkbe.

Eddig csak a 0 és 1 darab kommunikációról beszéltünk, de jogosan felmerül a kérdés hogy mi történik ha többet is megengedünk? Így eljutunk egy új IP osztályhoz [7], ami Interaktív Polinomiális időt jelent, arra utalva hogy több kommunikáció is megtörténhet a gépek között, egészen pontosan polinomiális számút engedünk meg. Fontos megjegyezni, hogy csak bizonyos valószínűség mellett várjuk el a jó választ. IP(k)-val jelöljük azon eseteket, ahol megszorítjuk a kommunikációk számát k-ra. Egy ehhez tartozó eredmény Goldwasser, Micali és Rackoff [8] nevéhez kötődik, hogy a gráf izomorfia probléma IP(2)-beli. Ezzel is elősegítve, hogy jobban el tudjuk helyezni a gráf izomorfia nehézségét.

3. Cayley gráfok izomorfiája

Ahogy korábban is láthattuk, a gráf izomorfia egy sokkal nehezz probléma, mint ahogy az elsőnek tűnhet. Most áttérünk a probléma vizsgálatára algebrai eszközökkel a Li[10] áttekintése alapján. Fontos megjegyezni, hogy véges Cayley gráfokról fogunk beszélni.

3.1. Csoportelmélet

A Cayley gráfok témaköre több alapvető csoportelméleti fogalomra és a definíciójukban szereplő tulajdonságaikra is építkezik, így az átláthatóság kedvéért felsorolunk párat. [4]

3.1.1. Definíció. *Egy G nem üres halmaz csoport, ha értelmezett rajta egy $\star$ bináris művelet, a következő három tulajdonsággal:*

- *Tetszőleges $a, b, c \in G$ elemkre teljesül, hogy $(a \star b) \star c = a \star (b \star c)$, tehát asszociatív.*
- *Létezik $n \in G$ elem, amire tetszőleges $a \in G$ elemre teljesül, hogy $n \star a = a \star n = a$, tehát G tartalmaz neutrális elemet. Szokás identitás elemnek is hívni és id-vel jelölni.*
- *Tetszőleges $a \in G$ elemre és $n \in G$ neutrális elemre teljesül, hogy létezik $a^{-1} \in G$ inverz elem, melyre $a^{-1} \star a = a \star a^{-1} = n$.*

3.1.2. Definíció. *Legyen G csoport. Ha S részhalmaza G-nek és csoport a G-beli műveletekre nézve, akkor $S \subset G$-et G részcsoportjának hívjuk. Ha nem várjuk el, hogy csoport legyen, akkor részhalmaznak nevezzük.*

3.1.3. Definíció. *Legyen $(G, \star_G)$ és $(T, \star_T)$ két csoport, $\varphi : G \to T$ leképezés és $g_1, g_2 \in G$ tetszőleges elemek, ekkor ha teljesül, hogy*

$$\varphi(g_1 \star_G g_2) = \varphi(g_1) \star_T \varphi(g_2)$$

tehát művelettartó, akkor homomorfizmusról beszélünk.

3.1.4. Definíció. *Legyen G egy csoport, ami hat egy S halmazon. Ekkor $s \in S$ elem stabilizátorának hívjuk az alábbi halmazt:*

$$G_s = \{g \in G | g \cdot s = s\},$$

tehát G azon elemeinek a halmaza, amelyek helyben hagyják s-t.

3.1.5. Definíció. *Egy G permutáció csoportot S halmazon regulárisnak hívunk, ha minden $s_1, s_2 \in S$ elemre létezik $g \in G$ elem, hogy $g \cdot s_1 = s_2$, tehát tranzitív, és pontosan egy ilyen elem létezik.*

3.1.6. Definíció. *Legyen G és T csoportok, és legyen φ egy homomorfizmus T-ből $Aut(G)$-be. Ekkor a $G \rtimes T$ szorzatot szemi-direkt szorzatnak hívjuk, ahol az új csoport elemei $g \in G$ és $t \in T$ csoportok elemeinek rendezett (g, t) párjai és a csoportművelete:*

$$(g_1, t_1)(g_2, t_2) = (g_1\varphi_{t_1}(g_1), t_1t_2), \text{ ahol } g_1, g_2 \in G \text{ és } t_1, t_2 \in T.$$

3.2. Cayley gráfok

A már hamarabb megismert izomorfia problémát vizsgálnánk Cayley gráfok esetén, de ehhez először szükségünk lesz a Cayley gráf deffiníciójára.

3.2.1. Definíció. *Adott egy G csoport és ennek egy $S \subset G$ részhalmaza. Ekkor az ehhez tartozó irányított Cayley gráf a $Cay(G, S)$, ahol a csúcsok a G elemei, és v-ből pontosan akkor fut él w-be, ha $wv^{-1} \in S$.*

A definícióból adódik pár könnyen belátható tulajdonság:

3.2.2. Következmény. *Egy $Cay(G, S)$ gráf minden csúcsának a kifoka $|S|$.*

3.2.3. Állítás. *Pontosan akkor összefüggő egy $Cay(G, S)$ gráf, ha az S részhalmaz elemei generálják G-t.*

Bizonyítás: Mivel S elemei generálják G-t, G minden eleme felírható S elemeinek egy kombinációjaként, így ezeket felhasználva tudunk mutatni sétát és így összefüggő a gráf. A másik irányhoz, mivel minden (v, w) élre $(v, w) \in S$ a Cayley gráf definíciója alapján, a séta mentén kapunk S elemeinek egy kombinációját, így definíció alapján S elemei generálják G-t. □

3.2.4. Állítás. *Egy S részhalmaz pontosan akkor tartalmazza G neutrális elemét, ha $Cay(S, G)$ minden csúcsához tartozik hurok él.*

Bizonyítás: Ha minden csúcshoz tartozik hurok él, akkor minden $g \in G$-re $gg^{-1} \in S$, ami a neutrális elem definíciója. Ha S tartalmazza G neutrális elemét, akkor minden $g \in G$-re fog futni $(g, gn) = (g, g)$ él, így pedig hurok élt kapunk minden csúcsra. □

3.2.5. Állítás. *Ha minden $s \in S$ elemre $s^{-1} \in S$, akkor a $Cay(G, S)$ gráf tekinthető irányítatlannak is.*

Bizonyítás: Egy tetszőleges $g \in G$-re legyen w^{-1} olyan, hogy $gw^{-1} \in S$, mivel az inverze is benne van $(gw^{-1})^{-1} \in S$. Ekkor

$$(gw^{-1})^{-1} = (w^{-1}g)^{-1} = (w^{-1})^{-1}g^{-1} = wg^{-1}.$$

Így van (g, w) és (w, g) irányított él, amit összevonhatunk egy irányítatlan élbe. □

3.3. Cayley gráfok izomorfia problémája

Most hogy definiáltuk a Cayley gráfokat, adódik a kérdés, mikor izomorf két Cayley gráf. Az alábbi részben ezzel a problémával fogunk foglalkozni.

3.3.1. Definíció. *Legyen $Cay(G, S)$ és $Cay(G', S')$ két Cayley gráf. Ekkor a $Cay(G, S) \stackrel{?}{\cong} Cay(G', S')$ problémát a Cayley gráf izomorfia problémájának hívjuk.*

3.3.2. Definíció. *Legyen $\Gamma = (V, E)$ egy irányított gráf, ahol V a csúcsok, az E pedig az élek halmaza. Ekkor Γ egy Cayley gráf, ha létezik G csoport és $S \subset G$ részhalmaza, hogy $\Gamma \cong Cay(G, S)$. Ahol V elemeit megfeleltetjük G elemeivel, és minden $e \in E$ élre, ahol $e = (g_1, g_2)$ és $g_1, g_2 \in G$, teljesül hogy $g_2 g_1^{-1} \in S$. Ekkor a (G, S) párost a Γ gráf Cayley reprezentációjának hívjuk.*

Az utóbbinál felmerülhet a kérdés, hogy egyértelmű-e egy Cayley reprezentáció. Ehhez vizsgáljunk meg egy példát.:

*P*élda: Legyen $\Gamma = K_n$ az n csúcsú irányítatlan teljes gráf. Ekkor bármilyen n-ed rangú G csoportra és $S = G \setminus \{id\}$ részhalmazára teljesül, hogy Cayley reprezentációja Γ-nak, ahol id a neutrális elem.

A példából láthatjuk, hogy több Cayley reprezentációja is lehet egy adott Γ gráfnak. Egy adott Γ-ra az összes Cayley reprezentáció meghatározásának a problémáját Cayley reprezentációs problémának hívjuk.

Mivel egy Cayley reprezentáció nem egyértelmű felmerülhet a kérdés, hogy mikor tekintünk két Cayley reprezentációt ekvivalensnek?

3.3.3. Definíció. *A $\Gamma = (V, E)$ gráfnak a (G, S) és (G', S') Cayley reprezentációit ekvivalensnek tekintjük, ha létezik $\sigma \in Sym(V)$ permutáció, ami izomorfizmust indukál G és G' között, illete $S^\sigma = S'$. Ahol a $Sym(V)$ a V önmagára vett összes bijekciójának a halmaza a függvénykompozícióval ellátva.*

A definíció következménye, hogyha (G, S) és (G', S') ekvivalens Cayley reprezentációk, akkor G és G' izomorf csoportok.

3.3.4. Definíció. *Legyen $\Gamma = (V, E)$ Cayley gráf. A V csúcshalmazon vett permutációt automorfizmusnak nevezzük, ha megtartja Γ csúcs-szomszédsági viszonyát, tehát minden $(v_1, v_2) \in E$ élre és σ automorfizmusra $(\sigma(v_1), \sigma(v_2)) \in E$ is él, ahol $\sigma(v_1), \sigma(v_2) \in V^\sigma$. A Γ összes automorfizmusának a csoportját $Aut(\Gamma)$-val jelöljük.*

Vegyünk egy szokásos (G, S) párost és egy $\Gamma = (V, E)$ gráfot. Tegyük fel, hogy a Γ gráfunk egy Cayley gráfja a (G, S) párosnak és feleltessük meg V-t G-vel. Ekkor $Aut(\Gamma)$ tartalmaz egy $\hat{G}$ részcsoportot, amit a jobb oldali G csoport művelete indukál, tehát

$$\forall \hat{g} \in \hat{G}\text{-re } \hat{g} : x \to xg, \forall x \in V.$$

A $\hat{G}$ részcsoportot a G csoport reguláris reprezentációjának hívjuk.

Legyen $\hat{g} = \hat{x}^{-1}\hat{y} \in \hat{G}$ és $x, y \in V$ tetszőleges elemek, ekkor

$$\hat{g}(x) = xx^{-1}y = y$$

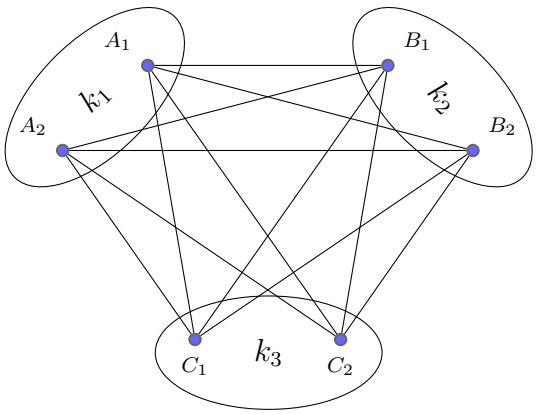
tehát $\hat{g}$ x-et y-ba képzi és így $\hat{G}$ tranzitív V-n. Ezen felül $Aut(\Gamma)$ tranzitív V-n és a Cayley gráfok csúcs-tranzitívak. Mivel $\hat{g} = \hat{x}^{-1}\hat{y}$ a $\hat{G}$ egyetlen eleme, ami x-et y-ba képzi, így $\hat{G}$ reguláris részcsoportja $Aut(\Gamma)$-nak. Mivel a Cayley gráfok csúcs-tranzitívak, ezért ha egy $Cay(G, S)$ Cayley gráf nem összefüggő, akkor az összes komponense (amik között nem fut él) izomorf a $Cay(\langle S\rangle, S)$ gráfra, ahol $\langle S\rangle$ az S elemei által generált csoport. Így adódik az alábbi állítás:

3.3.5. Állítás. *Egy irányított Γ gráf egy Cayley gráfja G csoportnak pontosan akkor, ha $Aut(\Gamma)$ tartalmaz egy részcsoportot, ami izomorf G-re és reguláris V-re.*

Az utóbbi állítás bizonyítása Sabidussi [13] nevéhez kötődik és összeköti a Cayley reprezentációs problémát az $Aut(\Gamma)$ vizsgálatával.

3.4. Cayley izomorfia $K_{m,k}$ gráfokon

Az alábbi fejezetben a $K_{m,k}$ gráfok Cayley reprezentációját fogjuk vizsgálni példaként. $K_{m,k}$-val jelöljük azt az irányított teljes k-részes gráfot, ahol m a partíciók mérete, illetve $m, k \in \mathbb{N}$ pozitív egész számok. Például $K_{2,3}$ gráfja:



$K_{2,3}$ gráfja

3.4.1. Állítás. *Egy G csoportnak pontosan akkor létezik Cayley gráfja ami izomorf $K_{m,k}$-ra, ha G $m \cdot k$-ad rendű és létezik m-ed rendű részcsoportja.*

Bizonyítás: Legyen G egy n-ed rendű csoport, $G_0 \subset G$ egy m-ed rendű részcsoport és $S = (G \setminus G_0)$, tehát azon G-beli elemek halmaza amelyek nincsenek benne G_0-ban. Ekkor a $Cay(G, S)$ gráf izomorf a teljes k-részes $K_{m,k}$ gráfra, ahol $k = n \cdot m^{-1}$. Legyen S_n egy $n \in \mathbb{N}$-ed fokú szimmetrikus csoport, ami a $(1, 2, 3, \cdots, n)$ permutációs csoportja, ekkor mivel $Aut(\Gamma) \cong S_m^k \rtimes S_k$, ahol S_m^k a k darab blokk külön-külön vett belső permutációi és S_k a blokkok permutációi. Így a $K_{m,k}$ kifejezhető mint egy Cayley gráf minden n rendű csoportra, aminek van egy m rendű részcsoportja.

A másik irányhoz tegyük fel, hogy $\Gamma = (G, S)$ egy Cayley reprezentációja a $K_{m,k}$ gráfnak. Ekkor a k részei $K_{m,k}$-nak egy $Aut(\Gamma)$-invariáns partíciót határoznak meg V-n, hiszen egy adott k_i blokkban lévő csúcs semelyik más csúccsal nincs összekötve k_i blokkban, de minden más csúccsal igen. Mivel a partíció $Aut(\Gamma)$-invariáns, ezért $\hat{G}$-invariáns is, hiszen $\hat{G} \subset Aut(\Gamma)$. Ekkor minden k_i blokkra igaz, hogy a $\hat{G}_{k_i} = \{\hat{g} \in \hat{G} | k_i^{\hat{g}} = k_i\}$ stabilizátor egy m-ed rangú részcsoportja $\hat{G}$-nek, így van karakterizációnk G-re. □

4. CI-gráfok

Az alábbi fejezetben azokat a Cayley izomorfia problémákat fogjuk vizsgálni, amik ugyan arra a csoportra hatnak.

Legyen szokásosan G csoport, $S \subset G$ részhalmaz és $Cay(G, S)$ a Cayley gráf. Mivel ugyan azon a csoporton dolgozunk a $Cay(G', S')$ esetén $G = G'$. Ha $S' = S$ a probléma megoldása triviális, így a továbbiakban feltesszük, hogy a két részhalmaz különböző.

Vegyük G egy σ automorfizmusát és legyen $S' = S^\sigma$. Ekkor σ indukál egy izomorfiát $Cay(G, S)$ és $Cay(G, S')$ között. Az utóbbi gondolatmenet ismerős lehet a 3.3.3-as Definícióból, annyi különbséggel, hogy itt ugyan azon a G csoporton dolgozunk.

Felmerülhet a kérdés, hogy lehetséges-e hogy létezik izomorfizmus $Cay(G, S)$ és $Cay(G, S')$ között, de nem tudunk mutatni σ által indukált izomorfiát S-ből S'-be? A válasz röviden igen, de erre bűvebben nem térünk ki.

Tehát itt csak azokról az esetekről beszélünk, ahol létezik $\sigma \in Aut(G)$ automorfizmus, hogy $S^\sigma = S'$.

4.0.1. Definíció. *Egy $Cay(G, S)$ gráfot G Cayley izomorfia gráfjának hívjuk, ha tetszőleges $Cay(G, S')$ gráfra teljesül, hogy ha $Cay(G, S)$ izomorf $Cay(G, S')$-val, akkor létezik $\sigma \in Aut(G)$ automorfizmus, amire $S^\sigma = S'$.*

A továbbiakban az egyszerűség kedvéért *Cayley izomorfia gráf* helyett *CI-gráf* rövidítést fogunk használni.

A definícióból adódik egy kérdés: G mely Cayley gráfjai CI-gráfok? A kérdésre a 7 fejezetben mutatunk egy algoritmust, ami eldönti, hogy egy adott Cayley gráf rendelkezik-e a CI tulajdonsággal.

A következőkben megnézzük pár következményét a definíciónak.

Tegyük fel, hogy $Cay(G, S)$ egy CI-gráf, ekkor ahhoz hogy eldöntsük egy $Cay(G, S')$ gráfról, hogy izomorf-e $Cay(G, S)$ gráfhoz, csak azt kell belátnunk, hogy létezik $\sigma \in Aut(G)$ automorfizmus, ami S-t S'-be képzi. Az egész $Aut(G)$ meghatározása egy nehéz probléma, de egy ilyen σ automorfizmus létezésének a meghatározása általában egyszerűbb.

Legyen G egy csoport, ekkor minden $\Gamma = Cay(G, S)$ gráfra és minden

$\sigma \in Aut(G)$automorfizmusra Γ^σ is egy $Cay(G, S^\sigma)$ Cayley gráf, így $Aut(G)$ hatva G-n indukál egy műveletet a G Cayley gráfjainak a halmazán.

4.0.2. Definíció. *Legyen $\Gamma = Cay(G, S)$ gráf és $S \subset G$ csoportok. Ekkor a $\mathcal{ISO}(G, \Gamma)$-val jelöljük G azon Cayley gráfjainak a halmazát, amelyek izomorfak Γ-ra.*

Felhasználva az utóbbi, illetve a 4.0.1 Definíciót az alábbi állításhoz jutunk:

4.0.3. Állítás. *Egy $\Gamma = Cay(G, S)$ gráf CI-gráfja G-nek pontosan akkor, ha $Aut(G)$ tranzitív $\mathcal{ISO}(G, \Gamma)$-n.*

Ekkor a részcsoportja $Aut(G)$-nek, ami rögzíti Γ-t az

$$Aut(G, S) = \{\sigma \in Aut(G) | S^\sigma = S\}.$$

Az $Aut(G, S)$ egy részcsoportja Γ automorfizmusainak, és $Aut(G, S)$ minden eleme fixálja G normálelemét. Ezért $Aut(G, S)$ egy részcsoportja a G normál elemének megfelelő csúcs csúcs-stabilizátorának.

4.1. Kritérium CI-gráfokra

Eddig a gráfizomorfia problémát gráfelméleti és csoportelméleti eszközökkel vegyesen vizsgáltuk. Viszont a következő tételünk a CI-gráf eldöntési problémát átalakítja egy csak csoportelméleti problémára.

4.1.1. Tétel. *Legyen Γ egy Cayley gráfja egy véges G csoportnak. Ekkor Γ pontosan akkor CI-gráfja G-nek, ha minden reguláris részcsoportja $Aut(\Gamma)$-nak, amik izomorfak G-re, konjugálnak.*

Az itt szereplő bizonyítás Li bizonyítása[10], Babai bizonyítása[1] alapján.

Bizonyítás: Legyen $\hat{G}$ a reguláris reprezentációja $Aut(\Gamma)$-nak, $Sym(G)$ a szimmetrikus csoportja G-nek és $\tilde{G}$ egy tetszőleges részcsoportja $Aut(\Gamma)$-nak, ami izomorf G-re és reguláris G csúcshalmazán.

Az első irányhoz azt tesszük föl, hogy Γ a G CI-gráfja, Mivel G minden reguláris reprezentációja permutáció izomorf, létezik $\varphi : \hat{G} \to \tilde{G}$ és $\rho \in Sym(G)$ bijekció, hogy:

$$\forall \hat{g} \in \hat{G}, \forall g \in G\text{-re } (g^{\hat{g}})^{\rho} = (g^{\rho})^{\hat{g}^{\varphi}} \Rightarrow \hat{g}\rho = \rho\hat{g}^{\varphi}.$$

Mivel ρ bijektív, ezért vehetjük az inverzét ρ^{-1}-t. Először ρ^{-1}-val balról szorozva, majd alkalmazva $\hat{G}$-re, az alábbit kapjuk:

$$\rho^{-1}\hat{g}\rho = \rho^{-1}\rho\hat{g}^{\varphi} \overset{\rho^{-1}}{\Longrightarrow} \rho^{-1}\hat{g}\rho = \hat{g}^{\varphi} \overset{\hat{G}}{\Rightarrow} \hat{G}^{\rho} = \rho^{-1}\hat{G}\rho = \hat{G}^{\varphi} = \tilde{G}.$$

Legyen $\Sigma = \Gamma^{\rho^{-1}}$, ahol Σ csúcsainak a halmaza G és tartja Γ csúcs-szomszédsági viszonyát. Ha alkalmazzuk ρ-t $Aut(\Sigma)$-ra:

$$Aut(\Sigma)^{\rho} = \rho^{-1}Aut(\Sigma)\rho = Aut(\Gamma)$$

Felhasználva, hogy $\hat{G} \leq Aut(\Gamma)$ a 3.3.5 Állítás szerint Σ egy Cayley gráfja G-nek. Felhasználva a feltevésünket létezik $\sigma \in Aut(\Gamma)$ automorfizmus, ami izomorfizmust indukál Γ és Σ között. Ekkor felhasználva, hogy $\Sigma = \Gamma^{\rho^{-1}}$:

$$\Gamma^{\sigma} = \Sigma = \Gamma^{\rho^{-1}} \Rightarrow \Gamma^{\sigma\rho} = \Gamma.$$

Tehát $\sigma\rho$ egy automorfizmusa Γ-nak. Így $\hat{G}$-re alkalmazva:

$$\hat{G}^{\sigma\rho} = \hat{G}^{\sigma^{-1}\sigma\rho} = \hat{G}^{\rho} = \tilde{G}.$$

Tehát $\hat{G}$ és $\tilde{G}$ egymás konjugáltjai $Aut(\Gamma)$-ban, és mivel $\tilde{G}$ tetszőleges reguláris részcsoportja volt $Aut(\Gamma)$-nak az első irányt beláttuk.

A második irányhoz azt tesszük föl, hogy minden G-re izomorf reguláris részcsoportjai az $Aut(\Gamma)$-nak egymás konjugáltjai. Legyen Σ egy Γ-ra izomorf Cayley gráfja G-nek, és φ egy izomorfia Σ és Γ között. Mivel Σ és Γ Cayley gráfjai G-nek, ezért $\varphi \in Sym(G)$ és a $\hat{G}^{\varphi}$ konjugáció reguláris G-n. Mivel $\hat{G} \leq Aut(\Sigma)$, $\hat{G}$-ra alkalmazva φ-t, a $\hat{G}^{\varphi} \leq Aut(\Gamma)$-t kapjuk.
Ezután felhasználva a feltevésünket, létezik $\sigma \in Aut(\Gamma)$, hogy:

$$\hat{G}^{\varphi} = \hat{G}^{\sigma} \Rightarrow \hat{G}^{\varphi\sigma^{-1}} = \hat{G}.$$

Így $\varphi\sigma^{-1} \in Sym(G)$ és normalizálja $\hat{G}$-t. Tehát $\varphi\sigma^{-1}$ indukál egy automorfizmust $\hat{G}$-n. Mivel φ egy izomorfizmus Σ és Γ között, és $\sigma^{-1} \in Aut(\Gamma)$, a $\varphi\sigma^{-1}$ is izomorfizmus Σ és Γ között, ezzel beláttuk a második irányt. □

A 4.1.1 Tétel fontos szerepet játszott a véges Cayley gráfok izomorfia vizsgálatában. A felhasználásának a bemutatására mutatunk egy példát a $DGRR$-re vonatkozóan.

4.1.2. Definíció. *Legyen $\Gamma = Cay(G, S)$ és $Aut(\Gamma) = \hat{G}$, ekkor az irányított Γ gráfot a G irányított grafikus reguláris reprezentációjának hívjuk és $DGRR$-el jelöljük.*

A $DGRR$-t szokás DRR-el is jelölni, a *digraphical regular reprezentation* elnevezés után.

Alkalmazzuk a 4.1.1 tételt a $DGRR$ definíciójára. Mivel $\hat{G}$ definíció szerint reguláris azt kapjuk, hogy ha egy Γ egy $DGRR$-je G-nek, akkor Γ egy CI-gráfja G-nek.

A $DGRR$ kapcsán eljuthatunk a normál csoportok fogalmához.

5. Normál Cayley gráfok

5.0.1. Definíció. *[4] Legyen G csoport és $S \in G$ részcsoportja. Ekkor S-t a G normál részcsoportjának hívjuk és $S \triangleleft G$-vel jelöljük, ha S egy tetszőleges elemét conjugálva a G egy tetszőleges elemével, egy S-be tartozó elemet kapunk.*

A definícióban a G-t megfeleltetve $Aut(\Gamma)$-val és S-t megfeleltetve $\hat{G}$-vel a Cayley gráfok egy új típusához érkezünk.

5.0.2. Definíció. *[10] Egy $\Gamma = Cay(G, S)$ gráfot G normál Cayley gráfjának hívunk, ha $\hat{G}$ normál részcsoportja $Aut(\Gamma)$-nak.*

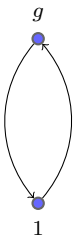
Így minden *DGRR*, a $Aut(\Gamma) = \hat{G}$ miatt, egy normális Cayley gráf.

A következőkben jelöljük Q_n-el az n dimenziós hiperkockát, ahol például $n = 2$ esetén a négyzetet, illetve $n = 3$ esetén az ismert kockát kapjuk.

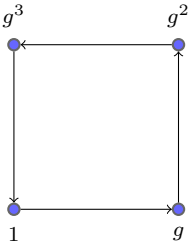
5.0.3. Definíció. *[4] Ha egy G csoportot generálja egy eleme, akkor ciklikus csoportnak hívjuk és Z_n-el jelöljük, ahol n a csoport rendje.*

Felhasználva a generálás definícióját, adható egy másik definíció is a ciklikus csoportokra: Ha egy G gráf minden eleme előáll egy $g \in G$ elemének egy hatványaként, akkor G egy ciklikus csoport. Az utóbbi definíció miatt szokás Z_n elemeit a $Z_n = \{1, g^1, g^2, \cdots, g^{n-1}\}$ módon is jelölni, ahol 1 a neutrális eleme G-nek.

*P*élda: Vegyük Z_2 és Z_4 Cayley gráfjait g generáló elem mellett.



$Cay(Z_2, \{g\})$

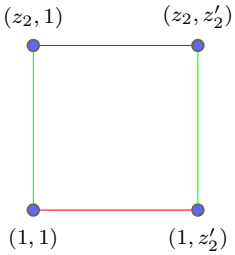


$Cay(Z_4, \{g\})$

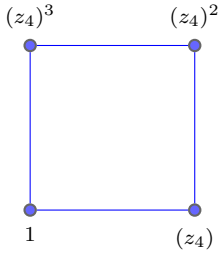
A következőkben bevezetjük a direct product-ot és összehasonlítjuk $Z_2 \times Z_2$ és Z_4-et.

5.0.4. Definíció. *[4] Legyen G és H csoport. Egy új M csoportot a G és H direct product-jának hívjuk és $G \times H$-vel jelöljük, ha az M csoport elemei rendezett párok G és H Descartes szorzata alapján, és az új csoport művelet pedig a $(g,h) \times (g',h') = (g \star_G g', h \star_H h')$, ahol $\star_G$ a G csoport művelete és $\star_H$ a H csoport művelete.*

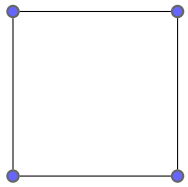
Most vegyük $z_2 \in Z_2$, $z_2' \in Z_2'$ és $z_4 \in Z_4$ generáló elemeket és vizsgáljuk meg a $Cay(Z_2 \times Z_2', \{(z_2,1),(1,z_2')\})$ és a $Cay(Z_4, \{z_4, z_4^{-1}\})$ Cayley gráfokat. Az egyszerűség kedvéért összevonjunk az irányított éleket irányítatlanná.



$Cay(Z_2 \times Z_2', \{(z_2,1),(1,z_2')\})$



$Cay(Z_4, \{z_4, z_4^{-1}\})$



Q_2 Hiperkocka

Ahogy láthatjuk a $Cay(Z_2 \times Z_2', \{(z_2,1),(1,z_2')\})$ és $Cay(Z_4, \{z_4, z_4^{-1}\})$ Cayley gráfokkal ábrázolhatjuk a Q_2 hiperkocka gráfját. Ez önmagában nem egy izgalmas eredmény, de ha megvizsgáljuk a két példánkat a 5.0.2 definíció alapján egy érdekes állításhoz jutunk. Először is szükségünk lesz a Q_2 hiperkocka automorfizmus csoportjára, $Aut(Q_2)$-re.

$Aut(Q_2)$ meghatározása

Egy négyzet szimmetriáinál a forgatás és tükrözés könnyen adódik. Jelöljük r-el a 90°-os forgatást, és m-el a Q_2 egy átlójára való tükrözést. Mivel a negatív irányba való forgatások kifejezhetők a pozitív irányba való forgatások ismétlésével, és bármely tükrözés kifejezhető az m átlónkra való tükrözés és a forgatások kombinációjaként, az alábbi táblázatot kapjuk:

	id	r	r^2	r^3	m	mr	mr^2	mr^3
id	id	r	r^2	r^3	m	mr	mr^2	mr^3
r	r	r^2	r^3	id	mr^3	m	mr	mr^2
r^2	r^2	r^3	id	r	mr^2	mr^3	m	mr
r^3	r^3	id	r	r^2	mr	mr^2	mr^3	mr
m	m	mr	mr^2	mr^3	id	r	r^2	r^3
mr	mr	mr^2	mr^3	m	r^3	id	r	r^2
mr^2	mr^2	mr^3	m	mr	r^2	r^3	id	r
mr^3	mr^3	m	mr	mr^2	r	r^2	r^3	id

Ezt a táblázatot Cayley táblázatnak hívjuk és egy csoport struktúráját mutatja be. Mivel egy n rendű csoportnál $(n+1)(n+1)$-es a táblázat, ezért nagyobb csoportoknál időigényes a használata, de szép képet tud adni egy csoportról.

Mivel a táblázatban nem szerepel más elem, mint a megadott 8 darab (a táblázat első sora), a $\langle r, m : r^4 = m^2 = id, rm = mr^{-1} \rangle$ generálja $Aut(Q_2)$-t, ami a D_4. Általánosan egy szabályos n-szög szimmetria csoportját diéder csoportnak hívjuk és D_n-el jelöljük. Tehát az $Aut(Q_2)$:

$$Aut(Q_2) = \{id, r, r^2, r^3, m, mr, mr^2, mr^3\}.$$

Igaz e, hogy $\hat{Z_2^2} \triangleleft Aut(Q_2)$?

Definíció alapján szükségünk lesz $Z_2 \times Z_2'$ reguláris reprezentációjára, amit jelöljünk $\hat{Z_2^2}$-vel és definíció alapján:

$$\forall \hat{z} \in \hat{Z_2^2}\text{-re } \hat{z} : x \mapsto xz, \forall x \in Z_2 \times Z_2'$$

Legyen $a \in Aut(Q_2)$ és $\hat{z} \in \hat{Z_2^2}$, így $\hat{Z_2^2} \triangleleft Aut(Q_2)$-hoz:

$$\forall a, \hat{z}\text{-re: } a^{-1}\hat{z}a \in \hat{Z_2^2}$$

Mivel $Aut(Q_2)$ elemei előállnak r és m kombinációiként, illetve felhasználva, hogy $mr^i = r^{-i}m$, $m^{-1} = m$ és $(mr^i)^{-1} = mr^i$. Ha tudjuk hogy $m^{-1}\hat{z}m \in \hat{Z_2^2}$ teljesül minden $\hat{z}$-re:

$$(mr^i)^{-1}\hat{z}(mr^i) \to mr^i\hat{z}mr^i \to (r^i)^{-1}(m^{-1}\hat{z}m)r^i \to (r^i)^{-1}\hat{z}'r^i \overset{?}{\in} \hat{Z_2^2}$$

Tehát ha tudjuk a tükrözésről hogy megfelelő, akkor elég csak a forgatásokat vizsgálni.

Mivel a csoport minden eleme előáll a generátor elemeinek egy kombinációjaként, elég a generátokat vizsgálni.

Bizonyítás: Legyen g_i a generátorja G-nek és $\hat{g} \in \hat{G}$ egy reguláris reprezentációja. Mivel a reguláris reprezentáció egy automorfizmus, a képe is G-be tartozik. Így ha $\hat{g_i}$ a g_i-vel való jobboldali szorzás, akkor:

$$\hat{g_i} : x \mapsto xg_i \Rightarrow \hat{g_i} : xg_i \mapsto (xg_i)g_i = xg_i^2 \Rightarrow \hat{g_i} : xg_i^2 \mapsto (xg_i^2)g_i = xg_i^3 \Rightarrow \cdots$$

Tehát $\hat{g_i}$ ismétlésével elő tudjuk állítani az $\hat{G}$ összes elemét. □

Továbbá $\hat{Z_2^2}$ elemeire gondolhatunk úgy, mint:

- $\star(1, 1)$, az identitás
- $\star(z_2, 1)$, egy m tükrözés, mivel a tükrözések forgatással átvihetők egymásba
- $\star(1, z_2')$, egy r^2-el elforgatott m tükrözés, tehát mr^2
- $\star(z_2, z_2')$, egy r^2 elforgatás.

Először vegyük a $\star(z_2, 1)$ elemet r^i forgatásra és m tükrözésre:

- $(r^i)^{-1}mr^i = mr^{2i}$, ami $\pmod 4$ mellet, vagy m vagy mr^2, amik elemei $\hat{Z_2^2}$-nek
- $mmm = m$, ami eleme $\hat{Z_2^2}$-nek.

Másodszor vegyük $\star(1, z_2')$ elemet r^i forgatásra és m tükrözésre

- $(r^i)^{-1}mr^2r^i = mr^{i+2}r^i = mr^{2i+2}$, ami $\pmod 4$ mellet, vagy m vagy mr^2, amik elemei $\hat{Z_2^2}$-nek
- $mmr^2m = mr^{-2}$, ami $\pmod 4$ mellet mr^2 ami m eleme $\hat{Z_2^2}$-nek.

Tehát a Q_2 **normál** cayley gráfja Z_2-nek.

Igaz e, hogy $\hat{Z_4} \triangleleft Aut(Q_2)$?

Az előző esethez hasonlóan járunk el. Legyen $\hat{Z_4}$ a Z_4 reguláris reprezentációja és $\hat{z_4} \in \hat{Z_4}$ egy elem.

Itt az tűnhet fel, hogy a $\hat{Z}_4$ elemei a forgatáshoz hasonlítanak:

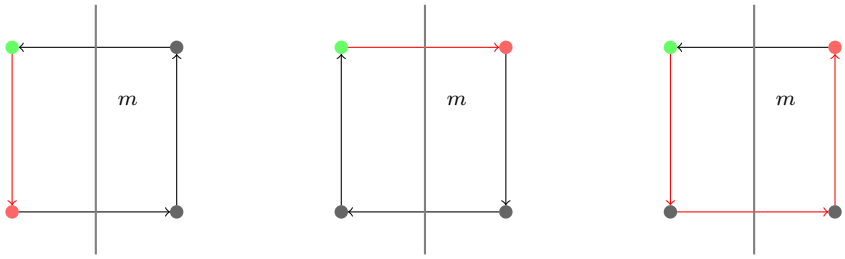
- $\star 1$, az identitás
- $\star q$, egy r forgatás
- $\star q^2$, egy r^2 forgatás
- $\star q^3$, egy r^3 forgatás

Vegyük a $\star q$ elemet r^i forgatásra és m tükrözésre:

- $(r^i)^{-1} r r^i = r^{2i+1}$, ami $\pmod 4$ mellet, vagy r vagy r^3, amik elemei $\hat{Z}_4$-nek
- $mrm = mmr^{-1}$, ami $\pmod 4$ mellet r^3, ami eleme $\hat{Z}_4$-nek.

Tehát a Z_4 normál részcsoportja $Aut(Q_2)$-nek, így a Q_2 **normál** Cayley gráfja Z_4-nek.

Hogy jobban látható legyen, hogy mi történik a konjugálás közben, egy ábrán is megmutatjuk a $m^{-1}(\star q)m$ konjugálásra. Eredetileg az összes pontra hatna, de az átláthatóság kedvéért csak egy pontra ábrázoljuk.



$\star q$ $\quad$ $m^{-1}(\star q)$ $\quad$ $m^{-1}(\star q)m$

Mivel itt maga a reguláris reprezentációt tükrözzük, csak az éleket kell tükröznünk m-re.

Most vegyük a $Z_2 \times Z_2$ gráfot és vizsgáljuk rá, hogy normál Cayley gráfja-e $Z_2 \times Z_2$-nak, vagy Z_4-nek. Ekkor azt kapjuk, hogy $Z_2 \times Z_2$-re normál, viszont Z_4-re nem. [14]

Tehát a vizsgálatból adódik, hogy $\Gamma = Cay(G, S)$ esetén, hogy a Γ normál Cayley gráfja legyen G-nek, nem csak a G-től, hanem Γ-tól is függ.

Sok eredmény született a normál Cayley gráfok témakörében, először is alkalmazzuk a 4.1.1-tételt normál Caylok gráfok esetén, ekkor az alábbit kapjuk.

5.0.5. Következmény. *[10] Legyen G egy véges csoport és Γ normál Cayley gráfja G-nek. Ekkor Γ pontosan akkor CI-gráfja G-nek, ha $\hat{G}$ az egyetlen reguláris részcsoportja $Aut(\Gamma)$-nak, ami izomorf G-re.*

5.0.6. Tétel. *[10] Legyen G egy véges nem kommutatív csoport, úgy hogy tartalmaz egy g elemet, ami nem konjugál g^{-1}-hez és legyen $S = \{x^{-1}gx | x \in G\}$. Ekkor a $Cay(G, S)$ gráf normál Cayley gráf, de nem CI-gráf.*

Ez a tétel alapján tudunk szerkeszteni irányítatlan normál Cayley gráfokat, amik nem CI-gráfok.

6. A.Ádám sejtése $n = p$ esetén

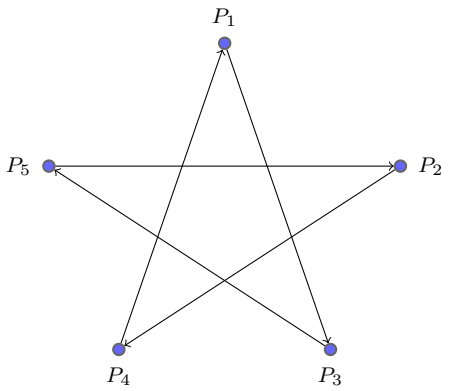
Az alábbi fejezetben megvizsgálunk egy A.Ádám [15] által felvetett elégséges feltételt egy gráfról és a hozzá tartozó sejtését. Továbbá belátjuk a sejtésről Djokovic [3] bizonyítása alapján, hogy prím n esetén igaz.

6.1. Elégséges feltétel és a sejtés egy más formája

Az említett gráfot definiáljuk az alábbi módon:
Legyenek $0 < k_1 < k_2 < \cdots < k_m < n$ egész számok. Legyen $G_n(k_1, \cdots, k_m)$ irányított gráf a $P_1, \cdots, P_n$ csúcsokkal, ahol P_i-ből fut él P_j-be, ha

$$\exists t, 0 \leq t \leq n : k_t \equiv j - i \pmod{n}.$$

*P*élda: Legyen $k = 2$ és $n = 5$, így az élek $i \equiv j - 2 \pmod{n}$.



$G_5(2)$ gráf

Legyen $G_n(k_1, \cdots, k_m)$ és $G'_n(k'_1, \cdots, k'_m)$ gráfok. Ekkor A.Ádám elégséges feltétele, hogy ha $G_n(k_1, \cdots, k_m)$ és $G'_n(k'_1, \cdots, k'_m)$ izomorf, akkor kell hogy létezzen $0 \leq r \leq n$ ami relatív prímje n-nek, és σ permutációja $\{1, \cdots, n\}$-nek, hogy $k'_t \equiv r k_{\sigma(t)} \pmod{n}$.

6.1.1. Sejtés. *Ez nem csak elégséges, hanem szükséges feltétel is.*

6.1.2. Állítás. *Ha n prím, akkor az állítás nem csak elégséges, hanem szükséges is.*

A továbbiakban jelöljük $S(x)$-el egy véges nem negatív egész számokat tartalmazó halmaz polinómját ahol $S(x) = \sum_{s \in S} x^s$, $|S|$-el az S halmaz

rendjét. Legyen $Z_n = \{0, 1, \cdots, n-1\}$ egy gyűrű a $(\mathrm{mod}\ n)$-es összeadással és szorzással, és jelöljük $z_1 \star z_2$-vel $z_1, z_2 \in Z_n$ szorzatát. Legyen $N = \{0, 1, \cdots, n-1\} \subset Z_n$, ekkor Z_n invertálható elemeinek az M multiplikatív csoportja hat N részhalmazainak a halmazán. Egy $a \in M$ művelet $K \subset N$-n az alábbi módon definiálunk:

$$K \xrightarrow{a} a \star K = \{a \star k | k \in K\}.$$

Ekkor feltűnhet, hogy egy ilyen a művelet nem változtat K rendjén. Így jelöljük $K' \sim K$-el, ha létezik a, hogy $K' = a \star K$.

6.1.3. Következmény. *Ekkor a 6.1.1 sejtésben a szükségesség értelmezhető az alábbi módon is:*

$$\textit{Ha } G_n(K) \cong G_n(K'), \textit{ akkor } K' \sim K.$$

A Z_n-t tekinthetjük ciklikus csoportnak, hiszen az összeadásra nézve az elem 1 generálja a csoportot. Az M multiplikatív csoportra gondolhatunk úgy, mint N automorfizmus csoportjára Z_n felett. Ha pedig N jelölését lecseréljük G-re, és K jelölését lecseréljük S-re, akkor egy ismerős problémához jutunk.

$$G_n(S) \cong G'_n(S') \Rightarrow \exists a \in Aut(G)_{Z_n} : a \star S = S'$$

Ha $G_n(S)$-t lerögzítjük és $G'_n(S')$-t szabadon választjuk mellé, akkor arra a kérdésre jutunk, hogy $G_n(S)$ CI-gráfja-e G-nek Z_n felett. A következő fejezetben belátjuk, hogy $n = p$ esetén elég csak a sajátértékeit vizsgálni Z_p ciklikus csoport felett.

6.2. A sejtés bizonyítása

Legyen A a szomszédsági mátrixa $G_n(K)$-nak, ekkor A egy sor-ciklikus mátrix és így az első oszlopban csak a K elemeinél van 1. Vegyük $G_n(K)$-nak a A'- mátrixát hasonló módon. Ekkor ha feltesszük, hogy $G_n(K) \cong G_n(K')$, akkor létezik P permutáció mátrix, hogy:

$$A' = P^{-1}AP$$

Ekkor felhasználva a v_A sajátvektor és λ_A sajátérték definícióját:

$$\lambda_A v_A = Av_A \xrightarrow{P\star} P\lambda_A v_A = PAv_A = PAP^{-1}Pv_A = A'(Pv_A) = \lambda_A(Pv_A)$$

Tehát a sajátértékeik megegyeznek, így a [11]-es forrás ciklikus mátrixokra vonatkozó része alapján:

$$K(\epsilon^k) = K'(\epsilon^{\pi(k)}), k \in N$$

ahol ϵ az n-edik primitív gyöke az egység elemnek, és π az N egy permutációja.

A 6.1.2 állítás bizonyításához, először bevezetünk egy jelölést, majd kimondunk egy lemmát. Ha $S \subset N$, akkor S^t-vel jelöljük azon $s \in S$ elemek halmazát, melyek oszthatóak p^t-vel, de nem oszthatóak p^{t+1}-el.

6.2.1. Lemma. *Ha $G_n(K) \cong G_n(K')$ és $n = p^t$, akkor létezik $K_1 \sim K$, $K_1' \sim K'$ és egy π permutációja N-nek, hogy $\pi(k) = k$ ahol $k \in N^0$, és $K_1(\epsilon^k) = K_1'(\epsilon^{\pi(k)}), k \in N$*

Bizonyítás:[6.1.2] Legyen $G_n(K) \cong G_n(K')$, ekkor a 6.2.1 Lemma szerint létezik

$$K_1(\epsilon^k) = K_1'(\epsilon^k), k \in N$$

ahol $K \sim K_1$ és $K' \sim K_1'$. Most tegyük fel, hogy $K_1(\epsilon^k) - K_1'(\epsilon^k) \neq 0$. Ekkor a különbség egy olyan polinóm, aminek egészértéűek az együtthatói és a foka $\leq p-1$, konstans rész nélkül. Egy ilyen polinóm nem lehet a minimum polinóm többszöröse ami:

$$1 + x + \cdots + x^{p-1}$$

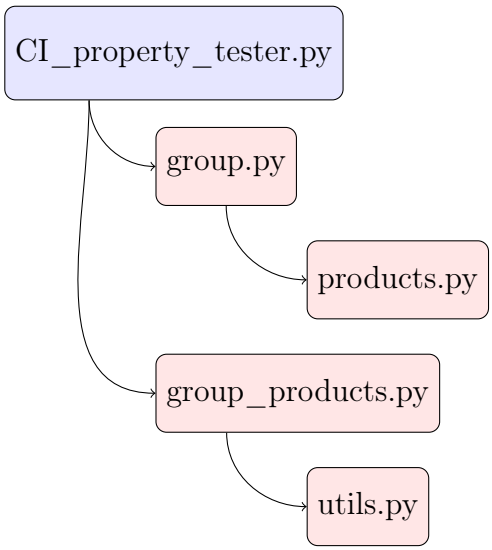
Tehát $K_1(\epsilon^k) - K_1'(\epsilon^k) = 0$, így $K_1 = K_1'$. Felhasználva, hogy $K \sim K_1$ és $K' \sim K_1'$, így $K \sim K'$ teljesül. Ezzel belátva, hogy prím n esetén igaz a sejtés. □

7. CI-ellenörző

Az alábbi fejezetben megnézzük az általam Python-ban készített CI-ellenörző felépítését, működését és a futási idejét. A kód célja, hogy meghatározza, hogy egy adott Cayley gráf rendelkezik-e a CI-tulajdonsággal.

A kódot modulárisan építettem fel, tehát külön fájlokban és függvényekben vannak megvalósítva az algoritmusok, mint például a Cayley gráf generálása. Az ilyen tipúsú felépítés egyszerűvé teszi az algoritmus bővítését és optimalizálását.

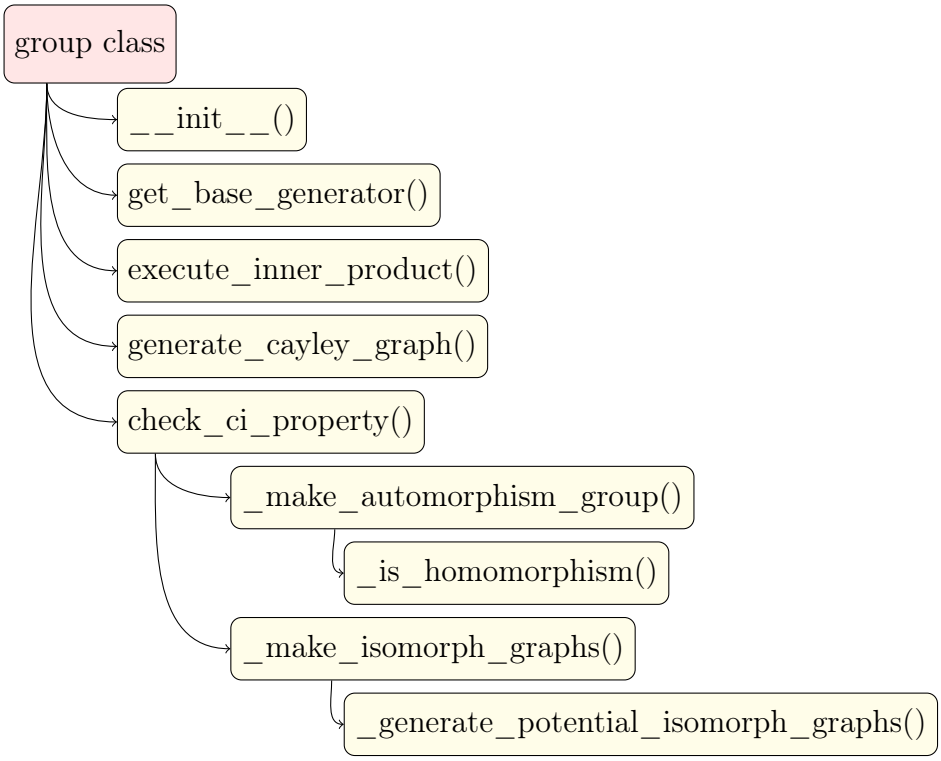
A programkód 5 fájlból tevődik össze, az alábbi módon:



A *CI_property_tester.py* tartalmazza a *main()* függvényt és felelős a program futasáért.

7.1. group.py felépítése és működése

A *group* klassz felelős a programban használt csoport szerkezetének a létrehozásáért, illetve ez tartalmazza a függvények többségét is. Az alábbi ábrán látható a klassz felépítése, a továbbiakban pedig a függvényeinek a vizsgálata.



7.1.1. __init__()

Ez a konstruktor inicializálja a csoportot 2 kötelező és 1 választható paraméterrel $O(1)$ időben:

- **base_set:** a csoport elemei.
- **product:** a csoport művelete.
- **base_generator:** a Cayley gráfhoz szükséges $S \subset G$ részhalmaz, amit nem kötelező megadni a klassz működéséhez.

7.1.2. get_base_generator()

Az opcionálisan megadott $S \subset G$ részhalmaz lekérdezője, ami hiba üzenetet ad, ha nem lett megadva, de használni akarnánk. $O(1)$ időben lekérhető.

7.1.3. execute_inner_product()

A csoporthoz tarzozó kettő általunk megadott elemnek a szorzatát adja vissza. A futási ideje a *products.py*-ban megadott csoport művelet alapján változik. A továbbiakban $O(1)$-nek veszem.

7.1.4. generate_cayley_graph()

Egy adott halmaz alapján létrehozza a Cayley gráfot. Mivel a csoport minden elemére vizsgálnia kell a részhalmazzal való szorzatát, a futási ideje $O(|G| \cdot |S|)$.

7.1.5. _is_homomorphism()

A *_make_automorphism_group()* által vizsgált permutációra teszteli, hogy homomorfizmus-e. A futási ideje $O(|G|^2)$, ahol $|G|$ a permutáció elemszáma.

7.1.6. _make_automorphism_group()

Kiszámolja a csoport automorfizmus csoportját az által, hogy létrehozza az összes permutációját a csoport elemeinek és a *_is_homomorphism()* meghívva teszteli, hogy homomorfizmusok-e.

Egy $|G|$ elemű csoportnak $|G|!$ darab permutációja lehet, csak ennek a függvénynek a futási ideje $O(|G|!)$. Mivel minden permutációra meghívja a *_is_homomorphism()* függvényt az összesített futási ideje $O(|G|! \cdot |G|^2)$

7.1.7. _generate_potential_isomorph_graphs()

Mivel a gráf csúcsainak a fokszáma függ a $S \subset G$ részhalmaz elemszámától, így elég csak azokat a Cayley gráfokat vizsgálnunk, amelyeknél a $T \subset G$ részhalmaz elemszáma megegyezik S elemszámával. A függvény kigenerálja az összes ilyen Cayley gráfot és átadja a *_make_isomorph_graphs()* függvénynek.

Mivel G-ből kiválasztanunk az összes $|S|$ elemszámú részhalmazt az iteráció lépésszáma $O(\binom{|G|}{|S|})$. Viszont mivel minden lépésben létre is kell hoznunk a gráfokat a *generate_cayley_graph()* segítségével az összesített futási időnk $O(\binom{|G|}{|S|} \cdot |G| \cdot |S|)$.

7.1.8. _make_isomorph_graphs()

A függvény először meghívja a *generate_cayley_graph()* függvényt, majd megvizsgálja a *_generate_potential_isomorph_graphs()* által megadott lehetséges gráfokat, hogy közülük melyek az eredeti gráfunkra ténylegesen izomorfak. Az utóbbit a *nx.isomorphism.DiGraphMatcher()* függvény által teszi, aminek a működésére később térünk ki.

Az iteráció lépésszáma a *_generate_potential_isomorph_graphs()* szerint $O(\binom{|G|}{|S|})$. A *nx.isomorphism.GraphMatcher()* függvény legrosszabb futási ideje $O(|G|! \cdot |G|)$, így összegezve a függvény legfeljebb

$$O(|G| \cdot |S|) + O(\tbinom{|G|}{|S|} \cdot |G| \cdot |S| + \tbinom{|G|}{|S|} \cdot |G|! \cdot |G|) \text{ lépést tesz meg.}$$

7.1.9. check_ci_property()

A függvény az eredeti gráfunk CI tulajdonságát ellenörzi le. Az utóbbihoz végig kell néznie, hogy a gráfunkra minden izomorf Cayley gráfhoz szükséges részhalmazra létezik-e automorfizmus ami átviszi az eredeti gráfunk részhalmazára.

Az izomorf gráfok listáját a *_make_isomorph_graphs()* függvény alapján határozza meg $O(|G| \cdot |S|) + O(\tbinom{|G|}{|S|} \cdot |G| \cdot |S| + \tbinom{|G|}{|S|} \cdot |G|! \cdot |G|)$ lépésben.

A csoport automorfizmus csoportját a *_make_automorphism_group()* függvény alapján határozza meg $O(|G|! \cdot |G|^2)$ lépésben.

Az iterációk összesített futási ideje $O(\tbinom{|G|}{|S|} \cdot |G|! \cdot |S|)$, hiszen:

- Az első iteráció lépésszáma legfeljebb $O(\binom{|G|}{|S|})$ lehet, hiszen legfeljebb ennyi féle képpen tudunk kiválasztani $|S|$ darab generátort G-ből.
- A második iteráció lépészszáma legfeljebb $O(|G|!)$ lehet, hiszen legfeljebb ennyi permutációval dolgozhatunk.
- A harmadik iteráció $|S|$ méretű halmazon fut végig, így a lépésszáma $O(|S|)$.

Összegezve a futási időket az alábbit kapjuk:

$$O(|G|\cdot|S|)+O(\tbinom{|G|}{|S|}\cdot|G|\cdot|S|+\tbinom{|G|}{|S|}\cdot|G|!\cdot|G|)+O(|G|!\cdot|G|^2)+O(\tbinom{|G|}{|S|}\cdot|G|!\cdot|S|)$$

Ha feltesszük, hogy $|S| \neq 0$, akkor $O(\binom{|G|}{|S|} \cdot |G|! \cdot |G|)$.

7.1.10. nx.isomorphism.DiGraphMatcher()

A függvény a networkx könyvtár gráf izomorfia vizsgáló függvénye és az úgy nevezett VF2-es algoritmus az alapja.[9]

Az algoritmusunk egy rekurzív algoritmus, ami először megvizsgál pár alapvető feltételt, például hogy a két gráf csúcsszáma megegyezik-e. Ha a két gráf megfelelt az elővizsgálatnak az algoritmus belép a rekurzív részébe. Az algoritmus próba szerűen megpróbálja csúcsonként megfelelteni a két gráfot egymásnak és minden lépésben ellenörzi, hogy a hozzávett csúcsoknak és éleknek köszönhetően sérül-e az izomorfia csúcs-szomszédsági tulajdonsága. Az utóbbi ellenörzést $O(deg(v) + deg(w))$ lépésben teszi meg.

Mivel a mi gráfunk nem rendelkezik semilyen különleges csúcs- vagy él-tulajdonsággal, mint például súllyal, az ehhez tartozó függvény ránk nem vonatkozik.

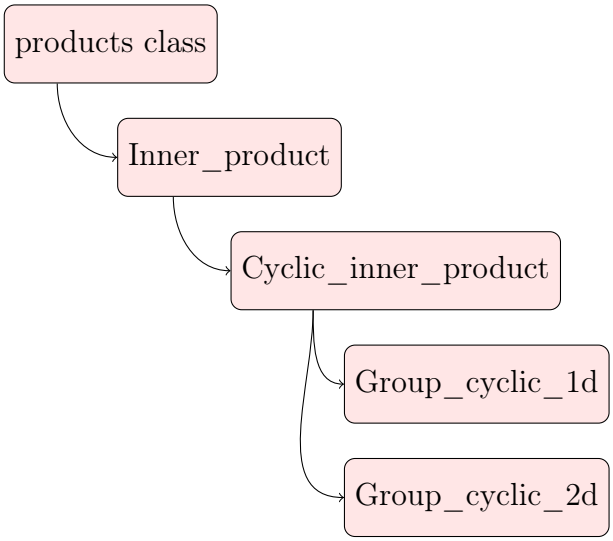
Az algoritmusnak szüksége van az eddig vizsgált és nem vizsgált csúcsok halmazára, így ezt is frissíti minden rekurzív meghívás során $O(|V|)$ lépésben.

A legrosszabb esetben az összes lehetőséget végig kell próbálnia, ami $O(|V|!)$ lépést jelent. Így a *nx.isomorphism.DiGraphMatcher()* összesített futási ideje:

$$O(|V|!) \cdot (O(deg(v) + deg(w)) + O(|V|)) \approx O(|V|! \cdot |V|).$$

7.2. products.py felépítése és működése

A *products* klassz felelős a csoport műveletének az elkódolásáért. A felépítésének az előnye, hogy könnyen bővíthető, akár általunk tetszőlegesen meghatározott művelettel is.

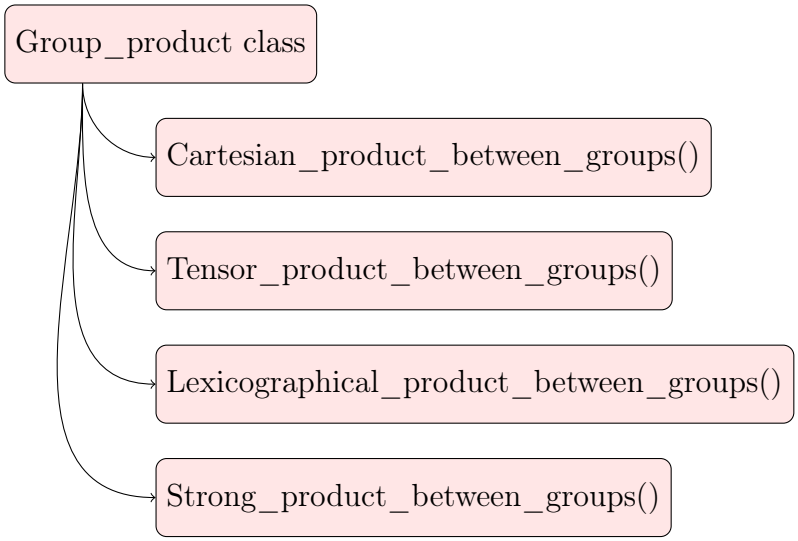


Általam két csoport művelet lett definiálva, mindkettő az additív ciklikus csoportokra vonatkozóan.

- *Group_ cyclic_ 1d*: Olyan ciklikus csoport melynek az elemei számok.
- *Group_ cyclic_ 2d*: olyan ciklikus csoport melynek az elemei vektor szerűen 2 darab számból állnak, például $(0, 1)$.

7.3. group_products.py felépítése és működése

Az alábbi fájl a programkód felhasználásához szükséges. A célja, hogy két Cayley gráf gráfszorzata esetén előállítsa az új $Cay(G, S)$ gráfhoz tartozó G és $S \subset G$ csoportokat.



7.4. ultis.py

Ez a fájl tartalmazza a kód futásához szükséges, de nem az algoritmushoz tartozó függvényt. A *nested_tupple_reductor()* feladata, hogy az algoritmus során felmerülő tuple egymásba ágyazást megoldja.

7.5. CI_property_tester.py felépítése és működése

A *CI_property_tester.py* felelős a kód futtatásáért, így ez tartalmazza *main()* függvényt. Ezen felül tartalmaz 3 függvényt, melyek a példák futtatásához szükségesek:

- *generate_base_set()*: Legenerálja a ciklikus csoport halmazát a megadott mérettel.
- *generate_cyclic_group()*: Legenerálja a ciklikus csoportot a megadott mérettel.
- *generate_random_generator()*: Legenerálja egy adott méretű csoportnak egy véletlenszerű részhalmazát, véletlenszerű mérettel.

7.6. Példák a kód felhasználására

Az alábbi alfejezetben különböző felhasználási módjaira hozunk példákat a programkódnak. Fontos megjegyezni, hogy mivel egyedül az additív ciklikus csoportoknak a művelete van inicializálva, így a példák is csak ezeket fogják tartalmazni. A példák sorban a *main()*, *main2()* és *main3()* alatt találhatók meg, melyeknek a futását a *if __name__ == "__main__":* függvény alatt tudjuk állítani.

7.6.1. Véletlenszerű részhalmazra CI vizsgálat

Az első példánk a *main()* függvény alatt található meg. A programkód feladata, hogy egy általunk megadott csoport méretre legenerálja az additív ciklikus csoportunkat (Z_n^+), illetve generáljon hozzá egy tetszőleges részhalmazát ($S \subseteq Z_n^+$). Végül kiszámolja és kiírja, hogy a hozzájuk tartozó $Cay(Z_n^+, S)$ Cayley gráf rendelkezik-e a CI tulajdonsággal.

Az ilyen típusú kódnak sok gyakorlati haszna nincs, de nagyban megkönnyebbítette a tesztelés folyamatát.

7.6.2. Egy csoport összes Cayley gráfjának a CI vizsgálata

A második példánk a *main2()* függvény alatt található meg. A programkód feladata, mint az előzőnél példánál, hogy egy általunk adott méretre legenerálja az additív ciklikus csoportunkat (Z_n^+).

A különbség az előzőhöz képest, hogy itt a programkód végignézi az összes lehetséges részhalmazt ($S \subseteq Z_n^+$) és megnézi melyek mellett rendelkezik a CI tulajdonsággal a $Cay(Z_n^+, S)$ Cayley gráfunk.

A következő táblázatban feltüntetjük, hogy a $Z_1, Z_2, Z_3, \cdots, Z_{10}$ esetekre mit ad vissza a kód.

Csoport	Részhalmazok, amikre a Cayley gráf nem CI
Z_1	None
Z_2	None
Z_3	None
Z_4	None
Z_5	None
Z_6	None
Z_7	None
Z_8	(6, 7, 3), (2, 5, 1), (6, 5, 1), (2, 7, 3), (2, 5, 4, 1), (6, 5, 1, 0), (6, 7, 0, 3), (2, 5, 1, 0), (6, 4, 7, 3), (2, 4, 7, 3), (2, 7, 0, 3), (6, 5, 4, 1), (6, 4, 7, 0, 3), (2, 5, 4, 1, 0), (6, 5, 4, 1, 0), (2, 4, 7, 0, 3)
Z_9	(2, 5, 8, 3), (4, 1, 7, 3), (6, 4, 1, 7), (6, 2, 5, 8), (6, 2, 5, 8, 0), (6, 4, 1, 7, 0), (4, 1, 7, 0, 3), (2, 5, 8, 0, 3)
Z_{10}	None

7.6.3. Gráfszorzatra a CI vizsgálata

A harmadik példánk a *main3()* függvény alatt található meg. A programkód feladata, hogy vizsgálja CI tulajdonsággal rendelkező és nem rendelkező gráfok szorzatának a CI tulajdonságát.

A programkódban a Cartesian gráf szorzat van megvalósítva, de ahogy a *group_products.py* fájban is látható több féle szorzatra is elvégezhetnénk.

Az előző példákhoz hasonlóan itt is additív ciklikus csoportokkal dolgozunk, de természetesen mi magunk is meghatározhatunk egy csoportot és hozzá csoportműveletet, illetve a Cayley gráf generálásához szükséges részhalmazt is.

A programkód először létrehoz nekünk 2 additív ciklikus csoportot részhalmazukkal együtt és kiszámolja, hogy teljesül-e a Cayley gráfjukra a CI tulajdonság. Ezután kiszámítja a két Cayley gráf szorzata során létrejövő szorzat csoportot és részhalmazát is. Végül a szorzat gráfra is kiszámolja, hogy teljesül-e a CI tulajdonság.

Legyen G_1 és $S_1 \subseteq G_1$ az első csoportunk és részhalmaza, illetve G_2 és $S_2 \subseteq G_2$ a második csoportunk és részhalmaza. Ekkor a szorzat csoportunk

elemszáma $|G_1| \cdot |G_2|$, illetve részhalmazának az elemszáma $|G_1| + |G_1|$. A 7.1.9 alfejezetben szereplő futási idő alapján látható, hogy a példa jelenlegi problémája, hogy már kis értékeknél is túl hosszú ideig fut.

Itt is megnézzük a kód futását pár kis példára. Először feltüntetjük a csoportot, majd a részhalmazát, végül pedig, hogy teljesül-e a CI tulajdonság.

G	$S \subseteq G$	CI	H	$T \subseteq H$	CI	$G \times H$	$P \subseteq G \times H$	CI
Z_2	(1,)	Yes	Z_2	(1,)	Yes	$Z_2 \times Z_2$	(1, 0), (0, 1)	Yes
Z_2	(1,)	Yes	Z_3	(2,)	Yes	$Z_2 \times Z_3$	(1, 0), (0, 2)	Yes
Z_3	(1,)	Yes	Z_3	(1,)	Yes	$Z_3 \times Z_3$	(1, 0), (0, 1)	Yes
Z_3	(2,)	Yes	Z_3	(1,)	Yes	$Z_3 \times Z_3$	(0, 1), (2, 0)	Yes
Z_3	(1,), (2,)	Yes	Z_3	(1,)	Yes	$Z_3 \times Z_3$	(1, 0), (2, 0), (0, 1)	Yes
Z_2	(1,)	Yes	Z_4	(3,)	Yes	$Z_2 \times Z_4$	(1, 0), (0, 3)	Yes
Z_2	(1,)	Yes	Z_4	(1,), (2,)	Yes	$Z_2 \times Z_4$	(1, 0), (0, 2), (0, 1)	Yes
Z_2	(1,)	Yes	Z_5	(2,)	Yes	$Z_2 \times Z_5$	(1, 0), (0, 2)	Yes
Z_2	(1,)	Yes	Z_5	(2,), (3,)	Yes	$Z_2 \times Z_5$	(1, 0), (0, 2), (0, 3)	Yes
Z_2	(1,)	Yes	Z_5	(1,), (2,), (3,)	Yes	$Z_2 \times Z_5$	(1, 0), (0, 2), (0, 3), (0, 1)	Yes

Az előbbi mintában látható $CI \square CI \to CI$ esetek után felmerülhet a kérdés, hogy csak akkor kaphatunk-e CI gráfot, ha a szorzat mind a két tagja CI volt-e. Tehát lehetséges-e, hogy egy nem-CI Cayley gráf és egy tetszőleges Cayley gráf Cartesian gráf szorzata CI gráf lesz?

Mivel a legkisebb ciklikus csoport, aminek létezik részhalmaza, hogy a Cayley gráfjuk nem-CI a Z_8, így sajnos a futási idő miatt ezt nem tudjuk ellenőrizni a programmal. Helyette megvizsgáljuk a $Z_8 \times Z_3$ szorzatot tételek által. Először is szükségünk lesz az alábbi két tételre.

7.6.4. Tétel. *[4] Egy n-ed rendű Z_n^+ additív ciklikus csoportnak az automorfizmus csoportja az n relatív prímeivel való szorzásoknak felel meg* $\mod n$.

7.6.5. Tétel. *[5] Legyen Z_n és Z_m ciklikus csoportok. Ha $(n, m) = 1$ relatív prímek, akkor*

$$Aut(Z_n \times Z_m) = Aut(Z_n) \times Aut(Z_m).$$

Először felhasználva a 7.6.5-s tételt azt kapjuk, hogy:

$$Aut(Z_8 \times Z_3) = Aut(Z_8) \times Aut(Z_3).$$

Utána felhasználva a 7.6.4-s tételt, azt kapjuk hogy az első tagra mod 8 mellett a $\{\cdot 1, \cdot 3, \cdot 5, \cdot 7\}$ hat , még a második tagra a mod 3 mellett a $\{\cdot 1, \cdot 2\}$ hat. Ezekből adódóan meghatározhatjuk a teljes automorfizmus csoportot:

(1,1)	(3,1)	(5,1)	(7,1)	(1,2)	(3,2)	(5,2)	(7,2)
(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)	(0,0):(0,0)
(1,0):(1,0)	(1,0):(3,0)	(1,0):(5,0)	(1,0):(7,0)	(1,0):(1,0)	(1,0):(3,0)	(1,0):(5,0)	(1,0):(7,0)
(2,0):(2,0)	(2,0):(6,0)	(2,0):(2,0)	(2,0):(6,0)	(2,0):(2,0)	(2,0):(6,0)	(2,0):(2,0)	(2,0):(6,0)
(3,0):(3,0)	(3,0):(1,0)	(3,0):(7,0)	(3,0):(5,0)	(3,0):(3,0)	(3,0):(1,0)	(3,0):(7,0)	(3,0):(5,0)
(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)	(4,0):(4,0)
(5,0):(5,0)	(5,0):(7,0)	(5,0):(1,0)	(5,0):(3,0)	(5,0):(5,0)	(5,0):(7,0)	(5,0):(1,0)	(5,0):(3,0)
(6,0):(6,0)	(6,0):(2,0)	(6,0):(6,0)	(6,0):(2,0)	(6,0):(6,0)	(6,0):(2,0)	(6,0):(6,0)	(6,0):(2,0)
(7,0):(7,0)	(7,0):(5,0)	(7,0):(3,0)	(7,0):(1,0)	(7,0):(7,0)	(7,0):(5,0)	(7,0):(3,0)	(7,0):(1,0)
(0,1):(0,1)	(0,1):(0,1)	(0,1):(0,1)	(0,1):(0,1)	(0,1):(0,2)	(0,1):(0,2)	(0,1):(0,2)	(0,1):(0,2)
(1,1):(1,1)	(1,1):(3,1)	(1,1):(5,1)	(1,1):(7,1)	(1,1):(1,2)	(1,1):(3,2)	(1,1):(5,2)	(1,1):(7,2)
(2,1):(2,1)	(2,1):(6,1)	(2,1):(2,1)	(2,1):(6,1)	(2,1):(2,2)	(2,1):(6,2)	(2,1):(2,2)	(2,1):(6,2)
(3,1):(3,1)	(3,1):(1,1)	(3,1):(7,1)	(3,1):(5,1)	(3,1):(3,2)	(3,1):(1,2)	(3,1):(7,2)	(3,1):(5,2)
(4,1):(4,1)	(4,1):(4,1)	(4,1):(4,1)	(4,1):(4,1)	(4,1):(4,2)	(4,1):(4,2)	(4,1):(4,2)	(4,1):(4,2)
(5,1):(5,1)	(5,1):(7,1)	(5,1):(1,1)	(5,1):(3,1)	(5,1):(5,2)	(5,1):(7,2)	(5,1):(1,2)	(5,1):(3,2)
(6,1):(6,1)	(6,1):(2,1)	(6,1):(6,1)	(6,1):(2,1)	(6,1):(6,2)	(6,1):(2,2)	(6,1):(6,2)	(6,1):(2,2)
(7,1):(7,1)	(7,1):(5,1)	(7,1):(3,1)	(7,1):(1,1)	(7,1):(7,2)	(7,1):(5,2)	(7,1):(3,2)	(7,1):(1,2)
(0,2):(0,2)	(0,2):(0,2)	(0,2):(0,2)	(0,2):(0,2)	(0,2):(0,1)	(0,2):(0,1)	(0,2):(0,1)	(0,2):(0,1)
(1,2):(1,2)	(1,2):(3,2)	(1,2):(5,2)	(1,2):(7,2)	(1,2):(1,1)	(1,2):(3,1)	(1,2):(5,1)	(1,2):(7,1)
(2,2):(2,2)	(2,2):(6,2)	(2,2):(2,2)	(2,2):(6,2)	(2,2):(2,1)	(2,2):(6,1)	(2,2):(2,1)	(2,2):(6,1)
(3,2):(3,2)	(3,2):(1,2)	(3,2):(7,2)	(3,2):(5,2)	(3,2):(3,1)	(3,2):(1,1)	(3,2):(7,1)	(3,2):(5,1)
(4,2):(4,2)	(4,2):(4,2)	(4,2):(4,2)	(4,2):(4,2)	(4,2):(4,1)	(4,2):(4,1)	(4,2):(4,1)	(4,2):(4,1)
(5,2):(5,2)	(5,2):(7,2)	(5,2):(1,2)	(5,2):(3,2)	(5,2):(5,1)	(5,2):(7,1)	(5,2):(5,1)	(5,2):(3,1)
(6,2):(6,2)	(6,2):(2,2)	(6,2):(6,2)	(6,2):(2,2)	(6,2):(6,1)	(6,2):(2,1)	(6,2):(6,1)	(6,2):(2,1)
(7,2):(7,2)	(7,2):(5,2)	(7,2):(3,2)	(7,2):(1,2)	(7,2):(7,1)	(7,2):(5,1)	(7,2):(3,1)	(7,2):(1,1)

Tehát az automorfizmus csoport számolásigénye nagyban lecsökkenthető, ha a csoportok rendjei relatív prímek.

8. Függelék

Az alábbi fejezet tartalmazza a CI-ellenörző fájljaiban található programkódokat.

8.1. group.py

```
import typing
from typing import Dict
from products import Inner_product
import itertools
from itertools import permutations, combinations
import networkx as nx
import matplotlib.pyplot as plt

class Group:

    def __init__(self, base_set:set[tuple],
                 product:Inner_product,
                 base_generator = None):
        self.base_set = base_set
        self.product = product
        self._base_generator = base_generator

    def get_base_generator(self) -> set[tuple]:
        """The function returns the given generators of a
        group, if it is not None."""
        if self._base_generator is not None:
            return self._base_generator
        else:
            raise Exception("Base generator is None")

    def execute_inner_product(self, list1:tuple, list2:tuple):
        """The function takes in two tuples and calculates
        their product assoicated with their group."""
        result = self.product.group_product(list1, list2)
        return result


    def _is_homomorphism(self,
                         mapping:Dict[tuple, tuple]) -> bool:
        """The function takes in a dictionary containing
        tuples and checks if it keeps up the hommomorphism
        property, returning True or False based on the
        calculation."""
```

```
        for a in self.base_set:
            for b in self.base_set:
                m1 = mapping[self.execute_inner_product(a, b)]
                m2 = self.execute_inner_product(mapping[a],
                                                mapping[b])
                if m1 != m2:
                    return False
        return True

    def _make_automorphism_group(self)  -> list[Dict[
                                                tuple, tuple]]:
        """The function calculates the automorphism group
        of the given group."""
        automorphisms = []
        elements = list(self.base_set)
        for perm in permutations(elements):
            # Create a mapping based on the permutation
            mapping = {elements[i]: perm[i]
                       for i in range(len(elements))}
            # Check if it is a homomorphism and bijection
            if self._is_homomorphism(mapping):
                automorphisms.append(mapping)
        return automorphisms

    def generate_cayley_graph(self,
                          generators:set[tuple]) -> nx.DiGraph:
        """The function takes in a set, then creates a Cayley
        graph based on product assoicated with the group."""
        G = nx.DiGraph()
        G.add_nodes_from(self.base_set)

        for set_element in self.base_set:
            for gen in generators:
                target = self.execute_inner_product(
                                            set_element, gen)
                if target in self.base_set:
                    G.add_edge(set_element, target,
                               label=str(gen))
        return G


    def _make_isomorph_graphs(self, generator:set
                              [tuple]) -> list[
                              list[nx.DiGraph | dict]]:
        """This function takes in a set, then sorts out only
```

```
84          the isomorphic graphs and their isomorphisms."""
85          isomorphic_graphs = []
86          graph = self.generate_cayley_graph(generator)
87          potential_graphs = \
88              self._generate_potential_isomorph_graphs(generator)
89          for potential_graph in potential_graphs:
90              gm = nx.isomorphism.DiGraphMatcher(graph,
91                                                 potential_graph[0])
92              if gm.is_isomorphic():
93                  isomorphic_graphs.append([potential_graph,
94                                            gm.mapping])
95          return isomorphic_graphs
96
97
98
99      def _generate_potential_isomorph_graphs(self, generator:
100                                              tuple) -> list[
101                                                  nx.DiGraph]:
102         """This function takes in a set, then gets all the
103         graphs that have the potential to be isomorphic to
104         the given graph."""
105         potential_generators = list(combinations(
106                                     self.base_set, len(generator)))
107         list_of_potential_graphs = []
108         for potential_generator in potential_generators:
109             list_of_potential_graphs.append((
110                 self.generate_cayley_graph(potential_generator),
111                  potential_generator))
112         return list_of_potential_graphs
113
114
115     def check_ci_property(self, generator:set[tuple]) -> bool:
116         """This function takes in a set, then checks for the
117         CI property."""
118         aut_group = self._make_automorphism_group()
119         isomorphic_graphs = self._make_isomorph_graphs(generator)
120         for isom in isomorphic_graphs:
121             flag = False
122             for aut in aut_group:
123                 mapped_isom_gen = set()
124                 for isom_gen in isom[0][1]:
125                     mapped_isom_gen.add(aut.get(isom_gen))
126                 if set(generator) ==  mapped_isom_gen:
127                     flag = True
128             if flag == False:
```

```
                return False
        return True
```

8.2. products.py

```
from abc import ABC, abstractmethod
import utils


class Inner_product(ABC):
    @abstractmethod
    def group_product(list1:tuple, list2:tuple)-> tuple:
        pass


class Cyclic_inner_product(Inner_product):
    def __init__(self, group_sizes:tuple):
        self.group_size = group_sizes


class Group_cyclic_1d(Cyclic_inner_product):
    def group_product(self, list1:tuple, list2:tuple)-> tuple:
        c = (list1[0] + list2[0]) % self.group_size[0]
        return (c,)


class Group_cyclic_2d(Cyclic_inner_product):
    def group_product(self, list1:tuple, list2:tuple) -> tuple:
        d = ((list1[0] + list2[0]) % self.group_size[0],
             (list1[1] + list2[1]) % self.group_size[1])
        return d
```

8.3. group_products.py

```
from abc import ABC, abstractmethod
from group import Group
from products import Group_cyclic_2d
from utils import nested_tupple_reductor


class Group_product(ABC):
    @abstractmethod
    def product_btw_groups(group1:Group, group2:Group)-> Group:
        pass

class Cartesian_product_between_groups(Group_product):
    @staticmethod
    def product_btw_groups(group1:Group, group2:Group)-> Group:

        group_3 = {(g1, g2) for g1 in group1.base_set for g2 in
                   group2.base_set}
        group_3 = nested_tupple_reductor(group_3)
        gen_3 = {(g1, (0,)) for g1 in
                 group1.get_base_generator()} \
                | {((0,), g2) for g2 in
                   group2.get_base_generator()}
        gen_3 = nested_tupple_reductor(gen_3)
        group_3_size = (len(group1.base_set),
                        len(group2.base_set))
        group_cartesian_product_2d = Group_cyclic_2d(
                                                group_3_size)
        result = Group(group_3, group_cartesian_product_2d,
                       gen_3)
        return result

class Tensor_product_between_groups(Group_product):
    @staticmethod
    def product_btw_groups(group1:Group, group2:Group)-> Group:

        group_3 = {(g1, g2) for g1 in group1.base_set for g2
                   in group2.base_set}
        group_3 = nested_tupple_reductor(group_3)
        gen_3 = {(g1, g2) for g1 in group1.get_base_generator()
                 for g2 in group2.get_base_generator()}
        gen_3 = nested_tupple_reductor(gen_3)
        group_3_size = (len(group1.base_set),
                        len(group2.base_set))
        group_cartesian_product_2d = Group_cyclic_2d(
```

```
                                                    group_3_size)
        result = Group(group_3, group_cartesian_product_2d,
                       gen_3)
        return result

class Lexicographical_product_between_groups(Group_product):
    @staticmethod
    def product_btw_groups(group1:Group,
                           group2:Group)-> Group:

        group_3 = {(g1, g2) for g1 in group1.base_set for g2
                   in group2.base_set}
        group_3 = nested_tupple_reductor(group_3)
        gen_3 =  {((0,), g2) for g2 in
                  group2.get_base_generator()} \
                | {(g1, g2) for g1 in
                   group1.get_base_generator()
                   for g2 in group2.base_set}
        gen_3 = nested_tupple_reductor(gen_3)
        group_3_size = (len(group1.base_set),
                        len(group2.base_set))
        group_cartesian_product_2d = Group_cyclic_2d(
                                                    group_3_size)
        result = Group(group_3, group_cartesian_product_2d,
                       gen_3)
        return result

class Strong_product_between_groups(Group_product):
    @staticmethod
    def product_btw_groups(group1:Group, group2:Group)-> Group:

        group_3 = {(g1, g2) for g1 in group1.base_set for g2
                   in group2.base_set}
        group_3 = nested_tupple_reductor(group_3)
        gen_3 = {(g1, (0,)) for g1 in
                 group1.get_base_generator()} \
                | {((0,), g2) for g2 in
                   group2.get_base_generator()} \
                | {(g1, g2) for g1 in
                   group1.get_base_generator()
                   for g2 in group2.get_base_generator()}
        gen_3 = nested_tupple_reductor(gen_3)
        group_3_size = (len(group1.base_set),
                        len(group2.base_set))
        group_cartesian_product_2d = Group_cyclic_2d(
```

```
                                                    group_3_size)
        result = Group(group_3, group_cartesian_product_2d,
                       gen_3)
        return result
```

8.4. utils.py

```
def nested_tupple_reductor(inp:set[tuple
                           [tuple,tuple]]) -> set[tuple]:
    """The function takes in a tupple of tupples
    and converts it into a set of tupples."""
    result = set()
    for element in inp:
        a = element[0][0]
        b = element[1][0]
        tmp = (a, b)
        result.add(tmp)
    return result
```

8.5. CI_property_tester.py

```
import networkx as nx
import random
import matplotlib.pyplot as plt
import itertools
from itertools import permutations, product, combinations

from group import Group
from group_products import Cartesian_product_between_groups
from group_products import Lexicographical_product_between_groups
from group_products import Strong_product_between_groups
from group_products import Tensor_product_between_groups
from products import Group_cyclic_1d

def graph_product_calculator(G, H):
        return nx.cartesian_product(G, H)

def generate_cyclic_group(size:int):
    """Based on the given size generates a cyclic group,
    without the generator."""
    group_set = generate_base_set(size)
    product_cyclic = Group_cyclic_1d(tuple([size]))
    group = Group(group_set, product_cyclic)
    return group

```

```
def generate_base_set(size):
    """Based on the given size generates the base set of
    the group."""
    group_set = { tuple([x]) for x in range(size) }
    return group_set

def generate_random_generator(size):
    """Based on the given size generates a random generator."""
    base_set = generate_base_set(size)
    num_generators = random.randint(1, size)
    generators = random.sample(sorted(base_set), num_generators)
    return generators

def yes_not_CI_getter(yes_not_CI):
    if len(yes_not_CI) == 0:
        return None
    else:
        return yes_not_CI

def anw_transformer(solution):
    if solution is True:
        return "CI"
    else:
        return "not-CI"



def main():
    k = 9
    cyclic_group_k = generate_cyclic_group(k)
    random_generator_k = generate_random_generator(k)
    cyclic_group_k._base_generator = random_generator_k
    solution = cyclic_group_k.check_ci_property(
                            cyclic_group_k.get_base_generator())

    print(solution)
    if solution is True:
        answ = "CI."
    elif solution is False:
        answ = "non-CI."
    print("For the cyclic group with order", k,
            "and the subset of", random_generator_k,
            ", the Cayley-graph is", answ)

    #Potential to draw the Cayley-graph:
```

```
70      """
71      G = cyclic_group_k.generate_cayley_graph(
72                                  cyclic_group_k.get_base_generator())
73      nx.draw(G, with_labels=True, node_color='lightblue',
74              node_size=500, font_size=15)
75      plt.show()
76      """
77
78
79  def main2():
80      k = 4
81      yes_CI = set()
82      not_CI = set()
83      cyclic_group_k = generate_cyclic_group(k)
84      for gen_size in range(0, len(cyclic_group_k.base_set) + 1):
85          for gen in combinations(cyclic_group_k.base_set,
86                                  gen_size):
87              cyclic_group_k._base_generator = gen
88              solution = cyclic_group_k.check_ci_property(
89                                  cyclic_group_k.get_base_generator())
90
91              if solution is True:
92                  yes_CI.add(gen)
93              elif solution is False:
94                  not_CI.add(gen)
95      print("The number of possible subgroups is",
96            2**len(cyclic_group_k.base_set), ". Out of these",
97            len(yes_CI),"are CI and", len(not_CI), "are not CI.")
98      print("The list of subsets which generate a \
99            CI Cayley-graph:", yes_not_CI_getter(yes_CI))
100     print("The list of subsets which does not generate a \
101           CI Cayley-graph:", yes_not_CI_getter(not_CI))
102
103
104 def main3():
105     #The first cyclic group (k).
106     k = 3
107     cyclic_group_k = generate_cyclic_group(k)
108     random_generator_k = generate_random_generator(k)
109     cyclic_group_k._base_generator = random_generator_k
110     solution_k = cyclic_group_k.check_ci_property(
111                                 cyclic_group_k.get_base_generator())
112
113
114     #The second cyclic group (m).
```

```
m = 4
cyclic_group_m = generate_cyclic_group(m)
random_generator_m = generate_random_generator(m)
cyclic_group_m._base_generator = random_generator_m
solution_m = cyclic_group_m.check_ci_property(
                        cyclic_group_m.get_base_generator())

#The group based on the Cartesian graph product of the two
# previous groups
product_group = Cartesian_product_between_groups\
        .product_btw_groups(cyclic_group_k, cyclic_group_m)
solution_product_graph = product_group.check_ci_property(
                        product_group.get_base_generator())

print("The Cartesian graph product of the",
      anw_transformer(solution_k),
      "Cayley-graph of the additive cyclic group of order",
      k, "with subgroup", random_generator_k, "and the",
      anw_transformer(solution_m),
      "Cayley-graph of the additive cyclic group of order",
      m, "with subgroup", random_generator_m, ", creates a",
      anw_transformer(solution_product_graph),
      "Cayley-graph with the underlying group",
      product_group.base_set, "and subset with",
      product_group.get_base_generator(),".")


#Potential to draw the Cayley-graphs:
"""
K = cyclic_group_k.generate_cayley_graph(
                        cyclic_group_k.get_base_generator())
nx.draw(K, with_labels=True, node_color='lightblue',
        node_size=500, font_size=15)
plt.show()

M = cyclic_group_m.generate_cayley_graph(
                        cyclic_group_m.get_base_generator())
nx.draw(M, with_labels=True, node_color='lightblue',
        node_size=500, font_size=15)
plt.show()

K_M = product_group.generate_cayley_graph(
                        product_group.get_base_generator())
nx.draw(K_M, with_labels=True, node_color='lightblue',
        node_size=500, font_size=15)
```

```
160     plt.show()
161     """
162
163
164
165 if __name__ == "__main__":
166     main()
167     #main2()
168     #main3()
```

Összefoglalás

A szakdolgozatban először bevezettük az izomorfia fogalmát gráfokra, majd egy példán keresztül beláttuk, hogy több izomorfia is lehet két gráf között. Definiáltuk a P és NP osztályt és mivel csak sejtések vannak P-be tartozására és NP-teljességre mutattunk két irányt, amikkel tovább tudjuk vizsgálni a bonyolultságát. Az elsőben a saját bonyolultsági osztályaként kezeltük, még a másodikban bevezettük az IP(2) osztályt.

Áttértünk a gráf izomorfia vizsgálatára algebrai eszközökkel, ahol bevezettük a Cayley gráfok fogalmát és beláttuk négy közvetlen következményét a definíciójának. Kimondtuk, hogy mikor tekinünk egy Γ gráfot egy (G, S) páros Cayley reprezentációjának. Egy példán keresztül megfigyeltük, hogy több Cayley reprezentáció is lehetséges, így definiáltuk, hogy mely Cayley reprezentációkat tekintjük ekvivalensnek. Az automorfizmus csoport és a reguláris reprezentáció definiálása után beláttuk, hogy egy gráf pontosan akkor Cayley gráfja egy csoportnak, ha a csoport automorfizmusa tartalmaz egy speciális részcsoportot. Példaként egy $K_{m,k}$ gráfon vizsgáltuk a (G, S) párosokat, melyekre Cayley reprezentáció.

Megvizsgáltuk, hogy mi történik hogyha ugyan azon a csoporton dolgozunk és bevezettük a CI-gráfokat. A CI-gráfok következményeként definiáltuk az $Aut(G, S)$ csoportot.

Kimondtunk és beláttunk egy CI-gráfokról szóló tételt, ami a CI-gráf eldöntési problémát átalakítja egy csak csoportelméleti problémára. Illetve megvizsgáltuk az utóbbi tételt a $DGRR$-ekre.

A normál részcsoport definiálása után bevezettük a normál Cayley gráfok fogalmát. Definiáltuk a ciklikus csoportokat és a direct productot-ot, majd megállapítottuk, hogy $\hat{Z_2^2} \triangleleft Aut(Q_2)$ és $\hat{Z}_4 \triangleleft Aut(Q_2)$. Viszont $\hat{Z_2^2} \triangleleft Aut(Z_2 \times Z_2)$ és $\hat{Z}_4 \not\triangleleft Aut(Z_2 \times Z_2)$, így megállapítottuk, hogy a csoporttól és a gráftól egyaránt függ a normál Cayley gráf tulajdonság.

Megnéztük az A.Ádám sejtésére prím n esetén adott bizonyítást.

Végül pedig bemutattunk egy moduláris programkódot, amely megállapítja egy Cayley gráfról, hogy rendelkezik-e a CI tulajdonsággal. Továbbá mutattunk 3 példát a kód felhasználására.

Hivatkozások

[1] L Bafai. Isomorphism problem for a class of point-symmetric structures. *Acta Mathematica Hungarica*, 29(3-4):329–336, 1977.

[2] David A Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51(2):61–66, 1994.

[3] DŹ Djoković. Isomorphism problem for a special class of graphs. *Acta Mathematica Hungarica*, 21(3-4):267–270, 1970.

[4] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2003.

[5] Szakács István Erdős Gábor, Kiss Emil. Algebra 3 jegyzet, 2018.

[6] Scott Fortin. The graph isomorphism problem. 1996.

[7] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, pages 203–225. 2019.

[8] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[9] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[10] Cai Heng Li. On isomorphisms of finite cayley graphs—a survey. *Discrete Mathematics*, 256(1):301–334, 2002.

[11] Marvin Marcus and Henryk Minc. *A survey of matrix theory and matrix inequalities*, volume 14. Courier Corporation, 1992.

[12] Rudolf Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.

[13] Gert Sabidussi. Vertex-transitive graphs. *Monatshefte für Mathematik*, 68(5):426–438, 1964.

[14] Ming-Yao Xu. Automorphism groups and isomorphisms of cayley digraphs. *Discrete Mathematics*, 182(1-3):309–319, 1998.

[15] M. Ádám. Research problem 2–10. *Journal of Combinatorial Theory*, 2:393, 1967.