УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ



Ђорђе Б. Јовановић

ИМПЛЕМЕНТАЦИЈА АЛГОРИТАМА ЗА НАБРАЈАЊЕ НЕИЗОМОРФНИХ ОБЈЕКАТА У ПРОЈЕКТИВНОЈ ГЕОМЕТРИЈИ

мастер рад

Београд, 2026.

Ментор:

др Филип Марић, редовни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Владица Андрејић, ванредни професор
Универзитет у Београду, Математички факултет

др Милан Банковић, доцент
Универзитет у Београду, Математички факултет

Датум одбране:

Наслов мастер рада: Имплементација алгоритама за набрајање неизоморфних објеката у пројективној геометрији

Резиме: Овај рад се бави проблемом класификације неизоморфних комбинаторних објеката у коначним пројективним просторима, са посебним освртом на паковања у $\mathrm{PG}(3, q)$. Описана је примена Шмалцовог алгоритма, у облику који је Бетен користио за класификовање паковања, као и реализација тог поступка у програмском језику C++. Алгоритам се заснива на постепеном проширивању парцијалних паковања кроз фазе *Lift* и *Classify*, уз коришћење групе колинеација и конструктивног препознавања орбита.

Имплементација је испитана на просторима $\mathrm{PG}(3, 2)$ и $\mathrm{PG}(3, 3)$. За $\mathrm{PG}(3, 2)$ извршена је потпуна класификација и добијена су два неизоморфна паковања, у складу са познатим резултатима. У случају $\mathrm{PG}(3, 3)$ класификација је спроведена до парцијалних паковања величине пет. Добијени бројеви за величине један, два и три поклапају се са резултатима из литературе. За величине четири и пет добијено је 219 066, односно 15 884 059 класа изоморфизма.

Добијени резултати омогућавају проверу имплементације на познатим примерима и показују практичне могућности, али и ограничења, примене Шмалцовог приступа на класификацију комбинаторних структура у коначној пројективној геометрији.

Кључне речи: коначни пројективни простори, паковања, парцијална паковања, Шмалцов алгоритам, колинеације, класификација неизоморфних објеката

Садржај

1 Увод **1**

1.1 Теорија група 2

1.2 Пројективна геометрија 8

2 Шмалцов алгоритам **22**

2.1 Основни појмови 22

2.2 *Up*-орбите и *down*-орбите 23

2.3 Опис алгоритма 25

2.4 Примена на класификовање паковања 31

3 Имплементација **38**

3.1 Константе простора 38

3.2 Представљање и уређење објеката 39

3.3 Генерисање тачака 39

3.4 Генерисање правих 43

3.5 Генерисање покривача 45

3.6 Колинеације и њихово дејство 48

3.7 Класификовање парцијалних паковања 58

4 Експериментални резултати **78**

4.1 Простор $\mathrm{PG}(3,2)$ 78

4.2 Простор $\mathrm{PG}(3,3)$ 80

5 Закључак **83**

Библиографија **84**

Глава 1

Увод

Пројективни простори над коначним пољима представљају природно окружење за многе комбинаторне структуре. У овом раду посматрамо пре свега покриваче правих и њихова паковања у просторима $\mathrm{PG}(3, q)$. Покривач разлаже скуп тачака на међусобно дисјунктне праве, док паковање разлаже скуп свих правих на међусобно дисјунктне покриваче. Поред питања постојања таквих структура, важан проблем је њихова класификација до на изоморфизам.

Једну класичну мотивацију за овај проблем даје Киркманов проблем петнаест ученица, постављен 1850. године [1]. Потребно је распоредити петнаест ученица током седам дана у по пет редова од по три ученице тако да се сваки пар ученица нађе у истом реду тачно једном. У пројективном простору $\mathrm{PG}(3, 2)$ тачке се могу посматрати као ученице, праве као тројке ученица, покривачи као дневни распореди, а паковање као целокупан седмодневни распоред. Ова аналогија даје једноставну конкретну интерпретацију објеката које ћемо касније формално дефинисати. Везу са Киркмановим проблемом прецизираћемо након одређивања основних параметара простора $\mathrm{PG}(3, 2)$.

Број могућих објеката брзо расте са параметром q, па непосредно набрајање и поређење свих конфигурација постаје непрактично. Природан приступ је зато да се искористи дејство групе колинеација и да се уместо свих објеката одреди по један представник сваке њихове орбите. У раду примењујемо Шмалцов алгоритам за класификацију комбинаторних објеката, у облику који је Бетен прилагодио класификовању паковања простора $\mathrm{PG}(3, 3)$ [9, 10, 1]. Алгоритам постепено проширује већ класификована парцијална паковања и састоји се од фаза *Lift* и *Classify*, уз чување стабилизатора и података за конструктивно препознавање орбита.

Главни циљ рада је имплементација овог поступка у програмском језику C++ и анализа његовог рада на просторима $\mathrm{PG}(3,2)$ и $\mathrm{PG}(3,3)$. Случај $\mathrm{PG}(3,2)$ довољно је мали да се класификација спроведе до потпуних паковања, што омогућава проверу имплементације на познатом резултату. Простор $\mathrm{PG}(3,3)$ већ показује изражену комбинаторну експлозију. За парцијална паковања величина један, два и три добијају се резултати који се поклапају са познатим вредностима из литературе, а овом имплементацијом класификација је настављена до парцијалних паковања величине пет. При томе је већ на прелазу са величине четири на величину пет забележен изражен раст меморијске потрошње, па су даље меморијске оптимизације и наставак експерименталне класификације остављени за будући рад.

У наставку овог поглавља уводимо појмове из теорије група и пројективне геометрије који су потребни за формални опис проблема. Друго поглавље посвећено је Шмалцовом алгоритму, фазама *Lift* и *Classify*, конструктивном препознавању и њиховој примени на парцијална паковања. У трећем поглављу описујемо имплементацију, почев од генерисања тачака, правих и покривача, преко представљања колинеација, до класификације парцијалних паковања. У четвртом поглављу приказујемо експерименталне резултате, а у закључку су сумирани добијени резултати и могући правци даљег рада.

1.1 Теорија група

Групе, подгрупе и количничке групе

Дефиниција 1. Нека је на скупу X дефинисана бинарна операција $\circ\colon X \times X \to X$. Тада кажемо да је структура $(X, \circ)$ *група* уколико важе:

- ✓ асоцијативност $(\forall x, y, z \in X)\,(x \circ y) \circ z = x \circ (y \circ z)$,
- ✓ постојање неутралног елемента $(\exists e \in X)(\forall x \in X)\, e \circ x = x \circ e = x$,
- ✓ постојање инверзних елемената $(\forall x \in X)(\exists x' \in X)\, x \circ x' = x' \circ x = e$.

Дефиниција 2. Групу $(X, \circ)$ у којој важи комутативност

$$(\forall x, y \in X)\, x \circ y = y \circ x$$

називамо *Абеловом групом*.

Дефиниција 3. Нека су $(G, \circ)$ и $(H, \star)$ две групе. Ако важи

$$H \subseteq G \quad \text{и} \quad x \star y = x \circ y \text{ за све } x, y \in H$$

онда кажемо да је $(H, \star)$ *подгрупа* групе $(G, \circ)$ и означавамо са $H \leqslant G$.

Дефиниција 4. Нека је H подгрупа групе $(G, \circ)$ и $g \in G$ произвољан елемент. *Леви косет* одређен елементом g и подгрупом H дефинише се као скуп

$$gH = \{\, g \circ h \mid h \in H \,\}.$$

Пре наредне теореме уведимо потребне ознаке. Скуп свих левих косета одређених подгрупом H групе G означаваћемо са G/H. Број елемената скупа X означаваћемо са $|X|$.

Теорема 1. *(Лагранж)* Нека је G коначна група и H њена подгрупа. Тада је број левих косета одређених подгрупом H у G једнак $|G/H| = \dfrac{|G|}{|H|}$.

Доказ. Може се природно увести релација еквиваленције на скупу G ($g_1 \sim g_2 \Leftrightarrow g_1 \in g_2H$) чије су класе еквиваленције управо леви косети ($G/\sim \; = G/H$). Према томе, леви косети разлажу групу G на дисјунктне подскупове.

$$G = \bigsqcup_{gH \in G/H} gH$$

Закључујемо да је број елемената групе G једнак збиру бројева елемената свих различитих левих косета $|G| = \sum_{gH \in G/H} |gH|$.

Како између скупова H и gH постоји бијекција ($h \mapsto g \circ h$) онда та два скупа имају исти број елемената.

$$|gH| = |H|$$

Дакле, леви косети gH разлажу G и сваки има по $|H|$ елемената, па следи да је број левих косета $\dfrac{|G|}{|H|}$. □

Дефиниција 5. Ако је H подгрупа групе $(G, \circ)$ и ако за све $g \in G$ и $h \in H$ важи

$$g \circ h \circ g^{-1} \in H$$

онда кажемо да је H *нормална* подгрупа групе G и пишемо $H \triangleleft G$.

Дефиниција 6. Нека је H нормална подгрупа групе $(G, \circ)$. Ако је на скупу G/H дефинисана операција $\cdot$ на следећи начин

$$g_1H \cdot g_2H = (g_1 \circ g_2)H$$

онда структуру $(G/H, \cdot)$ називамо *количничком групом*.

Операција је добро дефинисана јер не зависи од избора представника косета, тј. за друге $g_1' \in g_1H$ и $g_2' \in g_2H$ добија се исти резултат примене операције $g_1'H \cdot g_2'H = g_1H \cdot g_2H$, односно $(g_1' \circ g_2')H = (g_1 \circ g_2)H$.

Из $g_1' \in g_1H$ следи $g_1' = g_1 \circ h_1$ за неко $h_1 \in H$. На исти начин се закључује и $g_2' = g_2 \circ h_2$ за неко $h_2 \in H$. Онда имамо да је $g_1' \circ g_2' = g_1 \circ h_1 \circ g_2 \circ h_2 = g_1 \circ g_2 \circ \underbrace{g_2^{-1} \circ h_1 \circ \left(g_2^{-1}\right)^{-1}}_{\in H} \circ h_2 \in (g_1 \circ g_2)H$, тј. због нормалности подгрупе важи $(g_1' \circ g_2')H \subseteq (g_1 \circ g_2)H$. Аналогно се показује и супротна инклузија.

Лако се види да $(G/H, \cdot)$ заиста јесте група (неутрал је H, а инверз елемента gH је $g^{-1}H$).

Дејства група, орбите и стабилизатори

Дефиниција 7. Нека је $(G, \star)$ група са неутралом e и X непразан скуп. Под *дејством групе* G на скуп X подразумевамо функцију $\Theta\colon X \times G \to X$ (при чему ћемо, због једноставности, уместо $\Theta(x, g)$ користити степену нотацију x^g), за коју важи:

- ✓ $x^e = x$, за све $x \in X$,
- ✓ $(x^g)^h = x^{g \star h}$ за све $x \in X$ и $g, h \in G$.

Ако група G дејствује на скупове $X_1, \ldots, X_n$, ова дејства одређују дејство на Декартов производ $X_1 \times \cdots \times X_n$, дато за $x_i \in X_i$ са

$$(x_1, \ldots, x_n)^g = (x_1^g, \ldots, x_n^g).$$

Када су сви скупови $X_1, \ldots, X_n$ једнаки скупу X, за њихов Декартов производ такође користимо степену нотацију и пишемо X^n.

Дејство групе G на скуп X природно проширујемо на његов партитивни скуп $\mathcal{P}(X)$. За $S \in \mathcal{P}(X)$ и $g \in G$ дефинишемо

$$S^g = \{x^g \mid x \in S\}.$$

Како је $x^g \in X$ за свако $x \in X$, важи $S^g \in \mathcal{P}(X)$. Такође је $S^e = \{x^e \mid x \in S\} = S$ и $(S^g)^h = \{(x^g)^h \mid x \in S\} = \{x^{g\star h} \mid x \in S\} = S^{g\star h}$, па је овим заиста дефинисано дејство групе G на скуп $\mathcal{P}(X)$. Понављањем истог поступка добијамо дејство и на скуп[1] $\mathcal{P}^n(X)$ за свако $n \geqslant 0$.

На пример, ако група G дејствује на скуп тачака простора тако да је слика сваке праве поново права, описано дејство на партитивном скупу можемо ограничити на скуп правих (посматраних као скупова тачака). Произвољно дејство на скуп тачака не мора да испуњава овај услов.

У применама које ћемо посматрати, елементи групе G биће бијекције $g\colon X \to X$, а дејство ће бити задато њиховом применом, па ћемо писати $x^g = g(x)$. Да би за ову степену нотацију важило $(x^g)^h = x^{g\star h}$, операцију $\star$ дефинишемо обрнутим редоследом композиције функција:

$$g \star h = h \circ g.$$

У записима који се односе на дејства група подразумеваћемо операцију $\star$ и писати gh уместо $g \star h$.

Степену нотацију користимо и при записивању орбите.

Дефиниција 8. Нека група G дејствује на скуп X. *Орбита* елемента $x \in X$ је скуп $x^G = \{x^g \mid g \in G\}$ свих његових слика под дејством групе.

На основу дејства групе G на скуп X, може се на скупу X природно увести релација еквиваленције ($x \sim y \Leftrightarrow x \in y^G$) чије су класе еквиваленције управо орбите (које називамо и G-орбите), које разлажу скуп X.

Дакле, степену нотацију, осим за уобичајено степеновање бројева, користимо и за Декартове производе, за дејства група на елементе скупова, на објекте изграђене од тих елемената, на n-торке таквих објеката, као и за орбите. Из контекста ће бити јасно на шта се мисли.

Дефиниција 9. Нека група G дејствује на скуп X. *Стабилизатор* елемента $x \in X$ је скуп

$$G_x = \{g \in G \mid x^g = x\}.$$

Лако се показује да је G_x подгрупа групе G.

[1] Рекурзивно дефинишемо $\mathcal{P}^{n+1}(X) = \mathcal{P}(\mathcal{P}^n(X))$ и $\mathcal{P}^0(X) = X$.

Коначна поља

Дефиниција 10. Нека је на скупу X, поред бинарне операције $\circ$, дефинисана и бинарна операција $\star$: $X \times X \to X$. Структуру $(X, \circ, \star)$ називамо *пољем* ако:

- ✓ $(X, \circ)$ је Абелова група (са неутралом e),
- ✓ $(X \setminus \{e\}, \star|_{(X\setminus\{e\})^2})$ је Абелова група,
- ✓ за све $x, y, z \in X$ важи дистрибутивност операције $\star$ према $\circ$

$$\text{слева } x \star (y \circ z) = (x \star y) \circ (x \star z) \text{ и}$$

$$\text{здесна } (x \circ y) \star z = (x \star z) \circ (y \star z).$$

Тврђење 1. Ако је p прост број и $\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$, а операције $+$ и $\cdot$ су сабирање и множење по модулу p, онда је $(\mathbb{F}_p, +, \cdot)$ поље.

Доказ. Лако се показује скоро све (затворености операција, асоцијативности, комутативности, дистрибутивности, да је за $+$ неутрал 0 а инверз од x је $-x \equiv_p p - x$, да је неутрал за $\cdot$ једнак 1). Докажимо индукцијом да и за операцију $\cdot$ сваки елемент из $\mathbb{F}_p \setminus \{0\}$ има инверз.

1° Инверз елемента $n = 1$ је $1^{-1} = 1$ јер је $1 \cdot 1 = 1$.

2° Претпоставимо да постоји инверз сваког елемента мањег од $n \in \mathbb{F}_p \setminus \{0\}$. Докажимо да онда постоји и инверз елемента $n > 1$.
Нека се при дељењу p са n добија остатак r. Тада важи

$$p = \left\lfloor \frac{p}{n} \right\rfloor \cdot n + r, \qquad 0 < r < n.$$

Важи $r \neq 0$, јер би из $r = 0$ следило $n \mid p$, што није могуће за $1 < n < p$ и прост број p. Одатле применом операција по модулу p добијамо

$$r = p - \left\lfloor \frac{p}{n} \right\rfloor \cdot n \equiv_p - \left\lfloor \frac{p}{n} \right\rfloor \cdot n \equiv_p \left(p - \left\lfloor \frac{p}{n} \right\rfloor \right) \cdot n.$$

Како је $0 < r < n$, по индуктивној хипотези знамо да постоји инверз r^{-1}, па по модулу p важи $1 = r \cdot r^{-1} = \left(p - \left\lfloor \frac{p}{n} \right\rfloor\right) \cdot n \cdot r^{-1}$, тј.

$$n \cdot \left(p - \left\lfloor \frac{p}{n} \right\rfloor \right) \cdot r^{-1} = 1,$$

одакле закључујемо да постоји инверз и за n:

$$n^{-1} = \left(p - \left\lfloor \frac{p}{n} \right\rfloor \right) \cdot r^{-1}.$$

Из 1°, 2° и принципа потпуне индукције следи да за операцију $\cdot$ сваки елемент из $\mathbb{F}_p \setminus \{0\}$ има инверз. $\square$

У $\mathbb{F}_2 = \{0, 1\}$ имамо врло једноставне таблице инверза за сабирање и множење по модулу 2.

x	0	1
$-x$	0	1

x	0	1
x^{-1}	$\nexists$	1

Како је и 3 прост број, слично можемо и на скупу $\mathbb{F}_3 = \{0, 1, 2\}$ да направимо таблице инверза за сабирање и множење по модулу 3.

x	0	1	2
$-x$	0	2	1

x	0	1	2
x^{-1}	$\nexists$	1	2

Видели смо да постоје поља са простим бројем елемената. Показује се да коначно поље са q елемената постоји ако и само ако је $q = p^e$, где је p прост број, а e позитиван цео број. По дефиницији, поље мора да садржи бар два елемента (неутрале за обе операције), па ћемо (иако то већ следи из облика $q = p^e$) увек претпостављати да важи $q \geqslant 2$ када говоримо о пољима $\mathbb{F}_q$. С обзиром на то да ће у овом раду фокус бити на пољима малог реда (конкретно $q \in \{2, 3\}$), нећемо се детаљније бавити конструкцијом и особинама коначних поља реда p^e за $e > 1$.

Хомоморфизми и изоморфизми

Дефиниција 11. Нека су $\mathcal{R}_i \subseteq G^{n_i}$ и $\mathcal{S}_i \subseteq H^{n_i}$ произвољне релације и $(G, \mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k)$ и $(H, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k)$ произвољне структуре. Функција $f\colon G \to H$ је *хомоморфизам* уколико за све i и све $x_j \in G$ важи:

$$(x_1, \ldots, x_{n_i}) \in \mathcal{R}_i \quad \Rightarrow \quad (f(x_1), \ldots, f(x_{n_i})) \in \mathcal{S}_i.$$

У случају када су структуре групе (имају једну релацију која је заправо бинарна операција) говоримо о хомоморфизму група, а када су структуре поља онда је у питању хомоморфизам поља.

Тврђење 2. Ако је $f\colon G \to H$ хомоморфизам група $(G, \circ)$ и $(H, \star)$, онда важи $f(e_G) = e_H$ и $f\left(x^{-1}\right) = f(x)^{-1}$.

Доказ. Како је f хомоморфизам, важи $f(e_G) = f(e_G \circ e_G) = f(e_G) \star f(e_G)$. Када применимо $f(e_G)^{-1}$ операцијом $\star$ на обе стране добијене једнакости $f(e_G) = f(e_G) \star f(e_G)$ добијамо $e_H = f(e_G)$.

Такође је $e_H = f(e_G) = f(x \circ x^{-1}) = f(x) \star f(x^{-1})$. Применом $f(x)^{-1}$ операцијом $\star$ на обе стране добијене једнакости $e_H = f(x) \star f(x^{-1})$ добијамо $f(x)^{-1} = f(x^{-1})$. □

Дефиниција 12. Ако су $(G, \circ)$ и $(H, \star)$ две групе, e_H неутрални елемент у групи $(H, \star)$ и $f\colon G \to H$ хомоморфизам између те две групе, онда скуп

$$\operatorname{Ker} f = \{g \in G \mid f(g) = e_H\}$$

називамо *језгром* хомоморфизма f.

Лако се види да важи $\operatorname{Ker} f \triangleleft G$.

Дефиниција 13. Хомоморфизам који је бијекција и чија је инверзна функција такође хомоморфизам називамо *изоморфизмом*.

Када постоји изоморфизам између две структуре G и H кажемо да су оне изоморфне, што пишемо $G \cong H$. Лако се види да је $\cong$ релација еквиваленције. Показује се да су сва коначна поља са истим бројем елемената међусобно изоморфна.

Наведимо сада једну познату теорему која повезује хомоморфизме, језгро, количничке групе и слику функције $\operatorname{Im} f = \{f(x) \mid x \in \mathcal{D}(f)\}$ (где је са $\mathcal{D}(f)$ означен домен функције).

Теорема 2. *(Прва теорема о изоморфизму)* Ако је $f\colon G \to H$ хомоморфизам група, онда важи

$$G/\operatorname{Ker} f \cong \operatorname{Im} f.$$

Дефиниција 14. Изоморфизам структуре на саму себе називамо *аутоморфизмом*.

Кажемо да аутоморфизам представља симетрију алгебарске структуре.

1.2 Пројективна геометрија

За $n \in \mathbb{N}_0$, користићемо уобичајене операције у векторским просторима: множење вектора $x = (x_1, \ldots, x_{n+1}) \in \mathbb{F}_q^{n+1}$ скаларом $\lambda \in \mathbb{F}_q$

$$\lambda x = (\lambda x_1, \ldots, \lambda x_{n+1}),$$

сабирање два вектора $x = (x_1, \ldots, x_{n+1})$ и $y = (y_1, \ldots, y_{n+1})$

$$x + y = (x_1 + y_1, \ldots, x_{n+1} + y_{n+1}),$$

као и ознаку за нула вектор $\mathbf{0} = (\underbrace{0, 0, \ldots, 0}_{n+1}) \in \mathbb{F}_q^{n+1}$.

У наставку ћемо претпостављати да је $n \in \mathbb{N}_0$ и да је $q = p^e$ (за позитиван цео број e и прост број p), односно да је $\mathbb{F}_q$ поље.

Тачке

Дефиниција 15. Нека је $\sim$ релација на скупу $\mathbb{F}_q^{n+1} \setminus \{\mathbf{0}\}$ дефинисана на следећи начин

$$x \sim y \quad \overset{\text{деф}}{\iff} \quad (\exists \lambda \in \mathbb{F}_q \setminus \{0\})\, \lambda x = y.$$

Тада је $\sim$ релација еквиваленције. Количнички скуп $(\mathbb{F}_q^{n+1} \setminus \{\mathbf{0}\})/\sim$ називамо *пројективним простором димензије n над коначним пољем* $\mathbb{F}_q$ и означавамо са $\mathrm{PG}(n, q)$. Елементе пројективног простора називамо *тачкама*.

Дакле, цела једна класа еквиваленције $[x] = \{\lambda x \mid \lambda \in \mathbb{F}_q \setminus \{0\}\}$ јесте једна тачка простора $\mathrm{PG}(n, q)$, коју представља било који елемент тог скупа. С обзиром на то да су $(n+1)$-торке из класе међусобно сразмерне, тачки пројективног простора придружујемо било коју $(n+1)$-торку њене класе, што некад означавамо и са $[x_1 : x_2 : \ldots : x_{n+1}]$ а називамо *хомогеним координатама* дате пројективне тачке.

На пример, простор $\mathrm{PG}(2, 2) = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$ садржи следећих 7 тачака: $x_0 = [(0, 0, 1)]$, $x_1 = [(0, 1, 0)]$, $x_2 = [(0, 1, 1)]$,
$x_3 = [(1, 0, 0)]$, $x_4 = [(1, 0, 1)]$, $x_5 = [(1, 1, 0)]$, $x_6 = [(1, 1, 1)]$.

Дефиниција 16. Нека је $[n]_q = \dfrac{q^n - 1}{q - 1}$ и $[n]_q! = \prod_{i=1}^{n} [i]_q$.
За $0 \leqslant k \leqslant n$ *Гаусов биномни коефицијент* дефинише се са

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \frac{[n]_q!}{[k]_q!\,[n-k]_q!} = \prod_{i=0}^{k-1} \frac{q^{n-i} - 1}{q^{k-i} - 1}.$$

Тврђење 3. Пројективни простор $\mathrm{PG}(n, q)$ има укупно $[n+1]_q$ тачака.

Доказ. Пошто сваки представник тачке има $n + 1$ координата, а $\mathbb{F}_q$ има q елемената, на свакој од тих $n + 1$ координата постоји по q избора, што даје

q^{n+1}, али како $(n+1)$-торка $\mathbf{0}$ не представља пројективну тачку, њу онда не би требало бројати, па добијамо $q^{n+1}-1$ могућности. Делимо са $q-1=|\mathbb{F}_q\setminus\{0\}|$, што је број фактора који не мењају тачку множењем, тј. то је број елемената класе еквиваленције. Ти елементи су сви међусобно различити зато што не можемо множењем $(n+1)$-торке са два различита фактора добити исту $(n+1)$-торку, јер $\lambda_1(x_1,\ldots,x_{n+1})=\lambda_2(x_1,\ldots,x_{n+1})$ значи да је $\lambda_1 x_i=\lambda_2 x_i$ за свако i, а пошто је $(x_1,\ldots,x_{n+1})\neq\mathbf{0}$ онда постоји неки индекс i такав да је $x_i\neq 0$, па следи да је $\lambda_1=\lambda_2$. □

Тако добијамо да $\mathrm{PG}(2,2)$ заиста садржи укупно $[2+1]_2=\frac{2^3-1}{2-1}=7$ тачака, а да је број тачака простора облика $\mathrm{PG}(3,q)$ једнак

$$[3+1]_q=\frac{q^4-1}{q-1}=\frac{(q^2-1)(q^2+1)}{q-1}=(q+1)(q^2+1).$$

Дакле, простор $\mathrm{PG}(3,2)$ има $[3+1]_2=\frac{2^4-1}{2-1}=(2+1)(2^2+1)=15$ тачака.

$x_0=[(0,0,0,1)]$ $x_1=[(0,0,1,0)]$ $x_2=[(0,0,1,1)]$ $x_3=[(0,1,0,0)]$ $x_4=[(0,1,0,1)]$
$x_5=[(0,1,1,0)]$ $x_6=[(0,1,1,1)]$ $x_7=[(1,0,0,0)]$ $x_8=[(1,0,0,1)]$ $x_9=[(1,0,1,0)]$
$x_{10}=[(1,0,1,1)]$ $x_{11}=[(1,1,0,0)]$ $x_{12}=[(1,1,0,1)]$ $x_{13}=[(1,1,1,0)]$ $x_{14}=[(1,1,1,1)]$

Слично, добијамо да $\mathrm{PG}(3,3)$ има $[3+1]_3=\frac{3^4-1}{3-1}=(3+1)(3^2+1)=40$ тачака.

$x_0=[(0,0,0,1)]=\{(0,0,0,1),(0,0,0,2)\}$ $x_1=[(0,0,1,0)]=\{(0,0,1,0),(0,0,2,0)\}$
$x_2=[(0,0,1,1)]=\{(0,0,1,1),(0,0,2,2)\}$ $x_3=[(0,0,1,2)]=\{(0,0,1,2),(0,0,2,1)\}$
$x_4=[(0,1,0,0)]=\{(0,1,0,0),(0,2,0,0)\}$ $x_5=[(0,1,0,1)]=\{(0,1,0,1),(0,2,0,2)\}$
$x_6=[(0,1,0,2)]=\{(0,1,0,2),(0,2,0,1)\}$ $x_7=[(0,1,1,0)]=\{(0,1,1,0),(0,2,2,0)\}$
$x_8=[(0,1,1,1)]=\{(0,1,1,1),(0,2,2,2)\}$ $x_9=[(0,1,1,2)]=\{(0,1,1,2),(0,2,2,1)\}$
$x_{10}=[(0,1,2,0)]=\{(0,1,2,0),(0,2,1,0)\}$ $x_{11}=[(0,1,2,1)]=\{(0,1,2,1),(0,2,1,2)\}$
$x_{12}=[(0,1,2,2)]=\{(0,1,2,2),(0,2,1,1)\}$ $x_{13}=[(1,0,0,0)]=\{(1,0,0,0),(2,0,0,0)\}$
$x_{14}=[(1,0,0,1)]=\{(1,0,0,1),(2,0,0,2)\}$ $x_{15}=[(1,0,0,2)]=\{(1,0,0,2),(2,0,0,1)\}$
$x_{16}=[(1,0,1,0)]=\{(1,0,1,0),(2,0,2,0)\}$ $x_{17}=[(1,0,1,1)]=\{(1,0,1,1),(2,0,2,2)\}$
$x_{18}=[(1,0,1,2)]=\{(1,0,1,2),(2,0,2,1)\}$ $x_{19}=[(1,0,2,0)]=\{(1,0,2,0),(2,0,1,0)\}$
$x_{20}=[(1,0,2,1)]=\{(1,0,2,1),(2,0,1,2)\}$ $x_{21}=[(1,0,2,2)]=\{(1,0,2,2),(2,0,1,1)\}$
$x_{22}=[(1,1,0,0)]=\{(1,1,0,0),(2,2,0,0)\}$ $x_{23}=[(1,1,0,1)]=\{(1,1,0,1),(2,2,0,2)\}$
$x_{24}=[(1,1,0,2)]=\{(1,1,0,2),(2,2,0,1)\}$ $x_{25}=[(1,1,1,0)]=\{(1,1,1,0),(2,2,2,0)\}$
$x_{26}=[(1,1,1,1)]=\{(1,1,1,1),(2,2,2,2)\}$ $x_{27}=[(1,1,1,2)]=\{(1,1,1,2),(2,2,2,1)\}$
$x_{28}=[(1,1,2,0)]=\{(1,1,2,0),(2,2,1,0)\}$ $x_{29}=[(1,1,2,1)]=\{(1,1,2,1),(2,2,1,2)\}$
$x_{30}=[(1,1,2,2)]=\{(1,1,2,2),(2,2,1,1)\}$ $x_{31}=[(1,2,0,0)]=\{(1,2,0,0),(2,1,0,0)\}$

$x_{32} = [(1,2,0,1)] = \{(1,2,0,1),(2,1,0,2)\}$ $x_{33} = [(1,2,0,2)] = \{(1,2,0,2),(2,1,0,1)\}$
$x_{34} = [(1,2,1,0)] = \{(1,2,1,0),(2,1,2,0)\}$ $x_{35} = [(1,2,1,1)] = \{(1,2,1,1),(2,1,2,2)\}$
$x_{36} = [(1,2,1,2)] = \{(1,2,1,2),(2,1,2,1)\}$ $x_{37} = [(1,2,2,0)] = \{(1,2,2,0),(2,1,1,0)\}$
$x_{38} = [(1,2,2,1)] = \{(1,2,2,1),(2,1,1,2)\}$ $x_{39} = [(1,2,2,2)] = \{(1,2,2,2),(2,1,1,1)\}$

Праве

Тачке простора $\mathrm{PG}(n,q)$ можемо поистоветити са потпросторима димензије 1 векторског простора $\mathbb{F}_q^{n+1}$. Аналогно можемо дефинисати праву уз претпоставку $n > 0$, јер простор $\mathrm{PG}(0,q)$ садржи само једну тачку.

Дефиниција 17. *Права* простора $\mathrm{PG}(n,q)$ је скуп свих пројективних тачака садржаних у потпростору димензије 2 векторског простора $\mathbb{F}_q^{n+1}$.

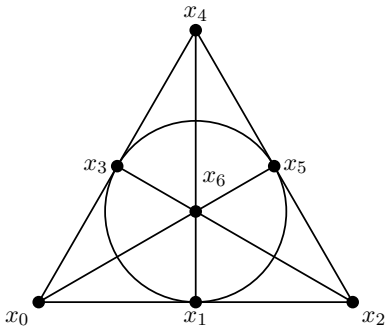
Такав потпростор садржи све векторе облика $\lambda A + \mu B$, где су A и B линеарно независни вектори у $\mathbb{F}_q^{n+1}$, а коефицијенти λ и μ припадају $\mathbb{F}_q$.

У случају $q = 2$ постоје три могућности за (λ, μ), јер скуп $\mathbb{F}_2 = \{0,1\}$ има два елемента, а комбинација $\lambda = \mu = 0$ даје нула вектор који не одређује пројективну тачку. Дакле, у случају $q = 2$, потпростор димензије 2 одређен линеарно независним векторима A и B од пројективних тачака садржи само $[A]$, $[B]$ и $[A+B]$.

У случају $q = 3$, права кроз тачке $[A]$ и $[B]$ садржи тачке $[A]$, $[B]$, $[A+B]$ и $[A+2B]$. Примећујемо да $[2A] = [A]$, $[2B] = [B]$, $[2A+B] = [2(A+2B)] = [A+2B]$ и $[2A+2B] = [2(A+B)] = [A+B]$ не представљају различите тачке од претходно наведене четири.

Користећи исте ознаке $x_0, \ldots, x_6$ за тачке простора $\mathrm{PG}(2,2)$ као раније, наводимо праве тог простора.

$p_0 = \{x_0, x_1, x_2\}$ $p_1 = \{x_0, x_3, x_4\}$
$p_2 = \{x_0, x_5, x_6\}$ $p_3 = \{x_1, x_3, x_5\}$
$p_4 = \{x_1, x_4, x_6\}$ $p_5 = \{x_2, x_3, x_6\}$
$p_6 = \{x_2, x_4, x_5\}$



Слика 1.1: Фанова раван $\mathrm{PG}(2,2)$

Модел простора $\mathrm{PG}(2,2)$ познат је као Фанова[2] раван.

[2]Gino Fano (1871–1952), италијански математичар

Тврђење 4. Права у $\mathrm{PG}(n,q)$ садржи $[2]_q = q+1$ тачака.

Доказ. Права кроз две различите тачке $[A]$ и $[B]$ састоји се од тачака $[\lambda A + \mu B]$, где је $(\lambda,\mu) \in \mathbb{F}_q^2 \setminus \{(0,0)\}$. За $\lambda = 0$ у питању је тачка $[B]$, а за фиксирано $\lambda \neq 0$ тачке праве су $[A + \lambda^{-1}\mu B]$, $\mu \in \mathbb{F}_q$ (укључена је 0 у овом случају за μ јер је $\lambda \neq 0$). Стога, тачке праве можемо описати скупом $\{[A+tB] \mid t \in \mathbb{F}_q\} \cup \{[B]\}$. Остаје да проверимо да је $[B]$ различита од свих тачака $[A+tB]$ и да различите вредности t дају различите тачке.

Тачку $[B]$ (различиту од $[A]$) не можемо добити ни за једно $t \in \mathbb{F}_q$. Из $[A+tB] = [B]$ би следило $A + tB = \lambda B$ за неко $\lambda \in \mathbb{F}_q$, односно $A = (\lambda - t)B$, што би заправо значило $[A] = [B]$.

Такође, знамо још и да свако $t \in \mathbb{F}_q$ даје различиту тачку, јер из $[A + t_1B] = [A + t_2B]$ следи $A + t_1B = \lambda(A + t_2B)$ за неко $\lambda \in \mathbb{F}_q$, тј. $A = (1-\lambda)^{-1}(\lambda t_2 - t_1)B$ за $\lambda \neq 1$ (када постоји инверз за $1-\lambda$ у пољу $\mathbb{F}_q$), што поново значи $[A] = [B]$. За $\lambda = 1$ се добија $(t_1 - t_2)B = \mathbf{0}$, а то за различите $t_1 \neq t_2$ даје $B = \mathbf{0}$, што је такође контрадикција. Према томе, права садржи $q+1 = [2]_q$ тачака. □

Из претходног тврђења следи да права у $\mathrm{PG}(n,2)$ садржи 3 тачке, а права у $\mathrm{PG}(n,3)$ садржи 4 тачке. Користећи одговарајуће ознаке $x_0, \ldots, x_{14}$ за тачке простора $\mathrm{PG}(3,2)$ које смо раније експлицитно написали, сада наводимо и праве тог простора.

$p_0 = \{x_0, x_1, x_2\}$	$p_1 = \{x_0, x_3, x_4\}$	$p_2 = \{x_0, x_5, x_6\}$	$p_3 = \{x_0, x_7, x_8\}$
$p_4 = \{x_0, x_9, x_{10}\}$	$p_5 = \{x_0, x_{11}, x_{12}\}$	$p_6 = \{x_0, x_{13}, x_{14}\}$	$p_7 = \{x_1, x_3, x_5\}$
$p_8 = \{x_1, x_4, x_6\}$	$p_9 = \{x_1, x_7, x_9\}$	$p_{10} = \{x_1, x_8, x_{10}\}$	$p_{11} = \{x_1, x_{11}, x_{13}\}$
$p_{12} = \{x_1, x_{12}, x_{14}\}$	$p_{13} = \{x_2, x_3, x_6\}$	$p_{14} = \{x_2, x_4, x_5\}$	$p_{15} = \{x_2, x_7, x_{10}\}$
$p_{16} = \{x_2, x_8, x_9\}$	$p_{17} = \{x_2, x_{11}, x_{14}\}$	$p_{18} = \{x_2, x_{12}, x_{13}\}$	$p_{19} = \{x_3, x_7, x_{11}\}$
$p_{20} = \{x_3, x_8, x_{12}\}$	$p_{21} = \{x_3, x_9, x_{13}\}$	$p_{22} = \{x_3, x_{10}, x_{14}\}$	$p_{23} = \{x_4, x_7, x_{12}\}$
$p_{24} = \{x_4, x_8, x_{11}\}$	$p_{25} = \{x_4, x_9, x_{14}\}$	$p_{26} = \{x_4, x_{10}, x_{13}\}$	$p_{27} = \{x_5, x_7, x_{13}\}$
$p_{28} = \{x_5, x_8, x_{14}\}$	$p_{29} = \{x_5, x_9, x_{11}\}$	$p_{30} = \{x_5, x_{10}, x_{12}\}$	$p_{31} = \{x_6, x_7, x_{14}\}$
$p_{32} = \{x_6, x_8, x_{13}\}$	$p_{33} = \{x_6, x_9, x_{12}\}$	$p_{34} = \{x_6, x_{10}, x_{11}\}$	

Тврђење 5. Број правих у $\mathrm{PG}(n,q)$ је $\begin{bmatrix} n+1 \\ 2 \end{bmatrix}_q$.

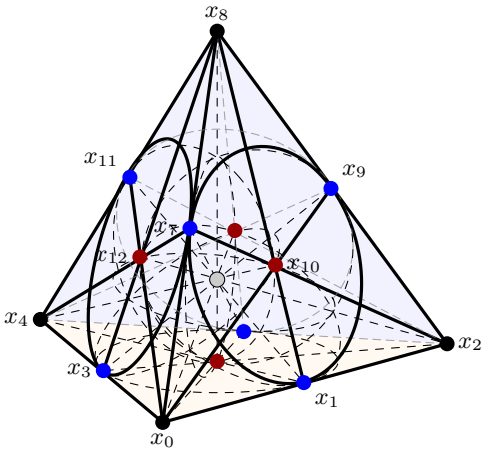
Доказ. Број правих је $\frac{\binom{|\mathrm{PG}(n,q)|}{2}}{\binom{q+1}{2}}$. Сваке две различите тачке одређују јединствену праву, па број свих парова тачака делимо бројем парова тачака на

једној правој. Број парова тачака у простору $\mathrm{PG}(n,q)$ је

$$\binom{|\mathrm{PG}(n,q)|}{2} = \frac{\frac{q^{n+1}-1}{q-1}\cdot\left(\frac{q^{n+1}-1}{q-1}-1\right)}{2!} = \frac{(q^{n+1}-1)(q^n-1)q}{2(q-1)^2},$$

а број парова тачака на свакој правој је $\binom{q+1}{2} = \frac{(q+1)q}{2}$. За тражени количник добијамо $\frac{(q^{n+1}-1)(q^n-1)}{(q-1)^2(q+1)} = \frac{(q^{n+1}-1)(q^n-1)}{(q^2-1)(q-1)} = \begin{bmatrix} n+1 \\ 2 \end{bmatrix}_q$. □

Заменом конкретних вредности у формулу добијамо да је број правих простора $\mathrm{PG}(2,2)$ једнак $\begin{bmatrix} 2+1 \\ 2 \end{bmatrix}_2 = \frac{(2^3-1)(2^2-1)}{(2^2-1)(2-1)} = \frac{7\cdot 3}{3} = 7$, док у случају $\mathrm{PG}(3,2)$ добијамо да је број правих $\begin{bmatrix} 3+1 \\ 2 \end{bmatrix}_2 = \frac{(2^4-1)(2^3-1)}{(2^2-1)(2-1)} = \frac{15\cdot 7}{3} = 35$, што нам даје потврду да су спискови правих тих простора које смо навели заиста потпуни.



Слика 1.2: Модел простора $\mathrm{PG}(3,2)$.

За $\mathrm{PG}(3,3)$ број правих је $\begin{bmatrix} 3+1 \\ 2 \end{bmatrix}_3 = \frac{(3^4-1)(3^3-1)}{(3^2-1)(3-1)} = \frac{80\cdot 26}{8\cdot 2} = 130$. Скуп тачака $\{x_0, x_1, x_2, x_3\}$ тог простора представља пример једне праве. Посебно ће нас занимати одређене конфигурације правих.

Покривачи и паковања

Дефиниција 18. *Покривач* (енгл. spread) простора $\mathrm{PG}(n,q)$ је скуп међусобно дисјунктних правих таквих да свака тачка простора припада некој правој из тог скупа.

Дакле, покривач разлаже скуп свих тачака на дисјунктне праве. Пример једног покривача простора $\mathrm{PG}(3,2)$ је $\{p_0, p_{19}, p_{25}, p_{30}, p_{32}\}$.

Тврђење 6. Ако покривач простора $\mathrm{PG}(n,q)$ постоји, онда је број правих у њему $\dfrac{[n+1]_q}{[2]_q} = \dfrac{q^{n+1}-1}{q^2-1}$.

Доказ. Свака тачка простора лежи на тачно једној правој из датог покривача, а свака права садржи по $q+1$ тачака. Према томе, број правих у покривачу је

$$\frac{|\mathrm{PG}(n,q)|}{q+1} = \frac{q^{n+1}-1}{(q-1)(q+1)} = \frac{q^{n+1}-1}{q^2-1}.$$

□

Заменом $n=3$ добијамо да је број правих у покривачу простора $\mathrm{PG}(3,q)$ једнак

$$\frac{q^4-1}{q^2-1} = \frac{(q^2-1)(q^2+1)}{q^2-1} = q^2+1.$$

У простору $\mathrm{PG}(3,2)$ сваки покривач се састоји од $2^2+1=5$ правих, док у $\mathrm{PG}(3,3)$ покривач има $3^2+1=10$ правих. Пример једног покривача простора $\mathrm{PG}(3,3)$ чине праве $\{x_0,x_1,x_2,x_3\}$, $\{x_4,x_{13},x_{22},x_{31}\}$, $\{x_5,x_{16},x_{26},x_{36}\}$, $\{x_6,x_{19},x_{30},x_{38}\}$, $\{x_7,x_{15},x_{27},x_{39}\}$, $\{x_8,x_{18},x_{28},x_{32}\}$, $\{x_9,x_{21},x_{23},x_{34}\}$, $\{x_{10},x_{14},x_{29},x_{35}\}$, $\{x_{11},x_{17},x_{24},x_{37}\}$, $\{x_{12},x_{20},x_{25},x_{33}\}$.

Тврђење 7. Покривач простора $\mathrm{PG}(n,q)$ постоји ако и само ако је n непаран број.

Доказ. Претпоставимо да покривач постоји. Број правих у покривачу мора да буде цео број, па због Тврђења 6 закључујемо да важи $q^2-1 \mid q^{n+1}-1$. Одатле следи $q+1 \mid q^{n+1}-1$, јер је $q^2-1=(q-1)(q+1)$. Како је $q \equiv_{q+1} -1$, имамо да је $q^{n+1}-1 \equiv_{q+1} (-1)^{n+1}-1$, што је 0 само кад је $n+1$ парно (због $q \geqslant 2$), тј. n непарно.

Доказ супротног смера импликације може се наћи у [8]. □

Дакле, не постоји покривач простора $\mathrm{PG}(2,2)$.

Дефиниција 19. *Паковање* (енгл. packing) простора $\mathrm{PG}(n,q)$ је скуп међусобно дисјунктних покривача такав да свака права простора припада неком покривачу из тог скупа.

То значи да паковање разлаже скуп свих правих на дисјунктне покриваче. Пример паковања простора $\mathrm{PG}(3,2)$ чине покривачи $\{p_0,p_{19},p_{25},p_{30},p_{32}\}$, $\{p_1,p_9,p_{18},p_{28},p_{34}\}$, $\{p_2,p_{10},p_{17},p_{21},p_{23}\}$, $\{p_3,p_{11},p_{14},p_{22},p_{33}\}$, $\{p_4,p_{12},p_{13},p_{24},p_{27}\}$, $\{p_5,p_7,p_{16},p_{26},p_{31}\}$, $\{p_6,p_8,p_{15},p_{20},p_{29}\}$.

Тврђење 8. Ако паковање простора $\mathrm{PG}(n,q)$ постоји, онда је број покривача у њему $[n]_q = \dfrac{q^n-1}{q-1}$.

Доказ. Према Тврђењу 5 правих укупно има $\begin{bmatrix} n+1 \\ 2 \end{bmatrix}_q$. Свака права простора припада тачно једном од покривача у паковању, а према Тврђењу 6 сваки покривач садржи по $\frac{[n+1]_q}{[2]_q}$ правих. Стога број покривача у паковању је

$$\frac{\begin{bmatrix} n+1 \\ 2 \end{bmatrix}_q}{\frac{[n+1]_q}{[2]_q}} = \frac{\frac{[n+1]_q \cdot [n]_q}{[2]_q \cdot [1]_q}}{\frac{[n+1]_q}{[2]_q}} = [n]_q = \frac{q^n-1}{q-1}.$$

□

Заменом $n=3$ добијамо да је број покривача у паковању простора $\mathrm{PG}(3,q)$ једнак

$$[3]_q = \frac{q^3-1}{q-1} = \frac{(q-1)(q^2+q+1)}{q-1} = q^2+q+1.$$

У $\mathrm{PG}(3,2)$ постоји разлагање скупа од 35 правих на $2^2+2+1=7$ покривача (сваки од 5 правих), а у $\mathrm{PG}(3,3)$ праве (130 њих) се могу расподелити на $3^2+3+1=13$ покривача од по 10 правих.

Паковање се често назива и *паралелизам* (енгл. parallelism), јер је аналогно разлагању свих правих афиног простора на паралелне класе: покривач има улогу једне паралелне класе, а паковање улогу целог разлагања. Ова аналогија се огледа и у наредном тврђењу.

Тврђење 9. Ако паковање простора $\mathrm{PG}(n,q)$ постоји, онда је број правих које пролазе кроз произвољну тачку тог простора једнак броју покривача у том паковању.

Доказ. Фиксирајмо тачку A и паковање P. Постоји бијекција између скупа правих које пролазе кроз тачку A и скупа покривача паковања P. Свакој правој l која пролази кроз A придружимо јединствени покривач s_l из паковања P који садржи l. Тај покривач постоји и јединствен је зато што паковање разлаже скуп свих правих простора на дисјунктне покриваче. Ово пресликавање је инјективно јер две различите праве кроз A не могу припадати истом покривачу – оне се секу у тачки A, док су праве сваког покривача међусобно дисјунктне. Пресликавање је и сурјективно јер сваки покривач из P садржи праву кроз A, пошто његове праве разлажу скуп свих тачака простора. □

Општи критеријум за постојање паковања није познат. Према Тврђењу 7, покривач простора $\mathrm{PG}(n, q)$ постоји само ако је n непаран, а како се свако паковање састоји од покривача, непарност броја n је потребан услов и за постојање паковања. Није познато да ли је овај услов и довољан, али су познати одређени довољни услови. У наредном тврђењу наводимо два таква услова, чији докази могу да се нађу у [5], [6] и [7].

Тврђење 10. Паковање у $\mathrm{PG}(n, q)$ постоји ако је испуњен бар један од следећих услова:

- n је непаран број и $q \in \{2, 3, 4, 8, 16\}$,
- $n = 2^k - 1$ за неки позитиван цео број k.

Без употребе рачунара теже је одредити укупан број покривача и паковања и извршити њихову класификацију. Испоставиће се да у пројективном простору $\mathrm{PG}(3, 2)$ постоји укупно 56 покривача (сви су међусобно изоморфни), а у $\mathrm{PG}(3, 3)$ постоји укупно 8 424 покривача (подељених у две класе изоморфизма). У простору $\mathrm{PG}(3, 2)$ постоје тачно два неизоморфна паковања, док у простору $\mathrm{PG}(3, 3)$ постоји укупно 73 343 неизоморфна паковања. Паковања простора $\mathrm{PG}(3, 3)$ проучавана су још почетком седамдесетих година XX века, али је њихова потпуна класификација извршена тек више од четири деценије касније, у раду Бетена [1].

Колинеације

Дефиниција 20. За три тачке кажемо да су *колинеарне* ако постоји права која их садржи.

Дефиниција 21. Аутоморфизам пројективног простора $\mathrm{PG}(n, q)$ са релацијом колинеарности називамо *колинеацијом*.

Дакле, колинеације сликају праве у праве. Због бијективности, следи да колинеације сликају и покриваче у покриваче, а самим тим и паковања у паковања. Даља разматрања биће усмерена на колинеације, у односу на које ћемо посматрати изоморфност свих ових објеката. Ради касније програмске имплементације, сада ћемо додатно повезати геометријски и алгебарски приступ.

Дефиниција 22. За пресликавање $f\colon \mathbb{F}_q^{n+1} \to \mathbb{F}_q^{n+1}$ кажемо да је *семилинеарно* ако постоји аутоморфизам $\sigma\colon \mathbb{F}_q \to \mathbb{F}_q$ поља $(\mathbb{F}_q, +, \cdot)$ и за све $x, y \in \mathbb{F}_q^{n+1}$ и све $\lambda \in \mathbb{F}_q$ важи

$$f(x+y) = f(x) + f(y) \quad \text{и} \quad f(\lambda x) = \sigma(\lambda) f(x).$$

Дефиниција 23. За $c\colon \mathrm{PG}(n,q) \to \mathrm{PG}(n,q)$ кажемо да је *пројективно семилинеарно* пресликавање ако постоји бијективно семилинеарно пресликавање $f\colon \mathbb{F}_q^{n+1} \to \mathbb{F}_q^{n+1}$ и за све $[x] \in \mathrm{PG}(n,q)$ важи

$$c([x]) = [f(x)].$$

Функција је добро дефинисана, јер из $[x] = [y]$ следи $x = \lambda y$ за неко $\lambda \in \mathbb{F}_q$, што даје $c([x]) = [f(\lambda y)] = [\sigma(\lambda) f(y)] = [f(y)] = c([y])$. Инјективност функције f гарантује $f(x) \neq \mathbf{0}$ за све $[x] \in \mathrm{PG}(n,q)$, тј. $c([x]) \in \mathrm{PG}(n,q)$.

Скуп свих пројективних семилинеарних пресликавања простора $\mathrm{PG}(n,q)$ означавамо са $\mathrm{P\Gamma L}(n+1,q)$. Из бијективности функције f следи да је и функција c бијекција. Заједно са операцијом композиције функција скуп $\mathrm{P\Gamma L}(n+1,q)$ чини једну групу, коју називамо *пројективном семилинеарном групом*.

Теорема 3. *(Основна теорема пројективне геометрије)* За $n \geqslant 2$ скуп свих колинеација простора $\mathrm{PG}(n,q)$ једнак је скупу $\mathrm{P\Gamma L}(n+1,q)$.

Доказ. Покажимо да је свако пројективно семилинеарно пресликавање колинеација. Нека је $c \in \mathrm{P\Gamma L}(n+1,q)$. По дефиницији, постоје аутоморфизам $\sigma\colon \mathbb{F}_q \to \mathbb{F}_q$ и бијективно семилинеарно пресликавање $f\colon \mathbb{F}_q^{n+1} \to \mathbb{F}_q^{n+1}$ у односу на σ, такви да је $c([x]) = [f(x)]$ за све $[x] \in \mathrm{PG}(n,q)$.

Нека су $[A], [B], [C] \in \mathrm{PG}(n,q)$ колинеарне тачке. Ако је $[A] = [B]$, онда је $c([A]) = c([B])$, па су тачке $c([A])$, $c([B])$ и $c([C])$ колинеарне. У супротном постоје $\lambda, \mu \in \mathbb{F}_q$ такви да је $C = \lambda A + \mu B$, па важи $c([C]) = [f(C)] = [f(\lambda A + \mu B)] = [\sigma(\lambda) f(A) + \sigma(\mu) f(B)]$. Како је $c([A]) = [f(A)]$ и $c([B]) = [f(B)]$, последња једнакост показује да $c([C])$ припада правој кроз $c([A])$ и $c([B])$. Дакле, c чува колинеарност.

Функција c је инјекција. Из $c([A]) = c([B])$ следи $[f(A)] = [f(B)]$, па за неко $\lambda \in \mathbb{F}_q$ важи $f(A) = \lambda f(B) = f\big(\sigma^{-1}(\lambda) B\big)$. Из инјективности функције f следи $A = \sigma^{-1}(\lambda) B$, односно $[A] = [B]$. Функција c је и сурјекција. Нека је $[B] \in \mathrm{PG}(n,q)$ произвољна тачка. Из сурјективности функције f следи да

постоји $A \in \mathbb{F}_q^{n+1}$ такав да је $f(A) = B$. Како је $B \neq \mathbf{0}$, из инјективности функције f и $f(\mathbf{0}) = \mathbf{0}$ следи $A \neq \mathbf{0}$, па је $[A] \in \mathrm{PG}(n, q)$ и $c([A]) = [B]$. Према томе, c је бијекција.

Остаје да проверимо да и c^{-1} чува колинеарност. Пошто је f бијекција, постоји инверзно пресликавање f^{-1}. Из адитивности функције f следи $f\big(f^{-1}(x)+f^{-1}(y)\big) = f\big(f^{-1}(x)\big)+f\big(f^{-1}(y)\big) = x+y$, па је и инверзна функција f^{-1} такође адитивна, тј. добијамо да важи $f^{-1}(x+y) = f^{-1}(x)+f^{-1}(y)$. Поред тога, за свако $\lambda \in \mathbb{F}_q$ важи $f\big(\sigma^{-1}(\lambda)f^{-1}(x)\big) = \sigma\big(\sigma^{-1}(\lambda)\big)f\big(f^{-1}(x)\big) = \lambda x$, па је $f^{-1}(\lambda x) = \sigma^{-1}(\lambda)f^{-1}(x)$. Дакле, f^{-1} је семилинеарно у односу на аутоморфизам σ^{-1}. Како је $c^{-1}([x]) = [f^{-1}(x)]$, претходно разматрање показује да и c^{-1} чува колинеарност. Стога је c аутоморфизам простора $\mathrm{PG}(n, q)$ са релацијом колинеарности, односно колинеација.

Доказ да је свака колинеација простора $\mathrm{PG}(n, q)$ пројективно семилинеарно пресликавање може се наћи у [3]. □

Семилинеарна и пројективна семилинеарна пресликавања у специјалном случају за $\sigma = \mathrm{id}$ називају се *линеарна* и *пројективна линеарна* пресликавања, редом. Скуп свих пројективних линеарних пресликавања означавамо са $\mathrm{PGL}(n+1, q)$, који заједно са композицијом функција чини групу коју називамо *пројективном линеарном групом*. Очигледно важи $\mathrm{PGL}(n+1, q) \leqslant \mathrm{P\Gamma L}(n+1, q)$.

Тврђење 11. Када је q прост број, важи $\mathrm{PGL}(n+1, q) = \mathrm{P\Gamma L}(n+1, q)$.

Доказ. По дефиницији увек важи $\mathrm{PGL}(n+1, q) \subseteq \mathrm{P\Gamma L}(n+1, q)$, тако да остаје да се докаже да за прост број q важи $\mathrm{P\Gamma L}(n+1, q) \subseteq \mathrm{PGL}(n+1, q)$, тј. да је $\sigma = \mathrm{id}$ (односно $\sigma(x) = x$ за свако x). Знамо да су сва коначна поља са истим бројем елемената изоморфна, а видели смо у Тврђењу 1 како изгледа једно коначно поље са простим бројем елемената, па $\mathbb{F}_q$ можемо поистоветити са пољем $\{0, 1, \ldots, q-1\}$. У таквом пољу знамо да за свако $x \in \mathbb{F}_q$ важи

$$\sigma(x) = \sigma(\underbrace{1 + \cdots + 1}_{x}) = \underbrace{\sigma(1) + \cdots + \sigma(1)}_{x} = x \cdot \sigma(1) = x$$

јер аутоморфизам увек мора да слика сваки неутрал у самог себе ($\sigma(1) = 1$). □

Групу свих бијективних линеарних пресликавања простора $\mathbb{F}_q^{n+1}$ (са операцијом композиције функција) означавамо са $\mathrm{GL}(n+1, q)$. Познато је да

је та група, у односу на стандардну базу, изоморфна групи инвертибилних квадратних матрица димензије $n+1$ над пољем $\mathbb{F}_q$ (са операцијом множења матрица), па ћемо те две групе у наставку поистовећивати. При томе, колоне матрице која одговара неком пресликавању представљају слике вектора стандардне базе.

Тврђење 12. Нека је $\mathrm{Z}(n+1,q) = \{\lambda E_{n+1} \mid \lambda \in \mathbb{F}_q \setminus \{0\}\}$, где је E_{n+1} јединична матрица димензије $n+1$. Тада важи

$$\mathrm{PGL}(n+1,q) \cong \mathrm{GL}(n+1,q)/\mathrm{Z}(n+1,q).$$

Доказ. Дефинишимо функцију $f\colon \mathrm{GL}(n+1,q) \to \mathrm{PGL}(n+1,q)$ која инвертибилну матрицу $A \in \mathrm{GL}(n+1,q)$ слика у одговарајуће пројективно линеарно пресликавање које она одређује $[x] \mapsto [Ax]$. Ово пресликавање је добро дефинисано јер из инвертибилности A следи $Ax \neq \mathbf{0}$ за $x \neq \mathbf{0}$ и пресликавање не зависи од избора представника x јер за свако $\lambda \in \mathbb{F}_q \setminus \{0\}$ важи $[A(\lambda x)] = [\lambda Ax] = [Ax]$.

Свако пројективно линеарно пресликавање је, по дефиницији, одређено неким бијективним линеарним пресликавањем баш на начин на који смо дефинисали функцију f (а у $\mathrm{GL}(n+1,q)$ се налазе сва бијективна линеарна пресликавања), одакле следи да је $\operatorname{Im} f = \mathrm{PGL}(n+1,q)$.

Функција је хомоморфизам јер важи $f(AB)([x]) = [ABx] = f(A)(f(B)([x]))$ за свако $[x] \in \mathrm{PG}(n,q)$, тј. $f(AB) = f(A) \circ f(B)$.

Очигледно важи $\mathrm{Z}(n+1,q) \subseteq \operatorname{Ker} f$ (јер је $[\lambda E_{n+1}x] = [x]$). Докажимо $\operatorname{Ker} f \subseteq \mathrm{Z}(n+1,q)$. За $A \in \operatorname{Ker} f$ важи $f(A) = \mathrm{id}$, тј. $[Ax] = [x]$, за све $[x] \in \mathrm{PG}(n,q)$ (односно за све $x \in \mathbb{F}_q^{n+1} \setminus \{\mathbf{0}\}$). Специјално, ово важи и за све векторе стандардне базе $e_i = (0,\ldots,0,1,0,\ldots,0)$ (на i-тој координати се налази 1, а све остале су 0), као и за све $e_i + e_j$ $(i \neq j)$. Како за све i важи $[Ae_i] = [e_i]$, закључујемо да се у матрици A налазе нуле свуда осим на главној дијагонали (јер је Ae_i заправо i-та колона матрице A), тј. $A = \operatorname{diag}(a_1,\ldots,a_{n+1})$. Из $[A(e_i+e_j)] = [e_i+e_j]$ следи $A(e_i+e_j) = \lambda(e_i+e_j)$ за неко $\lambda \in \mathbb{F}_q$, односно $a_i = a_j = \lambda$ (јер је $A(e_i+e_j) = a_ie_i + a_je_j$). Пошто ово важи за све различите парове i и j, закључујемо да су сви елементи главне дијагонале матрице A једнаки. Не могу бити 0 јер је $A \in \operatorname{Ker} f \subseteq \mathrm{GL}(n+1,q)$ инвертибилна матрица. Закључујемо $A \in \mathrm{Z}(n+1,q)$.

Пошто важи $\mathrm{Z}(n+1,q) = \operatorname{Ker} f$, применом прве теореме о изоморфизму, добијамо тражени закључак. □

У доказу теореме смо добили $\mathrm{Z}(n+1,q) = \operatorname{Ker} f$, а знамо да је језгро нормална подгрупа домена, тј. $\operatorname{Ker} f \triangleleft \mathrm{GL}(n+1,q)$. Одатле закључујемо $\mathrm{Z}(n+1,q) \triangleleft \mathrm{GL}(n+1,q)$.

Тврђење 13. Ако је $n \geqslant 2$ и q прост број, онда је број колинеација простора $\mathrm{PG}(n,q)$ једнак $\dfrac{1}{q-1}\displaystyle\prod_{i=0}^{n}(q^{n+1}-q^i)$.

Доказ. Из основне теореме пројективне геометрије закључујемо да је број колинеација простора $\mathrm{PG}(n,q)$ једнак броју елемената скупа $\mathrm{P\Gamma L}(n+1,q)$. Како је q прост број, из Тврђења 11 закључујемо да је тражени број заправо једнак $|\mathrm{PGL}(n+1,q)|$. На основу Тврђења 12 закључујемо да је број колинеација једнак $|\mathrm{GL}(n+1,q)/\mathrm{Z}(n+1,q)|$.

Како свака инвертибилна матрица реда $n+1$ представља један елемент групе $\mathrm{GL}(n+1,q)$, број елемената те групе се може одредити бројем начина да се изаберу линеарно независне колоне матрице. Прву колону можемо изабрати на $q^{n+1}-1$ начина (било који ненула вектор), другу на $q^{n+1}-q$ начина (вектор који није у простору који генерише прва колона), трећу на $q^{n+1}-q^2$ начина (вектор који није у простору генерисаном са прве две колоне) итд. Сходно томе, имамо следећи закључак

$$|\mathrm{GL}(n+1,q)| = \prod_{i=0}^{n}(q^{n+1}-q^i).$$

Како је $|\{\lambda E_{n+1} \mid \lambda \in \mathbb{F}_q \setminus \{0\}\}| = |\mathbb{F}_q \setminus \{0\}|$, следи $|\mathrm{Z}(n+1,q)| = q-1$. Пошто је $\mathrm{Z}(n+1,q) \leqslant \mathrm{GL}(n+1,q)$, из Лагранжове теореме добијамо

$$|\mathrm{PGL}(n+1,q)| = |\mathrm{GL}(n+1,q)/\mathrm{Z}(n+1,q)| = \frac{|\mathrm{GL}(n+1,q)|}{|\mathrm{Z}(n+1,q)|},$$

што доказује тврђење. □

Тако добијамо да у $\mathrm{PG}(3,2)$ има $|\mathrm{PGL}(4,2)| = 20\,160$ колинеација, а у $\mathrm{PG}(3,3)$ их има $|\mathrm{PGL}(4,3)| = 12\,130\,560$.

Вредности за $\mathrm{PG}(3,2)$ и $\mathrm{PG}(3,3)$

Наредна табела обједињује све наведене вредности, а подебљано су означене оне које не следе непосредно из претходних формула.

Карактеристика	**PG(3, 2)**	**PG(3, 3)**
Број тачака	15	40
Број правих	35	130
Број тачака на правој	3	4
Број правих у покривачу	5	10
Број покривача у паковању	7	13
Број свих покривача	**56**	**8 424**
Број неизоморфних покривача	**1**	**2**
Број неизоморфних паковања	**2**	**73 343**
Број колинеација	20 160	12 130 560

Вредности за $\mathrm{PG}(3,2)$ имају следеће тумачење у Киркмановом проблему петнаест ученица: тачке одговарају ученицама, праве редовима од по три ученице, покривачи дневним распоредима, а паковање целокупном седмодневном распореду. Сваке две тачке одређују јединствену праву. Пошто свака права припада тачно једном покривачу паковања, сваки пар ученица распоређен је у исти ред током тачно једног дана. Структура пројективног простора намеће додатна ограничења која Киркманов проблем не захтева, па свако паковање простора $\mathrm{PG}(3,2)$ даје решење овог проблема, али обрнуто не важи. Зато неизоморфна паковања простора $\mathrm{PG}(3,2)$ дају два од седам неизоморфних решења Киркмановог проблема [1].

У наставку разматрамо рачунарски приступ генерисању и класификовању покривача и парцијалних паковања. У случају $\mathrm{PG}(3,3)$ тај приступ захтева ефикасан алгоритам који може да се избори са проблемом комбинаторне експлозије.

Глава 2

Шмалцов алгоритам и његова примена у класификовању паковања простора $\mathbf{PG}(n, q)$

Класификациони алгоритми се често заснивају на постепеном проширивању већ класификованих подобјеката. Шмалц је развио један такав алгоритам у коме се током поступка чувају и касније користе већ пронађена пресликавања између изоморфних објеката [9, 10]. Бетен је Шмалцов алгоритам прилагодио и применио на класификовање паковања простора $\mathrm{PG}(3, 3)$. У наставку излажемо Бетенову формулацију Шмалцовог алгоритма [1].

2.1 Основни појмови

Нека су $\mathcal{X}$ и $\mathcal{Y}$ дисјунктни коначни скупови, при чему је $\mathcal{Y}$ скуп објеката које желимо да класификујемо, а $\mathcal{X}$ скуп њихових подобјеката које ћемо при томе користити. Претпостављамо да коначна група G дејствује на скупове $\mathcal{X}$ и $\mathcal{Y}$. Класификовање ће се заснивати на орбитама ових дејстава. Према раније уведеној конструкцији за Декартов производ, група G дејствује и на скуп $\mathcal{X} \times \mathcal{Y}$. Релацијом $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ формализујемо појам подобјекта: $X \in \mathcal{X}$ је подобјекат објекта $Y \in \mathcal{Y}$ ако и само ако је $(X, Y) \in \mathcal{R}$.

Дефиниција 24. За релацију $\mathcal{R}$ кажемо да је *G-инваријантна* ако за све $X \in \mathcal{X}$, $Y \in \mathcal{Y}$ и $g \in G$ важи

$$(X, Y) \in \mathcal{R} \quad \Rightarrow \quad (X, Y)^g \in \mathcal{R}.$$

У наставку подразумевамо да је овај услов испуњен. Одатле следи да група G дејствује и на скуп $\mathcal{R}$. Подобјекте бирамо тако да их је лакше класификовати, али и да су довољно велики да се могу ефикасно проширити до објеката из скупа $\mathcal{Y}$ са којима су у релацији $\mathcal{R}$.

Дефиниција 25. За $X \in \mathcal{X}$ дефинишемо *up*-скуп, а за $Y \in \mathcal{Y}$ *down*-скуп:

$$\mathrm{Up}(X) = \mathcal{R} \cap (\{X\} \times \mathcal{Y}) \quad \text{и} \quad \mathrm{Down}(Y) = \mathcal{R} \cap (\mathcal{X} \times \{Y\}).$$

Скуп $\mathcal{X}$ бирамо тако да сваки $Y \in \mathcal{Y}$ има бар један подобјекат из $\mathcal{X}$, односно да је $\mathrm{Down}(Y) \neq \varnothing$.

2.2 *Up*-орбите и *down*-орбите

Нека је $(X, Z) \in \mathrm{Up}(X)$ и $g \in G_X$. Због G-инваријантности релације $\mathcal{R}$ важи $(X, Z)^g = (X^g, Z^g) = (X, Z^g) \in \mathcal{R}$, па је $(X, Z^g) \in \mathrm{Up}(X)$. Дакле, стабилизатор G_X дејствује на $\mathrm{Up}(X)$. Аналогно, ако је $(Z, Y) \in \mathrm{Down}(Y)$ и $g \in G_Y$, онда је $(Z, Y)^g = (Z^g, Y^g) = (Z^g, Y) \in \mathrm{Down}(Y)$, па стабилизатор G_Y дејствује на $\mathrm{Down}(Y)$.

Дефиниција 26. Орбите дејства групе G_X на $\mathrm{Up}(X)$ називамо *up*-орбитама придруженим елементу X, а орбите дејства групе G_Y на $\mathrm{Down}(Y)$ називамо *down*-орбитама придруженим елементу Y.

Нека су $P_1, \ldots, P_m$ представници G-орбита у скупу $\mathcal{X}$, а $Q_1, \ldots, Q_r$ представници G-орбита у скупу $\mathcal{Y}$. Како орбите разлажу ове скупове, важи

$$\mathcal{X} = \bigsqcup_{i=1}^{m} P_i^G \quad \text{и} \quad \mathcal{Y} = \bigsqcup_{j=1}^{r} Q_j^G.$$

За свако $i \in \{1, \ldots, m\}$ посматрамо *up*-орбите придружене елементу P_i. Из сваке од њих изаберимо по једног представника и означимо их са $T_{i,1}, \ldots, T_{i,m_i}$. *Up*-орбиту чији је представник $T_{i,a}$ означавамо са

$$\mathcal{T}_{i,a} = T_{i,a}^{G_{P_i}}, \qquad a \in \{1, \ldots, m_i\},$$

па је

$$\mathrm{Up}(P_i) = \bigsqcup_{a=1}^{m_i} \mathcal{T}_{i,a}.$$

Аналогно, за свако $j \in \{1, \ldots, r\}$ посматрамо *down*-орбите придружене елементу Q_j. Из сваке од њих изаберимо по једног представника и означимо их са $S_{j,1}, \ldots, S_{j,r_j}$. Те орбите означимо са

$$\mathcal{S}_{j,b} = S_{j,b}^{G_{Q_j}}, \qquad b \in \{1, \ldots, r_j\},$$

па је

$$\mathrm{Down}(Q_j) = \bigsqcup_{b=1}^{r_j} \mathcal{S}_{j,b}.$$

Дефинишимо скупове

$$\mathfrak{T}_i = \{\mathcal{T}_{i,a} \mid a \in \{1, \ldots, m_i\}\} \quad \text{и} \quad \mathfrak{S}_j = \{\mathcal{S}_{j,b} \mid b \in \{1, \ldots, r_j\}\},$$

као и

$$\mathfrak{T} = \bigsqcup_{i=1}^{m} \mathfrak{T}_i \quad \text{и} \quad \mathfrak{S} = \bigsqcup_{j=1}^{r} \mathfrak{S}_j.$$

У наредној леми наводимо основну везу између *up*-орбита и *down*-орбита [1].

Лема 1. Постоји бијекција $\psi \colon \mathfrak{T} \to \mathfrak{S}$.

Доказ. Свака *up*-орбита $\mathcal{T}_{i,a} \in \mathfrak{T}$, као и свака *down*-орбита $\mathcal{S}_{j,b} \in \mathfrak{S}$, садржана је у тачно једној G-орбити у скупу $\mathcal{R}$. Постојање следи из $G_{P_i} \subseteq G$ и $G_{Q_j} \subseteq G$, а јединственост из чињенице да G-орбите разлажу $\mathcal{R}$. Показаћемо и да свака G-орбита у скупу $\mathcal{R}$ садржи тачно једну *up*-орбиту из $\mathfrak{T}$ и тачно једну *down*-орбиту из $\mathfrak{S}$.

Нека је $\mathcal{O}$ произвољна G-орбита у скупу $\mathcal{R}$, па је $\mathcal{O} \subseteq \mathcal{R}$. Пошто група G дејствује на парове по координатама, све прве координате парова из $\mathcal{O}$ припадају истој G-орбити у скупу $\mathcal{X}$. Како су $P_1, \ldots, P_m$ представници тих орбита, постоји јединствено $i \in \{1, \ldots, m\}$ такво да све прве координате парова из $\mathcal{O}$ припадају P_i^G. Зато за произвољно $(X, Y) \in \mathcal{O}$ можемо изабрати $g \in G$ тако да је $X^g = P_i$. Тада је $(P_i, Y^g) = (X, Y)^g \in \mathcal{O} \cap \mathrm{Up}(P_i)$, па је овај пресек непразан.

Покажимо да је $\mathcal{O} \cap \mathrm{Up}(P_i)$ једна од *up*-орбита придружених елементу P_i. Ако је $(P_i, Z) \in \mathcal{O} \cap \mathrm{Up}(P_i)$, онда важи $(P_i, Z)^{G_{P_i}} \subseteq \mathcal{O} \cap \mathrm{Up}(P_i)$, јер је $G_{P_i} \subseteq G$ и сваки елемент групе G_{P_i} фиксира P_i. С друге стране, ако су $(P_i, Y_1), (P_i, Y_2) \in \mathcal{O} \cap \mathrm{Up}(P_i)$, постоји $h \in G$ такав да је $(P_i, Y_1)^h = (P_i, Y_2)$. Поређењем првих координата добијамо $P_i^h = P_i$, па је $h \in G_{P_i}$. Дакле,

$\mathcal{O} \cap \mathrm{Up}(P_i) = \mathcal{T}_{i,a}$ за јединствено $a \in \{1, \ldots, m_i\}$. За $k \neq i$ важи $\mathcal{O} \cap \mathrm{Up}(P_k) = \varnothing$, јер P_i и P_k припадају различитим G-орбитама у скупу $\mathcal{X}$.

Аналогно, постоје јединствени $j \in \{1, \ldots, r\}$ и $b \in \{1, \ldots, r_j\}$ такви да је

$$\mathcal{O} \cap \mathrm{Down}(Q_j) = \mathcal{S}_{j,b},$$

а за $\ell \neq j$ важи $\mathcal{O} \cap \mathrm{Down}(Q_\ell) = \varnothing$. Дакле, свака G-орбита у скупу $\mathcal{R}$ садржи тачно једну *up*-орбиту из $\mathfrak{T}$ и тачно једну *down*-орбиту из $\mathfrak{S}$.

Дефинишимо пресликавање $\psi\colon \mathfrak{T} \to \mathfrak{S}$ тако што свакој $\mathcal{T}_{i,a} \in \mathfrak{T}$ придружујемо јединствену $\mathcal{S}_{j,b} \in \mathfrak{S}$ садржану у G-орбити која садржи $\mathcal{T}_{i,a}$. Пошто је ово придруживање једнозначно у оба смера, ψ је бијекција. □

На основу доказане бијекције уводимо *класификациони граф* $\Gamma_{\mathcal{X},\mathcal{Y},G}$. То је неусмерен бипартитан граф са скупом чворова $\{P_1, \ldots, P_m\} \sqcup \{Q_1, \ldots, Q_r\}$. Између чворова P_i и Q_j постоји по једна грана за сваки пар орбита $\mathcal{T}_{i,a} \in \mathfrak{T}_i$ и $\mathcal{S}_{j,b} \in \mathfrak{S}_j$ за који је $\psi(\mathcal{T}_{i,a}) = \mathcal{S}_{j,b}$. Истом пару чворова може одговарати више таквих парова орбита, па је класификациони граф мултиграф.

2.3 Опис алгоритма

Ако је $\psi(\mathcal{T}_{i,a}) = \mathcal{S}_{j,b}$, онда представници $T_{i,a}$ и $S_{j,b}$ припадају истој G-орбити у скупу $\mathcal{R}$, па постоји $g \in G$ такав да је $T_{i,a}^g = S_{j,b}$. Такав елемент не мора бити јединствен, јер за све $u \in G_{T_{i,a}}$ и $v \in G_{S_{j,b}}$ важи $T_{i,a}^{ugv} = S_{j,b}$. За потребе алгоритма довољно је пронаћи и сачувати један такав елемент.

Дефиниција 27. Елемент $g \in G$ за који важи $T_{i,a}^g = S_{j,b}$ називамо *Шмалцовим елементом изоморфизма* између орбита $\mathcal{T}_{i,a}$ и $\mathcal{S}_{j,b}$.

Одговарајућој грани класификационог графа придружујемо један Шмалцов елемент изоморфизма као ознаку. Идеја о чувању таквих елемената потиче од Шмалца [9, 10]. Поступак који за дато $X \in \mathcal{X}$ проналази $i \in \{1, \ldots, m\}$ и $g \in G$ такве да је $X^g = P_i$ називамо *конструктивним препознавањем* G-орбита у скупу $\mathcal{X}$. Сачуване Шмалцове елементе изоморфизма касније користимо при конструктивном препознавању G-орбита у скупу $\mathcal{Y}$.

Претпостављамо да су познати представници $P_1, \ldots, P_m$ и њихови стабилизатори, као и да је доступно конструктивно препознавање G-орбита у скупу $\mathcal{X}$. У Бетеновом опису основни Шмалцов алгоритам има две главне фазе:

Lift и *Classify*. На крају алгоритма добијамо представнике $Q_1, \ldots, Q_r$ свих G-орбита у скупу $\mathcal{Y}$, њихове стабилизаторе и по један Шмалцов елемент изоморфизма за сваку грану класификационог графа $\Gamma_{\mathcal{X},\mathcal{Y},G}$.

Фаза *Lift*

За сваког представника P_i одређујемо скуп $\mathrm{Up}(P_i)$ и његову поделу на *up*-орбите $\mathcal{T}_{i,a}$, односно на орбите дејства стабилизатора G_{P_i}. За сваку од њих бирамо представника $T_{i,a}$ и одређујемо његов стабилизатор $G_{T_{i,a}}$. Пошто је $T_{i,a} \in \mathrm{Up}(P_i)$, он је облика $T_{i,a} = (P_i, Y_{i,a})$, где је $Y_{i,a} \in \mathcal{Y}$. Сваки елемент стабилизатора $G_{T_{i,a}}$ фиксира обе координате, па је

$$G_{T_{i,a}} \leqslant G_{P_i} \quad \text{и} \quad G_{T_{i,a}} \leqslant G_{Y_{i,a}}.$$

Зато $G_{T_{i,a}}$ можемо одредити издвајањем оних елемената групе G_{P_i} који фиксирају и $Y_{i,a}$.

Поред представника *up*-орбита, за сваки $T \in \mathcal{T}_{i,a}$ бележимо по један елемент $h_2 \in G_{P_i}$ за који је $T^{h_2} = T_{i,a}$. Касније, при одређивању Шмалцових елемената изоморфизма и конструктивном препознавању, за дати елемент T на основу сачуваних података проналазимо одговарајући елемент h_2. Из једнакости $T^{h_2} = T_{i,a}$ затим добијамо представника $T_{i,a}$, а тиме и *up*-орбиту $\mathcal{T}_{i,a}$ којој T припада.

Након обраде свих представника P_i формирамо скуп кандидата за представнике G-орбита у $\mathcal{Y}$

$$\mathcal{C} = \{Y_{i,a} \mid i \in \{1, \ldots, m\},\ a \in \{1, \ldots, m_i\}\} \subseteq \mathcal{Y}.$$

Скуп $\mathcal{C}$ садржи бар по један елемент из сваке G-орбите у скупу $\mathcal{Y}$. Заиста, за произвољно $Y \in \mathcal{Y}$ из $\mathrm{Down}(Y) \neq \varnothing$ следи да постоји $X \in \mathcal{X}$ за који је $(X, Y) \in \mathcal{R}$. Пошто су $P_1, \ldots, P_m$ представници, постоје $i \in \{1, \ldots, m\}$ и $g \in G$ такви да је $X^g = P_i$. Због G-инваријантности је $(X, Y)^g = (P_i, Y^g) \in \mathrm{Up}(P_i)$, па овај пар припада некој *up*-орбити $\mathcal{T}_{i,a}$. Представник те *up*-орбите је $T_{i,a} = (P_i, Y_{i,a})$, па Y^g и $Y_{i,a}$ припадају истој G_{P_i}-орбити, а самим тим Y и $Y_{i,a} \in \mathcal{C}$ припадају истој G-орбити.

При формирању скупа $\mathcal{C}$ већ смо, засебно међу проширењима сваког елемента P_i, елиминисали изоморфне копије које се добијају дејством стабилизатора G_{P_i}. Преостале кандидате још треба класификовати под дејством целе групе G. Фазу *Lift* можемо сажето описати на следећи начин:

1. За сваког представника P_i одредити скуп $\mathrm{Up}(P_i)$.
2. Разложити $\mathrm{Up}(P_i)$ на *up*-орбите $\mathcal{T}_{i,a}$. При томе:
   - изабрати представника $T_{i,a}$ сваке *up*-орбите,
   - одредити стабилизатор $G_{T_{i,a}}$ изабраног представника,
   - за сваки $T \in \mathcal{T}_{i,a}$ забележити по један $h_2 \in G_{P_i}$ за који је $T^{h_2} = T_{i,a}$.
3. Од других координата представника $T_{i,a} = (P_i, Y_{i,a})$ формирати $\mathcal{C}$.

Фаза *Classify*

Сада из скупа $\mathcal{C}$ издвајамо представнике $Q_1, \ldots, Q_r$ свих G-орбита у скупу $\mathcal{Y}$ и одређујемо њихове стабилизаторе. Поред тога, одређујемо и чувамо Шмалцове елементе изоморфизма. Кораком *Define* једног кандидата из скупа $\mathcal{C}$ проглашавамо за наредног представника Q_j и уклањамо га из скупа $\mathcal{C}$. Кораком *Eliminate* затим из скупа $\mathcal{C}$ уклањамо све преостале кандидате који припадају G-орбити представника Q_j и одређујемо стабилизатор G_{Q_j}. Ова два корака наизменично понављамо све док скуп $\mathcal{C}$ не постане празан.

У кораку *Define* из скупа $\mathcal{C}$ бирамо једног кандидата $Y_{i,a}$. У фази *Lift* већ је одређен представник *up*-орбите $T_{i,a} = (P_i, Y_{i,a})$ чија је друга координата изабрани кандидат, па је познат и одговарајући представник P_i. Кандидата $Y_{i,a}$ проглашавамо за наредног представника Q_j и уклањамо из скупа $\mathcal{C}$, па је $T_{i,a} = (P_i, Q_j)$. У фази *Lift* одредили смо групу $G_{T_{i,a}}$ и утврдили да је $G_{T_{i,a}} \leqslant G_{Y_{i,a}}$. Пошто је $Y_{i,a} = Q_j$, добијамо $G_{T_{i,a}} \leqslant G_{Q_j}$. Групу $G_{T_{i,a}}$ користићемо у кораку *Eliminate* при одређивању целог стабилизатора G_{Q_j}.

У кораку *Eliminate* за сваки пар $(X, Q_j) \in \mathrm{Down}(Q_j)$ поступамо на следећи начин:

1. Конструктивним препознавањем проналазимо представника P_k и елемент $h_1 \in G$ за који је $X^{h_1} = P_k$.
2. Због G-инваријантности релације $\mathcal{R}$ важи

   $$(X, Q_j)^{h_1} = (P_k, Q_j^{h_1}) \in \mathrm{Up}(P_k).$$

   У фази *Lift* за пар $(P_k, Q_j^{h_1})$ сачуван је елемент $h_2 \in G_{P_k}$ који овај пар слика у представника $T_{k,c}$ одговарајуће *up*-орбите:

   $$\left(P_k, Q_j^{h_1}\right)^{h_2} = T_{k,c} = (P_k, Y_{k,c}).$$

3. Одатле следи $Q_j^{h_1h_2} = Y_{k,c}$. Нека је $\mathcal{S}_{j,b}$ јединствена *down*-орбита која садржи пар (X, Q_j). Из једнакости $(X, Q_j)^{h_1h_2} = T_{k,c}$ следи да су $\mathcal{T}_{k,c}$ и $\mathcal{S}_{j,b}$ садржане у истој G-орбити у скупу $\mathcal{R}$. Пошто ψ упарује орбите из $\mathfrak{T}$ и $\mathfrak{S}$ садржане у истој G-орбити, важи $\psi(\mathcal{T}_{k,c}) = \mathcal{S}_{j,b}$, па овом пару орбита по дефиницији класификационог графа одговара једна грана. Разликујемо два случаја:

   - Ако је $(k, c) \neq (i, a)$, кандидат $Y_{k,c}$ уклањамо из скупа $\mathcal{C}$, а елемент $(h_1h_2)^{-1}$ чувамо као ознаку те гране.
   - Ако је $(k, c) = (i, a)$, онда је $Y_{k,c} = Q_j$, па чувамо елемент $h_1h_2 \in G_{Q_j}$, док као ознаку те гране узимамо неутрални елемент групе G.

   Слика представника $T_{k,c}$ под дејством одговарајуће ознаке припада орбити $\mathcal{S}_{j,b}$, па ту слику узимамо за представника $S_{j,b}$. Стога је ознака гране Шмалцов елемент изоморфизма. За сваку грану чувамо само по једну ознаку.

Покажимо да се у кораку *Eliminate* из скупа $\mathcal{C}$ уклањају тачно сви преостали кандидати из G-орбите представника Q_j. Сваки уклоњени кандидат $Y_{k,c}$ припада овој орбити, јер је при његовом проналажењу одређен елемент $h_1h_2 \in G$ за који је

$$Q_j^{h_1h_2} = Y_{k,c}.$$

Нека сада преостали кандидат $Y_{\ell,d} \in \mathcal{C}$ припада G-орбити представника Q_j. Тада постоји $g \in G$ такав да је $Y_{\ell,d}^g = Q_j$, па је

$$T_{\ell,d}^g = (P_\ell^g, Q_j) \in \mathrm{Down}(Q_j).$$

При обради овог пара, поступком описаним у претходна три корака, одређују се елементи h_1 и h_2 и представник $T_{k,c}$ за које важи

$$\left(T_{\ell,d}^g\right)^{h_1h_2} = T_{k,c}.$$

Одатле следи $T_{k,c} \in T_{\ell,d}^G$. Пошто је

$$\mathcal{T}_{\ell,d} \subseteq T_{\ell,d}^G \quad \text{и} \quad \mathcal{T}_{k,c} \subseteq T_{k,c}^G = T_{\ell,d}^G,$$

према доказаној леми, $T_{\ell,d}^G$ садржи тачно једну *up*-орбиту из $\mathfrak{T}$, па је $\mathcal{T}_{k,c} = \mathcal{T}_{\ell,d}$, односно $(k, c) = (\ell, d)$. Како је $Q_j = Y_{i,a}$ већ уклоњен из скупа $\mathcal{C}$, преостали

кандидат $Y_{\ell,d}$ није једнак Q_j. Зато је $(k,c) \neq (i,a)$, па се у трећем кораку кандидат $Y_{k,c} = Y_{\ell,d}$ уклања из скупа $\mathcal{C}$.

Дакле, након корака *Eliminate* у скупу $\mathcal{C}$ не остаје ниједан кандидат из G-орбите представника Q_j. Како је пре почетка фазе *Classify* скуп $\mathcal{C}$ садржао бар по један елемент из сваке G-орбите у скупу $\mathcal{Y}$, понављањем корака *Define* и *Eliminate* добијамо тачно по једног представника сваке од тих орбита.

Да бисмо од познате подгрупе $G_{T_{i,a}} \leqslant G_{Q_j}$ одредили цео стабилизатор G_{Q_j}, довољно је да пронађемо по једног представника сваког левог косета $gG_{T_{i,a}}$, где је $g \in G_{Q_j}$. Покажимо да се након обраде свих парова из $\mathrm{Down}(Q_j)$ чува тачно по један такав представник.

Нека су h_1h_2 и $h_1'h_2'$ сачувани при обради парова (X, Q_j) и (X', Q_j). Тада је

$$(X, Q_j) = T_{i,a}^{(h_1h_2)^{-1}} \quad \text{и} \quad (X', Q_j) = T_{i,a}^{(h_1'h_2')^{-1}}.$$

Ако h_1h_2 и $h_1'h_2'$ припадају истом левом косету, онда је $h_1'h_2' = h_1h_2u$ за неко $u \in G_{T_{i,a}}$. Пошто u^{-1} фиксира $T_{i,a}$, важи

$$T_{i,a}^{(h_1'h_2')^{-1}} = T_{i,a}^{u^{-1}(h_1h_2)^{-1}} = \left(T_{i,a}^{u^{-1}}\right)^{(h_1h_2)^{-1}} = T_{i,a}^{(h_1h_2)^{-1}},$$

тј. $(X, Q_j) = (X', Q_j)$. Пошто се сваки пар обрађује једном и при његовој обради за одређивање стабилизатора чува највише један елемент h_1h_2, два различита сачувана елемента не могу припадати истом левом косету.

Остаје да покажемо да ниједан леви косет није пропуштен. Нека је $g \in G_{Q_j}$ произвољан. Како је $T_{i,a} = (P_i, Q_j) \in \mathcal{R}$, важи

$$T_{i,a}^{g^{-1}} = (P_i^{g^{-1}}, Q_j) \in \mathrm{Down}(Q_j).$$

У кораку *Eliminate* обрађују се сви елементи скупа $\mathrm{Down}(Q_j)$, међу њима и $T_{i,a}^{g^{-1}}$. Описаним поступком тада се одређују елементи h_1 и h_2 и представник $T_{k,c}$ за које важи

$$\left(T_{i,a}^{g^{-1}}\right)^{h_1h_2} = T_{k,c}.$$

Из наведене једнакости следи $T_{k,c} \in T_{i,a}^G$, па је $T_{k,c}^G = T_{i,a}^G$. Пошто је $G_{P_i} \subseteq G$ и $G_{P_k} \subseteq G$, важи

$$\mathcal{T}_{i,a} \subseteq T_{i,a}^G \quad \text{и} \quad \mathcal{T}_{k,c} \subseteq T_{k,c}^G = T_{i,a}^G.$$

Према доказаној леми, $T_{i,a}^G$ садржи тачно једну *up*-орбиту из $\mathfrak{T}$, па је $\mathcal{T}_{i,a} = \mathcal{T}_{k,c}$, односно $(k,c) = (i,a)$. Према правилу из трећег корака, елемент h_1h_2 се зато чува и важи

$$T_{i,a}^{g^{-1}h_1h_2} = T_{i,a}.$$

Отуда је $g^{-1}h_1h_2 \in G_{T_{i,a}}$, односно

$$h_1h_2 \in gG_{T_{i,a}}.$$

Дакле, у произвољном левом косету $gG_{T_{i,a}}$ налази се сачувани елемент h_1h_2. Он је представник тог косета.

Према томе, сачуван је тачно по један представник сваког левог косета $gG_{T_{i,a}}$, где је $g \in G_{Q_j}$. Како ови косети разлажу групу G_{Q_j}, сваки њен елемент може се записати у облику $(h_1h_2)u$, где је h_1h_2 један од сачуваних представника, а $u \in G_{T_{i,a}}$.

Конструктивно препознавање

Конструктивно препознавање G-орбита у скупу $\mathcal{Y}$ заснивамо на претпостављеном конструктивном препознавању G-орбита у скупу $\mathcal{X}$. За произвољан објекат $Y \in \mathcal{Y}$ на следећи начин одређујемо представника Q_j његове G-орбите и елемент $g \in G$ такав да је $Y^g = Q_j$:

1. Изаберемо произвољан $X \in \mathcal{X}$ за који је $(X, Y) \in \mathrm{Down}(Y)$.

2. Конструктивним препознавањем у скупу $\mathcal{X}$ проналазимо представника P_i и елемент $g_1 \in G$ за који је $X^{g_1} = P_i$.

3. У фази *Lift* за сваки $T \in \mathrm{Up}(P_i)$ сачуван је по један елемент групе G_{P_i} који га слика у представника његове *up*-орбите. Пошто је $(P_i, Y^{g_1}) = (X, Y)^{g_1} \in \mathrm{Up}(P_i)$, проналазимо елемент $g_2 \in G_{P_i}$ придружен пару (P_i, Y^{g_1}). Тада је $(P_i, Y^{g_1})^{g_2} = T_{i,c}$, где је $T_{i,c} = (P_i, Y_{i,c})$ представник *up*-орбите $\mathcal{T}_{i,c}$ која садржи пар (P_i, Y^{g_1}). Поређењем других координата добијамо $Y^{g_1g_2} = Y_{i,c}$.

4. Ако је $T_{i,c}$ пар коришћен у кораку *Define* при избору представника Q_j, за g_3 узимамо неутрални елемент групе G. У супротном узимамо елемент $(h_1h_2)^{-1}$ сачуван у кораку *Eliminate* за *up*-орбиту $\mathcal{T}_{i,c}$. У оба случаја g_3 је Шмалцов елемент изоморфизма и важи $Y_{i,c}^{g_3} = Q_j$.

Тада је $Y^{g_1g_2g_3} = Q_j$, па је $g_1g_2g_3$ тражени елемент добијен поступком конструктивног препознавања. Ако $\mathcal{Y}$ преузме улогу скупа подобјеката у наредној примени алгоритма, добијени представници, њихови стабилизатори и описани поступак конструктивног препознавања чине улаз за фазе *Lift* и *Classify*.

2.4 Примена на класификовање паковања

Дефиниција 28. Скуп од k међусобно дисјунктних покривача називамо *парцијалним паковањем величине* k.

Паковање је, дакле, парцијално паковање величине $[n]_q$. За једну примену алгоритма фиксирамо целе бројеве $0 \leqslant k < \ell \leqslant [n]_q$. Скуп $\mathcal{X}$ чине сва парцијална паковања величине k, а скуп $\mathcal{Y}$ сва парцијална паковања величине ℓ. Ови скупови су дисјунктни јер се састоје од парцијалних паковања различитих величина.

Тада је $\mathcal{R}$ релација инклузије $\subset$, а за $n \geqslant 2$ групу $G = \mathrm{P\Gamma L}(n+1, q)$ чине све колинеације простора $\mathrm{PG}(n, q)$. Колинеације пресликавају парцијална паковања у парцијална паковања исте величине, па група G дејствује на скупове $\mathcal{X}$ и $\mathcal{Y}$. Стога G-орбите у скуповима $\mathcal{X}$ и $\mathcal{Y}$ представљају класе изоморфизма. Из $X \subset Y$ следи да за свако $g \in G$ важи $X^g = \{s^g \mid s \in X\} \subset \{s^g \mid s \in Y\} = Y^g$, па релација инклузије испуњава услов G-инваријантности.

Парови из $\mathrm{Up}(X)$ одређују сва проширења парцијалног паковања X која припадају скупу $\mathcal{Y}$, док парови из $\mathrm{Down}(Y)$ одређују сва парцијална паковања величине k садржана у Y. Свако $Y \in \mathcal{Y}$ садржи бар једно такво парцијално паковање, па је $\mathrm{Down}(Y) \neq \varnothing$.

Лексикографско прорећивање класификационог графа

У примени алгоритма на парцијална паковања могуће је смањити број разматраних грана класификационог графа коришћењем лексикографског прорећивања које примењује Бетен [1]. Нека је на скупу свих покривача фиксиран линеаран поредак $<$. Елементе парцијалних паковања записујмо растуће у односу на овај поредак. За парцијална паковања исте величине

$$A = \{a_0, \ldots, a_{r-1}\}, \qquad B = \{b_0, \ldots, b_{r-1}\},$$

где је $a_0 < \cdots < a_{r-1}$ и $b_0 < \cdots < b_{r-1}$, дефинишемо лексикографски поредак

$$A \prec B \quad \overset{\text{деф}}{\Longleftrightarrow} \quad (\exists j)\,(a_j < b_j \ \wedge\ (\forall i < j)\, a_i = b_i).$$

Представнике G-орбита бирамо тако да буду лексикографски најмањи елементи својих орбита. Конкретан линеарни поредак покривача који се користи у имплементацији описан је у одељку 3.2.

Специјална релација и прорeђени граф. У наставку претпостављамо да је $k \geqslant 1$. За $X \in \mathcal{X}$ и $Y \in \mathcal{Y}$ дефинишемо специјалну релацију $\mathcal{R}^* \subseteq \mathcal{R}$ условом

$$(X,Y) \in \mathcal{R}^* \quad \Leftrightarrow \quad (X,Y) \in \mathcal{R} \text{ и } \min s^{G_X} > \max X \quad \text{за свако } s \in Y \setminus X.$$

Специјални *up*-скуп дефинишемо са $\mathrm{Up}^*(X) = \mathcal{R}^* \cap (\{X\} \times \mathcal{Y})$.

Тврђење 14. Скуп $\mathrm{Up}^*(X)$ је G_X-инваријантан.

Доказ. Нека је $(X,Y) \in \mathrm{Up}^*(X)$ и $h \in G_X$. Због G-инваријантности релације $\mathcal{R}$ важи

$$(X,Y)^h = (X,Y^h) \in \mathcal{R}.$$

Нека је $t \in Y^h \setminus X$. Тада постоји $s \in Y \setminus X$ за који је $t = s^h$. Пошто је $h \in G_X$, важи

$$t^{G_X} = (s^h)^{G_X} = \{s^{ha} \mid a \in G_X\} = s^{G_X},$$

јер елементи ha, када a пролази кроз G_X, такође пролазе кроз целу групу G_X. Зато је

$$\min t^{G_X} = \min s^{G_X} > \max X.$$

Ово важи за сваки $t \in Y^h \setminus X$, па је $(X,Y^h) \in \mathrm{Up}^*(X)$. □

Из G_X-инваријантности скупа $\mathrm{Up}^*(X)$ следи да се он састоји од целих *up*-орбита придружених елементу X. Заиста, за сваки $T \in \mathrm{Up}^*(X)$ из G_X-инваријантности следи

$$T^{G_X} \subseteq \mathrm{Up}^*(X).$$

Пошто *up*-орбите разлажу скуп $\mathrm{Up}(X)$, свака од њих је или цела садржана у $\mathrm{Up}^*(X)$ или је са њим дисјунктна. *Up*-орбите садржане у $\mathrm{Up}^*(X)$ називаћемо специјалним *up*-орбитама.

Прорeђени класификациони граф $\Gamma^*_{\mathcal{X},\mathcal{Y},G}$ добијамо из класификационог графа $\Gamma_{\mathcal{X},\mathcal{Y},G}$ задржавањем само оних грана чије су придружене *up*-орбите специјалне.

Очување свих G-орбита.

Тврђење 15. Нека је Q_j лексикографски најмањи представник произвољне G-орбите у скупу $\mathcal{Y}$. Тада постоји G-орбита у скупу $\mathcal{X}$ са изабраним представником P_i тако да важи

$$(P_i, Q_j) \in \mathrm{Up}^*(P_i).$$

Доказ. Нека је

$$Q_j = \{s_0, \ldots, s_{\ell-1}\}, \qquad s_0 < \cdots < s_{\ell-1},$$

и означимо са $P = \{s_0, \ldots, s_{k-1}\}$ скуп првих k елемената представника Q_j.

Најпре покажимо да је P лексикографски најмањи у својој G-орбити. Претпоставимо супротно, да за неко $g \in G$ важи $P^g \prec P$. Нека је P' скуп првих k елемената растућег записа скупа Q_j^g. Скуп P^g је подскуп скупа Q_j^g величине k. Пошто P' чине k најмањих елемената скупа Q_j^g, он је лексикографски најмањи међу његовим подскуповима величине k. Зато је $P' = P^g$ или $P' \prec P^g$. У оба случаја из $P^g \prec P$ следи $P' \prec P$. Скупови P' и P чине првих k елемената растућих записа скупова Q_j^g и Q_j, редом, па из $P' \prec P$ следи $Q_j^g \prec Q_j$. Ово противречи избору Q_j као лексикографски најмањег елемента своје G-орбите. Према томе, P је лексикографски најмањи у својој G-орбити, па је $P = P_i$ за неког од изабраних представника P_i.

Нека су $s \in Q_j \setminus P_i$ и $h \in G_{P_i}$ произвољни. Ако би било $s^h \in P_i$, из $P_i^{h^{-1}} = P_i$ следило би $s \in P_i$, супротно избору s. Зато је $s^h \notin P_i$.

Претпоставимо да је $s^h < \max P_i$. Пошто је $P_i^h = P_i$, скуп Q_j^h садржи све елементе скупа P_i, а садржи и додатни елемент $s^h \notin P_i$ пре $\max P_i$ у растућем запису. Како су елементи скупа P_i управо првих k елемената растућег записа скупа Q_j, при првој разлици растућих записа скупова Q_j^h и Q_j мањи елемент био би у Q_j^h. Следило би $Q_j^h \prec Q_j$, што противречи минималности Q_j у његовој G-орбити. Према томе, $s^h > \max P_i$ за свако $h \in G_{P_i}$, па важи

$$\min s^{G_{P_i}} > \max P_i \quad \text{за свако } s \in Q_j \setminus P_i.$$

Следи

$$(P_i, Q_j) \in \mathrm{Up}^*(P_i),$$

па је *up*-орбита којој овај пар припада специјална. Одговарајућа грана класификационог графа зато остаје у прорећеном графу и инцидентна је представнику Q_j. □

Из претходног тврђења следи да се у фази *Lift* могу разматрати само специјалне *up*-орбите, а да се при томе не изгуби ниједна G-орбита објеката из скупа $\mathcal{Y}$.

Измене алгоритма. У односу на основни алгоритам, лексикографско прорeђивање уводи три измене:

- у фази *Lift* разматрају се само специјалне *up*-орбите,
- у кораку *Eliminate* и даље се разматрају парови из $\mathrm{Down}(Q_j)$, али се након конструктивног препознавања поступак наставља само ако добијени пар припада $\mathcal{R}^*$,
- при конструктивном препознавању објекта Y парови из $\mathrm{Down}(Y)$ разматрају се редом, док се не пронађе пар за који се након препознавања прве координате добија пар из специјалног *up*-скупа.

Размотримо детаљније измену корака *Eliminate*. За сваки пар

$$(X, Q_j) \in \mathrm{Down}(Q_j)$$

конструктивним препознавањем најпре се одређују представник P_i и елемент $h_1 \in G$ за који је

$$X^{h_1} = P_i.$$

Тада је

$$(X, Q_j)^{h_1} = (P_i, Q_j^{h_1}) \in \mathrm{Up}(P_i).$$

Поступак се наставља само ако важи

$$(P_i, Q_j^{h_1}) \in \mathcal{R}^*.$$

Ако овај услов није испуњен, *up*-орбита која садржи $(P_i, Q_j^{h_1})$ није специјална, па одговарајућа грана није задржана у прорeђеном класификационом графу. У том случају се прелази на следећи пар из $\mathrm{Down}(Q_j)$. Показаћемо у наставку да се овим прескакањем не пропушта ниједан задржани кандидат којег је потребно елиминисати.

Важно је да се услов специјалности проверава тек након конструктивног препознавања прве координате. Услов

$$\min s^{G_X} > \max X$$

зависи од фиксираног линеарног поретка покривача. Произвољна колинеација не мора да чува тај поредак: из $s_1 < s_2$ не мора да следи $s_1^g < s_2^g$. Зато специјалност пара не мора бити сачувана применом произвољног елемента

групе G. Из тог разлога не проверавамо да ли је полазни пар (X, Q_j) специјалан. Најпре одређујемо h_1 за који је $X^{h_1} = P_i$, а затим услов специјалности проверавамо за пар $(P_i, Q_j^{h_1})$.

Покажимо и да овако модификован корак *Eliminate* и даље проналази сваки други задржани кандидат из G-орбите текућег кандидата. Нека су Y и Y' два различита задржана кандидата из исте G-орбите, при чему је

$$(P_i, Y) \in \mathrm{Up}^*(P_i) \quad \text{и} \quad (P_k, Y') \in \mathrm{Up}^*(P_k).$$

Нека је $Y^g = Y'$ за неко $g \in G$ и поставимо

$$X = P_k^{g^{-1}}.$$

Пошто је $P_k \subset Y'$, применом g^{-1} добијамо $X \subset Y$, па се пар (X, Y) разматра у кораку *Eliminate*. Конструктивним препознавањем одређујемо $h_1 \in G$ за који је

$$X^{h_1} = P_k.$$

Из $X = P_k^{g^{-1}}$ следи

$$P_k^{g^{-1}h_1} = X^{h_1} = P_k,$$

па је

$$g^{-1}h_1 \in G_{P_k}.$$

Поред тога,

$$Y^{h_1} = (Y^g)^{g^{-1}h_1} = (Y')^{g^{-1}h_1}.$$

Из G_{P_k}-инваријантности скупа $\mathrm{Up}^*(P_k)$ и $(P_k, Y') \in \mathrm{Up}^*(P_k)$ зато следи

$$(P_k, Y^{h_1}) = (P_k, Y')^{g^{-1}h_1} \in \mathrm{Up}^*(P_k).$$

Зато је услов за припадност специјалној релацији $\mathcal{R}^*$ испуњен. Парови (P_k, Y^{h_1}) и (P_k, Y') припадају истој специјалној *up*-орбити, па их подаци сачувани у фази *Lift* воде до истог представника те орбите. Зато се кандидат Y' проналази и елиминише.

Према томе, прескакањем парова за које након конструктивног препознавања услов специјалности није испуњен не пропушта се ниједан други задржани кандидат из G-орбите текућег кандидата.

Одређивање стабилизатора. Проверимо да лексикографско пророређивање не спречава ни одређивање целог стабилизатора G_{Q_j}. Нека је $T_{i,a} = (P_i, Q_j)$ представник специјалне *up*-орбите одабране у кораку *Define*. За произвољно $g \in G_{Q_j}$ посматрамо пар

$$T_{i,a}^{g^{-1}} = (X, Q_j) \in \mathrm{Down}(Q_j), \qquad X = P_i^{g^{-1}}.$$

Конструктивно препознавање за X одређује представника P_i, јер је $X \in P_i^G$, и елемент $h_1 \in G$ такав да је $X^{h_1} = P_i$. Тада важи

$$P_i^{g^{-1}h_1} = X^{h_1} = P_i,$$

па је $g^{-1}h_1 \in G_{P_i}$. Из G_{P_i}-инваријантности скупа $\mathrm{Up}^*(P_i)$ следи

$$(P_i, Q_j^{h_1}) = T_{i,a}^{g^{-1}h_1} \in \mathrm{Up}^*(P_i).$$

Зато се при обради пара (X, Q_j) поступак не прекида провером припадности релацији $\mathcal{R}^*$.

Добијени пар припада истој *up*-орбити као $T_{i,a}$, па сачувани елемент $h_2 \in G_{P_i}$ задовољава

$$(P_i, Q_j^{h_1})^{h_2} = T_{i,a}.$$

Поређењем других координата добијамо $h_1h_2 \in G_{Q_j}$. Поред тога, важи $T_{i,a}^{g^{-1}h_1h_2} = T_{i,a}$, одакле следи

$$g^{-1}h_1h_2 \in G_{T_{i,a}}, \qquad h_1h_2 \in gG_{T_{i,a}}.$$

Дакле, ниједан леви косет подгрупе $G_{T_{i,a}}$ у G_{Q_j} није пропуштен. Доказ да различити обрађени парови не дају представнике истог левог косета остаје непромењен у односу на основни алгоритам. Према томе, и после лексикографског пророређивања добија се тачно по један представник сваког левог косета, што омогућава одређивање целог стабилизатора G_{Q_j}.

Конструктивно препознавање после пророређивања. У поступку из одељка 2.3 више не можемо изабрати произвољан пар из $\mathrm{Down}(Y)$. После пророређивања подаци потребни за одређивање елемената g_2 и g_3 сачувани су само за специјалне *up*-орбите.

Зато за дати објекат $Y \in \mathcal{Y}$ редом разматрамо парове $(X, Y) \in \mathrm{Down}(Y)$. Конструктивним препознавањем објекта X одређујемо представника P_i и елемент $g_1 \in G$ за који је $X^{g_1} = P_i$, а затим проверавамо услов

$$(P_i, Y^{g_1}) \in \mathrm{Up}^*(P_i).$$

Ако услов није испуњен, прелазимо на следећи пар из $\mathrm{Down}(Y)$. Чим пронађемо пар који испуњава услов, одређујемо g_2 и g_3 као у основном поступку и добијамо $Y^{g_1g_2g_3} = Q_j$, чиме је препознавање завршено.

Тврђење 16. За свако $Y \in \mathcal{Y}$ постоји пар $(X, Y) \in \mathrm{Down}(Y)$ са следећим својством: ако је $P_i \in \mathcal{X}$ изабрани представник G-орбите објекта X, онда за сваки $g_1 \in G$ за који је $X^{g_1} = P_i$ важи

$$(P_i, Y^{g_1}) \in \mathrm{Up}^*(P_i).$$

Доказ. Нека је $Y \in \mathcal{Y}$ произвољан објекат и Q_j изабрани представник његове G-орбите. Према Тврђењу 15, постоји изабрани представник $P_i \in \mathcal{X}$ за који је $(P_i, Q_j) \in \mathrm{Up}^*(P_i)$.

Изаберимо $g \in G$ тако да је $Y^g = Q_j$ и поставимо $X = P_i^{g^{-1}}$. Пошто група G дејствује на скуп $\mathcal{X}$, важи $X \in \mathcal{X}$. Из $(P_i, Q_j) \in \mathcal{R}$ и G-инваријантности релације $\mathcal{R}$ следи $(X, Y) = (P_i, Q_j)^{g^{-1}} \in \mathcal{R}$. Према томе, $(X, Y) \in \mathrm{Down}(Y)$.

Објекти X и P_i припадају истој G-орбити, јер је $X = P_i^{g^{-1}}$. Зато је P_i изабрани представник G-орбите објекта X. Узмимо произвољан $g_1 \in G$ за који је $X^{g_1} = P_i$. Тада важи

$$P_i^{g^{-1}g_1} = \left(P_i^{g^{-1}}\right)^{g_1} = X^{g_1} = P_i,$$

па је $g^{-1}g_1 \in G_{P_i}$.

Из $Y^g = Q_j$ следи $Q_j^{g^{-1}g_1} = Y^{g_1}$. Према Тврђењу 14, скуп $\mathrm{Up}^*(P_i)$ је G_{P_i}-инваријантан, па добијамо

$$(P_i, Y^{g_1}) = (P_i, Q_j)^{g^{-1}g_1} \in \mathrm{Up}^*(P_i).$$

Пошто је g_1 био произвољан елемент за који је $X^{g_1} = P_i$, доказано је тражено својство. □

У имплементацији описаној у наредној глави узимамо $\ell = k + 1$. Тада је сваки елемент скупа $\mathrm{Up}(X)$ облика

$$(X, X \cup \{s\}),$$

па се услов за припадност специјалном *up*-скупу своди на

$$(X, X \cup \{s\}) \in \mathrm{Up}^*(X) \quad \Leftrightarrow \quad (X, X \cup \{s\}) \in \mathrm{Up}(X) \text{ и } \min s^{G_X} > \max X.$$

Прелаз са величине 0 на величину 1 обрађује се посебно и при њему ово прорећивање није потребно. За наредне величине алгоритам се примењује узастопно, додавањем по једног покривача.

Глава 3

Имплементација

3.1 Константе простора

На почетку имплементације дефинишу се основни параметри пројективног простора $\mathrm{PG}(n, q)$. У нашем програму ови параметри су `n` – димензија простора, и `q` – број елемената поља $\mathbb{F}_q$. У овом раду посматрамо само тродимензионалне просторе, па је $n = 3$. За параметар q разматрамо вредности 2 и 3, односно просторе $\mathrm{PG}(3, 2)$ и $\mathrm{PG}(3, 3)$.

```
const int n = 3;          // dimenzija prostora (PG(3,q))
const int q = 2;          // broj elemenata polja (2 ili 3)
```

Број тачака у пројективном простору $\mathrm{PG}(n, q)$ познат је из теоријског дела рада и дат је формулом

$$|\mathrm{PG}(n, q)| = \frac{q^{n+1} - 1}{q - 1}.$$

Број тачака рачунамо функцијом `stepen` и чувамо у константи `BR_TACAKA`, коју користимо као аргумент шаблона `std::bitset<BR_TACAKA>`. Број битова у типу `std::bitset` мора бити познат током компилације, па аргумент `BR_TACAKA` мора бити константан израз.

```
constexpr int stepen(int b, int e) {
    int r = 1;
    for (int i = 0; i < e; ++i) r *= b;
    return r;
}

constexpr int BR_TACAKA = (stepen(q, n + 1) - 1) / (q - 1);
```

Функција `stepen(b,e)` рачуна степен b^e. Ознака `constexpr` омогућава да се њена вредност израчуна током компилације. При приказаној иницијализацији константе `BR_TACAKA` израчунавање се управо тада и обавља. Ознака `constexpr` уведена је стандардом C++11, али је тек од стандарда C++14 дозвољено да овако означене функције садрже петље и локалне променљиве. Зато је за компилацију приказаног програма потребан стандард C++14 или новији, на пример `-std=c++17`.

Слично, можемо да дефинишемо и број правих, који нам је такође познат из теоријског дела рада.

```
constexpr int BR_PRAVIH = (BR_TACAKA * (BR_TACAKA-1)) / ((q+1) * q);
```

Пошто су вредности n и q дефинисане као константе, променом једне вредности у изворном коду (`q = 2` или `q = 3`) програм се може поново компајлирати и добити све параметре специфичне за изабрано поље. Све зависне величине, као што су број тачака, број правих и величине `bitset`-ова, ажурирају се током компилације без додатног трошка током извршавања.

3.2 Представљање и уређење објеката

Тачке представљамо векторима координата, а праве, покриваче и парцијална паковања векторима индекса њихових елемената. За целобројне векторе исте дужине користимо лексикографски поредак $\prec$ уведен у одељку 2.4.

Ово уређење користимо у целој имплементацији. Низове тачака, правих и покривача, као и низове представника парцијалних паковања, чувамо у лексикографски растућем поретку. Индексе тачака у правој, правих у покривачу и покривача у парцијалном паковању такође записујемо растуће. Тиме сваки од ових скупова добија јединствен векторски запис, па не генеришемо и не чувамо различите пермутације истог објекта. Уређени низови омогућавају и проналажење индекса бинарном претрагом без додатних структура за претраживање, чиме се смањују време извршавања и меморијска потрошња.

3.3 Генерисање тачака

Тачке пројективног простора $\mathrm{PG}(n, q)$ генеришемо у хомогеним координатама. Подсетимо се да се у хомогеним координатама свака тачка простора $\mathrm{PG}(n, q)$ може представити као ненула вектор $(x_0, x_1, \ldots, x_n)$ над пољем $\mathbb{F}_q$, где

се вектори који се разликују за ненула скаларни множилац сматрају еквивалентним, тј. представљају исту тачку. Стога, у програму тачке представљамо као векторе дужине $n+1$, чије су координате елементи поља $\mathbb{F}_q$. Чувамо их у глобалној структури података

```
vector<vector<int>> tacke;
```

која садржи представнике свих тачака простора. За коришћење те структуре потребно је да укључимо библиотеку `vector`.

Пошто је у хомогеним координатама свака тачка одређена до множења ненула скаларом, потребно је изабрати по једног канонског представника за сваку класу. У овом програму као канонска се узима она тачка чија је **прва ненула координата једнака 1**, гледано слева надесно. На тај начин се добија једнозначан облик сваке тачке. Ова нормализација је могућа јер сваки ненула елемент поља има инверз. Ако је x_i прва ненула координата вектора $x = (x_0, \ldots, x_n)$, онда вектор $x_i^{-1}x$ припада истој класи као x, а његова прва ненула координата једнака је 1.

Све тачке у канонском облику генеришемо следећом функцијом:

```
void generisi_tacke() {
    vector<int> tacka(n + 1, 0);
    for (int i(n); i >= 0; --i) {
        tacka[i] = 1;  // prva nenula koordinata je 1
        popuni_rep_od(i + 1, tacka);
        tacka[i] = 0;
    }
}
```

Функција `generisi_tacke` иницијализује вектор дужине $n+1$ и поставља координате тако да се у сваком кораку добија нови канонски представник. Петља која поставља прву ненула вредност иде уназад, од последње координате ка првој, што нам омогућава да све комбинације генеришемо у лексикографски растућем поретку. У сваком кораку се једна координата фиксира на 1 (прва ненула координата), након чега се рекурзивно попуњавају преостале координате десно од те јединице, помоћу помоћне функције `popuni_rep_od`.

Помоћна функција дефинисана је на следећи начин:

```
void popuni_rep_od(int poz, vector<int> & tacka) {
    if (poz > n) tacke.push_back(tacka);
    else
        for (int k(0); k < q; ++k) {
            tacka[poz] = k;
            popuni_rep_od(poz + 1, tacka);
        }
}
```

Функција `popuni_rep_od` се рекурзивно позива за све координате почев од позиције `poz` надесно. Када се достигне крај вектора, тренутни вектор се додаје у глобални низ `tacke`. На овај начин добијају се све могуће комбинације координата које одговарају тачкама у $\mathrm{PG}(n, q)$, при чему су оне лексикографски сортиране захваљујући редоследу којим координате попуњавамо и редоследу вредности којима их попуњавамо.

Овакав поступак гарантује да ће свака тачка бити представљена једним канонским вектором, јер се свака хомогена класа генерише само једном – управо она чија је прва ненула координата 1. Истовремено, како се рекурзивно вредности k у петљи мењају редом од 0 до $q - 1$, добијене тачке су сортиране у лексикографском поретку, јер ће свака наредна тачка која се уписује у вектор `tacke` имати већу вредност на позицији `poz` или на некој позицији лево у односу на њу.

Број генерисаних тачака одговара познатој вредности $|\mathrm{PG}(n, q)|$, тако да је сложеност ове функције линеарна у односу на тај број. За $\mathrm{PG}(3, 2)$ добија се 15 тачака, а за $\mathrm{PG}(3, 3)$ укупно 40. Стога, и временски и просторни захтев овог дела програма је занемарљив.

Тачке ћемо генерисати једном, позивом функције `generisi_tacke`, на почетку главног програма. Након што су све тачке пројективног простора генерисане и сачуване у вектору `tacke`, потребно је обезбедити начин да се било која тачка препозна и пронађе у том низу. У ту сврху дефинисана је функција `nadji_tacku`, која враћа индекс тражене тачке у вектору `tacke`:

```
int nadji_tacku(const vector<int> & kanonizovana) {
    return lower_bound(tacke.begin(), tacke.end(), kanonizovana)
                                                    - tacke.begin();
}
```

Пошто су тачке у вектору `tacke` већ генерисане у лексикографски растућем поретку, ова функција користи стандардну C++ функцију `std::lower_bound` библиотеке `algorithm`, која на сортираном низу врши бинарну претрагу. Аргумент функције `nadji_tacku` представља канонски облик тражене тачке, тј. вектор чија је прва ненула координата једнака јединици. Функција `lower_bound` враћа итератор на први елемент у низу који није мањи од тражене тачке у лексикографском поретку, што у нашем случају одговара управо позицији тражене тачке. Одузимањем почетног итератора `tacke.begin()` добија се целобројни индекс те тачке у вектору. Функцију је могуће позвати јер су тачке сортиране растуће у лексикографском поретку.

Број поређења при бинарној претрази је логаритамски. За 15, односно 40 тачака, у најгорем случају потребна су 4, односно 6 поређења, па је временски трошак извршавања ове функције занемарљив. На овај начин, функција `nadji_tacku` омогућава да се било која тачка у канонском облику готово моментално пронађе у глобалном низу тачака. Ово ће нам бити корисно јер ћемо у наставку тачке представљати целим бројевима, тј. њиховим индексима у низу `tacke`.

Када нам се у произвољном облику јави тачка чији индекс у глобалном низу `tacke` желимо да нађемо, требаће нам функција која налази канонски облик те тачке да бисмо могли да применимо функцију `nadji_tacku`. Као што смо већ објаснили, за то је потребно „скалирање" вектора фактором $k = x^{-1}$, где је x прва ненула координата. Дакле, и инверзи $x^{-1} \in \mathbb{F}_q$ ће нам бити потребни.

Знамо да је $\mathbb{F}_q$ поље за све q који су степен простог броја. Пошто у програму разматрамо само $q \in \{2, 3\}$, а оба ова броја су проста, можемо да искористимо чињеницу да у том случају важи $\mathbb{F}_q \cong \mathbb{Z}/q\mathbb{Z} \cong \{0, 1, \ldots, q-1\}$, што нам омогућава да користимо уобичајену аритметику по модулу q. За све $x \in \{1, 2, \ldots, q-1\}$ можемо једним проласком да унапред израчунамо инверзе x^{-1}, користећи рекурентну формулу која нам је позната из теоријског дела.

$$1^{-1} = 1, \qquad x^{-1} = \left(q - \left\lfloor \frac{q}{x} \right\rfloor\right) \cdot (q \bmod x)^{-1},$$

где $q \bmod x$ означава остатак при дељењу q са x, и самим тим мора бити мањи од x. Имплементација је следећа:

```
vector<int> inv(q);  // x*inv[x] = 1 (mod q)
```

```
void generisi_inverze() {
    inv[1] = 1;
    for (int i(2); i < q; ++i)
        inv[i] = ((q - q / i) * inv[q % i]) % q;
}
```

Инверз од 0 не постоји, па га ни не дефинишемо. За мале вредности q које разматрамо, могуће је и ручно унети све вредности инверза. У оба случаја временски и меморијски трошак је занемарљив. Приказани поступак важи за просте вредности q, што обухвата оба случаја која разматрамо.

Сада можемо да имплементирамо канонизацију тачке у смислу који смо описали – множимо тачку са $k = x_i^{-1}$, где је x_i прва ненула координата.

```
void kanonizuj(vector<int> & tacka) {
    int i(0); // prva nenula koordinata
    while (!(tacka[i] % q)) ++i;
    int k = inv[tacka[i]];
    for (; i <= n; ++i)
        tacka[i] = (tacka[i] * k) % q;
}
```

Ова функција претвара сваку тачку у њен једнозначни канонски облик (у ком је прва ненула координата једнака 1). Обезбеђује нам доследност и могућност ефикасног проналажења у сортираном низу `tacke` помоћу функције `nadji_tacku`. Поступак је линеарне сложености по дужини вектора, што је за мале вредности n занемарљиво.

3.4 Генерисање правих

Сада ћемо описати поступак конструисања свих правих у пројективном простору $\mathrm{PG}(n, q)$, као и структуре података које бележе њихову инциденцију са тачкама. Права у $\mathrm{PG}(n, q)$ садржи тачно $q+1$ тачака. Сваке две различите тачке тог простора одређују јединствену праву. У имплементацији, сваку праву представљамо као $(q+1)$-торку индекса тачака из глобалног вектора `tacke`, уз пратећи битовски вектор инциденције.

Следе глобалне структуре:

```
vector<vector<int>> prave;  // spisak tacaka svake prave
vector<bitset<BR_TACAKA>> bitovi_pravih;
```

Функција `generisi_prave` гради све праве „провлачењем" кроз парове различитих тачака A, B (индекси `i<j`) и додавањем преосталих $q-1$ тачака облика $A+tB$ за $t \in \{1, \ldots, q-1\}$, уз обавезну канонизацију и проналажење индекса:

```
void generisi_prave() {
    vector<int> prava(q + 1);
    vector<int> tacka(n + 1);

    generisi_inverze();

    //"provlacimo" pravu kroz dve tacke i, j
    for (int i(0); i < (int)tacke.size() - q; ++i)
        for (int j(i + 1); j < (int)tacke.size() - q + 1; ++j) {
            bool nova = true;
            // [B], [A + t*B] (0 <= t < q)
            for (int t = 1; t < q && nova; ++t) {
                for (int k(0); k <= n; ++k)
                    tacka[k] = (tacke[i][k] + t * tacke[j][k]) % q;

                kanonizuj(tacka);
                int id = nadji_tacku(tacka);
                      // hocemo da i, j budu "najmanje" tacke na pravoj
                if (id > j) prava[t + 1] = id;
                else nova = false;
            }

            if (nova) {
                prava[0] = i;
                prava[1] = j;
                sort(prava.begin() + 2, prava.end());
                prave.push_back(prava);

                bitset<BR_TACAKA> bitset_prave;
                for(int t : prava) bitset_prave.set(t);
                bitovi_pravih.push_back(move(bitset_prave));
            }
        }
}
```

За сваки уређени пар `(i,j)` за који важи `i<j`, узимају се тачке $A =$ `tacke[i]` и $B =$ `tacke[j]`. За свако $t \in \{1, \ldots, q-1\}$ израчунава се канонски представник тачке $A + tB$, односно индекс `id` те тачке међу свим тачкама вектора `tacke`. На тај начин, уз већ познате A и B, добија се свих

$q+1$ тачака праве која пролази кроз те две тачке.

Да би се иста права појавила тачно једном, намећемо услов да су при проласку *две најмање* тачке на правој управо `i` и `j`. Томе служи услов `id > j`. Кад се за неко t добије `id` $\leqslant$ `j`, то значи да постоји „мањи" пар тачака `(i',j')`, од којих је једна `id`, на истој правој који бисмо раније нашли, па се текућа права прескаче као већ обрађена (`nova = false`). Када услов да су `i` и `j` најмање тачке праве важи на целој правој, сортирамо и преостале тачке праве растуће, и тада се у вектор `prave` уписује $(q+1)$-торка `[i,j,...]`. Како петље пролазе по индексима `i` и `j` растуће, и како су све остале тачке на правој по конструкцији веће од њих, низ `prave` се формира у лексикографском поретку по уређеним $(q+1)$-торкама тачака. Услов `id > j` осигурава да се иста права неће поновити више пута са другачијим избором почетног пара, па се свака права појављује тачно једном. Резултат је потпун и јединствен низ правих.

За сваку праву формира се и битовска маска `bitset<BR_TACAKA>`. Бит на позицији `t` је 1 ако и само ако је тачка `t` на посматраној правој. Ова представа омогућава ефикасне операције које ћемо користити касније за проверу инциденције. Конкретно, провером једног бита утврђујемо да ли права садржи дату тачку, а битовском конјункцијом са маском већ покривених тачака проверавамо да ли се сече са неком од већ изабраних правих.

3.5 Генерисање покривача

Покривач је скуп правих који покрива скуп свих тачака пројективног простора, при чему ниједна тачка није покривена више од једном правом. У овом делу описујемо алгоритам за генерисање свих таквих покривача у простору, имплементиран помоћу функције `generisi_pokrivace` и помоћне рекурзивне функције `popuni_pokrivace`. За чување резултата користимо глобалне структуре података сличне претходним, које бележе сваки покривач као низ индекса правих и одговарајући битовски вектор инциденције (који има бит 1 на позицији `p` ако и само ако покривач садржи праву p). Глобалне променљиве за покриваче дефинишемо на следећи начин:

```
// spisak pravih svakog pokrivaca
vector<vector<int>> pokrivaci;

// bitovski vektori incidencije pokrivaca i pravih
vector<bitset<BR_PRAVIH>> bitovi_pokrivaca;
```

Два покривача немају заједничку праву ако и само ако битовска конјункција њихових маски нема постављених битова. Ова провера се касније користи при одлучивању да ли се покривач може додати парцијалном паковању.

Функција `generisi_pokrivace` иницијализује процес тако што креира празан покривач и скуп покривених тачака, затим позива рекурзивну функцију која попуњава покриваче праву по праву. Имплементација ових функција дата је у наставку:

```
void popuni_pokrivace(vector<int> &pokrivac,
                                  bitset<BR_TACAKA> pokrivene_tacke) {
    if (pokrivene_tacke.all()) {
        bitset<BR_PRAVIH> bitset_pokrivaca;
        for (int p : pokrivac) bitset_pokrivaca.set(p);
        bitovi_pokrivaca.push_back(move(bitset_pokrivaca));
        pokrivaci.push_back(pokrivac);
    } else {
        int tacka = nadji_prvu_slobodnu_tacku(pokrivene_tacke);
        for (int p(0); p < prave.size(); ++p)
            if (bitovi_pravih[p].test(tacka) &&
                        !(bitovi_pravih[p] & pokrivene_tacke).any()) {
                         /* ako prava prolazi kroz prvu slobodnu tacku
            i ne prolazi kroz pokrivene tacke onda dodajemo tu pravu */
                pokrivac.push_back(p);
                popuni_pokrivace(pokrivac,
                                  pokrivene_tacke | bitovi_pravih[p]);
                pokrivac.pop_back();
            }
    }
}

void generisi_pokrivace() {
    bitset<BR_TACAKA> pokrivene_tacke;
    vector<int> pokrivac;
    popuni_pokrivace(pokrivac, pokrivene_tacke);
}
```

Посматрајмо најпре базни случај рекурзије у функцији `popuni_pokrivace`. Услов `pokrivene_tacke.all()` проверава да ли су све тачке већ покривене. Ако јесу, изабране праве покривају све тачке, па тренутни вектор `pokrivac` представља један покривач. Тада се тај покривач чува. Формира се његов

битовски приказ `bitset_pokrivaca` тако што се за сваку праву из покривача поставља одговарајући бит. Затим се и битовски вектор и сам вектор индекса правих тог покривача додају у глобалне низове `bitovi_pokrivaca` и `pokrivaci`. На тај начин бележимо једно решење – један комплетан покривач.

У супротном (ако још има непокривених тачака), прелази се на рекурзивни корак. Прво се одређује „најмања" непокривена тачка, помоћу функције `nadji_prvu_slobodnu_tacku`. Ова помоћна функција проналази најмањи индекс тачке чији бит у битсет маски `pokrivene_tacke` није постављен.

```
int nadji_prvu_slobodnu_tacku(const bitset<BR_TACAKA> & pokrivene) {
        for (int i(0); i < tacke.size(); ++i)
                if (!pokrivene.test(i))
                        return i;
        return -1;
}
```

И та тачка мора да буде покривена неком правом покривача, стога даље тражимо праву која може да је покрије. Тачније, све праве које садрже ту тачку и дисјунктне су са до сада изабраним правама разматрамо редом по растућим индексима. Најпре се разматра права са најмањим индексом, а након повратка из рекурзивног позива разматрају се и остале такве праве. Та дисјунктност је еквивалентна услову да права не садржи ниједну од до сада покривених тачака. Само ако су оба услова испуњена, праву додајемо у текући покривач. Тада обједињујемо њене тачке са већ покривеним тачкама (`pokrivene_tacke | bitovi_pravih[p]`), и рекурзивно позивамо `popuni_pokrivace` са овим проширеним скупом покривених тачака и проширеним покривачем. На повратку из рекурзивног позива уклања се додата права и враћа стање покривача на претходни корак да би се испробале друге могућности, све док се не испробају све праве које покривају дату тачку. Када се тиме исцрпе све опције за текућу непокривену тачку, рекурзија се враћа на претходни ниво.

Приметимо да функција `popuni_pokrivace` на рекурзиван начин систематски генерише све комбинације правих које покривају све тачке без преклапања. Рекурзија је природан начин за претраживање овако великог простора решења. Сваки рекурзивни позив сужава проблем на све мањи скуп непокривених тачака, док се на крају не достигне тривијални случај (нема непокривених тачака, тј. све су покривене – покривач је комплетан). Такав приступ сече грану претраге кад год установи да је немогуће конструисати

валидан покривач (ако ниједна доступна права не може покрити неку тачку без преклапања), чиме се избегава непотребно испробавање. Избегнуто је преклапање правих унутар једног покривача захваљујући наведеном услову: нова права се додаје само ако не садржи ниједну тачку која је већ покривена другом правом. То значи да су све изабране праве у покривачу дисјунктне, па је свака тачка покривена тачно једном правом, што је у складу са дефиницијом покривача. На тај начин алгоритам генерише искључиво исправне покриваче. Алгоритам генерише и све покриваче: за сваки покривач, у сваком кораку најмања непокривена тачка припада тачно једној његовој правој, а петља разматра и ту праву. Понављањем овог аргумента следи да постоји рекурзивна путања којом се добија тај покривач.

Тако конструисани покривачи појављују се сортирани у лексикографском поретку (по растућим индексима правих). У сваком рекурзивном кораку изабрана права садржи најмању непокривену тачку и, због услова дисјунктности, не садржи ниједну мању тачку. После њеног додавања најмања преостала непокривена тачка има већи индекс, па због лексикографске уређености низа `prave` и следећа изабрана права има већи индекс. Пошто петља на сваком нивоу разматра кандидате по растућим индексима, добијени вектори покривача генеришу се у лексикографском поретку. За фиксиран покривач, најмања непокривена тачка припада тачно једној правој тог покривача, јер његове праве покривају све тачке и међусобно су дисјунктне. Зато је у сваком кораку јединствено одређена права тог покривача која се следећа додаје, па исти покривач не може бити генерисан различитим редоследима. Резултат је потпун и јединствен лексикографски уређен низ покривача, уписан у вектор `pokrivaci` (са одговарајућим битовима у `bitovi_pokrivaca`), који ће се даље користити у наредним корацима имплементације.

3.6 Колинеације и њихово дејство

Представљање колинеација

Пошто су тачке у програму означене индексима $0, \ldots,$ `BR_TACAKA` $- 1$, колинеацију чувамо као низ у коме се на позицији `t` налази индекс слике тачке са индексом `t`. Како је колинеација бијекција скупа тачака, овај низ је пермутација њихових индекса. За тип низа којим представљамо колинеацију уводимо

назив `Kolineacija`:

```
using Kolineacija = array<uint8_t, BR_TACAKA>;
```

Тип `uint8_t` заузима један бајт, што је довољно за индексе тачака у оба простора која разматрамо: $\mathrm{PG}(3,2)$ има 15, а $\mathrm{PG}(3,3)$ има 40 тачака. За разлику од динамичког вектора, структура `array` има унапред познату дужину и не захтева посебну динамичку алокацију за своје елементе. Овакво представљање знатно смањује меморију потребну за чување великог броја колинеација. За коришћење типова `array` и `uint8_t` укључујемо заглавља `array` и `cstdint`.

У наставку су приказане функције за одређивање инверза и композиције колинеација, као и променљива `id`, намењена чувању идентичког пресликавања:

```
Kolineacija napravi_inverz(const Kolineacija& f) {
    Kolineacija inverz{};
    for (int t(0); t < BR_TACAKA; ++t)
        inverz[f[t]] = t;
    return inverz;
}

Kolineacija id{};

Kolineacija kompozicija(const Kolineacija& f, const Kolineacija& g) {
    Kolineacija h{};
    for (int t(0); t < BR_TACAKA; ++t)
        h[t] = f[g[t]];
    return h;
}
```

Пошто је колинеација `f` пермутација индекса тачака, из `f[t]` = `s` следи да инверзна колинеација индекс `s` слика у `t`. Зато функција `napravi_inverz` поставља `inverz[f[t]]` = `t` за сваки индекс `t`.

Идентичко пресликавање `id`, које је неутрални елемент групе колинеација, пре употребе задајемо у главном програму постављањем `id[t]` = `t` за све индексе тачака. Функција `kompozicija(f,g)` враћа колинеацију $f \circ g$, јер индекс `t` најпре пресликава помоћу `g`, а затим добијени индекс помоћу `f`. Према конвенцији из прве главе, производ fg означава композицију $g \circ f$, па се у програму добија позивом `kompozicija(g,f)`.

Дејство колинеација на праве и покриваче

При одређивању орбита потребно је више пута израчунавати слике правих и покривача под датом колинеацијом. Пошто колинеацију представљамо њеним дејством на индексе тачака, најпре одређујемо слику праве, а затим помоћу ње и слику покривача.

```
int slika_prave(int prava_id, const Kolineacija & kolineacija) {
    vector<int> nova_prava;

    for (int tacka : prave[prava_id])
        nova_prava.push_back(kolineacija[tacka]);

    sort(nova_prava.begin(), nova_prava.end());

    return lower_bound(prave.begin(), prave.end(), nova_prava)
                                                        - prave.begin();
}

int slika_pokrivaca(int pokrivac_id, const Kolineacija & kolineacija) {
    vector<int> novi_pokrivac;

    for (int prava : pokrivaci[pokrivac_id])
        novi_pokrivac.push_back(slika_prave(prava, kolineacija));

    sort(novi_pokrivac.begin(), novi_pokrivac.end());

    return lower_bound(pokrivaci.begin(),
                       pokrivaci.end(),
                       novi_pokrivac)
           - pokrivaci.begin();
}
```

Нека је `prava_id` индекс посматране праве. За сваки индекс тачке `tacka` на тој правој, вредност `kolineacija[tacka]` представља индекс њене слике. У одељку 1.2 показано је да колинеација слика праву у праву, па добијени индекси представљају све тачке једне праве. Те индексе сортирамо растуће како бисмо добили исти векторски запис који користимо за праве у вектору `prave`.

Према начину представљања из одељка 3.2, вектор `prave` садржи сваку праву тачно једном, а његови елементи поређани су растуће према лексикографском поретку. Зато функција `lower_bound` проналази позицију добијене

праве и `slika_prave` враћа њен индекс.

Слика покривача одређује се применом функције `slika_prave` на сваку његову праву. Како колинеације сликају покриваче у покриваче, добијени индекси представљају праве једног покривача. Њиховим сортирањем добијамо канонски векторски запис тог покривача. Вектор `pokrivaci` садржи сваки покривач тачно једном и његови елементи поређани су растуће према лексикографском поретку, па се индекс слике поново проналази функцијом `lower_bound`.

У оба случаја знамо да се тражени објекат налази у одговарајућем вектору, јер су претходно генерисане све праве и сви покривачи, а колинеације их пресликавају у објекте исте врсте. Зато након позива функције `lower_bound` није потребна додатна провера једнакости.

Разлагање групе колинеација у првом кораку

Пошто у програму разматрамо просторе $\mathrm{PG}(3, q)$, одговарајући векторски простор је $\mathbb{F}_q^4$. За $q \in \{2, 3\}$ број q је прост, па из резултата одељка 1.2 следи да је група свих колинеација $G = \mathrm{PGL}(4, q)$.

У првом кораку прелазимо са парцијалних паковања величине 0 на парцијална паковања величине 1. Једино парцијално паковање величине 0 је празан скуп, а парцијална паковања величине 1 одређена су појединачним покривачима. Пошто свака колинеација фиксира празан скуп, његов стабилизатор је цела група G. Зато у овом кораку треба одредити орбите свих покривача под дејством целе групе колинеација.

Општа функција `korak`, при преласку са парцијалних паковања величине k на парцијална паковања величине $k + 1$, користиће експлицитно сачуване стабилизаторе представника величине k. Да бисмо исти приступ применили за $k = 0$, морали бисмо као стабилизатор празног парцијалног паковања да сачувамо целу групу G. У случају $\mathrm{PG}(3, 3)$ она има 12 130 560 елемената. Како њено експлицитно чување није потребно, овај прелаз је издвојен у функцију `prvi_lift`, која пролази кроз све елементе групе, али их не чува истовремено.

Групу G набрајамо помоћу стабилизатора уређеног пара тачака. Фиксирање само једне тачке оставило би већи стабилизатор. Са друге стране, код три тачке не би било довољно захтевати само да су њихове слике међусобно различите, јер би требало водити рачуна и о колинеарности. Код уређеног

пара једини услов на његове слике јесте да буду различите, а одговарајући стабилизатор има једноставан матрични опис.

Према редоследу у ком су тачке генерисане, прве две тачке су $x_0 = [(0,0,0,1)] = [e_4]$ и $x_1 = [(0,0,1,0)] = [e_3]$. Нека је

$$H = \{h \in G \mid h(x_0) = x_0,\ h(x_1) = x_1\}$$

стабилизатор уређеног пара (x_0, x_1).

За сваки уређени пар различитих тачака (x_a, x_b) бирамо по једну колинеацију $\tau_{a,b}$ такву да је $\tau_{a,b}(x_0) = x_a$ и $\tau_{a,b}(x_1) = x_b$. Нека су A и B векторски представници тачака x_a и x_b. Пошто су ове тачке различите, вектори A и B су линеарно независни. Зато се уређени парови (e_4, e_3) и (A, B) могу допунити до база простора $\mathbb{F}_q^4$. Инвертибилно линеарно пресликавање које прву од изабраних база слика у другу индукује колинеацију са траженим сликама тачака x_0 и x_1.

Нека је $c \in G$ произвољна колинеација и нека су a и b јединствени индекси за које је $c(x_0) = x_a$ и $c(x_1) = x_b$. Из инјективности колинеације c следи $a \neq b$. За изабрану колинеацију $\tau_{a,b}$ поставимо $h = \tau_{a,b}^{-1} \circ c$. Тада је $h(x_0) = \tau_{a,b}^{-1}(x_a) = x_0$, а аналогно је и $h(x_1) = x_1$. Према томе, $h \in H$ и $c = \tau_{a,b} \circ h$.

Ово представљање је јединствено. Индекси a и b одређени су сликама тачака x_0 и x_1. Када су они познати, колинеација $\tau_{a,b}$ већ је изабрана, а затим је и $h = \tau_{a,b}^{-1} \circ c$ јединствено одређена. Зато важи дисјунктно разлагање

$$G = \bigsqcup_{\substack{0 \leqslant a,b < \texttt{BR_TACAKA} \\ a \neq b}} \{\, \tau_{a,b} \circ h \mid h \in H \,\}.$$

Проласком кроз све уређене парове различитих тачака и све елементе групе H добијамо, дакле, сваку колинеацију тачно једном. У наставку описујемо генерисање елемената групе H и изабраних колинеација $\tau_{a,b}$.

Генерисање стабилизатора уређеног пара

У теоријском делу показано је да сваки елемент групе $\mathrm{PGL}(4, q)$ може да се представи инвертибилном матрицом реда 4, при чему две такве матрице одређују исту колинеацију ако и само ако се разликују за ненула скаларни множилац. Нека је $u \in H$ и нека је M матрица која представља колинеацију u.

Пошто u фиксира тачку $x_0 = [e_4]$, вектор Me_4 мора бити ненула скаларни умножак вектора e_4. Како је Me_4 четврта колона матрице M, она је облика λe_4, где је $\lambda \neq 0$. Множењем целе матрице са λ^{-1} бирамо представника исте колинеације чија је четврта колона једнака e_4. Овај представник је јединствен, јер би даље множење ненула скаларом променило његову четврту колону.

Тачка $x_1 = [e_3]$ такође је фиксирана, па трећа колона овако нормализоване матрице мора бити облика Ke_3, где је $K \neq 0$. Према томе, сваки елемент групе H има јединственог матричног представника облика

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ e & f & K & 0 \\ g & h & 0 & 1 \end{bmatrix}.$$

Детерминанта ове матрице једнака је $K(ad - bc)$. Зато се параметри a, b, c, d бирају тако да је $ad - bc \neq 0$, параметар K као произвољан ненула елемент поља, док су e, f, g, h произвољни елементи поља $\mathbb{F}_q$.

Број инвертибилних горњих левих блокова реда 2 једнак је $(q^2-1)(q^2-q)$. Прва колона може бити произвољан ненула вектор, док се друга бира изван једнодимензионалног потпростора генерисаног првом колоном, који садржи q вектора. За параметре e, f, g, h постоји q^4 избора, а за K постоји $q-1$ избора. Стога је

$$|H| = (q^2-1)(q^2-q)q^4(q-1).$$

За $q = 2$ добијамо $|H| = 96$, а за $q = 3$ добијамо $|H| = 7\,776$.

Елементе групе H и њихово дејство на праве чувамо у следећим структурама:

```
vector<Kolineacija> stabilizator_0_1(
    (q * q - 1) * (q * q - q) * q * q * q * q * (q - 1));

vector<vector<int>> stab_0_1_na_pravu(
    (q * q - 1) * (q * q - q) * q * q * q * q * (q - 1),
    vector<int>(BR_PRAVIH));
```

Ознака `0_1` односи се на индексе фиксираних тачака x_0 и x_1. На позицији `s` вектора `stabilizator_0_1` чува се један елемент групе H, док `stab_0_1_na_pravu[s][p]` садржи индекс слике праве `p` под том колинеацијом. Слике правих одређујемо унапред, јер се исти елементи групе H касније примењују на велики број покривача.

Функција `generisi_stabilizator_0_1` петљама пролази кроз све описане вредности параметара. Услов `(a*d-b*c)%q` задржава инвертибилне горње леве блокове, док петља за `K` пролази кроз вредности од 1 до $q-1$. Пошто су петље кроз параметре већ описане, у наставку приказујемо само део функције који за сваку добијену матрицу израчунава њено дејство на тачке и праве:

```
for (int i(0); i < BR_TACAKA; ++i) {
    tacka[0] = (a*tacke[i][0] + b*tacke[i][1]) % q;
    tacka[1] = (c*tacke[i][0] + d*tacke[i][1]) % q;
    tacka[2] = (e*tacke[i][0] + f*tacke[i][1] + K*tacke[i][2]) % q;
    tacka[3] = (g*tacke[i][0] + h*tacke[i][1] + tacke[i][3]) % q;

    kanonizuj(tacka);
    kolineacija[i] = nadji_tacku(tacka);
}

stabilizator_0_1[s] = kolineacija;

for (int i(0); i < BR_PRAVIH; ++i)
    stab_0_1_na_pravu[s][i] = slika_prave(i, kolineacija);

s++;
```

Четири израза којима се попуњава вектор `tacka` представљају множење канонског представника тачке матрицом M. Пошто је M инвертибилна, ненула вектор се пресликава у ненула вектор, па затим можемо да применимо функцију `kanonizuj`. Функција `nadji_tacku` враћа индекс добијене пројективне тачке, који се уписује у одговарајућу позицију колинеације.

Описане петље пролазе кроз све матрице наведеног облика и сваку од њих генеришу једном. Ако би две генерисане матрице одређивале исту колинеацију, разликовале би се за ненула скаларни множилац. Међутим, њихове четврте колоне једнаке су e_4, па тај множилац мора бити једнак 1. Матрице су зато једнаке. Следи да се у вектор `stabilizator_0_1` уписују сви елементи групе H, сваки тачно једном.

Генерисање изабраних колинеација

Претходно смо показали да за сваки уређени пар различитих тачака (x_a, x_b) постоји колинеација $\tau_{a,b}$ која x_0 слика у x_a, а x_1 у x_b. Сада описујемо како програм бира једну такву колинеацију.

Нека су $A = (A_0, A_1, A_2, A_3)^\mathsf{T}$ и $B = (B_0, B_1, B_2, B_3)^\mathsf{T}$ канонски векторски представници тачака x_a и x_b, сачувани на позицијама `a` и `b` вектора `tacke`. Из $a \neq b$ следи $x_a \neq x_b$, па A и B нису скаларни умношци један другог. Зато су линеарно независни и матрица са колонама B и A има ранг 2, па бар један њен минор реда 2 није нула. Другим речима, постоје индекси редова $i < j$ за које је $B_iA_j - B_jA_i \neq 0$. Функција бира први такав пар индекса.

За $r \in \{0, 1, 2, 3\}$ означимо са $E_r = e_{r+1}$ одговарајући вектор стандардне базе. Нека су $k < l$ преостала два индекса из скупа $\{0, 1, 2, 3\}$. За изабрани пар (a, b) посматрамо матрицу

$$M_{a,b} = \begin{bmatrix} | & | & | & | \\ E_k & E_l & B & A \\ | & | & | & | \end{bmatrix}.$$

Развојем њене детерминанте по прве две колоне добија се, до на знак, вредност $B_iA_j - B_jA_i$. Она је ненула, па је матрица $M_{a,b}$ инвертибилна.

Трећа колона матрице $M_{a,b}$ једнака је B, а четврта A. Множење матрице одговарајућим вектором стандардне базе издваја ту колону, па је $M_{a,b}e_3 = B$ и $M_{a,b}e_4 = A$. Како је $x_1 = [e_3]$, а $x_0 = [e_4]$, пројективно линеарно пресликавање одређено матрицом $M_{a,b}$ задовољава $\tau_{a,b}(x_1) = x_b$ и $\tau_{a,b}(x_0) = x_a$.

Изабране колинеације и њихово дејство на праве чувамо у структурама

```
vector<vector<Kolineacija>> kolineacija_0_1_u(
    BR_TACAKA,
    vector<Kolineacija>(BR_TACAKA));

vector<vector<vector<int>>> kol_0_1_na_pravu(
    BR_TACAKA,
    vector<vector<int>>(
        BR_TACAKA,
        vector<int>(BR_PRAVIH)));
```

Вредност `kolineacija_0_1_u[a][b]` представља колинеацију $\tau_{a,b}$, док `kol_0_1_na_pravu[a][b][p]` садржи индекс слике праве `p` под том колинеацијом. Позиције за које је $a = b$ не користимо, јер колинеација не може различите тачке x_0 и x_1 пресликати у исту тачку. Прве две димензије обе структуре имају дужину `BR_TACAKA`, што омогућава непосредан приступ подацима за уређени пар индекса `a` и `b`.

Функција `generisi_0_1_kolineacije` најпре пролази кроз све парове $a \neq b$, а затим на претходно описани начин одређује индексе i, j, k, l. Тај

део кода изостављамо, јер је поступак непосредно изнад објашњен. За фиксиран пар (a, b) приказујемо део функције који израчунава дејство матрице $M_{a,b}$ на све тачке и праве. Ради јаснијег разликовања од индекса i, j, k, l, у приказаном коду бројач `t` означава индекс тачке, а `r` индекс њене координате:

```
for (int t(0); t < BR_TACAKA; ++t) {
    for (int r(0); r < 4; ++r)
        tacka[r] =
            ((r == k) * tacke[t][0]
           + (r == l) * tacke[t][1]
           + tacke[b][r] * tacke[t][2]
           + tacke[a][r] * tacke[t][3]) % q;

    kanonizuj(tacka);
    kolineacija_0_1_u[a][b][t] = nadji_tacku(tacka);
}

for (int p(0); p < BR_PRAVIH; ++p)
    kol_0_1_na_pravu[a][b][p] = slika_prave(p,kolineacija_0_1_u[a][b]);
```

У језику C++ изрази `r == k` и `r == l` имају вредност 1 када је једнакост испуњена, а иначе вредност 0. Зато прва два сабирка представљају множење прве две колоне E_k и E_l одговарајућим координатама полазне тачке. Наредна два сабирка представљају множење колона B и A њеном трећом и четвртом координатом. Добијени вектор је, дакле, производ $M_{a,b}X$, где је X канонски представник полазне тачке.

Пошто је X ненула вектор, а $M_{a,b}$ инвертибилна, добијени вектор је ненула, па га можемо канонизовати и функцијом `nadji_tacku` пронаћи индекс одговарајуће пројективне тачке. На крају се за сваку праву унапред израчунава њена слика. Ова табела се касније користи при примени колинеација $\tau_{a,b}$ на покриваче, без поновног израчунавања слика њихових тачака.

Примена разлагања на покриваче

При одређивању орбита покривача потребно је за велики број колинеација израчунати слику истог покривача. Могли бисмо за сваку колинеацију најпре формирати њено дејство на тачке, а затим позвати раније описану функцију `slika_pokrivaca`. Међутим, тиме бисмо при сваком позиву поново израчунавали слике тачака сваке праве. Пошто су слике правих под елементима групе

H и колинеацијама $\tau_{a,b}$ већ сачуване, непосредно користимо одговарајуће табеле.

```
int slika_pokrivaca_stab_0_1(int pokrivac_id, int stab_0_1_id) {
    vector<int> novi_pokrivac;

    for (int prava : pokrivaci[pokrivac_id])
        novi_pokrivac.push_back(stab_0_1_na_pravu[stab_0_1_id][prava]);

    sort(novi_pokrivac.begin(), novi_pokrivac.end());

    return lower_bound(pokrivaci.begin(),
                       pokrivaci.end(),
                       novi_pokrivac)
           - pokrivaci.begin();
}

int slika_pokrivaca_kol_0_1(int pokrivac_id, int a, int b) {
    vector<int> novi_pokrivac;

    for (int prava : pokrivaci[pokrivac_id])
        novi_pokrivac.push_back(kol_0_1_na_pravu[a][b][prava]);

    sort(novi_pokrivac.begin(), novi_pokrivac.end());

    return lower_bound(pokrivaci.begin(),
                       pokrivaci.end(),
                       novi_pokrivac)
           - pokrivaci.begin();
}
```

Прва функција одређује слику покривача под елементом `stabilizator_0_1[stab_0_1_id]` групе H, док друга одређује његову слику под колинеацијом $\tau_{a,b}$. Обе функције су специјализоване варијанте функције `slika_pokrivaca`. Разлика је у томе што индекс слике сваке праве читају из унапред попуњене табеле, уместо да га поново израчунавају помоћу дејства колинеације на тачке.

Добијене индексе правих поново сортирамо како бисмо покривач записали у канонском векторском облику. Његов индекс затим проналазимо функцијом `lower_bound`, на исти начин као у раније описаној функцији `slika_pokrivaca`.

Нека је h_s = `stabilizator_0_1[s]`. За колинеацију $c = \tau_{a,b} \circ h_s$ слика покривача са индексом `i` израчунава се следећим позивом:

```
int pokrivac =
    slika_pokrivaca_kol_0_1(slika_pokrivaca_stab_0_1(i, s), a, b);

Kolineacija c =
    kompozicija(kolineacija_0_1_u[a][b], stabilizator_0_1[s]);
```

Унутрашњи позив најпре примењује h_s, а спољашњи затим примењује $\tau_{a,b}$. То одговара композицији $c = \tau_{a,b} \circ h_s$. Исти редослед важи и у позиву функције `kompozicija`, која прво примењује свој други, а затим први аргумент.

Према претходно доказаном разлагању, проласком кроз све уређене парове $a \neq b$ и све индексе s пролазимо кроз сваку колинеацију групе G тачно једном. При томе се иста слика покривача може добити више пута ако покривач има нетривијалан стабилизатор. На овај начин можемо да одредимо целу орбиту покривача, као и све колинеације које га фиксирају, што ћемо користити у функцији `prvi_lift`.

3.7 Класификовање парцијалних паковања

Подаци који се чувају током класификације

После сваког преласка са парцијалних паковања величине k на парцијална паковања величине $k+1$, добијени резултати постају улаз за наредни прелаз. Зато програм чува изабране представнике орбита, њихове стабилизаторе и податке потребне за конструктивно препознавање. За то користимо следеће структуре:

```
vector<vector<vector<int>>> predstavnici;
vector<vector<vector<Kolineacija>>> stabilizatori;

struct PodaciPrepoznavanja {
    int predstavnik = -1;
    Kolineacija preslikavanje{};
};

vector<vector<map<int,PodaciPrepoznavanja>>> podaci_prepoznavanja;
```

Ради краћег записа уведимо ознаку $P_{k,r}$ = `predstavnici[k][r]`. Вектор `predstavnici[k]` садржи изабране представнике свих G-орбита парцијалних

паковања величине k, а $P_{k,r}$ је представник са индексом `r`. Парцијално паковање $P_{k,r}$ чувамо као вектор који садржи индексе његових k покривача, поређане растуће.

Пре позива функције `prvi_lift`, у главном програму иницијализујемо представнике величине 0:

```
predstavnici.push_back({{}});
```

Тако `predstavnici[0]` садржи један елемент, празан вектор, који представља једино парцијално паковање величине 0. Његов стабилизатор је цела група G, па су орбите које се у првом кораку добијају дејством стабилизатора већ G-орбите појединачних покривача. Зато у првом кораку фаза *Classify* није потребна. Функција `prvi_lift` непосредно одређује представнике парцијалних паковања величине 1, њихове стабилизаторе и податке за конструктивно препознавање. Сваки наредни позив функције `korak` спроводи обе фазе алгоритма и додаје представнике следеће величине.

За $k \geqslant 1$, у фази *Lift* преласка са парцијалних паковања величине k на парцијална паковања величине $k+1$, вектор `stabilizatori[k-1][r]` садржи све елементе стабилизатора $G_{P_{k,r}}$. Представници и њихови стабилизатори имају исти унутрашњи индекс `r`. Померање спољашњег индекса за један постоји зато што стабилизатор празног парцијалног паковања, који је једнак целој групи G, не чувамо експлицитно. Његово дејство обрађује се посебно у функцији `prvi_lift`.

Програм при прелазу са величине $k \geqslant 1$ на величину $k+1$ чува податке за проширења представника $P_{k,r}$ из специјалног *up*-скупа. За свако такво проширење чува којој G-орбити оно припада и једну колинеацију која га слика у изабраног представника те орбите. Нека је s индекс покривача за који $(P_{k,r}, P_{k,r} \cup \{s\}) \in \mathrm{Up}^*(P_{k,r})$. Елемент `podaci_prepoznavanja[k][r][s]` садржи две информације. Члан `predstavnik` садржи индекс t одговарајућег представника $P_{k+1,t}$, док члан `preslikavanje` садржи колинеацију f за коју важи

$$\big(P_{k,r} \cup \{s\}\big)^f = P_{k+1,t}.$$

Дакле, ови подаци омогућавају да за проширење $P_{k,r} \cup \{s\}$ непосредно одредимо представника његове орбите и пресликавање у тог представника. Почетна вредност -1 члана `predstavnik` означава да током формирања ових података коначни представник још није одређен.

Пошто се за фиксиране индексе `k` и `r` чувају само подаци за претходно описана специјална проширења, користимо структуру `map`, уместо да за сваког представника резервишемо место за све покриваче.

У првом кораку је $P_{0,0} = \varnothing$, па елемент `podaci_prepoznavanja[0][0][s]` одређује представника орбите једночланог парцијалног паковања $\{s\}$ и колинеацију која $\{s\}$ слика у тог представника. Ове податке, заједно са представницима појединачних покривача и њиховим стабилизаторима, попуњава функција `prvi_lift`.

Класификовање појединачних покривача

Функција `prvi_lift` разлаже скуп свих покривача на G-орбите. За представника сваке орбите бира покривач са најмањим индексом у вектору `pokrivaci` и одређује стабилизатор изабраног представника. За сваки покривач чува по једну колинеацију која га слика у изабраног представника орбите којој припада.

```
void prvi_lift() {
    predstavnici.push_back({});
    vector<bool> nadjen(pokrivaci.size(), false);
    vector<vector<Kolineacija>> G;
    map<int,PodaciPrepoznavanja> podaci;

    for (int i(0); i < pokrivaci.size(); ++i)
        if (!nadjen[i]) {
            nadjen[i] = true;
            int indeks_predstavnika = predstavnici.back().size();
            predstavnici.back().push_back({i});
            podaci[i].predstavnik = indeks_predstavnika;
            podaci[i].preslikavanje = id;

            vector<Kolineacija> stabilizator;

            for (int a(0); a < BR_TACAKA; ++a)
                for (int b(0); b < BR_TACAKA; ++b)
                    if (a != b)
                        for (int s(0); s<stabilizator_0_1.size(); ++s){
                            int pokrivac = slika_pokrivaca_kol_0_1(
                                        slika_pokrivaca_stab_0_1(i, s),
                                        a, b);
```

```
                            if (pokrivac == i) {
                                Kolineacija c = kompozicija(
                                            kolineacija_0_1_u[a][b],
                                            stabilizator_0_1[s]);
                                stabilizator.push_back(move(c));
                            } else if (!nadjen[pokrivac]) {
                                nadjen[pokrivac] = true;
                                podaci[pokrivac].predstavnik =
                                            indeks_predstavnika;

                                Kolineacija c = kompozicija(
                                            kolineacija_0_1_u[a][b],
                                            stabilizator_0_1[s]);
                                podaci[pokrivac].preslikavanje =
                                            napravi_inverz(c);
                            }
                        }

            G.push_back(move(stabilizator));
        }

    stabilizatori.push_back(move(G));
    podaci_prepoznavanja = {{move(podaci)}};
}
```

На почетку функције у вектор `predstavnici` додајемо празан вектор у који ће бити уписани представници величине 1. У овој функцији `i` и `pokrivac` означавају индексе у глобалном вектору `pokrivaci`, док `indeks_predstavnika` означава позицију у вектору `predstavnici[1]`.

Вредност `nadjen[p]` показује да ли је покривач са глобалним индексом `p` већ пронађен у некој од обрађених орбита. За представника $P_{1,r}$, вектор `G[r]` привремено чува све елементе стабилизатора $G_{P_{1,r}}$, док мапа `podaci` садржи податке потребне за конструктивно препознавање проширења празног парцијалног паковања.

Функција пролази кроз глобалне индексе покривача растућим редом. Ако `nadjen[i]` има вредност `false`, покривач са индексом `i` не припада ниједној раније обрађеној орбити, па једночлано парцијално паковање $\{i\}$ бирамо за представника нове орбите. Његова позиција у вектору `predstavnici[1]` чува се у променљивој `indeks_predstavnika`.

За изабрани покривач затим пролазимо кроз све колинеације групе G.

Према раније описаном разлагању, свака од њих има јединствен облик $c = \tau_{a,b} \circ h_s$, па угнежђене петље кроз индексе `a`, `b` и `s` разматрају сваку колинеацију тачно једном. Променљива `pokrivac` садржи глобални индекс слике изабраног покривача под колинеацијом c.

Ако је `pokrivac == i`, колинеација c фиксира изабрани покривач, па припада његовом стабилизатору. Како се разматрају сви елементи групе G, у вектор `stabilizator` уписују се сви елементи стабилизатора једночланог парцијалног паковања $\{i\}$.

У случају `pokrivac != i`, податке чувамо само ако добијени покривач раније није пронађен, што проверава услов `!nadjen[pokrivac]`. Пошто колинеација c слика изабрани покривач са индексом `i` у покривач са индексом `pokrivac`, њен инверз слика добијени покривач назад у изабраног представника. Зато у `podaci[pokrivac].preslikavanje` чувамо `napravi_inverz(c)`. Једно такво пресликавање довољно је за конструктивно препознавање, па се при каснијим појављивањима исте слике композиција и инверз више не израчунавају.

При обради једног представника означавају се сви покривачи из његове G-орбите. Зато следећи покривач за који `nadjen` има вредност `false` припада новој орбити. Пошто глобалне индексе обилазимо растућим редом, изабрани покривач има најмањи индекс у својој орбити.

Вектор `pokrivaci` уређен је лексикографски, па је покривач са најмањим индексом уједно лексикографски најмањи покривач у орбити. Зато је једночлано парцијално паковање $\{i\}$ лексикографски најмањи представник те орбите. Изабрани индекси покривача расту, па су и елементи вектора `predstavnici[1]` поређани лексикографски.

Стабилизатори се додају истим редом којим се представници бирају, па након обраде представника $P_{1,r}$ вектор `G[r]` садржи све елементе стабилизатора $G_{P_{1,r}}$. На крају функције вектор `G` премешта се у `stabilizatori[0]`, док се мапа `podaci` смешта на позицију `podaci_prepoznavanja[0][0]`, која одговара јединственом представнику $P_{0,0} = \varnothing$.

Општи корак алгоритма

Након класификовања појединачних покривача, сваки наредни прелаз обавља функција `korak`. Нека је при једном њеном позиву `nivo` $= d \geqslant 1$. Тада скуп $\mathcal{X}$ из друге главе чине парцијална паковања величине d, а скуп $\mathcal{Y}$

парцијална паковања величине $d+1$. Релација између њих је инклузија, па је сваки пар из $\mathrm{Up}(P_{d,r})$ облика

$$(P_{d,r}, P_{d,r} \cup \{s\}),$$

где је s покривач који се може додати представнику $P_{d,r}$.

На почетку функције референца `P` упућује на вектор `predstavnici[nivo]`. Зато је `P[r]` управо представник $P_{d,r}$, где `r` означава његов индекс међу представницима величине d. Функција најпре спроводи фазу *Lift*, у којој за сваки такав представник одређује орбите допуштених проширења под дејством његовог стабилизатора. Након тога фаза *Classify* класификује добијене кандидате под дејством целе групе G и формира представнике величине $d+1$, њихове стабилизаторе и податке за наредно конструктивно препознавање.

Допуштени додаци и специјални *up*-скуп

У имплементацији користимо лексикографско прорeђивање описано у одељку 2.4. Пошто функција `korak` прелази са парцијалних паковања величине d на парцијална паковања величине $d+1$, специјални *up*-скуп чине проширења

$$(P_{d,r}, P_{d,r} \cup \{s\})$$

за која је s допуштен додатак и важи

$$\min s^{G_{P_{d,r}}} > \max P_{d,r}.$$

У одељку 2.4 показано је да се задржавањем само оваквих проширења не губи ниједна G-орбита парцијалних паковања величине $d+1$.

Услов да је s допуштен додатак и да је $s > \max P_{d,r}$ проверавамо функцијом `pokrivaci_za_dodavanje`. Она проналази покриваче који немају заједничку праву са датим парцијалним паковањем и чији је индекс већи од индекса свих његових покривача.

```
vector<int> pokrivaci_za_dodavanje(const vector<int>& polu_pakovanje) {
    bitset<BR_PRAVIH> zauzete_prave;

    for (int pokrivac : polu_pakovanje)
        zauzete_prave |= bitovi_pokrivaca[pokrivac];

    vector<int> kandidati;
```

```
    for (int s = polu_pakovanje.empty() ? 0 : polu_pakovanje.back()+1;
         s < pokrivaci.size(); ++s)
        if (!(bitovi_pokrivaca[s] & zauzete_prave).any())
            kandidati.push_back(s);

    return kandidati;
}
```

Битовском дисјункцијом маски свих покривача из датог парцијалног паковања добијамо маску `zauzete_prave`, чији су постављени битови управо индекси већ употребљених правих. Покривач са индексом `s` дисјунктан је са свим покривачима парцијалног паковања ако и само ако са овом маском нема заједнички постављен бит.

Пошто индексе покривача у парцијалном паковању чувамо растуће, вредност `polu_pakovanje.back()` представља његов највећи индекс. Зато петља почиње од наредног индекса. На овај начин добијамо само кандидате $s > \max P_{d,r}$, док се преостали део услова за припадност специјалном *up*-скупу проверава у фази *Lift* помоћу стабилизатора $G_{P_{d,r}}$. Кандидати се разматрају растућим редом.

Фаза *Lift*

За фиксираног представника $P_{d,r}$ и допуштени додатак s посматрамо пар

$$T = (P_{d,r}, P_{d,r} \cup \{s\}) \in \mathrm{Up}(P_{d,r}).$$

Стабилизатор $G_{P_{d,r}}$ фиксира прву координату овог пара, па његово дејство на такве парове можемо посматрати као дејство на додати покривач s. Зато се одређивање *up*-орбита своди на одређивање орбита допуштених додатака под дејством $G_{P_{d,r}}$.

Почетак функције `korak` и део који спроводи фазу *Lift* изгледају овако:

```
void korak(int nivo) {
    vector<vector<int>>& P = predstavnici[nivo];

    vector<vector<int>> dodaci(P.size());
    vector<map<int,vector<Kolineacija>>> G(P.size());
    vector<map<int,PodaciPrepoznavanja>> novi_podaci(P.size());
    vector<map<int,vector<int>>> orbite(P.size());
    vector<map<int,int>> Gp_original(P.size());
```

```
vector<bool> trivijalan_stabilizator(P.size());

for(int k(0); k<P.size(); ++k) {
    trivijalan_stabilizator[k] =
        stabilizatori[nivo-1][k].size() == 1;

    vector<bool> nadjen(pokrivaci.size(),false);

    for(int pokrivac : pokrivaci_za_dodavanje(P[k])) {
        if(nadjen[pokrivac]) continue;
        nadjen[pokrivac] = true;

        if(trivijalan_stabilizator[k]) {
            dodaci[k].push_back(pokrivac);
            novi_podaci[k][pokrivac].preslikavanje=id;
            continue;
        }

        bool dodaj = true;
        vector<Kolineacija> G_k;
        map<int,Kolineacija> h2_pom;
        vector<int> orbita = {pokrivac};

        h2_pom[pokrivac]=id;

        for(auto& f : stabilizatori[nivo-1][k]) {
            int f_pokrivac = slika_pokrivaca(pokrivac,f);

            if(f_pokrivac < pokrivac) {
                dodaj = false;
                nadjen[f_pokrivac] = true;
                break;
            }

            if(f_pokrivac == pokrivac)
                G_k.push_back(f);
            else if(!nadjen[f_pokrivac]) {
                nadjen[f_pokrivac] = true;
                orbita.push_back(f_pokrivac);
                h2_pom[f_pokrivac]=napravi_inverz(f);
            }
        }
```

```
            if(dodaj) {
                dodaci[k].push_back(pokrivac);
                G[k][pokrivac]=move(G_k);
                orbite[k][pokrivac]=move(orbita);

                for(int p : orbite[k][pokrivac]) {
                    Gp_original[k][p]=pokrivac;
                    novi_podaci[k][p].preslikavanje=h2_pom[p];
                }
            }
        }
    }

    stabilizatori[nivo-1].clear();
    stabilizatori[nivo-1].shrink_to_fit();

    // faza Classify
    ...
}
```

У овом делу функције програмска променљива `k` означава индекс представника у вектору `P`, па је `P[k]` $= P_{d,k}$. Вектор `dodaci[k]` садржаће изабране представнике орбита допуштених додатака том парцијалном паковању. Структура `orbite[k][pokrivac]` чува индексе покривача из орбите чији је изабрани представник `pokrivac`, док `Gp_original[k]` омогућава да се за сваки елемент такве орбите непосредно пронађе њен представник.

За сваки $P_{d,k}$ вектор `nadjen` бележи покриваче чија је орбита већ разматрана. Пошто функција `pokrivaci_za_dodavanje` враћа кандидате растућим редом, желимо да као представника орбите задржимо покривач са најмањим индексом. За тренутног кандидата `pokrivac` зато пролазимо кроз све елементе стабилизатора $G_{P_{d,k}}$ и одређујемо његове слике. Ако се добије слика са мањим индексом, тренутни покривач није најмањи у својој орбити и не додаје се у `dodaci[k]`.

Ако мања слика не постоји, `pokrivac` је покривач са најмањим индексом у својој $G_{P_{d,k}}$-орбити. Како редослед индекса у глобалном вектору `pokrivaci` одговара лексикографском поретку покривача, најмањи индекс у орбити одговара лексикографски најмањем покривачу у тој орбити. Поред тога, функција `pokrivaci_za_dodavanje` разматра само индексе веће од последњег ин-

декса у $P_{d,k}$. Зато за задржани покривач p важи

$$\min p^{G_{P_{d,k}}} = p > \max P_{d,k}.$$

Дакле, у `dodaci[k]` задржава се тачно по један представник сваке специјалне *up*-орбите придружене представнику $P_{d,k}$.

Током проласка кроз стабилизатор истовремено се одређују и подаци који ће бити потребни у фази *Classify*. Ако важи `f_pokrivac == pokrivac`, колинеација `f` фиксира и $P_{d,k}$ и додати покривач. Зато вектор `G_k` на крају садржи све елементе стабилизатора пара

$$T = (P_{d,k}, P_{d,k} \cup \{\texttt{pokrivac}\}).$$

Тај стабилизатор се, за задржани додатак, смешта у `G[k][pokrivac]`.

Када колинеација `f` слика изабрани додатак са индексом `pokrivac` у додатак са индексом `f_pokrivac`, њен инверз слика добијени додатак назад у изабрани представник орбите. Пошто је `f` елемент групе $G_{P_{d,k}}$, исто важи и за њен инверз, па он при томе фиксира $P_{d,k}$. Зато се у помоћној мапи `h2_pom` за сваки пронађени покривач чува управо `napravi_inverz(f)`. То је елемент $h_2 \in G_{P_{d,k}}$ који се у теоријском опису фазе *Lift* користи за пресликавање елемента једне *up*-орбите у њеног представника.

За задржану орбиту ови подаци се затим преносе у `novi_podaci`. Члан `preslikavanje` за покривач `p` садржи одговарајући елемент h_2, док члан `predstavnik` у овој фази још није одређен. Његова коначна вредност биће постављена тек у фази *Classify*, када се утврди који представник G-орбите величине $d+1$ одговара посматраном проширењу.

Ако је стабилизатор $G_{P_{d,k}}$ тривијалан, свака његова орбита додатака има само један елемент. Тада нема потребе пролазити кроз стабилизатор, па се сваки допуштени кандидат непосредно додаје у `dodaci[k]`, а одговарајуће пресликавање h_2 једнако је идентичкој колинеацији. Овај случај програм обрађује посебно јер се код већих парцијалних паковања јавља велики број представника са тривијалним стабилизатором.

По завршетку фазе *Lift* стабилизатори представника величине d више нису потребни. Сви подаци који ће се користити у фази *Classify* већ су смештени у локалне структуре `G`, `orbite`, `Gp_original` и `novi_podaci`. Зато се `stabilizatori[nivo-1]` празни пре наставка функције, чиме се ослобађа меморија коју су заузимали стабилизатори претходног нивоа.

Конструктивно препознавање подобјеката

У фази *Classify*, при обради елемената скупа Down(Q), потребно је конструктивно препознати парцијална паковања која се јављају као подобјекти текућег кандидата Q. Функција `nadji_predstavnika` зато прима произвољно парцијално паковање `objekat` и одређује индекс `k` и колинеацију `f` такве да важи

$$\texttt{objekat}^f = P_{m,k},$$

где је m величина тог парцијалног паковања.

Поступак је рекурзиван. Базни случај чине једночлана парцијална паковања. Функција `prvi_lift` већ је под дејством целе групе G класификовала све појединачне покриваче и за сваки од њих сачувала представника његове G-орбите и колинеацију која га у тај представник пресликава, па се у овом случају тражени подаци непосредно читају из `podaci_prepoznavanja`.

За парцијално паковање веће величине функција редом разматра његове подобјекте добијене избацивањем по једног покривача. Сваки такав подобјекат најпре рекурзивно конструктивно препознаје, а затим проверава да ли за добијеног представника постоје раније сачувани подаци за одговарајуће проширење из специјалног *up*-скупа Up*. Ако такви подаци не постоје, прелази на следећи подобјекат. Из Тврђења 16 следи да међу разматраним подобјектима увек постоји бар један за који су сачувани потребни подаци. Зато функција увек може да одреди представника посматраног парцијалног паковања и колинеацију која га у тај представник пресликава. Чим пронађе први такав подобјекат, враћа добијене податке и преостале подобјекте више не разматра.

```
int nadji_predstavnika(const vector<int>& objekat, Kolineacija& f) {
    if(objekat.size() == 1) {
        f = podaci_prepoznavanja[0][0][objekat[0]].preslikavanje;
        return podaci_prepoznavanja[0][0][objekat[0]].predstavnik;
    }

    int nivo = objekat.size() - 1;
    for(int j(objekat.size() - 1); j >= 0; --j) {
        vector<int> podobjekat = objekat;
        int izbaceni = podobjekat[j];
        podobjekat.erase(podobjekat.begin() + j);

        Kolineacija g1 = id;
```

```
            int k = nadji_predstavnika(podobjekat,g1);
            int slika_izbacenog = slika_pokrivaca(izbaceni,g1);

            auto it = podaci_prepoznavanja[nivo][k].find(slika_izbacenog);
            if(it == podaci_prepoznavanja[nivo][k].end()) continue;

            f = kompozicija(it->second.preslikavanje, g1);
            return it->second.predstavnik;
        }

        return -1;
}
```

Нека `objekat` има $m > 1$ покривача. Тада је `nivo = m-1`, јер се у сваком проласку кроз петљу избацује један покривач и добија подобјекат величине $m - 1$. Петља редом разматра такве подобјекте, почев од оног који се добија избацивањем последњег покривача. За текући индекс `j`, из вектора `objekat` формира се `podobjekat` избацивањем елемента `objekat[j]`, чији се индекс чува у променљивој `izbaceni`.

За формирани `podobjekat` функција се позива рекурзивно, при чему као повратну вредност даје индекс `k`, док у променљиву `g1` уписује колинеацију за коју важи

$$\texttt{podobjekat}^{g_1} = P_{m-1,k}.$$

Ова колинеација има улогу елемента g_1 из поступка конструктивног препознавања описаног у одељку 2.3. Под истом колинеацијом израчунава се и слика избаченог покривача, чији се индекс чува у променљивој `slika_izbacenog`. Тада важи

$$\texttt{objekat}^{g_1} = P_{m-1,k} \sqcup \{\texttt{slika_izbacenog}\}.$$

Пошто је прелаз са парцијалних паковања величине $m - 1$ на парцијална паковања величине m завршен, у `podaci_prepoznavanja[nivo][k]` налазе се подаци за раније задржана проширења из специјалног *up*-скупа Up^*. Зато се функцијом `find` проверава да ли је `slika_izbacenog` кључ у овој мапи. Ако није, добијено проширење није међу сачуваним специјалним проширењима, па се прелази на следећи подобјекат.

Ако кључ постоји, сачувано пресликавање `it->second.preslikavanje` слика проширење

$$P_{m-1,k} \sqcup \{\texttt{slika_izbacenog}\}$$

у представника $P_{m,r}$, где је r = `it->second.predstavnik`. У нотацији из одељка 2.3, ово сачувано пресликавање одговара производу $g_2 g_3$. Пошто `g1` слика `objekat` у наведено проширење, позив

```
f = kompozicija(it->second.preslikavanje, g1)
```

даје колинеацију која `objekat` слика у $P_{m,r}$. Функција затим враћа `it->second.predstavnik`, чиме се њено извршавање завршава и преостали подобјекти се не разматрају.

Успешност функције следи индукцијом по величини m парцијалног паковања. За $m = 1$ потребни подаци непосредно се читају из табеле попуњене функцијом `prvi_lift`. За $m > 1$, по индуктивној претпоставци, рекурзивни позиви успешно препознају подобјекте величине $m-1$. Применом Тврђења 16 на $k = m - 1$ и $\ell = m$ добијамо да бар један од тих подобјеката води до специјалног проширења за које постоје сачувани подаци. Петља зато успешно завршава препознавање најкасније при обради тог подобјекта, па завршни `return -1` у исправном извршавању функције није достижан.

Фаза *Classify*

После фазе *Lift* у векторима `dodaci[i]` налазе се представници специјалних *up*-орбита придружених представнику $P_{d,i}$. Фаза *Classify* треба међу добијеним кандидатима да издвоји лексикографски најмањег представника сваке G-орбите, одреди његов стабилизатор и допуни податке потребне за конструктивно препознавање.

У програму се нови представници величине $d + 1$ привремено чувају у вектору `Q`. Да бисмо га разликовали од ове програмске променљиве, текући кандидат у наставку означаваћемо са

$$Y = P_{d,i} \cup \{p\}, \qquad p \in \texttt{dodaci[i]}.$$

Скуп `za_brisanje` садржи парове индекса `(i,p)` за које је већ утврђено да одговарају кандидату из G-орбите неког раније изабраног представника.

Део функције `korak` који спроводи фазу *Classify* дат је у наставку.

```
vector<vector<int>> Q;
vector<vector<Kolineacija>> G_Q;
set<pair<int,int>> za_brisanje;
```

```
for(int i(0); i<P.size(); ++i)
    for(int p : dodaci[i]) {
        if(za_brisanje.find({i,p}) != za_brisanje.end())
            continue;

        vector<Kolineacija> h1h2ovi;

        for(int j(P[i].size()-1); j>=0; --j) {
            vector<int> X = P[i];
            X.erase(X.begin() + j);
            X.push_back(p);

            Kolineacija h1 = id;
            int k = nadji_predstavnika(X,h1);

            int h1_na_j = slika_pokrivaca(P[i][j],h1);

            if(novi_podaci[k].find(h1_na_j) == novi_podaci[k].end())
                continue;

            int h1h2_na_j =
                trivijalan_stabilizator[k]
                ? h1_na_j
                : Gp_original[k][h1_na_j];

            if(za_brisanje.find({k,h1h2_na_j}) != za_brisanje.end())
                continue;

            Kolineacija h1h2 =
                kompozicija(novi_podaci[k][h1_na_j].preslikavanje, h1);

            if(k==i && h1h2_na_j==p)
                h1h2ovi.push_back(h1h2);
            else {
                za_brisanje.insert({k,h1h2_na_j});

                if(trivijalan_stabilizator[k]) {
                    novi_podaci[k][h1h2_na_j].predstavnik = Q.size();
                    novi_podaci[k][h1h2_na_j].preslikavanje =
                        napravi_inverz(h1h2);
```

```
                } else {
                    for(int s : orbite[k][h1h2_na_j]) {
                        novi_podaci[k][s].predstavnik = Q.size();
                        novi_podaci[k][s].preslikavanje =
                            kompozicija(
                                napravi_inverz(h1h2),
                                novi_podaci[k][s].preslikavanje);
                    }
                }
            }
        }

        if(trivijalan_stabilizator[i])
            novi_podaci[i][p].predstavnik=Q.size();
        else
            for(int s : orbite[i][p])
                novi_podaci[i][s].predstavnik=Q.size();

        Q.push_back(P[i]);
        Q.back().push_back(p);

        if(trivijalan_stabilizator[i]) {
            G_Q.push_back({id});
            for(auto& h : h1h2ovi)
                G_Q.back().push_back(h);
        } else {
            G_Q.push_back(G[i][p]);
            for(auto& g : G[i][p])
                for(auto& h : h1h2ovi)
                    G_Q.back().push_back(
                        kompozicija(g,h));
        }
    }

predstavnici.push_back(move(Q));
stabilizatori.push_back(move(G_Q));
podaci_prepoznavanja.push_back(move(novi_podaci));
```

Кандидати се обилазе прво по растућем индексу `i` представника $P_{d,i}$, а затим по растућим индексима `p` из вектора `dodaci[i]`. Пошто су представници величине d лексикографски уређени и за сваки додатак важи $p > \max P_{d,i}$, овим редоследом се и кандидати $P_{d,i} \cup \{p\}$ обилазе лексикографски. Ако се

пар `(i,p)` већ налази у скупу `za_brisanje`, раније је пронађена колинеација која показује да је тај кандидат изоморфан већ изабраном представнику, па га прескачемо. У супротном кандидат $Y = P_{d,i} \cup \{p\}$ постаје наредни представник, а пре његовог уписивања одређујемо остале кандидате које треба елиминисати и његов стабилизатор.

За сваки $j \in \{0, \ldots, d-1\}$ посматрамо подобјекат

$$X_j = Y \setminus \{P_{d,i}[j]\}.$$

Претходно описана функција `nadji_predstavnika` одређује индекс `k` и колинеацију h_1 за које је

$$X_j^{h_1} = P_{d,k}.$$

Покривач који је из Y избачен при формирању X_j није нестао из целог кандидата. Његова слика под h_1 добија се позивом `slika_pokrivaca(P[i][j],h1)` и њен индекс се чува у `h1_na_j`. Зато важи

$$Y^{h_1} = P_{d,k} \cup \{\texttt{h1_na_j}\}.$$

У фази *Lift* задржане су само специјалне *up*-орбите. После конструктивног препознавања подобјекта X_j добили смо

$$Y^{h_1} = P_{d,k} \cup \{\texttt{h1_na_j}\}.$$

Постојање кључа `h1_na_j` у `novi_podaci[k]` значи да овај пар припада некој од специјалних *up*-орбита које су задржане у фази *Lift*. Ако такав кључ не постоји, пар не припада специјалном *up*-скупу и не одговара грани проређеног класификационог графа, па се прелази на следећи елемент скупа $\mathrm{Down}(Y)$.

Ова провера је програмска реализација модификованог корака *Eliminate* из одељка 2.4, где је показано да се њоме не пропушта ниједан задржани кандидат из G-орбите текућег кандидата Y.

Ако такав кључ постоји, потребно је пронаћи представника $G_{P_{d,k}}$-орбите покривача са индексом `h1_na_j`. Када је стабилизатор тривијалан, орбита је једночлана и тај представник је сам покривач `h1_na_j`. У супротном је његов индекс већ сачуван у `Gp_original[k][h1_na_j]`. Ову вредност програм чува у `h1h2_na_j`. Ако је одговарајући кандидат већ означен за брисање, његова поновна обрада није потребна.

Нека је h_2 колинеација из `novi_podaci[k][h1_na_j].preslikavanje`. У овом тренутку она слика посматрано проширење у представника његове специјалне *up*-орбите. Зато колинеација

$$h = h_2 \circ h_1$$

слика текући кандидат Y у други кандидат добијен у фази *Lift*. У програму се она чува под именом `h1h2`, јер према конвенцији за групну операцију уведеној у првој глави композицији $h_2 \circ h_1$ одговара производ $h_1 h_2$. Позив `kompozicija(h2,h1)` управо израчунава ову композицију.

Ако су `k==i` и `h1h2_na_j==p`, добијени кандидат је поново Y. Тада $h_1 h_2$ припада стабилизатору G_Y, па се чува у вектору `h1h2ovi`. У супротном је пронађен други кандидат из исте G-орбите као Y. Његов пар индекса додаје се у `za_brisanje`, па тај кандидат касније неће бити изабран за новог представника.

Истовремено се за елиминисаног кандидата попуњавају подаци за конструктивно препознавање. Пошто $h_1 h_2$ слика Y у елиминисаног кандидата, његов инверз слика тог кандидата назад у изабраног представника Y. Ако је стабилизатор $G_{P_{d,k}}$ тривијалан, управо се тај инверз непосредно чува. У супротном, за сваки покривач s из одговарајуће стабилизаторске орбите већ постоји локално пресликавање које проширење $P_{d,k} \cup \{s\}$ слика у представника те орбите. Зато композиција

$$(h_1 h_2)^{-1} \circ \texttt{novi_podaci[k][s].preslikavanje}$$

слика и то проширење у Y. Управо ову композицију израчунава одговарајући позив функције `kompozicija`.

После обраде свих X_j, текући кандидат добија индекс `Q.size()` међу новим представницима. Исти индекс уписује се и за све елементе његове стабилизаторске орбите, јер су њихова локална пресликавања из фазе *Lift* већ усмерена ка изабраном додатку p. Затим се вектор $P_{d,i}$ копира у `Q` и на његов крај додаје `p`, чиме се у `Q` уписује представник Y величине $d+1$.

Остаје да се формира стабилизатор G_Y. Стабилизатор G_T пара $T = (P_{d,i}, Y)$ познат је из фазе *Lift* и важи $G_T \leqslant G_Y$. Ако је `trivijalan_stabilizator[i]` тачно, група $G_{P_{d,i}}$ је тривијална, па је и G_T тривијална. У супротном су сви елементи групе G_T сачувани у `G[i][p]`.

Према доказу одређивања стабилизатора у одељку 2.3, представник левог косета gG_T, где је $g \in G_Y$, добија се обрадом пара

$$T^{g^{-1}} = (P_{d,i}^{g^{-1}}, Y) \in \mathrm{Down}(Y).$$

При томе, различити парови који се пресликавају у T дају представнике различитих левих косета.

Основном косету G_T одговара сам пар T, који петља не разматра, јер су елементи тог косета већ познати из фазе *Lift*. За сваки $j \in \{0, \ldots, d-1\}$ унутрашња петља из $P_{d,i}$ избацује елемент $P_{d,i}[j]$ и додаје покривач p, чиме добија $X_j = Y \setminus \{P_{d,i}[j]\}$. Пошто је $Y = P_{d,i} \cup \{p\}$, тако се разматрају сви парови из $\mathrm{Down}(Y)$ осим T, сваки тачно једном. Међу њима су сви парови потребни за преостале косете.

За парове потребне за преостале косете у одељку 2.4 показано је да после конструктивног препознавања важи $(P_{d,k}, Y^{h_1}) \in \mathcal{R}^*$. Зато кључ `h1_na_j` постоји у мапи `novi_podaci[k]`, па провера помоћу `find` не прескаче те парове.

При обради таквог пара конструктивно препознавање даје `k == i`, јер његова прва координата припада G-орбити представника $P_{d,i}$. Добијени пар затим припада истој специјалној *up*-орбити као T, чији је изабрани додатак p, па је `h1h2_na_j == p`. Пар `(i,p)` није у скупу `za_brisanje` на почетку обраде кандидата, а током те обраде у скуп се додају само парови различити од њега. Зато провера припадности скупу `za_brisanje` не прескаче парове потребне за стабилизатор. Пресликавања из фазе *Lift* за текућу *up*-орбиту остају непромењена, јер се поље `preslikavanje` мења само за орбите елиминисаних кандидата. Стога вектор `h1h2ovi` садржи тачно по једног представника сваког левог косета различитог од G_T.

У вектор новог стабилизатора најпре се уписују елементи основног косета: само `id` када је `trivijalan_stabilizator[i]` тачно, а иначе садржај `G[i][p]`. За сваки елемент `h` вектора `h1h2ovi` затим се додају елементи косета hG_T. Када је $G_{P_{d,i}}$ тривијална, тај косет садржи само h. У супротном се за сваки елемент `g` групе G_T израчунава `kompozicija(g,h)`. Овај позив даје композицију $g \circ h$, што према усвојеној конвенцији за операцију групе одговара производу hg. Пошто различити леви косети немају заједничке елементе, а за фиксирано h различити елементи $g \in G_T$ дају различите производе hg, сваки елемент стабилизатора G_Y добија се тачно једном.

Коначно, кандидати се обрађују лексикографским редом. Према резултату из одељка 2.4, лексикографски најмањи елемент сваке G-орбите

има облик $P_{d,r} \cup \{s\}$, при чему $(P_{d,r}, P_{d,r} \cup \{s\}) \in \mathrm{Up}^*(P_{d,r})$. Мора важити $s = \min s^{G_{P_{d,r}}}$, јер би из $s^h < s$ за неко $h \in G_{P_{d,r}}$ следило $P_{d,r} \cup \{s^h\} \prec P_{d,r} \cup \{s\}$. Зато фаза *Lift* задржава овај објекат као кандидата.

Када тај кандидат дође на ред, он не може бити у скупу `za_brisanje`, јер би то значило да постоји раније обрађен, а тиме и лексикографски мањи, кандидат из исте G-орбите. Зато се он бира за представника. Претходно смо показали да се при његовој обради сви остали задржани кандидати из исте G-орбите означавају за брисање. Према томе, у `Q` се из сваке G-орбите уписује управо њен лексикографски најмањи елемент.

На крају функције добијени представници, њихови стабилизатори и подаци за конструктивно препознавање премештају се у глобалне структуре. Они тиме постају улаз за наредни позив функције `korak`.

Покретање програма

На крају повежимо описане делове имплементације у целину. Главна функција најпре генерише тачке, праве и покриваче посматраног простора, а затим припрема податке потребне за дејство колинеација. Након иницијализације празног парцијалног паковања позива се `prvi_lift`, а потом се функција `korak` узастопно позива за наредне величине парцијалних паковања.

У приказаној имплементацији почетни подаци се генеришу при сваком покретању програма. Могуће је у датотекама сачувати тачке, праве, покриваче и податке о колинеацијама и њиховом дејству на праве, па их при наредним покретањима за исте параметре учитавати, уз очување редоследа објеката и њихових индекса. За простор $\mathrm{PG}(3,3)$ припрема ових података пре позива `prvi_lift` је трајала приближно 0,113 s. То је мање од 1% укупног времена извршавања програма до четвртог нивоа, па је могућа временска уштеда мала. Чување и учитавање почетних података није имплементирано.

```
int main() {
    generisi_tacke();
    generisi_prave();
    generisi_pokrivace();

    for(int t(0); t < BR_TACAKA; ++t)
        id[t] = t;

    generisi_stabilizator_0_1();
```

```
    generisi_0_1_kolineacije();

    predstavnici.push_back({{}});
    prvi_lift();

    stabilizator_0_1.clear();
    stabilizator_0_1.shrink_to_fit();
    stab_0_1_na_pravu.clear();
    stab_0_1_na_pravu.shrink_to_fit();
    kolineacija_0_1_u.clear();
    kolineacija_0_1_u.shrink_to_fit();
    kol_0_1_na_pravu.clear();
    kol_0_1_na_pravu.shrink_to_fit();

    korak(1);
    korak(2);
    korak(3);

    return 0;
}
```

После класификовања појединачних покривача више нису потребне табеле `stabilizator_0_1`, `stab_0_1_na_pravu`, `kolineacija_0_1_u` и `kol_0_1_na_pravu`. Зато се после позива функције `prvi_lift` оне празне, а позивима `shrink_to_fit` омогућава се ослобађање меморије коју су заузимале.

Број позива функције `korak` зависи од дубине до које желимо да спроведемо класификацију. После позива `korak(d)` вектор `predstavnici[d+1]` садржи представнике неизоморфних парцијалних паковања величине $d+1$, па вредност `predstavnici[d+1].size()` даје њихов број. За потпуно паковање простора $\mathrm{PG}(3,2)$ потребно је достићи величину 7, док потпуно паковање простора $\mathrm{PG}(3,3)$ има 13 покривача. У експериментима описаним у наставку број позива прилагођаван је конкретном простору и расположивој меморији.

Глава 4

Експериментални резултати

Имплементирани програм испитан је на просторима $\mathrm{PG}(3,2)$ и $\mathrm{PG}(3,3)$. За сваку достигнуту величину парцијалних паковања забележени су број добијених класа изоморфизма и бројеви елемената стабилизатора њихових представника. На тај начин може се пратити читав ток класификације и упоредити добијене вредности са познатим резултатима.

Мерења су извршена на рачунару са процесором Intel Core i7-13700T и 16 GB радне меморије, под оперативним системом Ubuntu 24.04.4 LTS. Програм је компајлиран помоћу GCC 13.3.0, уз оптимизацију `-O3` и стандард C++17. Времена појединачних прелаза мерена су помоћу `std::chrono::steady_clock`, док су укупно време извршавања и максимална потрошња радне меморије мерени помоћу програма GNU `time`. Све наведене вредности времена и меморије су заокружене.

4.1 Простор $\mathrm{PG}(3,2)$

У простору $\mathrm{PG}(3,2)$ потпуно паковање садржи седам покривача. Праве овог простора већ су у првој глави означене са $p_0, \ldots, p_{34}$. Програм генерише укупно 56 покривача, које према њиховом редоследу у вектору `pokrivaci` означавамо са $s_0, \ldots, s_{55}$:

$s_0 = \{p_0, p_{19}, p_{25}, p_{30}, p_{32}\}$ $s_1 = \{p_0, p_{19}, p_{26}, p_{28}, p_{33}\}$ $s_2 = \{p_0, p_{20}, p_{25}, p_{27}, p_{34}\}$
$s_3 = \{p_0, p_{20}, p_{26}, p_{29}, p_{31}\}$ $s_4 = \{p_0, p_{21}, p_{23}, p_{28}, p_{34}\}$ $s_5 = \{p_0, p_{21}, p_{24}, p_{30}, p_{31}\}$
$s_6 = \{p_0, p_{22}, p_{23}, p_{29}, p_{32}\}$ $s_7 = \{p_0, p_{22}, p_{24}, p_{27}, p_{33}\}$ $s_8 = \{p_1, p_9, p_{17}, p_{30}, p_{32}\}$
$s_9 = \{p_1, p_9, p_{18}, p_{28}, p_{34}\}$ $s_{10} = \{p_1, p_{10}, p_{17}, p_{27}, p_{33}\}$ $s_{11} = \{p_1, p_{10}, p_{18}, p_{29}, p_{31}\}$
$s_{12} = \{p_1, p_{11}, p_{15}, p_{28}, p_{33}\}$ $s_{13} = \{p_1, p_{11}, p_{16}, p_{30}, p_{31}\}$ $s_{14} = \{p_1, p_{12}, p_{15}, p_{29}, p_{32}\}$
$s_{15} = \{p_1, p_{12}, p_{16}, p_{27}, p_{34}\}$ $s_{16} = \{p_2, p_9, p_{17}, p_{20}, p_{26}\}$ $s_{17} = \{p_2, p_9, p_{18}, p_{22}, p_{24}\}$
$s_{18} = \{p_2, p_{10}, p_{17}, p_{21}, p_{23}\}$ $s_{19} = \{p_2, p_{10}, p_{18}, p_{19}, p_{25}\}$ $s_{20} = \{p_2, p_{11}, p_{15}, p_{20}, p_{25}\}$
$s_{21} = \{p_2, p_{11}, p_{16}, p_{22}, p_{23}\}$ $s_{22} = \{p_2, p_{12}, p_{15}, p_{21}, p_{24}\}$ $s_{23} = \{p_2, p_{12}, p_{16}, p_{19}, p_{26}\}$
$s_{24} = \{p_3, p_7, p_{17}, p_{26}, p_{33}\}$ $s_{25} = \{p_3, p_7, p_{18}, p_{25}, p_{34}\}$ $s_{26} = \{p_3, p_8, p_{17}, p_{21}, p_{30}\}$
$s_{27} = \{p_3, p_8, p_{18}, p_{22}, p_{29}\}$ $s_{28} = \{p_3, p_{11}, p_{13}, p_{25}, p_{30}\}$ $s_{29} = \{p_3, p_{11}, p_{14}, p_{22}, p_{33}\}$
$s_{30} = \{p_3, p_{12}, p_{13}, p_{26}, p_{29}\}$ $s_{31} = \{p_3, p_{12}, p_{14}, p_{21}, p_{34}\}$ $s_{32} = \{p_4, p_7, p_{17}, p_{23}, p_{32}\}$
$s_{33} = \{p_4, p_7, p_{18}, p_{24}, p_{31}\}$ $s_{34} = \{p_4, p_8, p_{17}, p_{20}, p_{27}\}$ $s_{35} = \{p_4, p_8, p_{18}, p_{19}, p_{28}\}$
$s_{36} = \{p_4, p_{11}, p_{13}, p_{23}, p_{28}\}$ $s_{37} = \{p_4, p_{11}, p_{14}, p_{20}, p_{31}\}$ $s_{38} = \{p_4, p_{12}, p_{13}, p_{24}, p_{27}\}$
$s_{39} = \{p_4, p_{12}, p_{14}, p_{19}, p_{32}\}$ $s_{40} = \{p_5, p_7, p_{15}, p_{25}, p_{32}\}$ $s_{41} = \{p_5, p_7, p_{16}, p_{26}, p_{31}\}$
$s_{42} = \{p_5, p_8, p_{15}, p_{21}, p_{28}\}$ $s_{43} = \{p_5, p_8, p_{16}, p_{22}, p_{27}\}$ $s_{44} = \{p_5, p_9, p_{13}, p_{26}, p_{28}\}$
$s_{45} = \{p_5, p_9, p_{14}, p_{22}, p_{32}\}$ $s_{46} = \{p_5, p_{10}, p_{13}, p_{25}, p_{27}\}$ $s_{47} = \{p_5, p_{10}, p_{14}, p_{21}, p_{31}\}$
$s_{48} = \{p_6, p_7, p_{15}, p_{24}, p_{33}\}$ $s_{49} = \{p_6, p_7, p_{16}, p_{23}, p_{34}\}$ $s_{50} = \{p_6, p_8, p_{15}, p_{20}, p_{29}\}$
$s_{51} = \{p_6, p_8, p_{16}, p_{19}, p_{30}\}$ $s_{52} = \{p_6, p_9, p_{13}, p_{24}, p_{30}\}$ $s_{53} = \{p_6, p_9, p_{14}, p_{20}, p_{34}\}$
$s_{54} = \{p_6, p_{10}, p_{13}, p_{23}, p_{29}\}$ $s_{55} = \{p_6, p_{10}, p_{14}, p_{19}, p_{33}\}$

Први од њих, s_0, управо је пример покривача наведен у првој глави. Полазећи од ових покривача, програм спроводи класификацију парцијалних паковања све до потпуних паковања. За сваки број покривача k у наредној табели наведени су број добијених класа изоморфизма, изабрани представници $P_{k,r}$ и величине њихових стабилизатора $G_{P_{k,r}}$:

k	Број класа	Представник	$\lvert G_{P_{k,r}}\rvert$
1	1	$P_{1,0} = \{s_0\}$	360
2	1	$P_{2,0} = \{s_0, s_9\}$	24
3	2	$P_{3,0} = \{s_0, s_9, s_{18}\}$	24
		$P_{3,1} = \{s_0, s_9, s_{21}\}$	6
4	3	$P_{4,0} = \{s_0, s_9, s_{18}, s_{29}\}$	6
		$P_{4,1} = \{s_0, s_9, s_{18}, s_{30}\}$	6
		$P_{4,2} = \{s_0, s_9, s_{21}, s_{47}\}$	24
5	2	$P_{5,0} = \{s_0, s_9, s_{18}, s_{29}, s_{38}\}$	8
		$P_{5,1} = \{s_0, s_9, s_{18}, s_{30}, s_{37}\}$	8
6	2	$P_{6,0} = \{s_0, s_9, s_{18}, s_{29}, s_{38}, s_{41}\}$	24
		$P_{6,1} = \{s_0, s_9, s_{18}, s_{30}, s_{37}, s_{43}\}$	24
7	2	$P_{7,0} = \{s_0, s_9, s_{18}, s_{29}, s_{38}, s_{41}, s_{50}\}$	168
		$P_{7,1} = \{s_0, s_9, s_{18}, s_{30}, s_{37}, s_{43}, s_{48}\}$	168

Једини прелаз у овом примеру при коме фаза *Classify* елиминише кандидате јесте прелаз са три на четири покривача. После фазе *Lift* добија се пет кандидата, при чему се у фази *Classify* утврђује

$$\{s_0, s_9, s_{21}, s_{30}\} \cong \{s_0, s_9, s_{18}, s_{29}\}, \qquad \{s_0, s_9, s_{21}, s_{24}\} \cong \{s_0, s_9, s_{18}, s_{30}\}.$$

Зато на наредном нивоу остају три представника наведена у табели.

При следећем прелазу, за представника $P_{4,2} = \{s_0, s_9, s_{21}, s_{47}\}$, једини допуштени додаци лексикографски већи од s_{47} јесу покривачи s_{48} и s_{50}. Међутим, свака од њихових орбита под дејством стабилизатора $G_{P_{4,2}}$ садржи покривач чији индекс није већи од 47. Зато ниједно од ових проширења не припада специјалном *up*-скупу, па се у фази *Lift* из представника $P_{4,2}$ не задржава ниједан кандидат величине пет.

Последњи ниво садржи два представника, па су добијена два неизоморфна потпуна паковања простора $\mathrm{PG}(3, 2)$, што се поклапа са познатим резултатом. Овај пример тако приказује целокупан ток имплементираног поступка, од генерисаних покривача, преко свих нивоа парцијалних паковања, до потпуних паковања. У наведеном окружењу целокупно извршавање програма за овај простор трајало је мање од 0,1 секунде.

4.2 Простор $\mathrm{PG}(3, 3)$

За простор $\mathrm{PG}(3, 3)$ број парцијалних паковања расте знатно брже. За парцијална паковања са једним до пет покривача у наредној табели наведени су бројеви класа изоморфизма и времена појединачних прелаза алгоритма.

Број покривача	Број класа	Време прелаза
1	2	9,0 s
2	17	0,11 s
3	1 274	0,09 s
4	219 066	4,1 s
5	15 884 059	6 min 36 s

За вредност k из прве колоне, време у трећој колони представља време потребно за прелаз са парцијалних паковања са $k - 1$ покривача на парцијална паковања са k покривача. У та времена нису укључени припрема почетних података ни време потребно за добијање претходних нивоа. Укупна времена

извршавања програма до четири, односно пет покривача наведена су засебно у наставку.

За сваки број покривача k, у наредној табели приказана је расподела класа изоморфизма према броју елемената стабилизатора њихових представника. Запис a^b означава да постоји b класа чији представници имају стабилизатор са a елемената. Изложилац 1 се изоставља.

k	Расподела величина стабилизатора
1	5 760, 1 920
2	240, 120, 96, 48, 32^2, 24, 16^2, 12^2, 10, 8^2, 6, 4^2
3	240, 96, 72, 48^2, 32^2, 24^7, 16^8, 12^8, 10^2, 8^{32}, 6^{12}, 4^{85}, 3^{12}, 2^{341}, 1^{760}
4	72, 64, 48^3, 40, 36, 32^3, 24^{11}, 18, 16^{11}, 12^{20}, 10, 8^{127}, 6^{31}, 4^{848}, 3^{22}, $2^{9\,312}$, $1^{208\,672}$
5	320, 240, 160, 120, 48^8, 40^2, 32^3, 24^{14}, 20, 16^{35}, 12^{56}, 10^2, 8^{335}, 6^{249}, 5^{11}, $4^{4\,019}$, 3^{358}, $2^{108\,331}$, $1^{15\,770\,631}$

За парцијална паковања са једним, два и три покривача, и бројеви класа изоморфизма и расподеле величина стабилизатора у потпуности се поклапају са резултатима које је добио Бетен [1].

На петом нивоу тривијалан стабилизатор има 15 770 631 представник, што чини приближно 99,29% свих класа на том нивоу. Код ових представника једина колинеација која фиксира парцијално паковање као скуп покривача јесте идентичко пресликавање. Стабилизатор таквог представника не може да преслика један допуштени додатак у други. Зато код ових представника фаза *Lift* задржава све додатке који задовољавају лексикографски услов, па се елиминисање изоморфних копија међу тако задржаним кандидатима обавља искључиво у фази *Classify*.

Број класа брзо расте већ на почетним нивоима, па узастопно настављање поступка са фазама *Lift* и *Classify* постаје рачунарски захтевно. Укупно време извршавања програма до парцијалних паковања са четири покривача износило је 13,7 секунди, а максимална потрошња радне меморије износила је 188 MiB. При наставку класификације до парцијалних паковања са пет покривача укупно време износило је 6 минута и 55 секунди, а максимална потрошња радне меморије 7,35 GiB.

На истом рачунару покушан је и прелаз са пет на шест покривача. Оперативни систем прекинуо је програм због недостатка меморије током фазе *Lift*,

када је потрошња радне меморије програма достигла приближно 14,18 GiB. Покушај на платформи Kaggle, уз меморијско ограничење од 30 GiB, такође је прекинут због недостатка меморије током фазе *Lift*. Зато је овом имплементацијом класификација парцијалних паковања простора $\mathrm{PG}(3,3)$ завршена до величине пет.

Потпуно паковање простора $\mathrm{PG}(3,3)$ садржи 13 покривача. Бетен у потпуној класификацији не наставља поступак са фазама *Lift* и *Classify* узастопно кроз све величине парцијалних паковања. После класификације парцијалних паковања величине три, за сваки од 1 274 представника проблем допуњавања до потпуног паковања своди на проблем *Exact Cover*, који решава Кнутовим алгоритмом *Dancing Links* (DLX) [1]. На тај начин избегава се узастопно набрајање свих класа парцијалних паковања величина $4, 5, \ldots, 12$. Разматрање и примену поступка допуњавања решавањем проблема *Exact Cover* остављамо за даљи рад.

Глава 5

Закључак

У овом раду представљена је примена Шмалцовог алгоритма на класификовање паковања у коначним пројективним просторима, са посебним освртом на просторе $\mathrm{PG}(3,2)$ и $\mathrm{PG}(3,3)$. Најпре су изложене математичке основе потребне за опис пројективних простора, колинеација, орбита и стабилизатора, а затим су описане фазе *Lift* и *Classify* и њихова реализација у програмском језику C++.

У случају $\mathrm{PG}(3,2)$ имплементација омогућава да се поступак спроведе до потпуних паковања. Добијена су два неизоморфна паковања, што се поклапа са познатим резултатом.

За $\mathrm{PG}(3,3)$ број објеката нагло расте већ на почетним нивоима класификације. За парцијална паковања са једним, два и три покривача добијени бројеви класа изоморфизма и расподеле величина стабилизатора поклапају се са резултатима из литературе. За парцијална паковања величине четири и пет добијено је 219 066, односно 15 884 059 класа изоморфизма. Максимална потрошња радне меморије износила је 188 MiB при класификацији до четири покривача, а 7,35 GiB при наставку до пет покривача. Овај раст показује да меморијска потрошња брзо постаје практично ограничење, јер се истовремено чува велики број кандидата, података за конструктивно препознавање и колинеација. Зато потпуна класификација паковања простора $\mathrm{PG}(3,3)$ није извршена овом верзијом програма.

Меморијска потрошња могла би се смањити компактнијим представљањем колинеација и смањивањем употребе динамичких структура података. Даљи рад може обухватити и имплементацију Бетеновог поступка за директан прелаз са три на тринаест покривача решавањем проблема *Exact Cover*.

Библиографија

[1] A. Betten. (2016). The Packings of PG(3,3). *Designs, Codes and Cryptography*, 79(3).

[2] N. Magaud. (2022). Proof Pearl: Formalizing Spreads and Packings of the Smallest Projective Space PG(3,2) using the Coq Proof Assistant.

[3] E. Artin (1957). Geometric Algebra. *Interscience Publishers*

[4] M. Lavrauw, G. Van de Voorde. (2015). Field reduction and linear sets in finite geometry. *Contemporary Mathematics*, 632.

[5] R. D. Baker. (1976). Partitioning the planes of $AG_{2m}(2)$ into 2-designs. *Discrete Mathematics*, 15(3).

[6] L. Xu, T. Feng. (2023). The chromatic index of finite projective spaces. *Journal of Combinatorial Designs*, 31(9).

[7] A. Beutelspacher. (1974). On parallelisms in finite projective spaces. *Geometriae Dedicata*, 3.

[8] J. W. P. Hirschfeld. (1998). Projective Geometries over Finite Fields, second edition. *Oxford University Press.*

[9] B. Schmalz. (1990). Verwendung von Untergruppenleitern zur Bestimmung von Doppelnebenklassen. *Bayreuther Mathematische Schriften*, 31.

[10] B. Schmalz. (1992). t-Designs zu vorgegebener Automorphismengruppe. *Bayreuther Mathematische Schriften*, 41.

Биографија аутора

Ђорђе Јовановић је рођен 6. 7. 1994. у Београду. Завршио је Математичку гимназију 2013. и исте године се уписао на Математички факултет Универзитета у Београду. Од 2020. године запослен је на Факултету организационих наука.