

UNIVERZITA KOMENSKÉHO V BRATISLAVE  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



**VLASTNOSTI A SYMETRIE LINEÁRNYCH  
KÓDOV DEFINOVANÝCH INCIDENČNÝMI  
MATICAMI REGULÁRNYCH GRAFOV S  
MALÝM OBVODOM**

Diplomová práca

2022

Bc. Branislav Boráň

UNIVERZITA KOMENSKÉHO V BRATISLAVE  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



**VLASTNOSTI A SYMETRIE LINEÁRNYCH  
KÓDOV DEFINOVANÝCH INCIDENČNÝMI  
MATICAMI REGULÁRNYCH GRAFOV S  
MALÝM OBVODOM**

Diplomová práca

Študijný program: Aplikovaná informatika

Študijný odbor: 2511 Aplikovaná informatika

Školiace pracovisko: Katedra algebry a geometrie

Školiteľ: doc. RNDr. Róbert Jajcay, DrSc.

Bratislava, 2022

Bc. Branislav Boráň



Univerzita Komenského v Bratislave  
Fakulta matematiky, fyziky a informatiky

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Branislav Boráň

**Študijný program:** aplikovaná informatika (Jednoodborové štúdium,  
magisterský II. st., denná forma)

**Študijný odbor:** informatika

**Typ záverečnej práce:** diplomová

**Jazyk záverečnej práce:** anglický

**Sekundárny jazyk:** slovenský

**Názov:** Properties and symmetries of linear codes defined via incidence matrices of regular graphs with small girth

*Vlastnosti a symetrie lineárnych kódov definovaných incidenčnými maticami regulárnych grafov s malým obvodom*

**Anotácia:** Lineárne kódy sú podpriestory konečnorozmerných vektorových priestorov nad konečnými poľami. Majú preto bohaté grupy automorfizmov, ktoré zároveň obsahujú množstvo informácií o uvažovanom kóde. Určenie úplnej grupy automorfizmov kódu je výpočtovo náročná úloha. Namiesto určenia grupy automorfizmov pre daný kód sa preto uvažuje obrátená úloha zostrojenia kódu s predpísanou grupou automorfizmov. Cieľom práce je preskúmať oba smery tejto interakcie.

**Ciel:** Cieľom navrhovanej problematiky je poskytnúť študentovi výpočtovo zložitý problém vyžadujúci dôkladné porozumenie štruktúry uvažovaných objektov ako aj programátorské a organizačné schopnosti.

**Literatúra:** R. Hill, A first course in coding theory, Oxford University Press, 1993

S. Roman, Coding and information theory, Springer, 1992

R. Jajcay, P. Potocnik and Stephen E. Wilson, Half-cyclic, dihedral and half-dihedral codes,

J. of Applied Mathematics and Computing 64 (2020), 691-708.

**Kľúčové slová:** lineárny kód, grupa automorfizmov, konečné pole, regulárny graf malého obvodu

**Vedúci:** doc. RNDr. Róbert Jajcay, DrSc.

**Katedra:** FMFI.KAG - Katedra algebry a geometrie

**Vedúci katedry:** doc. RNDr. Pavel Chalmovianský, PhD.

**Dátum zadania:** 09.12.2020

**Dátum schválenia:** 10.12.2020

prof. RNDr. Roman Ďuríkovič, PhD.

garant študijného programu

.....  
študent

.....  
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

.....  
Bratislava, 2022

Bc. Branislav Boráň

# Pod'akovanie

Chcel by som sa v prvom rade pod'akovať môjmu školiteľovi doc. RNDr. Róbertovi Jajcayovi, DrSc. za odbornú pomoc a usmernenia pri písaní tejto práce, za materiály, cenné rady, ktoré mi veľmi pomohli pri riešení tejto diplomovej práce. V neposlednom rade chcem tiež pod'akovať celej mojej rodine, priateľke a kamarátom za podporu počas môjho štúdia.

# Abstrakt

V našej práci skúmame vlastnosti a symetrie lineárnych kódov definovaných incidenčnými maticami regulárnych grafov s malým obvodom. V kapitole „Analýza problému“ popisujeme teoretické poznatky potrebné na pochopenie uvažovanej oblasti skúmania. Uvádzame lineárny kód, jeho využitie v komunikácii a procese prenosu informácií. Rovnako skúmame špeciálny typ lineárneho kódu LDPC a jeho grafovú reprezentáciu, na ktorej je založené nami navrhované generovanie LDPC kódov. Na skúmanie vlastností a symetrií lineárnych kódov využívame okrem parametrov, ktoré definujú samotný kód (dĺžka kódu, maximálny počet kódových slov, minimálna Hammingová vzdialenosť v kóde) porovnávanie kódov s optimálnymi a perfektnými kódmi aj kontrolné matice, generujúce matice a grupy automorfizmov. V grafovej reprezentácii LDPC kódov skúmame informačné pomery, Moorove ohraničenie a grupy automorfizmov. V kapitole „Návrh riešenia“ uvádzame nami vyvinuté metódy skúmania vlastností a symetrií. Popisujeme programovací jazyk a prostredie, v ktorom sme realizovali skúmanie a bola v ňom navrhnutá logika samotného riešenia. Návrh dvoch softvérových diel, ktorých výstupmi sú výsledky našej práce, uvádzame v kapitole „Riešenie“. Softvérové diela sa zakladajú na využití knižnice SageMath dvoma spôsobmi. Prvý spôsob je implementovaný v online aplikácii CoCalc vo forme Sage skriptov. Druhý spôsob je implementovaný ako konzolová aplikácia v programovacom jazyku

Python, kde je knižnica SageMath importovaná. V kapitole „Výstupy navrhovaných riešení“ prinášame výstupy oboch riešení, výsledky oboch riešení a porovnávame výsledky konzolovej aplikácie s výsledkami online aplikácie Co-Calc. Niektoré výstupné údaje je možné uložiť a môžu v budúcnosti slúžiť ako vstupné dátá pre ďalšie oblasti skúmania.

Kľúčové slová: grupa automorfizmov, perfektné kódy, klietky

# Abstract

Our research focuses on the properties and symmetries of linear codes defined via incidence matrices of regular graphs with a small girth. The chapter “Problem analysis“ contains the theoretical knowledge needed to understand the considered area of research. We present linear codes, their use in communication and information transfer processes. We also present a special type of linear code, the LDPC code and its graphical representation, on which our proposed generation of LDPC codes is based. To examine the properties and symmetries of linear codes, in addition to the parameters that define the code itself (code length, maximum number of codewords, minimum code distance), we also use parity check matrices, generating matrices and automorphism groups to compare our generated codes with optimal and perfect codes. In the graphical representation of LDPC codes, we examine information ratios, Moore bounds and automorphism groups. In the chapter “Solution design“, we also examine our developed methods and use them for properties and symmetries of the considered codes. We describe the programming language and development environment in which our research was performed and where the logic of the solution was designed. The design of two softwares, the outputs of which are the results of our work, is presented in the chapter "Solutions". Our softwares are based on the use of the SageMath library in two ways. The first method is implemented in the online applica-

tion CoCalc in the form of a Sage script. The second method is implemented as a console application in the Python programming language, where the SageMath library is imported. In the "Outputs of the proposed solutions" chapter we demonstrate the outputs of both solutions, the results of both solutions and we compare console application results with online application CoCalc results.

Keywords: automorphism group, perfect codes, cages

# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
<b>2</b>	<b>Motivácia</b>	<b>2</b>
<b>3</b>	<b>Analýza problému</b>	<b>3</b>
3.1	Komunikácia a prenos informácií . . . . .	3
3.2	Lineárny kód . . . . .	5
3.2.1	Generujúca matica lineárneho kódu . . . . .	6
3.2.2	Kontrolná matica lineárneho kódu . . . . .	6
3.2.3	Dĺžka lineárneho kódu . . . . .	7
3.2.4	Maximálny počet slov v kóde a Optimálny kód . . . . .	7
3.2.5	Hammingova vzdialenosť v kóde . . . . .	9
3.2.6	Informačný pomer . . . . .	10
3.2.7	Perfektné lineárne kódy . . . . .	10
3.2.8	LDPC kódy . . . . .	11
3.3	Grafová reprezentácia LDPC kódov . . . . .	12
3.3.1	Incidenčná matica a graf vzdialenosťi . . . . .	12
3.3.2	Veľkosť obvodu grafu . . . . .	14
3.3.3	Automorfizmus grafu . . . . .	14
3.3.4	Automorfizmus lineárneho kódu . . . . .	14

3.4 Konštrukcia LDPC kódov . . . . .	15
3.4.1 Klietka a rekordný graf . . . . .	15
<b>4 Návrh riešenia</b>	<b>19</b>
4.1 Programovací jazyk a prostredie . . . . .	19
4.1.1 Návrh 1. riešenia - skripty v CoCalc . . . . .	20
4.1.2 Návrh 2. riešenia - vlastná konzolová aplikácia . . . . .	20
4.2 Návrh logiky samotného riešenia . . . . .	22
4.2.1 Generovanie klietok a rekordných grafov . . . . .	22
4.2.2 Validácia klietky a rekordného grafu . . . . .	24
4.2.3 Získavanie údajov z klietky alebo rekordného grafu . . . . .	24
4.2.4 Lineárny kód a generujúca matica získaná z kontrolnej matice . . . . .	26
4.2.5 Validácia lineárneho kódu . . . . .	28
4.2.6 Generovanie lineárnych kódov z grupy automorfizmov grafu . . . . .	28
4.2.7 Generovanie grupy automorfizmov z lineárneho kódu .	30
<b>5 Riešenie</b>	<b>32</b>
5.1 Prvé riešenie - CoCalc (SageMath skripty) . . . . .	32
5.1.1 Skript generateCageAndLinearCodeByParameters . .	33
5.1.2 Skript autCodesFromCages . . . . .	35
5.1.3 Skript mooreBoundsCageValidation . . . . .	37
5.1.4 Skript generateIncidnceMatrices . . . . .	38
5.1.5 Skript generateParityCheckMatrices . . . . .	40
5.1.6 Skript cagesData . . . . .	42
5.1.7 Skript generateAutCodesFromCages . . . . .	44
5.1.8 Skript generateGeneratorMatrices . . . . .	46

5.1.9	Skript linearCodesData . . . . .	48
5.1.10	Skript getLinearCodesParameter . . . . .	50
5.1.11	Skript minDistanceLinearCodes . . . . .	51
5.1.12	Skript maxWordsLinearCodes . . . . .	53
5.1.13	Skript sizeNLinearCodesData . . . . .	54
5.1.14	Skript autLinearCodesData . . . . .	56
5.1.15	Skript generateAutGroupLCode . . . . .	57
5.1.16	Skript linearCodeValidation . . . . .	59
5.1.17	Skript autGroupCode . . . . .	61
5.1.18	Skripty generujúce vstupné alebo výstupné textové súbory	63
5.1.19	Skripty informačného charakteru zobrazujúce detaľy a výpočty . . . . .	63
5.2	Druhé riešenie - vlastná konzolová aplikácia . . . . .	64
5.2.1	Výpočet Moorového ohraničenia . . . . .	67
5.2.2	Validácia klietok a rekordných grafov . . . . .	67
5.2.3	Konštrukcia klietky alebo rekordného grafu zo vstupných dát . . . . .	68
5.2.4	Generovanie klietky Cage(6,4) . . . . .	69
5.2.5	Parser na spracovanie textových súborov zoznamov susedností . . . . .	70
5.2.6	Výpočet parametra perfektného kódu . . . . .	71
5.2.7	Validácia lineárneho kódu . . . . .	71
5.2.8	Generovanie incidenčných a kontrolných matíc . . . . .	72
5.2.9	Ukladanie matice do textového súboru . . . . .	73
5.2.10	Získanie a uloženie grupy automorfizmov z grafu . . . . .	74
5.2.11	Získanie a uloženie grupy automorfizmov z lineárneho kódu . . . . .	75

5.2.12	Inicializácia počiatočných vrcholov a doplnenie grupy automorfizmov grafu . . . . .	76
5.2.13	Inicializácia počiatočných vrcholov, hrán a doplnenie grupy automorfizmov kódu . . . . .	77
5.2.14	Zobrazenie obrazu grafu, jeho incidenčná kontrolná a generujúca matica . . . . .	78
5.2.15	Formáter na formátovanie výstupných výpisov . . . . .	79
5.3	Ukladanie výsledkov . . . . .	81
<b>6</b>	<b>Výstupy navrhovaných riešení</b>	<b>82</b>
6.1	Klietky, rekordné grafy a ich reprezentujúce matice . . . . .	82
6.1.1	Graficky zobraziteľné klietky alebo rekordné grafy . . .	83
6.1.2	Graficky nezobraziteľné klietky a rekordné grafy . . . .	96
6.2	Konštrukcia lineárneho kódu z klietky a rekordného grafu . .	104
6.2.1	Lineárne kódy so všetkými uvažovanými parametrami .	105
6.2.2	Lineárne kódy bez parametra optimálneho kódu . . . .	106
6.2.3	Lineárne kódy bez minimálnej Hammingovej vzdialenosťi, parametrov perfektného a optimálneho kódu . .	109
6.2.4	Lineárne kódy bez parametrov . . . . .	111
6.3	Konštrukcia grupy automorfizmov z grafu . . . . .	114
6.4	Konštrukcia grupy automorfizmov z lineárneho kódu . . . .	121
6.5	Porovnanie výstupov . . . . .	128
<b>7</b>	<b>Záver</b>	<b>134</b>
<b>Príloha A: Používateľská príručka</b>		<b>141</b>
<b>Príloha B: Obsah repozitára</b>		<b>143</b>

# Zoznam skratiek

**|AutGroup(C)|** Počet automorfizmov získaný z lineárneho kódu. 27, 30, 48, 56–58, 62, 64, 104, 106, 108, 109, 111, 114, 121, 122, 124, 131

**|AutGroup(Cage)|** Počet automorfizmov získaný z klietky. 16, 17, 24, 25, 29, 43, 50, 64, 82, 91, 93, 97, 101, 106, 108, 109, 111, 114, 115, 118, 121, 128, 129, 131, 132, 135, 137

**|Hrows|** Počet riadkov kontrolnej matice. 26, 43

**|edg|** Počet hrán grafu. 24, 25, 35, 42, 64, 67, 82

**|ver|** Počet vrcholov grafu. 12, 15–18, 22–25, 29, 35, 42, 64, 67, 69, 76, 82, 91, 93, 97, 101

**AutGroup(C)** Grupa automorfizmov získaná z lineárneho kódu. 14, 26, 27, 31, 33, 35, 57–59, 61, 63, 64, 76, 81, 114, 121, 123, 125, 126, 132–134, 139

**AutGroup(Cage)** Grupa automorfizmov získaná z klietky. 14, 31, 33–35, 37, 44, 46, 64, 80, 81, 90, 97, 99, 102, 104, 114, 116, 117, 119–121, 126, 132, 134, 138

**C(n,m,d)** Lineárny kód. 6, 7, 10, 11, 15, 19, 24, 26–28, 30, 33–35, 48–52, 55, 57–59, 61, 62, 72, 75, 104–111, 114, 118, 121–124, 128, 130–134,

136–139

**C[n,r]** Lineárny kód. 6, 7, 10, 134

**Cage(k,g)** Klietka. 15, 17, 22–25, 29, 30, 33, 64, 65, 76, 81, 82, 90, 93, 96–99, 101–111, 114, 118–121, 125, 128, 129, 131, 132, 135, 136, 138

**d** Hammingova vzdialenosť v kóde. x, xxii, 6–10, 26–28, 35, 48, 49, 51–53, 64, 71, 104, 105, 107–111, 130, 131, 136, 137

**G** Generujúca matica lineárneho kódu. x, 6, 7, 10, 26–28, 30, 33–35, 37, 47, 48, 50–57, 59–61, 64, 72, 79, 81, 104–114, 118, 119, 121, 130–132, 136, 140

**g** Veľkosť obvodu grafu. x, 12, 14, 15, 23, 24, 34, 36, 39, 108, 131, 137, 141

**H** Kontrolná matica lineárneho kódu, Incidenčná matica klietky. 6, 7, 10, 11, 14, 15, 24–26, 28, 29, 33–35, 37, 40–44, 47, 48, 57–62, 64, 72, 79, 81–83, 90, 92, 93, 95–104, 114, 119, 121, 128–130, 132, 133, 135, 136, 138–140

**k** Stupeň vrcholov v grafe. 12, 15, 23, 24, 34, 36, 39, 141

**L** Incidenčná matica. 12, 13, 15, 25, 26, 29–31, 33–35, 37, 39–42, 45, 46, 62–64, 79, 81–83, 90–103, 114, 119, 121, 128–130, 132, 135, 138–140

**LDPC** Low density parity check codes - Lineárne kódy s malou hustotou matice H. x, xi, 11, 12, 14, 15, 25

**m** Maximálny počet slov v kóde. x, 6, 7, 11, 26–28, 35, 48, 49, 51, 53, 54, 64, 71, 104, 105, 107, 108, 110, 111, 130, 131, 136

**M(k,g)** Moorové ohraničenie pre klietku. 15, 23, 24, 35, 37, 38, 64, 69, 82, 90, 91, 93, 96, 97, 101, 128, 135

**MDPC** Moderate density parity check codes - Lineárne kódy s väčšou hustotou matice H. 12

**n** Dĺžka lineárneho kódu. x, xxii, 5–11, 25–27, 35, 48, 49, 51, 54–56, 64, 71, 104, 105, 107

**o(n,m,d)** Parameter optimálneho kódu. 26, 27, 104, 106, 108–112, 130, 131, 136

**p(n,m,d)** Parameter perfektného kódu. 27, 28, 35, 48, 50, 51, 64, 71, 104, 106, 108–112, 130, 131, 136, 137

**QC** Kvázickyklický blokový lineárny kód. 11

**QC-MDPC** Kvázickyklické lineárne kódy s väčšou hustotou matice H ako LDPC. 11

**R** Informačný pomer. x, 10, 24, 26, 34, 35, 43, 64, 82, 90, 93, 96, 97, 100, 128, 135

**r** Počet lineárne nezávislých vektorov, Dimenzia pod priestoru lineárneho kódu. 5–7, 10, 25, 26, 104–107, 109, 110, 130, 134, 136

**Rec(k,g)** Rekordný graf. 17, 18, 22–25, 29, 30, 33, 64, 65, 76, 81, 82, 90, 93, 96–99, 101–104, 109–111, 114, 118–121, 125, 128, 129, 131, 132, 135–138

**t(A)** Čas generovania a ukladania grupy automorfizmov klietky alebo rekordného grafu. 115, 116, 118–120, 122, 124, 125

**t(G)** Čas generovania a ukladania generujúcej matice lineárneho kódu. 104–  
107, 109, 110, 113, 130, 131, 136

**t(H)** Čas generovania a ukladania kontrolnej matice lineárneho kódu. 82,  
83, 90, 92, 93, 95, 96, 99, 100, 103, 130

**t(L)** Čas generovania a ukladania incidenčnej matice. 82, 83, 90, 91, 93, 94,  
96, 97, 100, 101

**V(n,2)** Množina všetkých usporiadaných n-tíc  $F_2^n$ . 5, 7

# Zoznam obrázkov

3.1	Prenos informácií od zdroja k prijímateľovi [3] . . . . .	4
3.2	Vzťah medzi grafom vzdialenosť a incidenčnou maticou [7] . .	13
3.3	Petersenov graf (vľavo) a Heawoodov graf (vpravo)[12] . . . .	16
3.4	McGeeho graf (vľavo) a Tutteov-Coxeterov graf (vpravo) [12] .	16
4.1	SageMath a CoCalc [18, 19] . . . . .	19
5.1	Vývojový diagram - generateCageAndLinearCodeByParameters skript . . . . .	34
5.2	Vývojový diagram - autCodesFromCages skript . . . . .	36
5.3	Vývojový diagram - mooreBoundsCageValidation skript . . . .	38
5.4	Vývojový diagram - generateIncidenceMatrices skript . . . .	39
5.5	Vývojový diagram - generateParityCheckMatrices skript . . .	41
5.6	Vývojový diagram - cagesData skript . . . . .	43
5.7	Vývojový diagram - generateAutCodesFromCages skript . . .	45
5.8	Vývojový diagram - generateGeneratorMatrices skript . . . .	47
5.9	Vývojový diagram - linearCodesData skript . . . . .	49
5.10	Vývojový diagram - getLinearCodesParameter skript . . . . .	50
5.11	Vývojový diagram - minDistanceLinearCodes skript . . . . .	52
5.12	Vývojový diagram - maxWordsLinearCodes skript . . . . .	53
5.13	Vývojový diagram - sizeNLinearCodesData skript . . . . .	55

5.14	Vývojový diagram - autLinearCodesData skript . . . . .	56
5.15	Vývojový diagram - generateAutGroupLCode skript . . . . .	58
5.16	Vývojový diagram - linearCodeValidation skript . . . . .	60
5.17	Vývojový diagram - autGroupCode skript . . . . .	62
5.18	Diagram tried - konzolová aplikácia . . . . .	66
5.19	Výpočet Moorového ohraničenia - riešenie . . . . .	67
5.20	Validácia klietok a rekordných grafov - riešenie . . . . .	67
5.21	Konštrukcia klietky alebo rekordného grafu - riešenie . . . . .	68
5.22	Konštrukcia Cage(6,4) - riešenie . . . . .	69
5.23	Parser zoznamu susedností - riešenie . . . . .	70
5.24	Výpočet parametra perfektného kódu - riešenie . . . . .	71
5.25	Validácia lineárneho kódu - riešenie . . . . .	71
5.26	Generovanie incidenčných a kontrolných matíc - riešenie . . . . .	72
5.27	Ukladanie matice do textoveho suboru - riešenie . . . . .	73
5.28	Získanie a uloženie grupy automorfizmov z grafu - riešenie . . . . .	74
5.29	Získanie a uloženie grupy automorfizmov lineárneho kódu - riešenie . . . . .	75
5.30	Doplnenie automorfizmov grafu a inicializácia vrcholov - riešenie	76
5.31	Doplnenie automorfizmov kódu a inicializácia vrcholov, hrán - riešenie . . . . .	77
5.32	Zobrazenie obrazu grafu, incidenčná, kontrolná a generujúca matica - riešenie . . . . .	78
5.33	Formátovanie výstupných výpisov - riešenie . . . . .	79
5.34	Vývojový diagram - konzolová aplikácia . . . . .	80
6.1	Cage(3,5) - výstupný graf [19] . . . . .	83
6.2	Cage(3,6) - výstupný graf [19] . . . . .	84
6.3	Cage(3,7) - výstupný graf [19] . . . . .	84

6.4	Cage(3,8) - výstupný graf [19] . . . . .	85
6.5	Cage(3,10) - výstupný graf [19] . . . . .	85
6.6	Cage(3,11) - výstupný graf [19] . . . . .	86
6.7	Rec(3,14) - výstupný graf . . . . .	86
6.8	Cage(4,5) - výstupný graf [19] . . . . .	87
6.9	Cage(4,7) - výstupný graf . . . . .	87
6.10	Cage(4,9) - výstupný graf . . . . .	88
6.11	Cage(6,4) - výstupný graf . . . . .	88
6.12	Cage(7,5) - výstupný graf [19] . . . . .	89
6.13	Časová náročnosť ukladania incidenčných matíc z CoCalc č. 1	91
6.14	Časová náročnosť ukladania kontrolných matíc z CoCalc č. 1 .	92
6.15	Časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 1 . . . . .	94
6.16	Časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 1 . . . . .	95
6.17	Časová náročnosť ukladania kontrolných matíc z CoCalc č. 2 .	98
6.18	Časová náročnosť ukladania kontrolných matíc z CoCalc č. 2 .	99
6.19	Časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 2 . . . . .	102
6.20	Časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 2 . . . . .	103
6.21	Časová náročnosť ukladania generujúcich matíc z CoCalc . . .	112
6.22	Časová náročnosť ukladania generujúcich matíc z konzolovej aplikácie . . . . .	113
6.23	Časová náročnosť ukladania grúp automorfizmov Cage/Rec z CoCalc . . . . .	116

6.24 Časová náročnosť ukladania grúp automorfizmov Cage/Rec z konzolovej aplikácie . . . . .	120
6.25 Časová náročnosť ukladania grúp automorfizmov lineárnych kódov z konzolovej aplikácie . . . . .	125
6.26 Obraz Cage(3,5) zostrojený z prvého automorfizmu C(15,64,5)	126
6.27 Obraz Cage(3,5) zostrojený z prvého automorfizmu C(15,64,5) s preusporiadanými vrcholmi . . . . .	127
6.28 Porovnanie obrazov z Cage(3,5) . . . . .	128
7.1 Cage(3,5) - získavanie údajov . . . . .	136
7.2 C(15,64,5) - získavanie údajov . . . . .	138
7.3 Porovnanie lineárneho kódu C(15,64,5) zostrojeného z klietky a z automorfizmu klietky . . . . .	139
7.4 Porovnanie obrazov z C(15,64,5) a Cage(3,5) . . . . .	140

# Zoznam tabuliek

3.1	Tabuľka známych $A_2(n, d)$ č. 1 [3, 16] . . . . .	8
3.2	Tabuľka známych $A_2(n, d)$ č. 2 [3, 16] . . . . .	9
3.3	Tabuľka známych rekordných grafov Rec(k,g) [12, 7] . . . . .	18
4.1	Tabuľka technických parametrov zariadenia . . . . .	21
6.1	Tabuľka výstupov graficky zobraziteľných klietok a rekordných grafov v CoCalc . . . . .	90
6.2	Tabuľka výstupov graficky zobraziteľných klietok a rekordných grafov v konzolovej aplikácii . . . . .	93
6.3	Tabuľka výstupov graficky nezobraziteľných klietok a rekordných grafov v CoCalc . . . . .	96
6.4	Tabuľka výstupov graficky nezobraziteľných klietok a rekordných grafov v konzolovej aplikácii . . . . .	100
6.5	Tabuľka výstupov lineárnych kódov so všetkými uvažovanými parametrami z CoCalc . . . . .	105
6.6	Tabuľka výstupov lineárnych kódov so všetkými uvažovanými parametrami z konzolovej aplikácie . . . . .	105
6.7	Tabuľka výstupov lineárnych kódov bez parametra optimálneho kódu z CoCalc . . . . .	106

6.8 Tabuľka výstupov lineárnych kódov bez parametra optimálneho kódu z konzolovej aplikácie . . . . .	107
6.9 Tabuľka výstupov lineárnych kódov bez minimálnej Hammingovej vzdialenosťi v kóde, parametrov perfektného a optimálneho kódu z CoCalc . . . . .	109
6.10 Tabuľka výstupov lineárnych kódov bez minimálnej Hammingovrj vzdialenosťi v kode, parametrov perfektného a optimálneho kódu z konzolovej aplikácie . . . . .	110
6.11 Tabuľka výstupov grúp automorfizmov získaných z klietok alebo rekordných grafov z CoCalc . . . . .	115
6.12 Tabuľka výstupov grúp automorfizmov získaných z klietok alebo rekordných grafov z konzolovej aplikácie . . . . .	118
6.13 Tabuľka výstupov grúp automorfizmov získaných z lineárnych kódov v CoCalc . . . . .	122
6.14 Tabuľka výstupov grúp automorfizmov získaných z lineárnych kódov v konzolovej aplikácii . . . . .	124

# Kapitola 1

## Úvod

Na konštrukciu lineárnych kódov využívame klietky a rekordné grafy [12], ktoré sú špecialnymi typmi grafu vzdialenosťi [7]. Graf vzdialenosťi nám v maticovej reprezentácii poskytuje incidenčnú maticu, na základe ktorej vieme vygenerovať lineárny kód [7, 11, 9]. Lineárny kód je možné skúmať z viacerých aspektov, ale v našej práci sa venujeme skúmaniu vlastností a symetrií lineárnych kódov. Pomocou grupy automorfizmov získanej z grafu vieme vygenerovať lineárny kód a porovnať ho s kódom získaným priamo z klietky alebo rekordného grafu. Grupu automorfizmov vieme získať aj z lineárnych kódov [17]. Skúmané lineárne kódy vieme vyhodnotiť. Návrh riešenia zahŕňa popis programovacieho jazyka, vývojového prostredia a návrh logiky samotného riešenia, kde detailne popisujeme, akým spôsobom získavame klietky, rekordné grafy, lineárne kódy a grupy automorfizmov. Samotné riešenie je interpretované v online aplikácii CoCalc [18] vo forme SageMath skriptov [19] a v konzolovej aplikácii s využitím programovacieho jazyka Python [8]. Konzolová aplikácia je spustiteľná len po inštalácii v SageMath konzole [19]. Výstupy navrhovaných riešení zahŕňajú okrem výsledkov spracovaných a zoobrazených v tabuľkách aj výstupné textové súbory oboch aplikácií.

# Kapitola 2

## Motivácia

Problémom pri komunikácii a prenose informácií cez médium alebo komunikačný kanál je možná strata alebo skreslenie informácie v dôsledku pôsobenia šumu [3]. Sekvencia kódov s rýchlosťou menšou ako je kapacita kanála má schopnosť opraviť všetky chyby spôsobené šumom pridaním redundantných symbolov k pôvodným údajom. Táto realizácia viedla k vývoju kódov na opravu chýb (ECC) [7], ktorých cieľom je zakódovať údaje pridaním určitého množstva redundancie do správy, aby bolo možné pôvodnú správu obnoviť a dekódovať [4]. Lineárne kódy sú podpriestory konečnorozmerných vektorových priestorov nad konečnými poľami [3]. S lineárnymi kódmi sme sa stretli už v našej bakalárskej práci [15], kde sme skúmali hustotu inverzií riedkych cyklických matíc v McElieceovom kryptosystéme. Zistili sme, že matica, ktorá predstavovala blok generujúcej matice, mala v niektorých prípadoch nízku hustotu a mohla by znamenať bezpečnostné riziko. V diplomovej práci sme nadviazali na tému lineárnych kódov a špecializujeme sa na generovanie lineárnych kódov z grafových štruktúr, skúmanie ich vlastností a symetrií. Lineárne kódy majú bohaté grupy automorfizmov, ktoré obsahujú množstvo informácií o uvažovanom kóde.

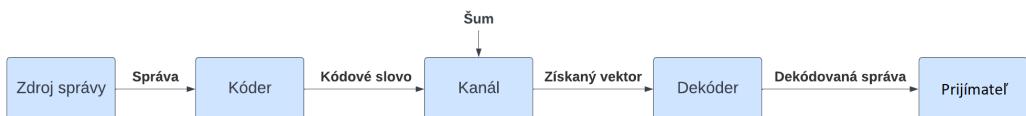
# Kapitola 3

## Analýza problému

### 3.1 Komunikácia a prenos informácií

Hlavným cieľom teórie kódovania je zabezpečiť, aby sa informácie dali posieľať z bodu A do bodu B bez toho, aby sa čokoľvek z nich stratilo alebo došlo k skresleniu pôvodnej informácie [4]. Medzi príklady komunikácie môžeme zaradiť internetovú komunikáciu, počúvanie hudby, komunikáciu v ľudskom jazyku, komunikáciu prostredníctvom telefónnej linky, vysokofrekvenčnú rádiovú komunikáciu, komunikáciu medzi satelitmi vo vesmíre alebo medzi sateľitom a Zemou [3]. Prenos informácií zo zdroja do cieľa je možný po preložení informácie do formátu, ktorý je následne pomocou komunikačného kanála alebo média možné odoslať [4]. Komunikačný kanál alebo médium si môžeme predstaviť ako vzduch, drôtové vedenia, optické vlákna [7], magnetické pamäťové zariadenia, kompaktné disky a akýkoľvek druh elektronických komunikačných zariadení, ako sú mobilné telefóny [5]. Prenos údajov môže byť do veľkej miery ovplyvnený vonkajším šumom, ktorý by mohol rušiť alebo skresľovať signál predstavujúci tieto údaje, a tým v nich spôsobiť chyby [7]. Vonkajší šum môžeme chápať ako ľudskú chybu pri posielaní správy, príliš

veľký hluk pri komunikácii medzi ľuďmi, blesk, tepelný šum, nedokonalosti zariadenia a iné [3]. V roku 1948 Shannon [1] identifikoval číslo nazývané kapacita kanála a dokázal, že ľubovoľne spoľahlivá komunikácia je možná v každom prípade, ak je rýchlosť prenosu dát nižšia ako kapacita kanála [5]. Sekvencia kódov s rýchlosťou menšou ako je kapacita kanála má schopnosť opraviť všetky chyby [7]. Detekcia a oprava chýb sa dosiahne pridaním redundantných symbolov k pôvodným údajom. Táto realizácia viedla k vývoju kódov na opravu chýb (ECC) [7]. Cieľom kódov na opravu chýb je zakódovať údaje pridaním určitého množstva redundancie do správy, aby bolo možné obnoviť pôvodnú správu a dekódovať ju so špecifickým stupňom presnosti v takom prípade, ak sa nevyskytlo príliš veľa chýb [3]. Na obrázku (Obr. 3.1) je znázornený prenos informácií od zdroja k prijímateľovi.



Obr. 3.1: Prenos informácií od zdroja k prijímateľovi [3]

Zdroj odošle správu. Bez úprav správy by sa správa prenesla priamo cez kanál, akýkoľvek šum by správu skreslil tak, že by sa nedala obnoviť. Z tohto dôvodu kóder pridáva redundanciu a takáto správa je v obrázku nazývaná kódové slovo. Kódové slovo sa prenáša cez kanál, kde šum vo forme chybového vektora narúša kódové slovo a vytvára získaný vektor [5]. Získaný vektor sa odošle dekóderu na dekódovanie, odstránenie chýb a nadbytočností. Dekóder vytvorí odhad pôvodnej správy [5] a dekódovaná správa sa odošle prijímateľovi.

Nech  $F_q$  je abeceda, ktorá je často považovaná za  $Z_q = \{0, 1, 2, \dots, q - 1\}$

[3]. Súbor sekvencií symbolov, kde každý symbol je vyberaný zo súboru  $F_q = \{\lambda_1, \lambda_2, \dots, \lambda_q\}$  q odlišných prvkov sa nazýva q-árny kód [3]. Ak je q prvočíslo, potom často považujeme abecedu  $F_q$  za konečné pole rádu q. 2-árne kódy sa nazývajú binárne kódy, 3-árne kódy sa označujú ako ternárne kódy [5, 3]. V našej práci sa budeme venovať binárnym kódom.

## 3.2 Lineárny kód

Uvažujeme binárny kód. Ak je výsledkom súčtu každých dvoch slov modulo dvomi kódové slovo, tak uvažovaný binárny kód je zároveň lineárny [7, 3]. V našej práci sa zameriavame na binárne lineárne kódy a ak budeme spomínať lineárny kód, máme na mysli práve binárny lineárny kód.

Nech  $F_2^n$  je priestor n-rozmerných vektorov. Lineárny kód je r-rozmerný lineárny podpriestor priestoru  $F_2^n$  a r nazývame dimensiou lineárneho kódu [5]. Množinu všetkých usporiadaných n-tíc priestoru  $F_2^n$  označujeme ako  $V(n,2)$  a jej prvky budeme nazývať vektory. V prípade, že lineárny kód je z  $V(n,2)$ , potom lineárny kód môžeme označiť ako  $[n,r]$ -kód [3]. Lineárny podpriestor rozmeru r obsahuje práve r lineárne nezávislých vektorov, ktoré tvoria bázu podpriestoru [13, 5].

### **Veta č. 1 (počet vektorov v lineárnom kóde)**

Predpokladajme, že  $\{v_1, v_2, \dots, v_r\}$  je bázou podpriestoru lineárneho kódu z  $V(n,2)$ . Potom každý vektor lineárneho kódu môže byť vyjadrený jednoznačne ako lineárna kombinácia základných vektorov a lineárny kód obsahuje presne  $2^r$  vektorov [3].

V prípade, že by sme chceli špecifikovať minimálnu Hammingovu vzdialenosť lineárneho kódu  $d$  (3.2.5), tak lineárny kód označujeme ako  $[n,r,d]$ -kód alebo  $(n,2^r,d)$ -kód (Veta č. 1) [3]. V našej práci budeme využívať najmä označenie  $(n,2^r,d)$ -kód, skrátene  $C(n,m,d)$ , kde  $n$  je dĺžka lineárneho kódu (3.2.3),  $m$  je maximálny počet slov (3.2.4) a  $d$  je minimálna Hammingova vzdialenosť v kóde (3.2.5).

### 3.2.1 Generujúca matica lineárneho kódu

Generujúca matica lineárneho kódu  $G$  je zostrojená z bázy lineárneho kódu  $C[n,r]$  (3.2). Riadky matice predstavujú prvky bázy, a zároveň predstavujú lineárne nezávislé vektory dĺžky  $n$  [13, 5]. Generujúcu maticu  $G$  si môžeme zadefinovať nasledovne. Nech je vektor  $\vec{s}$  vstup, teda nekódované slovo a vektor  $\vec{x}$  je výstup, teda kódované slovo, potom platí vzťah (3.1).

$$C[n, r] = \{\vec{s} \times G : \vec{s} \in F_2^n\}, \quad \vec{x} = \vec{s} \times G \quad (3.1)$$

### 3.2.2 Kontrolná matica lineárneho kódu

V  $r$ -rozmernom linearnom kóde  $C[n,r]$  v  $F_2^n$  existuje  $n-r$  lineárne nezávislých vektorov  $\vec{x}$  takých, že každé kódové slovo je kolmé na všetky tieto vektory. Keď týchto  $n-r$  vektorov zoberieme ako riadky matice, dostaneme kontrolnú maticu lineárneho kódu  $H$  [13, 5]. Ľubovoľný vektor  $\vec{x}$  je kódovým slovom práve vtedy, ak platí vzťah (3.2).

$$C[n, r] = \{\vec{x} \in F_2^n : H \times \vec{x}^T = 0\} \quad (3.2)$$

### 3.2.3 Dĺžka lineárneho kódu

Dĺžka lineárneho kódu  $n$  predstavuje dĺžku slov v kóde [13]. V generujúcej matici  $G$  predstavuje dĺžku lineárne nezávislých riadkov. V kontrolnej matici  $H$  reprezentuje takisto dĺžku riadkov matice [7]. V prípade, že uvažujeme bázu z  $V(n,2)$ , ktorá má  $n$  vektorov, tak platí vzťah (3.3) [3].

$$n = \dim(V(n, 2)) \quad (3.3)$$

### 3.2.4 Maximálny počet slov v kóde a Optimálny kód

Maximálny počet slov v kóde  $m$  je maximálny počet takých kódových slov, ktoré obsahuje lineárny kód  $C(n,m,d)$ . Nech parameter  $r$  predstavuje počet lineárne nezávislých riadkov generujúcej matice  $G$ , potom z vety o počte vektorov v lineárnom kóde (Veta č. 1) vyplýva vzťah (3.4) [3].

$$m = |C[n, r]| = 2^r \quad (3.4)$$

#### Veta č. 2 (Optimálny kód)

Nech  $A_2(n, d)$  je maximálny počet slov dĺžky  $n$  nad  $Z_2$  takých, že Hammingova vzdialenosť medzi ľubovoľnými dvoma kódovými slovami je aspoň  $d$ . Lineárny  $[n, r, d]$ -kód, pre ktorý platí  $2^r = A_2(n, d)$  sa nazýva optimálny lineárny kód  $C(n,m,d)$ , ak platí vzťah (3.5) [3].

$$A_2(n, d) = \max\{m = 2^r \mid \exists [n, r, d] \in F_2^n\} \quad (3.5)$$

V nasledujúcich tabuľkách (Tabuľka 3.1), (Tabuľka 3.2) si zobrazíme známe  $A_2(n, d)$  pre niektoré fixné dĺžky lineárneho kódu  $n$  a minimálne Hammingové vzdialosti v kóde  $d$  [3, 16].

<b>n</b>	<b>d = 3</b>	<b>d = 4</b>	<b>d = 5</b>	<b>d = 6</b>
5	4	-	2	-
6	8	4	2	2
7	16	8	2	2
8	20	16	4	2
9	40	20	6	4
10	72-79	40	12	6
11	144-158	72	24	12
12	256	144	32	24
13	512	256	64	32
14	1024	512	128	64
15	2048	1024	256	128
16	2816-3276	2048	258-340	256
17	5632-6552	2816-3276	512-673	258-340
18	10496-13104	5632-6552	1024-1237	512-673
19	20480-26168	10496-13104	2048-2279	1024-1237
20	40960-43688	20480-26168	2560-4096	2048-2279
21	81920-87333	40960-43688	4096-6941	2560-4096
22	163840-172361	81920-87333	8192-13674	4096-6941
23	327680-344308	163840-172361	16384-24106	8192-13674
24	$2^{19} - 599184$	327680-344308	17920-47538	16384-24106
25	$2^{20} - 1198368$	$2^{19} - 599184$	32768-84260	17920-47538
26	$2^{21} - 2396736$	$2^{20} - 1198368$	65536-157285	32768-84260
27	$2^{22} - 4792950$	$2^{21} - 2396736$	131072-291269	65536-157285
28	-	$2^{22} - 4792950$	-	131072-291269

Tabuľka 3.1: Tabuľka známych  $A_2(n, d)$  č. 1 [3, 16]

<b>n</b>	<b>d = 7</b>	<b>d = 8</b>	<b>d = 10</b>	<b>d = 12</b>	<b>d = 14</b>	<b>d = 16</b>
5	1	-	-	-	-	-
6	1	1	1	1	1	1
7	2	1	1	1	1	1
8	2	2	1	1	1	1
9	2	2	1	1	1	1
10	2	2	2	1	1	1
11	4	2	2	1	1	1
12	4	4	2	2	1	1
13	8	4	2	2	1	1
14	16	8	2	2	2	1
15	32	16	4	2	2	1
16	36-37	32	4	2	2	2
17	64	36	6	2	2	2
18	128	64	10	4	2	2
19	256	128	20	4	2	2
20	512	256	40	6	2	2
21	1024	512	42-47	8	4	2
22	2048	1024	64-84	12	4	2
23	4096	2048	80-150	24	4	2
24	4096-5421	4096	136-268	48	6	4
25	4104-9275	4096-5421	192-466	52-55	8	4
26	8192- 17099	4104-9275	384-836	64-96	14	4
27	16384- 32151	8192- 17099	512-1585	128-169	28	6
28	-	16384- 32151	1024-2817	178-288	56	8

Tabuľka 3.2: Tabuľka známych  $A_2(n, d)$  č. 2 [3, 16]

### 3.2.5 Hammingova vzdialenosť v kóde

Uvažujme vektory  $\vec{x}$  a  $\vec{y}$  ako kódové slová. Hammingová vzdialenosť vektorov  $d(\vec{x}, \vec{y})$ , pre ktoré platí, že  $\vec{x} \in F_2^n$  a  $\vec{y} \in F_2^n$  je počet koordinátov, na

ktorých sa vektory  $\vec{x}$  a  $\vec{y}$  líšia [13, 5]. Hammingova vzdialenosť v kóde d je taká najmenšia vzdialenosť v lineárnom kóde  $C(n,m,d)$ , ktorá je definovaná ako najmenšia vzdialenosť zo všetkých vzdialenosťí medzi kódovými slovami, pričom platí vzťah (3.6) [3].

$$d = \min\{d(\vec{x}, \vec{y}) | \vec{x}, \vec{y} \in C(n, m, d), \vec{x} \neq \vec{y}\} \quad (3.6)$$

Hammingova vzdialenosť v kóde je dôležitým parametrom lineárneho kódu  $C(n,m,d)$  z hľadiska opravy chýb [3].

### 3.2.6 Informačný pomer

Informačný pomer  $R$  (angl. Rate) udáva počet informácií alebo bitov nad celkovým počtom prenesených bitov [1]. V lineárnom kóde  $C[n,r]$  je možné informačný pomer vyjadriť ako podiel rozmeru lineárneho kódu  $r$  a dĺžky lineárneho kódu  $n$  [7]. V generujúcej matici  $G$  predstavuje  $r$  počet lineárne nezávislých riadkov a  $n$  reprezentuje dĺžku týchto riadov. V kontrolnej matici  $H$  predstavuje počet lineárne nezávislých riadkov rozdiel  $n - r$  (3.2.2). Dĺžku týchto riadkov, podobne ako v generujúcej matici  $G$  reprezentuje dĺžka lineárneho kódu  $n$ . Výpočet  $R$  je vyjadrený vzťahom (3.7) [7].

$$R = \frac{r}{n} \quad (3.7)$$

### 3.2.7 Perfektné lineárne kódy

#### Veta č. 3 (Sphere-packing ohraničenie)

Nech  $t$  je parameter, pre ktorý platí, že každé dve sféry polomeru  $t$  centrovane na odlišné kódové slová majú prázdný prienik. Pomocou parametra  $t$  môžeme vyjadriť minimálnu Hammingovu vzdialenosť v kóde d vzťahom (3.8) [3].

$$d = 2t + 1 \quad (3.8)$$

Pre binárny lineárny kód  $C(n,m,d)$  potom platí označenie  $C(n, m, 2t + 1)$  kód. Sphere-packing ohraničenie je dané vzťahom (3.9).

$$m \left\{ \binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{t} \right\} \leq 2^n \quad (3.9)$$

Počet všetkých vektorov v  $m$  sférach s polomerom  $t$  centrovaných na  $m$  kódových slov je daný ľavou stranou rovnice. Sphere-packing ohraničenie platí vždy vtedy, ak je ľavá strana rovnice menšia, nanajvýš rovnajúca sa  $2^n$ . Toto číslo vyjadruje celkový počet vektorov v  $F_2^n$ . V prípade, že sú obe strany rovnice v rovnosti, tak hovoríme, že lineárny kód  $C(n,m,d)$  je perfektný [3].

### 3.2.8 LDPC kódy

LDPC kódy sú lineárne samoopravné kódy, ktoré umožňujú prenos dát rýchlosťou blízkou kapacite kanálu a zároveň pre ne existujú vysoko účinné dekódovacie algoritmy [11, 9]. Tieto kódy majú veľmi riedku kontrolnú maticu  $H$ , pomocou ktorej sa dajú opraviť chyby v kódových slovách [13, 11]. Hlavnou nevýhodou väčšiny LDPC kódov je vysoká časová náročnosť ich kódovacieho algoritmu. Výhodou je paraleлизmus pri dekódovaní a jednoduché výpočtové operácie [7]. Téme LDPC kódov sme sa okrajovo venovali aj v bakalárskej práci, v ktorej sme skúmali hustotu inverzií riedkych cyklických matíc. Zamerali sme sa však na QC-MDPC McElieceov kryptosystém. Skratka QC reprezentuje kvázicyklický lineárny kód. V kvázicyklickom lineárnom kóde sa kontrolná matica  $H$  skladá z jednotlivých blokov, pričom platí, že jednot-

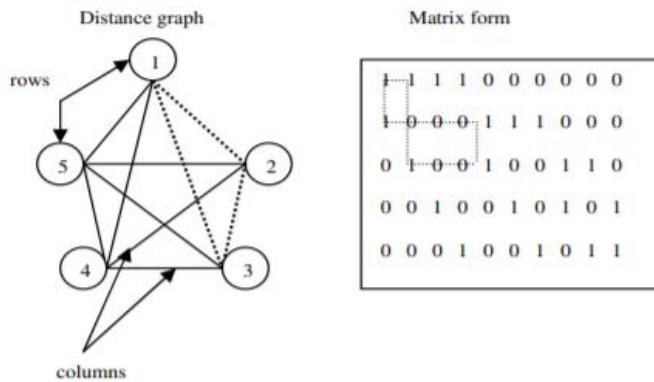
livé bloky sú tvorené z cyklických matíc [13]. Rozdiel medzi MDPC kódmi a LDPC je v hustote kontrolnej matice, ktorá môže byť pre MDPC kódy o niečo hustejšia ako pri LDPC kódoch [15]. V našej práci sa zameriavame na všeobecné LDPC kódy bez ohľadu na to, či sú kvázicyklické alebo nie.

### 3.3 Grafová reprezentácia LDPC kódov

#### 3.3.1 Incidenčná matica a graf vzdialenosťi

Nech  $L = (a_{i,j})$  je matica  $x \times y$  tvorená z prvkov hodnoty 0 alebo 1. Maticu L môžeme považovať za incidenčnú maticu elementov  $e_1, e_2, \dots, e_x$  oproti množinám  $S_1, S_2, \dots, S_y$ ; to znamená, že  $a_{i,j} = 0$  alebo  $a_{i,j} = 1$  podľa toho, či  $e_i$  je alebo nie je súčasťou  $S_j$  [2]. Graf vzdialenosťi je súvislý graf, ktorý má daný počet vrcholov  $|ver|$ , stupeň vrcholov k, v ktorom minimálna dĺžka cyklu medzi vrcholmi a hranami je g [7]. Vrcholy grafu vzdialenosťi reprezentujú riadky incidenčnej matice L a počet takýchto vrcholov v grafe vzdialenosťi zodpovedá počtu riadkov v incidenčnej matici L. Hrany grafu vzdialenosťi reprezentujú stĺpce incidenčnej matice L a počet takýchto hrán v grafe vzdialenosťi zodpovedá počtu stĺpcov v incidenčnej matici L. Stĺpec incidenčnej matice L potom predstavuje takú množinu hrán, ktorá formuje kompletnejší graf vzdialenosťi medzi vrcholmi spojenými v stĺpci [7].

Incidenčná matica L môže byť zobrazená ako graf vzdialenosťi [7]. Nasledujúci obrázok ilustruje vzťah medzi incidenčnou maticou a grafom vzdialenosťi z ktorého je incidenčná matica L odvodená na obrázku (Obr. 3.2).



Obr. 3.2: Vzťah medzi grafom vzdialenosť a incidenčnou maticou [7]

Graf vzdialenosť je formovaný cestami, ktoré sa skladajú buď z hrán alebo vrcholov. Cestu, ktorá sa začína a končí tým istým vrcholom alebo cestu, ktorá sa začína a končí tou istou hranou budeme nazývať obvod grafu vzdialenosť (3.3.3) [7]. Na obrázku (Obr. 3.2) si môžeme všimnúť, že v incidenčnej matici  $L$  je obvod reprezentovaný uzavretou cestou, ktorá sa skladá z pospájaných prvkov hodnoty 1 po vertikálnej a horizontálnej líni tak, ako je zobrazené na obrázku. Môžeme si všimnúť, že prvý riadok (vrchol 1) je spojený s druhým riadkom (vrchol 2) na prvej pozícii vertikálne, pretože oba riadky majú na začiatku prvok 1. V druhom riadku hľadáme horizontálne najbližší prvok hodnoty 1, ktorý by sme opäť mohli spojiť vertikálne s ďalším prvkom 1, nachádzajúcim sa v inom riadku. V prípade, že dokážeme vertikálne spojiť prvok 1 v ľubovoľnom riadku opäť s prvým riadkom, cesta sa uzavrie a doštávame obvod incidenčnej matice  $L$ . Východiskový riadok nemusí byť nutne prvý, ale aby bol obvod kompletný, je nutné východiskovým riadkom skončiť. Jednotlivé obvody v grafe vzdialenosť korešpondujú s obvodmi v incidenčnej matici  $L$ .

### 3.3.2 Veľkosť obvodu grafu

Veľkosť obvodu grafu  $g$  ovplyvňuje rýchlosť dekódovania LDPC kódu. V grafovej reprezentácii LDPC kódu sa jedná o najmenší cyklus v grafe. Jeho dĺžku vypočítame buď pomocou vrcholov alebo pomocou hrán. Ako sme spomínali, jednotlivé obvody v grafe vzdialenosť korešpondujú s obvodmi v kontrolnej matici  $H$ . V kontrolnej matici  $H$  kódu je dĺžka obvodu  $2g$  [6], pretože cyklus medzi riadkami a stĺpcami je vedený nielen vertikálne, ale aj horizontálne, pričom do veľkosti obvodu sa započítavaju všetky uvažované prepojenia s prvkami hodnoty 1. Z toho vyplýva, že cyklus grafu reprezentuje iba polovicu maticového kódu [6].

### 3.3.3 Automorfizmus grafu

Automorfizmus grafu je permutácia  $\phi$  všetkých vrcholov grafu, ktorá zachováva jeho štruktúru takým spôsobom, že akékoľvek 2 vrcholy  $U$  a  $V$  susedia iba vtedy a len vtedy ak platí, že  $\phi(U)$  susedí s  $\phi(V)$  [12]. Množina všetkých automorfizmov grafu tvorí grupu automorfizmov  $\text{AutGroup}(\text{Cage})$  [12].

### 3.3.4 Automorfizmus lineárneho kódu

Nech  $S_n$  je symetrická grupa, ktorá pôsobí na  $F_2^n$  akciou  $v\phi := (v\phi_{(1)}^{-1}, \dots, v\phi_{(n)}^{-1})$ , kde  $v = (v_1, \dots, v_n) \in F_2^n$  a  $\phi \in S_n$ . Nech  $C$  je binárny lineárny kód, potom ak  $v\phi \in C$  pre všetky  $v \in C$ ,  $\phi$  je automorfizmus binárneho lineárneho kódu  $C$  a množina všetkých automorfizmov binárneho lineárneho kódu  $C$  sa nazýva grupa automorfizmov binárneho lineárneho kódu  $\text{AutGroup}(C)$  [17].

### 3.4 Konštrukcia LDPC kódov

Na konštrukciu LDPC kódov môžeme využiť incidenčné matice L, ktoré získame z grafov vzdialenosťi. V prácach [11, 9, 7] autori používajú incidenčnú maticu L ako kontrolnú maticu H lineárneho kódu, z ktorej už je možné získať lineárny kód C(n,m,d) a skúmať jeho vlastnosti, ako aj vlastnosti samotných grafov.

#### 3.4.1 Klietka a rekordný graf

Grafy vzdialenosťi delíme na regulárne s vrcholmi rovnakého stupňa k a ne-regulárne s vrcholmi rôznych stupňov [7]. V našej práci sa budeme venovať regulárnym grafov vzdialenosťi. Klietka Cage(k,g) je k-regulárny graf obvodu g s počtom vrcholov  $|ver|$  väčším, nanajvyš rovnajúcim sa dolnému ohraničeniu počtu vrcholov, tiež známym ako Moorové ohraničenie  $M(k,g)$  [12]. Výpočet Moorovho ohraničenia sa líši podľa toho, či je obvod g uvažovannej klietky nepárny (3.10) alebo párný (3.11):

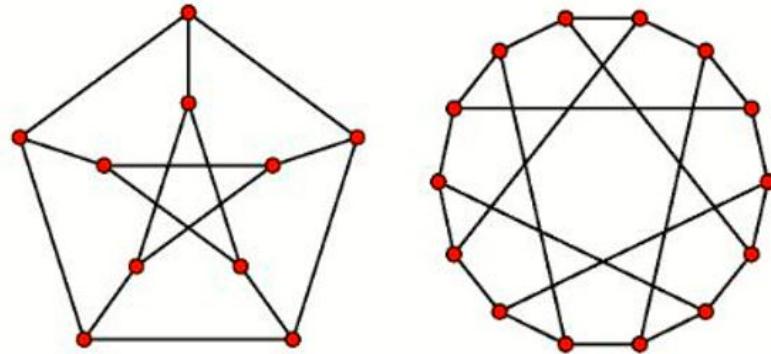
nepárny obvod:

$$M(k, g) \leq 1 + \sum_{i=0}^{(g-3)/2} k(k-1)^i = \frac{k(k-1)^{(g-1)/2} - 2}{k-2} \quad (3.10)$$

párný obvod:

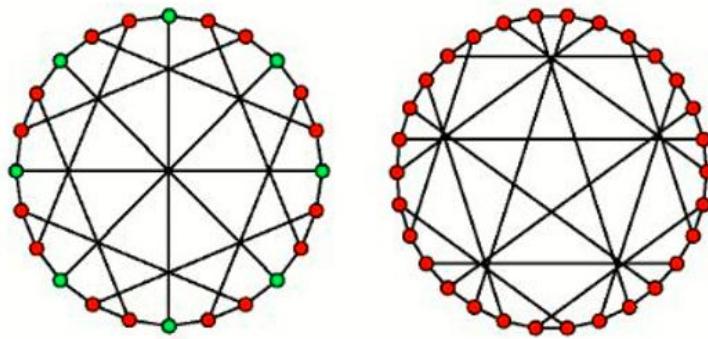
$$M(k, g) \leq 2 \sum_{i=0}^{(g-2)/2} k(k-1)^i = \frac{2(k-1)^{g/2} - 2}{k-2} \quad (3.11)$$

Aj keď neexistuje jednotná konštrukcia klietok, existuje niekoľko známych klietok pre stupeň vrchola k a obvod g [12]. Ukážeme si niektoré z nich.



Obr. 3.3: Petersenov graf (vľavo) a Heawoodov graf (vpravo)[12]

**Petersenov graf** (Obr. 3.3) predstavuje klietku Cage(3,5) a počet vrcholov  $|ver|$  má 10. **Heawoodov graf** (Obr. 3.3) je klietkou Cage(3,6), počet vrcholov  $|ver|$  má 14 a počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má 336 [12].



Obr. 3.4: McGeeho graf (vľavo) a Tutteov-Coxeterov graf (vpravo) [12]

**McGeeho graf** (Obr. 3.4) znázorňuje klietku Cage(3,7), počet vrcholov  $|ver|$  má 24 a počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má 32. **Tutteho-Coxterov graf** (Obr. 3.4) predstavuje klietku Cage(3,8), počet vrcholov  $|ver|$  má 30 a počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má 1440. **Balabanov graf** znázorňuje klietku Cage(3,11), počet vrcholov  $|ver|$  má 112 a počet automorfizmov

$|\text{AutGroup}(\text{Cage})|$  má 64. **Bensonov graf** je klietkou Cage(3,12), počet vrcholov  $|ver|$  má 126 a počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má 12096. **Robertsonov graf** predstavuje klietku Cage(4,5). **Exoo, McKay a Nadonov graf** znázorňujú klietku Cage(4,7). Cage(5,5) je skupina grafov, ktoré majú počty automorfizmov  $|\text{AutGroup}(\text{Cage})|$  20, 30 a 120. **Hoffmanov-Singletonov graf** je klietkou Cage(7,5). **O'Keefe a Wongov graf** predstavujú klietku Cage(7,6) [12].

Doposiaľ sme uvažovali známe klietky Cage( $k,g$ ). Jednalo sa o grafy, ktorých počty vrcholov  $|ver|$  sú preukázateľne najmenšie a také aj ostatné. Okrem klietok poznáme ešte **rekordné grafy Rec( $k,g$ )**. Rekordný graf Rec( $k,g$ ) je graf, ktorého počet vrcholov  $|ver|$  je najmenší v súčasnosti známy. Hoci niektoré z týchto grafov môžu byť v skutočnosti klietky, väčšina bude s najväčšou pravdepodobnosťou nakoniec nahradená menšími grafmi. [12] V nasledujúcej tabuľke (Tabuľka 3.3) zobrazujeme rekordné grafy Rec( $k,g$ ) so známym počtom vrcholov a automorfizmov.

<b>Rec(k,g)</b>	<b> ver </b>	<b> AutGroup(Cage) </b>
Rec(3,13)	272	-
Rec(3,14)	384	-
Rec(3,15)	620	14880
Rec(3,16)	960	96
Rec(3,17)	544	2176
Rec(3,18)	2560	640
Rec(3,19)	4324	51888
Rec(3,20)	5376	2688
Rec(3,21)	16028	-
Rec(3,22)	16206	-
Rec(3,23)	49326	-
Rec(3,24)	49608	-
Rec(3,25)	108906	-
Rec(8,5)	80	-
Rec(9,5)	96	-
Rec(10,5)	124	-
Rec(11,5)	154	-
Rec(12,5)	203	-
Rec(13,5)	230	-

Tabuľka 3.3: Tabuľka známych rekordných grafov Rec(k,g) [12, 7]

# Kapitola 4

## Návrh riešenia

V kapitole návrh riešenia uvádzame použité programovacie jazyky, vývojové prostredie a návrh samotnej logiky oboch riešení.

### 4.1 Programovací jazyk a prostredie

Riešenie navrhujeme v programovacom jazyku Python [8] aplikovaním knižnice SageMath [19]. SageMath ponúka množstvo funkcií určených na prácu s grafmi a lineárnymi kódmi  $C(n,m,d)$ . Navrhujeme dve riešenia. Prvé riešenie je implementované v online aplikácii CoCalc [18] vo forme SageMath skriptov. Druhé riešenie je implementované ako konzolová aplikácia v programovacom jazyku Python [8], kde je knižnica SageMath naimportovaná.



Obr. 4.1: SageMath a CoCalc [18, 19]

#### 4.1.1 Návrh 1. riešenia - skripty v CoCalc

CoCalc (z ang. Collaborative Calculation in the Cloud) je virtuálny online pracovný priestor určený na výpočty, výskum, spoluprácu a vytváranie dokumentov. Používateľ potrebuje iba webový prehliadač. Práca v CoCalc je realizovaná vo forme projektov. Každý projekt pozostáva zo súborov, ku ktorým máte prístup iba vy a vaši spolupracovníci. Tieto súbory je možné upravovať súčasne, čo znamená, že vaše zmeny sa medzi všetkými účastníkmi projektu synchronizujú v reálnom čase. Otvárajú sa v pridruženom online editore a používateľ môže interaktívne pracovať v prostredí CoCalc [18]. Napríklad súbory končiace na \*.sagews sú určené na prácu so SageMath knižnicou [19] v SageMath skriptoch, súbor \*.ipynb spúšťa implementácie CoCalc pre Ju- pyter Notebook a súbor \*.tex otvára editor pre dokumenty LaTeX. V našom navrhovanom riešení využijeme práve SageMath skripty. CoCalc je voľne dostupný vo forme skúšobnej verzie. Bez zakúpenia licencie je jej výkon 10-krát pomalší, používateľ nemôže inštalovať balíky, klonovať repozitáre z GitHub alebo stahovať datasety [18]. Nevýhodami Sage skriptov, ktoré nás limitujú sú pomalší výkon, výpočtovo náročný a neprehľadný prístup k výsledkom. Z týchto dôvodov navrhujeme aj druhé riešenie.

#### 4.1.2 Návrh 2. riešenia - vlastná konzolová aplikácia

Ako druhé riešenie navrhujeme vývoj konzolovej aplikácie v programovacom jazyku Python [8], do ktorého bude potrebné knižnicu SageMath [19] naimportovať. Keďže už nevyužívame online úložný priestor, ale pracovný priestor v našom notebooku, v tabuľke (Tabuľka 4.1) uvažujeme jeho technické parametre.

názov	<b>DELL Precision 15 (7550)</b>
RAM	<b>16 GB</b>
procesor	<b>Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz</b>
operačný systém	<b>Windows 10 Pro (procesor typu x64)</b>
grafické karty	<b>NVIDIA Quadro T2000</b>
úložné médium	<b>SSD</b>

Tabuľka 4.1: Tabuľka technických parametrov zariadenia

Využitím vlastného zariadenia do určitej miery eliminujeme problém prvého riešenia v pomalšom výkone. Stále však musíme brať do úvahy isté obmedzenia výkonu vyplývajúce z technických parametrov. Ako vývojové prostredie využijeme Visual Studio Code [14]. Aplikáciu budeme spúštať volaním hlavnej triedy, odkiaľ bude volaná trieda s hlavným menu. Okrem hlavného menu uvažujeme aj ďalšie vnorené menu, ku ktorým sa bude môcť používateľ dostať a k výsledkom sa vie navigovať výberom z rôznych možností. Týmto spôsobom zlepšíme prístup k výsledkom a znížime výpočtovú náročnosť, pretože nebude potrebné počítať všetky výpočty ako v prvom riešení, ale iba tie najnutnejšie, ktoré sú potrebné k dosiahnutiu požadovaného výsledku. Na spustenie aplikácie je potrebné stiahnuť si a nainštalovať aplikáciu SageMath [19], ktorá je voľne dostupná na stiahnutie, inštaláciu a podporuje aj skripty a projekty v jazyku Python [8]. Po stiahnutí získame 3 aplikácie: SageMath, SageMath Shell a SageMath Notebook [19]. Využívame iba konzolu SageMath, kde budeme spúštať našu aplikáciu načítaním takého Pythonového súboru, ktorý obsahuje hlavnú triedu na spustenie konzolovej aplikácie.

## 4.2 Návrh logiky samotného riešenia

### 4.2.1 Generovanie klietok a rekordných grafov

Existuje viacero možností ako vygenerovať klietku Cage( $k,g$ ) alebo rekordný graf Rec( $k,g$ ) v programe SageMath [19]. V našej práci uvažujeme 3 spôsoby generovania Cage( $k,g$ ) alebo Rec( $k,g$ ). Prvý spôsob je využitie grafov, ktoré SageMath ponúka ako vopred naimplementované grafy [19]. Druhý spôsob sa opiera o naprogramovanie vlastného parsera, ktorý dokáže spracovať vstupné textové súbory zoznamu susedností [21] a z nich vytvoriť Cage( $k,g$ ) alebo Rec( $k,g$ ). Tretí spôsob spočíva vo využití známych údajov, ktoré sú pre danú klietku Cage( $k,g$ ) všeobecne známe (počet vrcholov |ver|, typ grafu a podobne) [12, 6] a navrhnutie vlastného postupu, ktorý bude viesť k vyzenerovaniu Cage( $k,g$ ) alebo Rec( $k,g$ ).

#### Sage grafy ako vstupné dátá

Program Sage ponúka zopár vopred naimplementovaných grafov, ktoré môžeme využiť ako klietky Cage( $k,g$ ) [19]. Na generovanie využijeme triedu *sage.graphs.graph\_generators.GraphGenerators* [19]. Trieda pozostáva z konštruktorov pre niekoľko bežných grafov. Ku generátoru zástupcov pristupujeme cez funkciu *graphs()* [19]. Na zobrazenie známych grafov Sage využíva Spring-layout algoritmus [19]. Z grafov využívame: Petersenov graf - Cage(3,5), Heawoodov graf - Cage(3,6), McGeeho graf - Cage(3,7), Tutteho-Coxeterov graf - Cage(3,8), Balabanov graf - Cage(3,10), Robertsonov graf - Cage(4,5), Hoffman-Singletonov graf - Cage(7,5). Tieto grafy už majú vopred naimplementované všetky vrcholy a hrany medzi jednotlivými vrcholmi [19].

### Zoznam susedností ako vstupné dáta

Uvažujeme existujúce vstupné textové súbory, ktoré nesú informácie o susednostiach všetkých uvažovaných vrcholov jednotlivo pre 1 klietku  $Cage(k,g)$  alebo rekordný graf  $Rec(k,g)$  [21]. Riadok predstavuje zoznam vrcholov, s ktorými susedí konkrétny vrchol. Navrhнемe si vlastný parser, ktorý nám rozdelí tieto prebrate vstupné dáta [21] na zoznamy vrcholov. Z nich bude možné vytvoriť hrany. Tieto hrany bude možné pridať prázdnemu grafu ( pomocou funkcie  $EmptyGraph()$ ), [19] a tým z neho vytvoriť  $Cage(k,g)$  alebo  $Rec(k,g)$ .

### Vlastný návrh generovania

Generovanie klietok bez vstupných dát nemá jednotný prístup. Je potrebné k nim pristupovať jednotlivo alebo preskúmať skupiny, ktoré sa generujú podobným spôsobom [12]. Navrhujeme vlastný spôsob generovania klietky  $Cage(6, 4)$  na základe parametrov  $k$  a  $g$  bez ďalších vstupných parametrov. Budúcu klietku vygenerujeme ako bipartitný graf. V Sage použijeme funkciu  $DegreeSequenceBipartite(s,s)$  [19], ktorá bude mať 2 rovnaké parametre  $s$ , pričom každý predstavuje zoznam vrcholov. Najskôr je potrebné si vypočítať Moorové ohraničenie pre klietku  $M(6,4)$  pre minimálny počet vrcholov.  $Cage(6,4)$  je taký typ klietky, ktorej počet vrcholov  $|ver|$  je rovnaký ako minimálny počet vrcholov z  $M(k,g)$  [7]. Tieto vrcholy rozdelíme na 2 zoznamy takým spôsobom, že každý zoznam bude obsahovať  $\frac{m}{2}$  vrcholov stupňa  $k$ , a teda platí vzťah (4.1).

$$s = [k, k, k, k, k, k] \quad len(s) = \frac{m}{2} \quad (4.1)$$

Budúcej klietke  $\text{Cage}(k,g)$  nastavíme vrcholy pomocou funkcie  $\text{set\_vertex}()$  [19]. Výsledný graf bude  $\text{Cage}(6,4)$ .

#### 4.2.2 Validácia klietky a rekordného grafu

Na validáciu existencie klietok  $\text{Cage}(k,g)$  a rekordných grafov  $\text{Rec}(k,g)$  bude potrebné zistiť Moorové ohraničenie pre klietku  $M(k,g)$ , na základe ktorého vieme otestovať počet vrcholov  $|ver|$ , ako aj existenciu klietky  $\text{Cage}(k,g)$  podľa vzťahu (3.10). Ako vstup využijeme parametre stupeň  $k$  a obvod  $g$ . Pomocou vstupných parametrov vieme vypočítať Moorové ohraničenie pre klietku  $M(k,g)$ , ktoré porovnáme so skutočným počtom vrcholov  $|ver|$  našej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ . Následne vieme otestovať aj počet hrán  $|edg|$ , ktoré vieme rovnako na základe Moorového ohraničenia  $M(k,g)$  vypočítať a porovnať očakávaný počet hrán  $|edg|$  s naším skutočným počtom hrán našej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ .

#### 4.2.3 Získavanie údajov z klietky alebo rekordného grafu

Sage ponúka zopár vopred naimplementovaných funkcií na získanie údajov z klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ , niektoré však bude potrebné naprogramovať. Údaje, ktoré budeme pri  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  sledovať, budú vrcholy a hrany, ako aj počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  zistený priamo z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Okrem  $|\text{AutGroup}(\text{Cage})|$  sa budeme zaoberať aj samotnou grupou automorfizmov pre  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Špeciálny údaj bude aj incidenčná matica, ktorú budeme v ďalšej časti spracovávať v spojitosti s lineárnym kódom  $C(n,m,d)$ . Z incidenčnej matici vieme zistiť kontrolnú maticu  $H$  a informačný pomer  $R$ .

### Počet vrcholov, hrán a automorfizmov grafu

Z vygenerovanej klietky Cage( $k,g$ ) alebo rekordného grafu Rec( $k,g$ ) v podobe grafu, ktorému sme na základe zoznamu susednosti [21] zadefinovali vrcholy a hrany alebo na zaklade grafu, ktorý ich už mal zadefinované, vieme získať zoznam všetkých vrcholov pomocou funkcie *vertices()* [19], zoznam všetkých hrán (pomocou funkcie *edges()* [19]) a zoznam všetkých automorfizmov grafu (pomocou funkcie *automorphism\_group()* [19]). Pre zistenie počtu vrcholov  $|ver|$ , hrán  $|edg|$  a počtu automorfizmov  $|\text{AutGroup}(\text{Cage})|$  nám stačí iba zistiť veľkosť ich zoznamov. Skúmame aj samotnú grupu automorfizmov pre Cage( $k,g$ ) alebo Rec( $k,g$ ). Pre ďalšiu prácu so samotnou grupou automorfizmov bude dôležité označenie vrcholov. Tie musia byť označené kladnými číslami ako uvedieme neskôr v sekcii (4.2.6).

### Incidenčná matica a kontrolná matica

Z klietky Cage( $k,g$ ) alebo rekordného grafu Rec( $k,g$ ) vieme získať incidenčnú maticu (pomocou funkcie *incidence\_matrix()* [19]). V prácach [11, 9, 7] používajú incidenčnú maticu ako kontrolnú maticu lineárneho kódu  $H$ , avšak pomocou Gauss-Jordanovej eliminácie [10] incidenčnej matice  $L$  ľahko zistíme, že matica neobsahuje iba lineárne nezávislé riadky. Z tohto dôvodu sme sa rozhodli, že kontrolnou maticou LDPC kódu bude incidenčná matica  $L$ , ktorá po Gauss-Jordanovej eliminácii [10] nebude obsahovať žiadne lineárne závislé riadky, ostane v riadkovom echelónovom tvare [10] a bude obsahovať  $n - r$  lineárne nezávislých riadkov, ako je uvedené v sekcii (3.2.2). Pomocou funkcie *echelon\_form()* [19] získame z incidenčnej matice  $L$  maticu v echelónovom tvare [10], pričom ak obsahovala  $L$  lineárne závislé riadky, tak v echelónovom tvare získame vynulovaný taký počet riadkov, koľko ich bolo lineárne závislých. Napokon kontrolnú maticu  $H$  vytvoríme tým, že budeme

porovnávať riadky s nami vytvoreným nulovým vektorom dĺžky  $n$ , pričom si v nej ponecháme iba "nenulové" riadky, čiže tie, ktoré niesú tvorené iba z nulových prvkov.

### Informačný pomer

V práci [7] autor počíta informačné pomery z incidenčných matíc  $L$ , ktoré považuje za kontrolné matice  $H$ . V našej práci však budeme vychádzať z kontrolných matíc  $H$ , ktoré sme odvodili od incidenčných matíc vyššie v sekcií (4.2.3). Informačný pomer  $R$  udávajúci počet informácií alebo bitov nad celkovým počtom prenesených bitov vypočítame pomocou už známeho počtu riadkov kontrolnej matice  $H$  a známeho počtu stĺpcov kontrolnej matice  $H$  vychádzajúc zo vzťahu (3.7). Parameter  $n$  predstavuje počet stĺpcov matice  $H$  a z počtu riadkov matice  $H$  vieme odvodiť parameter  $r$  pomocou vzťahu (4.2):

$$r = n - |H\text{rows}| \quad (4.2)$$

Následne pomocou vzťahu (3.7) navrhнемe výpočet  $R$ .

#### 4.2.4 Lineárny kód a generujúca matica získaná z kontrolnej matice

Z kontrolnej matice  $H$  je možné získať lineárny kód  $C(n,m,d)$  pomocou existujúcej funkcie v Sage `codes.from_parity_check_matrix(H)` [19] a následne z neho generujúcu maticu lineárneho kódu  $G$  pomocou funkcie `systematic_generator_matrix()` [19]. V ďalšom kroku nás bude zaujímať minimálna Hammingova vzdialenosť  $d$ , počet kódových slov  $m$ , grupa automorfizmov  $\text{AutGroup}(C)$ , porovnanie s optimálnymi lineárnymi kódmi  $o(n,m,d)$

a s perfektnými lineárnymi kódmi  $p(n,m,d)$ .

### **Minimálna Hammingova vzdialenosť, maximálny počet slov a grupa automorfizmov**

Minimálnu Hammingovu vzdialenosť v kóde  $d$  je možné zistiť z lineárneho kódu  $C(n,m,d)$  pomocou funkcie *minimum\_distance()* [19]. Maximálny počet slov  $m$  je možné zistiť z generujúcej matice  $G$  pomocou vzťahu (3.4). Grupu automorfizmov  $\text{AutGroup}(C)$  je možné zistiť z lineárneho kódu  $C(n,m,d)$  pomocou funkcie *permutation\_automorphism\_group()* [19] a počet automorfizmov  $|\text{AutGroup}(C)|$  je možné zistiť z grupy automorfizmov  $\text{AutGroup}(C)$  pomocou funkcie *order()* [19]. Získavanie grupy automorfizmov lineárneho kódu  $\text{AutGroup}(C)$  podrobnejšie popisujeme v sekcií (4.2.7).

### **Porovnanie s optimálnymi kódmi**

Zadefinujeme si parameter  $o(n,m,d)$  na základe vety o optimálnom kóde (Veta č. 2) ako podiel nášho počtu slov  $m$  a tabuľkovej hodnoty pre optimálny kód  $A_2(n, d)$  podľa vzťahu (4.3).

$$o(n, m, d) = \frac{m}{A_2(n, d)} \quad (4.3)$$

Tento parameter v intervale  $<0,1>$  znázorňuje, akým spôsobom sa líšime od optimálneho kódu ( $o(n,m,d) = 1$  by znamenalo, že uvažujeme optimálny kód). Výpočet realizujeme mimo navrhovaných softvérových riešení.

### Porovnanie s perfektnými kódmi

Zadefinujeme si parameter  $p(n,m,d)$  na základe vety o perfektnom kóde (Veta č. 3), ak vieme definovať parameter  $d = 2t + 1$  podľa vzťahu (4.4).

$$p(n, m, d) = \frac{m \left\{ \binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{t} \right\}}{2^n} \quad (4.4)$$

Tento parameter v intervale  $<0,1>$  znázorňuje, akým spôsobom sa líšime od perfektného kódu ( $p(n,m,d) = 1$  by znamenalo, že kód je perfektný).

#### 4.2.5 Validácia lineárneho kódu

Na validáciu využijeme vzťah z definície lineárneho kódu  $C(n,m,d)$  a jeho kontrolnej matice  $H$  (3.2) a odvodíme vzťah (4.5)

$$G \times H^T = 0 \quad (4.5)$$

Vstupmi validácie sú kontrolná matica  $H$  a generujúca matica  $G$  toho istého lineárneho kódu. Súčin generujúcej matice  $G$  a transponovanej kontrolnej matice  $H$  sa musí rovnať nulovej matici [3], ktorá má počet riadkov rovnajúci sa počtu riadkov generujúcej matice  $G$  a počet stĺpcov rovnajúci sa počtu stĺpcov transponovanej kontrolnej matice  $H$ .

#### 4.2.6 Generovanie lineárnych kódov z grupy automorfizmov grafu

Grupu alebo v našom prípade zoznam všetkých automorfizmov grafu popisujeme v sekcií (3.3.3) a vieme ju získať pomocou funkcie *automorphism\_group()*

[19]. Na to, aby sme určili počet automorfizmov z klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$  nám stačí zistiť veľkosť zoznamu  $|\text{AutGroup}(\text{Cage})|$  všetkých automorfizmov. Funkcia *automorphism\_group()* [19] nám však nevracia automorfizmy v úplnom tvaru, aby sme z nich boli schopní vygenerovať obraz pôvodnej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ . Funkcia *Permutation()* [19], ktorej jedným z argumentov je automorfizmus, berie do úvahy iba kladné hodnoty vrcholov a automorfizmus musí obsahovať aj informáciu o vrcholoch, ktoré ostávajú na pôvodnom mieste aj v obraze  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Z tohto dôvodu je funkcia *automorphism\_group()* [19] sama o sebe nedostatočná a je nutné pridať automorfizmu informáciu o chýbajúcich vrcholoch. Ako prvý krok je nutné si vrcholy klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$  označiť tak, aby obsahovali iba kladné hodnoty pomocou funkcie *relabel()* [19], pričom jej vstup tvorí usporiadaný zoznam vrcholov od jedna až po počet vrcholov  $|V|$  daného grafu. Následne si upravíme automorfizmus pridaním informácie o chýbajúcich vrcholoch. Neskôr je možné použiť funkciu *Permutation()* [19], ktorej vstupom je automorfizmus a výstupom je zoznam vrcholov v takom poradí, akým budú označovať obraz  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Tento zoznam aplikujeme ako vstup funkcie *relabel()* [19], ktorú zavoláme opäťovne a dostaneme nový obraz  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Z tohto obrazu už pomocou nami implementovaných metód *getIncidenceMatrix()*, *getParityCheckMatrix()* a *getGeneratorMatrix()* získame zo všetkých grafových automorfizmov najskôr incidenčné matice  $L$ , ktoré porovnávame s incidenčnou maticou  $L$  získanou priamo z uvažovanej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ . Z incidenčných matíc automorfizmov grafu získame kontrolné matice  $H$  automorfizmov grafu, ktoré porovnávame s kontrolnou maticou  $H$  získanou priamo z uvažovanej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ . Z kontrolných matíc automorfizmov grafu

získame generujúce matice  $G$  automorfizmov grafu, ktoré porovnávame s generujúcou maticou  $G$  získanou priamo z uvažovanej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ .

#### 4.2.7 Generovanie grupy automorfizmov z lineárneho kódu

Grupu automorfizmov lineárneho kódu popisujeme v sekcii (3.3.4) a vieme ju zistiť pomocou funkcie *permutation\_automorphism\_group()* [19]. Následne je z nej možné získať zoznam automorfizmov pomocou funkcie *list()* [19]. Na to, aby sme určili počet automorfizmov z lineárneho kódu nám stačí zistiť veľkosť zoznamu  $|\text{AutGroup}(C)|$  všetkých automorfizmov. Funkcie *permutation\_automorphism\_group()* [19] a *list()* [19] nám vrátia zoznam permutácií hrán, ktoré sme doposiaľ v grafovej reprezentácii neuvažovali. V grafovej reprezentácii by bol automorfizmus lineárneho kódu permutácia  $\phi$  všetkých hrán grafu, ktorá zachováva jeho štruktúru takým spôsobom, že akékoľvek 2 hrany  $edgU$  a  $edgV$  susedia iba vtedy a len vtedy, ak platí, že  $\phi(edgU)$  susedí s  $\phi(edgV)$ . Pre lepšiu orientáciu v grafovej reprezentácii lineárneho kódu  $C(n,m,d)$  preto potrebujeme pridať označenie hrán. Pomocou nami naprogramovaných funkcií *setDefaultVertices(graph)*, použitím SageMath funkcie *relabel()* [19] a *setDefaultEdges(graph)*, použitím SageMath funkcie *set\_edge\_label* [19] si nastavíme pôvodné označenie hrán a vrcholov grafu, ktorý reprezentuje incidenčná matica  $L$ . Funkcia *permutation\_automorphism\_group()* [19] nám nevracia automorfizmy v úplnom tvaru, aby sme z nich vygenerovali obraz lineárneho kódu reprezentovaný grafoom incidenčnej matice potrebnej k získaniu kontrolnej matice spomínaného lineárneho kódu. Pomocou nami navrhnutej funkcie *getPermutationsOfAutomorphisms()* v lineárnom kóde  $C(n,m,d)$  pridávame zvyšné údaje permu-

tácií tak, aby permutácia obsahovala informácie o všetkých hranách grafu reprezentujúceho incidenčnú maticu  $L$ . Okrem zvyšných hrán pridávame aj označenie 2 vrcholov, medzi ktorými hrana existuje. Grupa automorfizmov lineárneho kódu  $\text{AutGroup}(C)$  v takejto forme nám umožňuje sledovať zmenu označenia hrán medzi vrcholmi, a tým pádom vieme získať obraz reprezentujúci automorfizmus. Tento obraz by sme už mali byť schopní porovnať s obrazmi, ktoré sme získali z grupy automorfizmov grafu  $\text{AutGroup}(\text{Cage})$ .

# Kapitola 5

## Riešenie

V kapitole riešenie detailne popisujeme obe riešenia z hľadiska jednotlivých tried, spustiteľných skriptov, vývojových diagramov, diagramu tried a zobrazujeme najdôležitejšie funkcie, ktoré bolo potrebné implementovať. Riešenie sa zakladá na dvoch spôsoboch využitia knižnice SageMath [19]. Prvé riešenie je implementované v online aplikácii CoCalc [18] priamo v skriptoch. Druhé riešenie je implementované ako konzolová aplikácia v programovacom jazyku Python [8], kde je knižnica SageMath [19] naimportovaná.

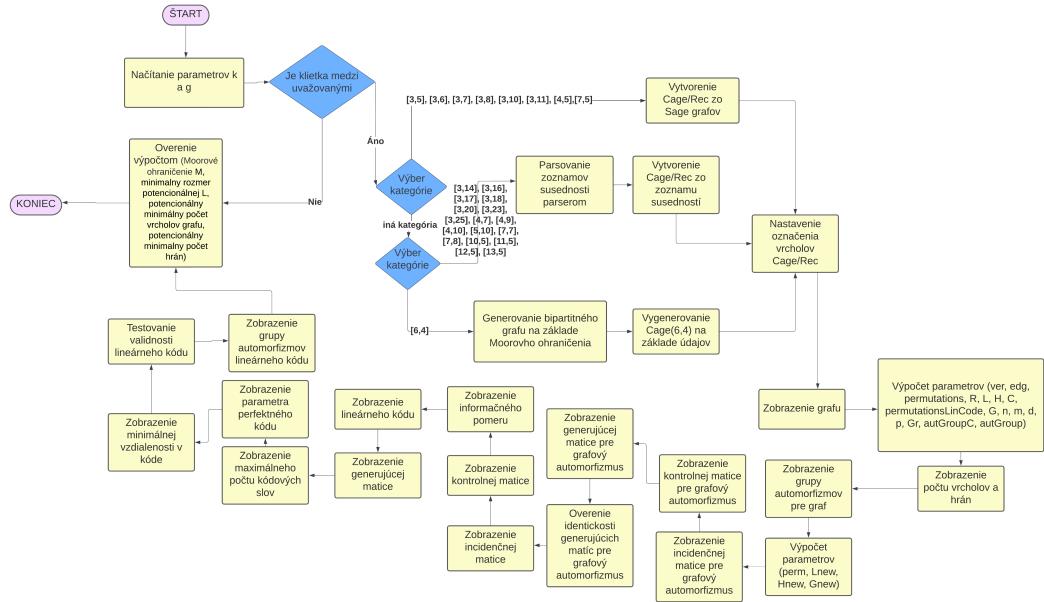
### 5.1 Prvé riešenie - CoCalc (SageMath skripty)

V prvom riešení uvažujeme sedemnásť SageMath skriptov [19], ktoré fungujú takmer samostatne nezávisle od seba. Niektoré skripty však generujú textové súbory, ktoré slúžia nielen ako výsledky, ale aj ako vstupy pre iné skripty. Tento prístup je výhodný, pretože dokážeme robiť výpočty pre všetky uvažované vstupné dátá vrámci jedného spustenia skriptu, avšak pre dosiahnutie relevantných výsledkov je potrebné zachovať poradie spúšťania niektorých skriptov. Na začiatku je potrebné spustiť skript, ktorý má nastarosti

vytvorenie textových súborov incidenčných matíc L. Tie slúžia ako vstupy pre generovanie kontrolných matíc H, ktoré sú tiež ukladané do textových súborov. Z L a H je potom možné získať detaily a parametre popisujúce samotné grafy, ich grupy automorfizmov AutGroup(Cage) a tvoria prechod ku samotným lineárnym kódom C(n,m,d). Kontrolné matice H slúžia ako vstupy pre generovanie lineárnych kódov C(n,m,d) spolu s ich generujúcimi maticami G. Generujúce matice G môžeme využiť tiež ako vstupné údaje, a preto je potrebné v nasledujúcom kroku spustiť skript, ktorý ich vygeneruje a uloží podobne do textových súborov. Z nich je potom možné získať detaily a parametre lineárnych kódov C(n,m,d), ako aj grupu automorfizmov AutGroup(C). Iný prístup k vstupným údajom berie do úvahy iba parametre grafu, a tým pádom skript generuje výsledky nie naraz, ale jednotlivo v závislosti od zmeny parametrov. Na základe vstupných a výstupných údajov skriptov vieme rozdeliť skripty na tie, ktoré spracovávajú dátu na základe parametrov a tie, ktoré spracovávajú dátu na základe vstupného textového súboru. Ďalej sú to skripty, ktoré generujú textové súbory a skripty, ktoré majú len informačný charakter, pretože nás informujú o rôznych vypočítaných parametroch alebo výsledkoch. V nasledujúcich sekciách popíšeme jednotlivo všetky skripty pomocou vývojových diagramov.

### 5.1.1 Skript generateCageAndLinearCodeByParameters

V skripte *generateCageAndLinearCodeByParameters.sagews* implementujeme všetky funkcionality okrem ukladania výsledkov do textových súborov vždy pre jednu Cage(k,g) alebo Rec(k,g). Skript spracúva a generuje dátu na základe vstupných parametrov. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.1).



Obr. 5.1: Vývojový diagram - generateCageAndLinearCodeByParameters skript

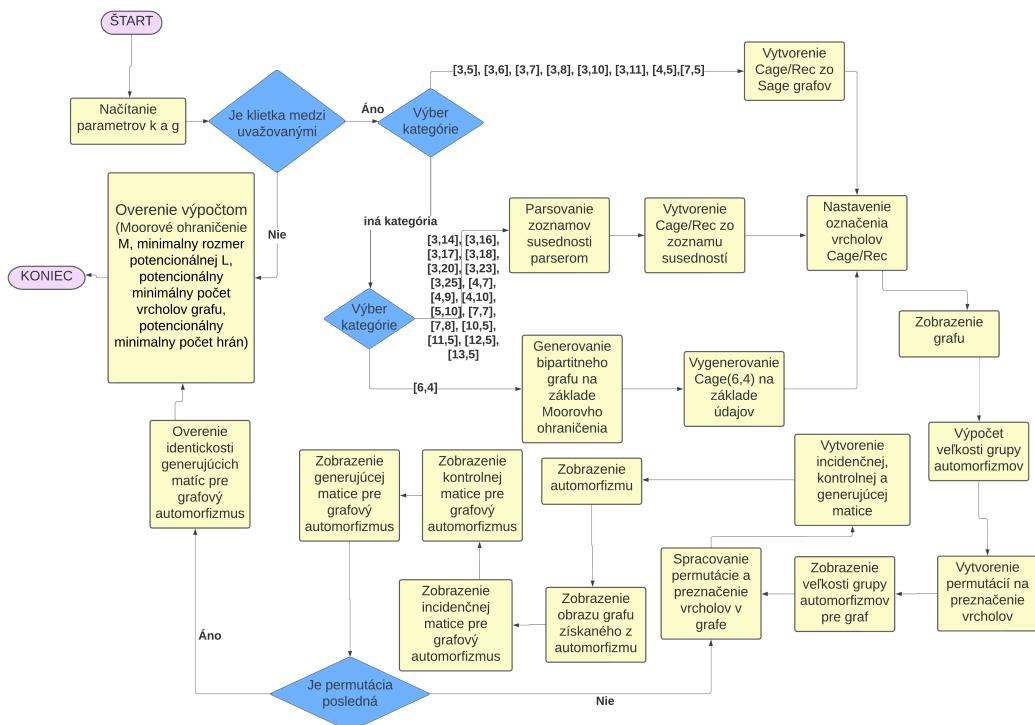
Na obrázku (Obr. 5.1) môžeme vidieť, že vstupom pre skript sú používateľom zadané parametre  $k$  a  $g$  (predpokladáme, že používateľ bude meniť obe hodnoty), ktoré skript načíta a následne sa rozhoduje, či má uvažovaný graf parametre z troch zoznamov dvojíc parametrov  $k$  a  $g$ . Ak ich nemá, dostane sa  $k$  overeniu výpočtom bez generovania grafu a skončí. Ak ich má, tak na základe zoznamu dvojíc parametrov vie skript zostrojiť graf pomocou Sage-Math grafov [19], zo zoznamu susedností [21] alebo využije generovanie na základe vlastných údajov. Získanému grafu sa nastaví označenie vrcholov z kladných čísel. Následne vygenerovaný graf zobrazíme. Postupne realizujeme výpočty na získanie počtu vrcholov, hrán, grupy automorfizmov grafu  $\text{AutGroup}(\text{Cage})$  a incidenčnej matice  $L$  z grafu. Z incidenčnej matice  $L$  zistíme kontrolnú maticu  $H$ , z ktorej vieme vypočítať informačný pomer  $R$  a získať lineárny kód  $C(n,m,d)$  spolu s jeho generujúcou maticou  $G$ . Z generujúcej

matice získame dĺžku kódu  $n$ , maximálny počet kódových slov  $m$ , minimálnu Hammingovu vzdialenosť  $d$ , parameter perfektného kódu  $p(n,m,d)$  a grupu automorfizmov lineárneho kódu  $\text{AutGroup}(C)$ . Po realizácii výpočtov skript začne postupne zobrazovať zistené údaje. Zobrazí počet vracholov  $|ver|$  a hrán  $|edg|$  grafu, grupy automorfizmov pre graf  $\text{AutGroup}(Cage)$ , následne pre každý automorfizmus zistí permutácie, incidenčnú  $L$ , kontrolnú  $H$  a generujúcu maticu  $G$ . Jednotlivé matice grafového automorfizmu skript takisto zobrazí. Skript overí, či sú všetky generujúce matice pre grafový automorfizmus identické. Skript postupne zobrazí aj incidenčnú  $L$  a kontrolnú maticu  $H$  pre pôvodný graf. Takisto sa zobrazí aj informačný pomer  $R$ , lineárny kód  $C(n,m,d)$ , generujúca matica a maximálny počet slov lin. kódu  $m$ , parameter perfektného lin. kódu  $p(n,m,d)$ , minimálnej Hammingovej vzdialnosti  $d$ . Skript otestuje, či je lineárny kód  $C(n,m,d)$  validný, zobrazí grupu automorfizmov lineárneho kódu  $\text{AutGroup}(C)$  a ako posledný krok overí výpočtom pomocou Moorového ohraničenia  $M(k,g)$  minimálny rozmer potencinálnej incidenčnej matice  $L$ , potencionálny obvod cyklu v incidenčnej matici  $L$ , ako aj potencionálny počet vracholov  $|ver|$  a hrán  $|edg|$ . Následne je skript ukončený. Skript *textgenerateCageAndLinearCodeByParameters.sagews* obsahuje takmer všetku funkcionality prvého riešenia, čo nám umožňuje získať informácie pre uvažovaný graf naraz, avšak pre zložitejšie grafy je menej a menej účinný a postupne získavame menej a menej výsledkov. Aby sme získali viac výsledkov aj zo zložitejších štruktúr, vytvorili sme viacero skriptov, ktoré sa zameriavajú na určitú časť funkcionality.

### 5.1.2 Skript autCodesFromCages

V skripte *autCodesFromCages.sagews* zobrazujeme automorfizmy, obrazy grafu, kontrolné  $H$  a generujúce matice  $G$  z automorfizmov. Skript spracúva a ge-

neruje dátu na základe vstupných parametrov. Vo vývojovom diagrame pre nás skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.2).



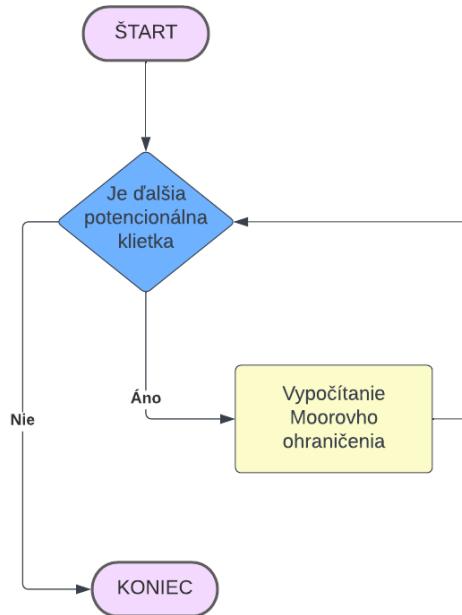
Obr. 5.2: Vývojový diagram - autCodesFromCages skript

Na obrázku (Obr. 5.2) môžeme vidieť, že vstupom pre skript je zoznam dvojíc parametrov k a g (užívateľ môže zoznam upraviť a pridať dvojice navyše), ktoré skript načíta a následne sa rozhoduje, či má uvažovaný graf parametre z troch zoznamov dvojíc parametrov k a g. Ak ich nemá, dostane sa k overeniu výpočtom bez generovania grafu a skončí. Ak ich má, tak na základe zoznamu dvojíc parametrov vie skript zostrojiť graf pomocou SageMath grafov [19], zo zoznamu susedností [21] alebo využije generovanie na základe vlastných údajov. Získanému grafu sa nastaví označenie vrcholov z kladných

čísel. Následne vygenerovaný graf zobrazíme. Vypočíta veľkosť grupy automorfizmov grafu  $\text{AutGroup}(\text{Cage})$  a vytvoria sa permutácie na preznačenie vrcholov. Permutácie sa spracujú a vrcholy v grafe sú preznačené kladnými číslami. Pre každý automorfizmus sa vytvorí incidenčná  $L$ , kontrolná  $H$  a generujúca matica  $G$ . Jednotlivé automorfizmy sa zobrazia. Zobrazí sa aj obraz grafu získaného z automorfizmu ako graf s novým usporiadaním vrcholov. Skript zobrazí všetky vytvorené matice a rozhoduje sa na základe toho, či je na konci vstupného zoznamu dvojíc. Ak nie je, vráti sa k spracovaniu permutácie a preznačeniu vrcholov v grafe. Ak je, tak už spraví iba overenie, či sú všetky generujúce matice grafového automorfizmu identické, overí sa výpočtom správnosť získaných údajov z Moorového ohraničenia  $M(k,g)$  a skript skončí. Skript *autCodesFromCages.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) jednotlivé automorfizmy grafu, im prislúchajúce obrazy a takisto incidenčné  $L$ , kontrolné  $H$  a generujúce matice  $G$  prislúchajúce jednotlivým automorfizmom.

### 5.1.3 Skript `mooreBoundsCageValidation`

V skripte *mooreBoundsCageValidation.sagews* zobrazujeme Moorove ohraničenia  $M(k,g)$  vypočítané na základe vstupných parametrov. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.3).

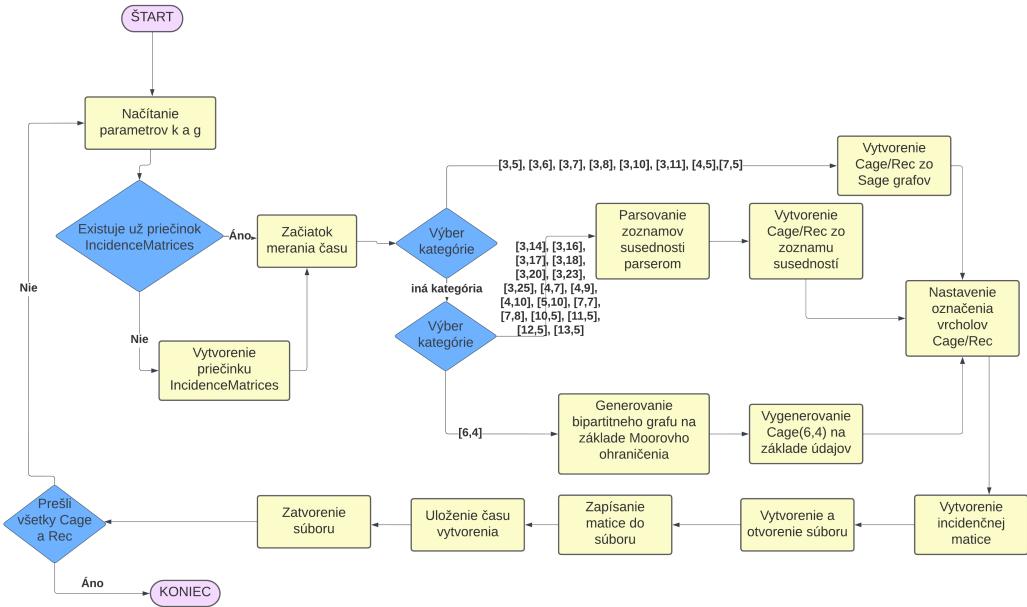


Obr. 5.3: Vývojový diagram - *mooreBoundsCageValidation* skript

Na obrázku (Obr. 5.3) môžeme vidieť proces výpočtu Moorového ohraničenia  $M(k,g)$  pre všetky uvažované klietky a rekordné grafy. Používateľ nezadáva ani neupravuje na vstupe nič. Skript *mooreBoundsCageValidation.sagews* nám umožňuje získať Moorové ohraničenie  $M(k,g)$  pre všetky uvažované klietky a rekordné grafy.

#### 5.1.4 Skript generateIncidenceMatrices

V skripte *generateIncidenceMatrices.sagews* generujeme incidenčné matice a ukladáme ich do textových súborov. Skript spracúva na vstupe zoznamy parametrov všetkých uvažovaných grafov. Vo vývojovom diagrame pre nás skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.4).

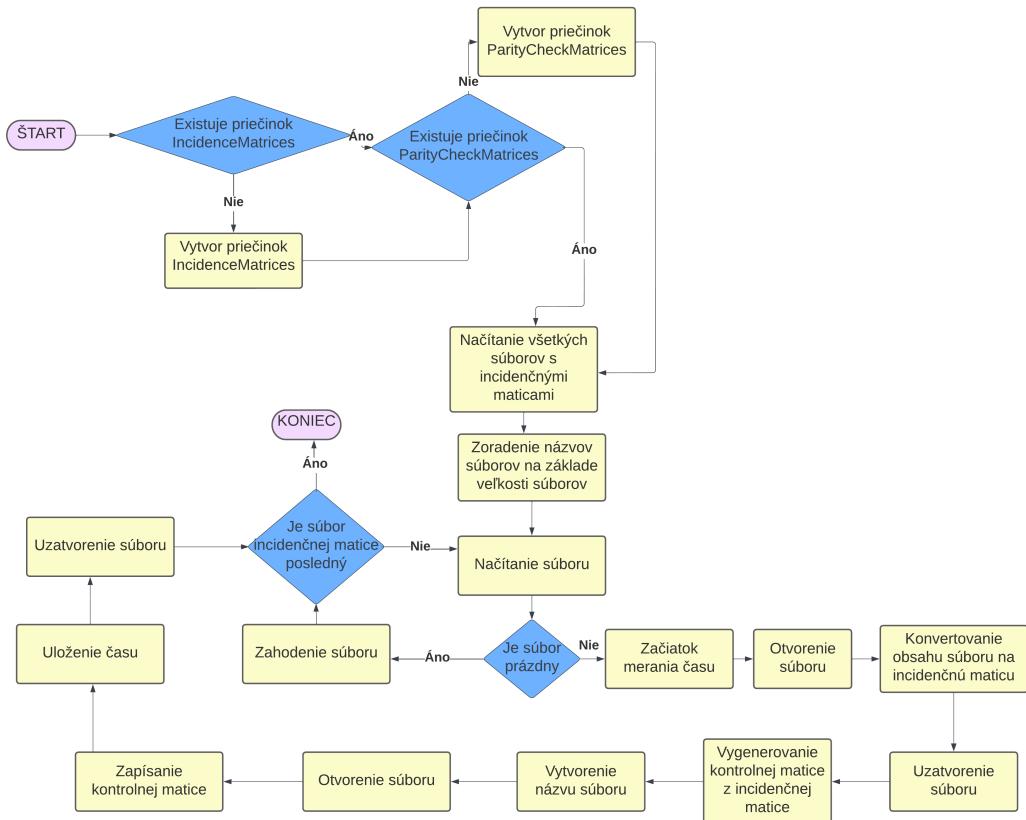
Obr. 5.4: Vývojový diagram - `generateIncidenceMatrices` skript

Na obrázku (Obr. 5.4) môžeme vidieť, že vstupom pre skript je zoznam dvojíc parametrov `k` a `g` (užívateľ na vstupe nič neupravuje), ktoré skript načíta. Skript následne musí overiť, či už existuje priečinok `IncidenceMatrices`, kde sa budú ukladať vygenerované incidenčné matice `L`. Ak taký priečinok ešte neexistuje, tak ho vytvorí a pokračuje na zapnutie časomieri. V opačnom prípade hned' pokračuje na zapnutie časomieri. V ďalšom kroku sa skript rozhoduje, z ktorého zoznamu dvojíc parametrov má uvažované parametre `k` a `g`. Na základe zoznamu dvojíc parametrov vie skript zostrojiť graf pomocou SageMath grafov [19], zo zoznamu susedností [21] alebo využije generovanie na základe vlastných údajov. Získanému grafu sa nastaví označenie vrcholov z kladných čísel. Z grafu vytvorí skript incidenčnú maticu `L`. Vytvorí a zapíše maticu do nového textového súboru, ktorý bude umiestnený v priečinku `IncidenceMatrices`. Po uložení skript zistí čas potrebný na vygenerovanie a uloženie matice a zatvorí súbor. Na záver skript sleduje, či prešli všetky dvojice.

jice parametrov s uvažovaného zoznamu. Ak neprešli, proces sa opakuje od načítania nových parametrov a nakoniec skript skončí. Skript *generateIncidenceMatrices.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) incidenčné matice L uvažovaných grafov a uloží ich do priečinku *IncidenceMatrices*.

### 5.1.5 Skript *generateParityCheckMatrices*

V skripte *generateParityCheckMatrices.sagews* generujeme kontrolné matice H a ukladáme ich do textových súborov. Skript spracúva na vstupe zoznamy parametrov všetkých uvažovaných grafov. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.5).



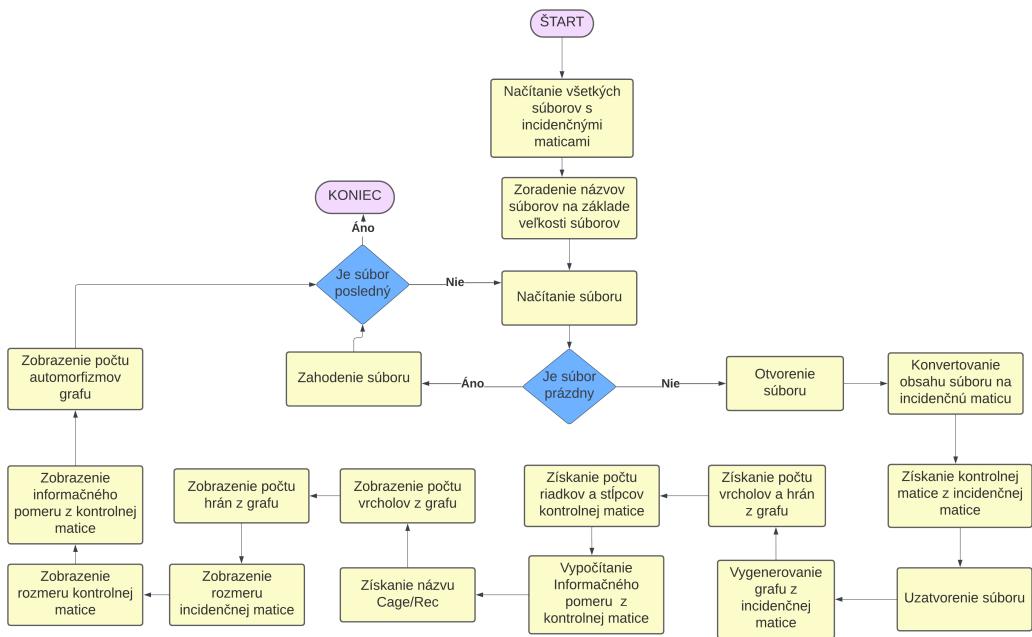
Obr. 5.5: Vývojový diagram - generateParityCheckMatrices skript

Na obrázku (Obr. 5.5) môžeme vidieť, že skript skúma existenciu priečinkov IncidenceMatrices a ParityCheckMatrices. Ak by priečinky chýbali, vytvorí ich. Následne skript načíta všetky súbory s incidenčnými maticami L a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s incidenčnou maticou L. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku IncidenceMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, spúšťa sa časomiera, otvorí sa súbor s incidenčnou maticou L, konvertuje sa obsah na incidenčnú maticu L a súbor sa uzavrie. Následne sa z incidenčnej matice L vygeneruje kontrolná matica H a vygeneruje sa aj nový názov súboru, do ktorého bude kontrolná matica H

uložená. Nový súbor sa otvorí, zapíše sa doňho kontrolná matica  $H$ , zastaví sa meranie času. Textový súbor s novou kontrolnou maticou  $H$  sa uzavrie a skript zistuje, či ešte má nejaké textové súbory s incidenčnými maticami  $L$  na vstupe alebo nie. Ak ešte má textový súbor na vstupe, tak pokračuje od jeho načítania, v opačnom prípade skript skončí. Skript *generateParityCheckMatrices.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) kontrolné matice z textových súborov incidenčných matíc  $L$ , vygenerovať z nich kontrolné matice  $H$  a uložiť ich do textových súborov v priečinku *ParityCheckMatrices*.

### 5.1.6 Skript *cagesData*

V skripte *cagesData.sagews* získavame informácie o uvažovaných grafoch ako sú počet hrán  $|edg|$  a vrcholov  $|ver|$ , názov grafu, informačný pomer a ďalšie. Skript spracúva na vstupe textové súbory kontrolných matíc  $H$ . Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.6).



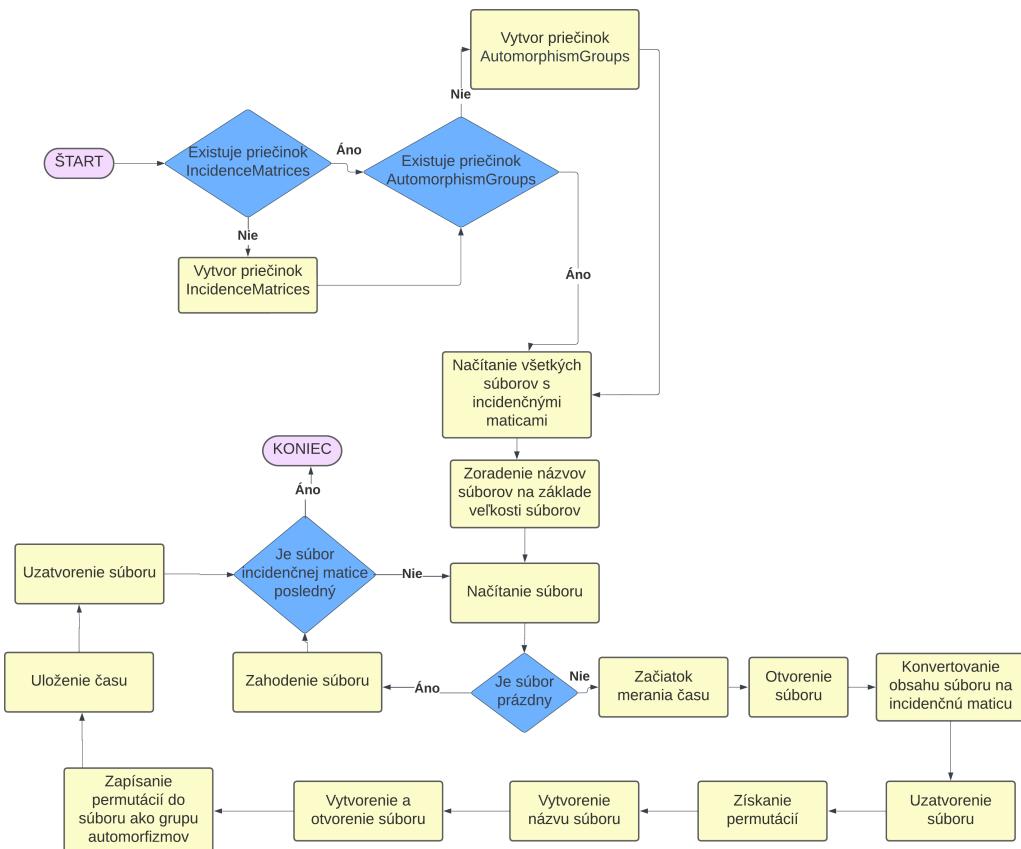
Obr. 5.6: Vývojový diagram - cagesData skript

Na obrázku (Obr. 5.6) môžeme vidieť, že skript najskôr načíta všetky súbory s incidenčnými maticami a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s incidenčnou maticou. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdny, zahodí sa, a ak sa nejedná o posledný súbor v priečinku IncidenceMatrices, zavolá sa ďalší. Ak súbor nie je prázdny, otvorí sa súbor s incidenčnou maticou, konvertuje sa obsah na incidenčnú maticu, z nej sa získa kontrolná matica H a súbor sa uzavrie. Z incidenčnej matice sa vygeneruje graf, z ktorého získame informáciu o počte vrcholov a hrán. Z kontrolnej matice H získame počet riadkov  $|Hrows|$  a informačný pomer R. Následne skript získava z názvu textového súboru incidenčnej matice názov klietky alebo rekordného grafu. Zobrazia sa informácie o počte vrcholov, počte hrán, rozmere incidenčnej matice, rozmere kontrolnej matice H, informačného pomeru R a zobrazí sa počet automorfizmov z grafu  $|\text{AutGroup}(\text{Cage})|$ .

Ak je súbor incidenčnej matice posledný, skript skončí. V opačnom prípade sa zavolá ďalší súbor s incidenčnou maticou. Skript *cagesData.sagews* nám umožňuje načítať naraz (pokiaľ je to výpočtovou náročnosťou možné) incidenčné matice z textových súborov a získať z nich informácie o grafoch a kontrolných maticiach H.

### 5.1.7 Skript generateAutCodesFromCages

V skripte *generateAutCodesFromCages.sagews* generujeme grupy automorfizmov grafov AutGroup(Cage) a ukladáme ich do textových súborov. Skript spracúva na vstupe zoznamy parametrov všetkých uvažovaných grafov. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.7).

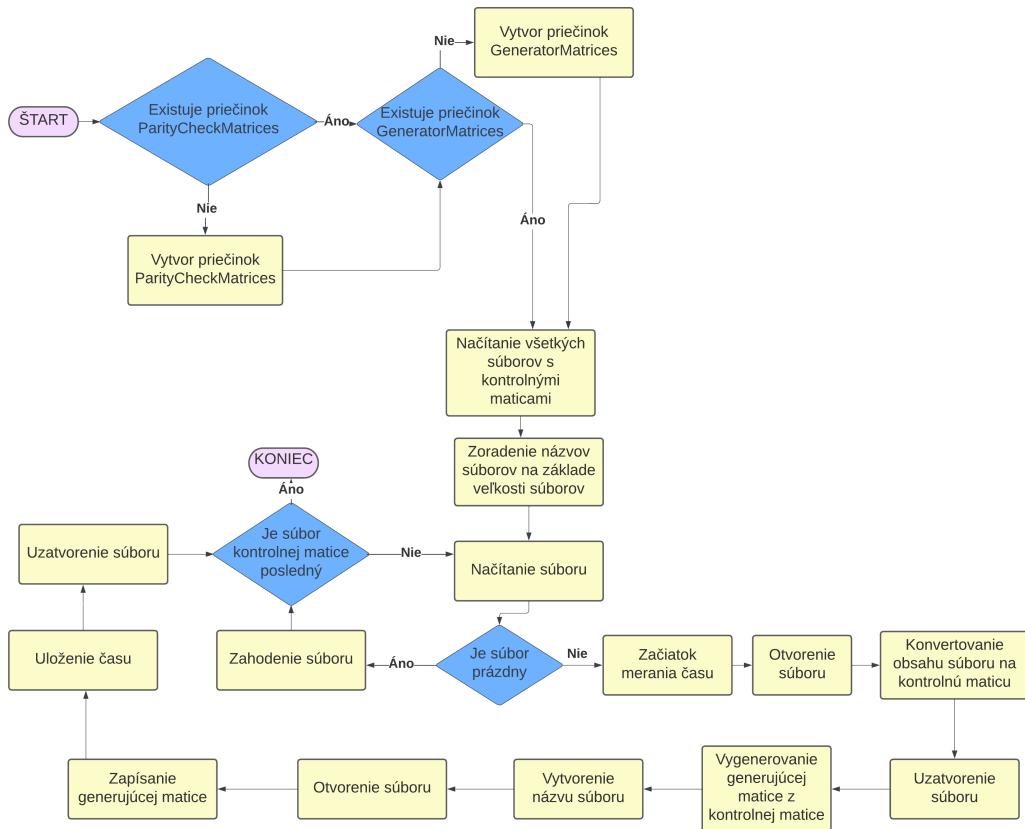
Obr. 5.7: Vývojový diagram - `generateAutCodesFromCages` skript

Na obrázku (Obr. 5.7) môžeme vidieť, že skript skúma existenciu priečinkov `IncidenceMatrices` a `AutomorphismGroups`. Ak by priečinky chýbali, vytvorí ich. Následne skript načíta všetky súbory s incidenčnými maticami L a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s incidenčnou maticou L. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku `IncidenceMatrices`, zavolá sa ďalší. Ak súbor nie je prázdný, spúšťa sa časomiera, otvorí sa súbor s incidenčnou maticou L, konvertuje sa obsah na incidenčnú maticu L a súbor sa uzavrie. Skript následne získá permutácie potrebné na určenie automorfiz-

mov. Z textového súboru incidenčnej matice  $L$  sa vytvorí názov súboru pre grupu automorfizmov  $\text{AutGroup}(\text{Cage})$ . Následne sa vytvorí a otvorí nový súbor, zapíšu sa doňho všetky permutácie a vytvorí sa grupa automorfizmov  $\text{AutGroup}(\text{Cage})$ . Zastaví sa meranie času a súbor sa uzavrie. Ak je textový súbor incidenčnej matice  $L$  posledný, skript skončí. V opačnom prípade sa zavolá ďalší súbor s incidenčnou maticou  $L$ . Skript *generateAutCodesFromCages.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) grupy automorfizmov grafov  $\text{AutGroup}(\text{Cage})$  z textových súborov incidenčných matíc  $L$  a uložiť ich do priečinku `AutomorphismGroups`.

### 5.1.8 Skript `generateGeneratorMatrices`

V skripte *generateGeneratorMatrices.sagews* generujeme generujúce matice a ukladáme ich do textových súborov. Skript spracúva na vstupe textové súbory kontrolných matíc. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.8).



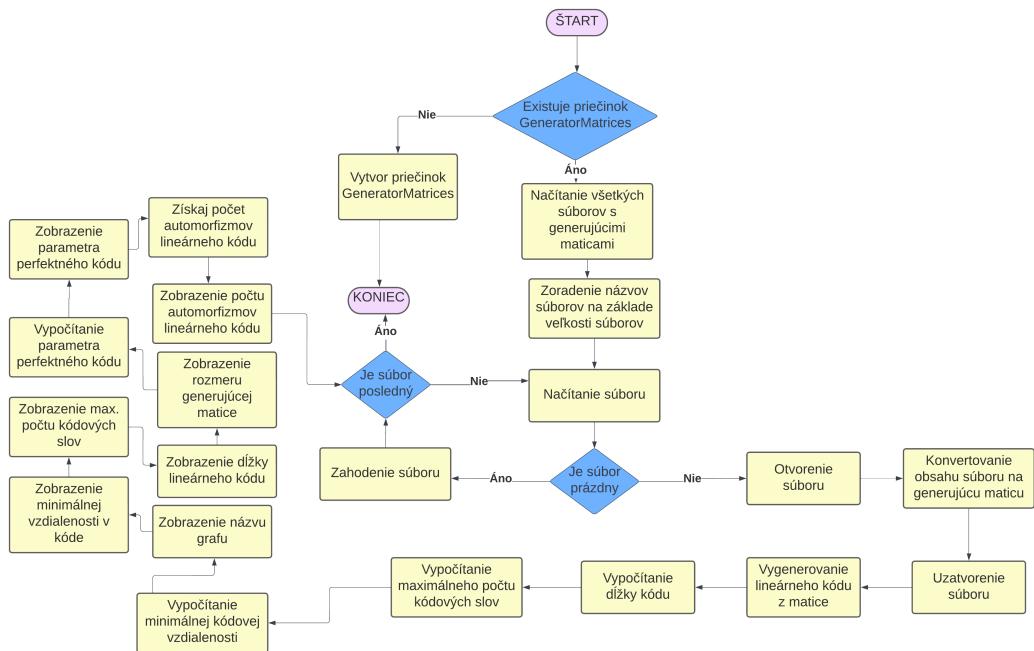
Obr. 5.8: Vývojový diagram - generateGeneratorMatrices skript

Na obrázku (Obr. 5.8) môžeme vidieť, že skript skúma existenciu priečinkov ParityCheckMatrices a GeneratorMatrices. Ak by priečinky chýbali, vytvorí ich. Následne skript načíta všetky súbory s kontrolnými maticami H a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s kontrolnou maticou H. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku ParityCheckMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, spúšťa sa časomiera, otvorí sa súbor s kontrolnou maticou H, konvertuje sa obsah na kontrolnú maticu H a súbor sa uzavrie. Následne sa z kontrolnej matice H vygeneruje generujúca matica G a vygeneruje sa aj nový názov súboru, do ktorého bude generujúca matica

G uložená. Nový súbor sa otvorí, zapíše sa doňho generujúca matica, zastaví sa meranie času. Textový súbor s novou generujúcou maticou sa uzavrie a skript zistíuje, či ešte má nejaké textové súbory s kontrolnými maticami H na vstupe alebo nie. Ak ešte má textový súbor na vstupe, tak pokračuje od jeho načítania, v opačnom prípade skript skončí. Skript *generateGeneratorMatrices.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) generujúce matice z textových súborov kontrolných matíc H a uloží ich do priečinku GeneratorMatrices.

### 5.1.9 Skript linearCodesData

V skripte *linearCodesData.sagews* generujeme dátá lineárnych kódov  $C(n,m,d)$ , ako sú dĺžka lin. kódu n, maximálny počet kódových slov m, minimálna Hammingova vzdialenosť v kóde d, názov zdrojového grafu, rozmer generujúcej matice, parameter perfektného kódu  $p(n,m,d)$  a počet automorfizmov z lin. kódu  $|AutGroup(C)|$ . Skript spracúva na vstupe textové súbory generujúcich matíc. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.9).



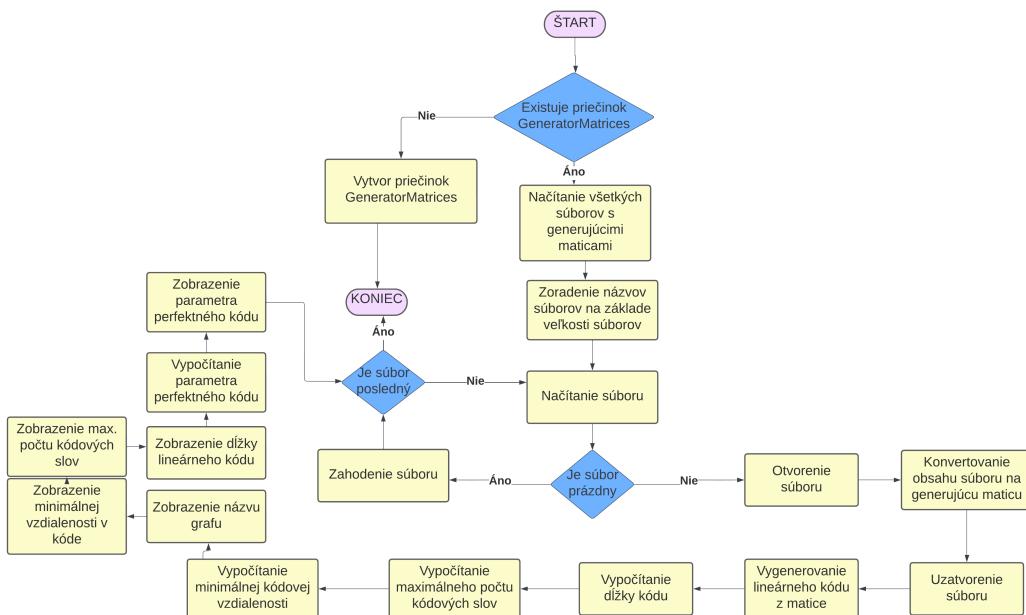
Obr. 5.9: Vývojový diagram - linearCodesData skript

Na obrázku (Obr. 5.9) môžeme vidieť, že skript skúma existenciu priečinku GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, otvorí sa súbor s generujúcou matricou, konvertuje sa obsah na generujúcu maticu a súbor sa uzavrie. Z generujúcej matice sa získa lineárny kód  $C(n,m,d)$ . Skript následne vypočíta dĺžku kódu  $n$ , maximálny počet kódových slov  $m$  a minimálnej Hammingovej vzdialenosťi  $d$ . Následne sa zobrazí názov grafu a postupne sa zobrazujú dôkladne lineárneho kódu  $C(n,m,d)$ . Zobrazí sa minimálna Hammingova vzdialenosť v kóde  $d$ , maximálny počet kódových slov  $m$ , dĺžka lineárneho kódu  $n$  a roz-

mer generujúcej matice. Vypočíta a zobrazí sa parameter perfektného kódu  $p(n,m,d)$ , získa sa a zobrazí počet automorfizmov lineárneho kódu  $|\text{AutGroup}(\text{Cage})|$ . Ak je súbor s generujúcou maticou posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou. Skript *linearCodesData.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) údaje z textových súborov generujúcich matíc.

### 5.1.10 Skript getLinearCodesParameter

V skripte *getLinearCodesParameter.sagews* zobrazujeme parametre lineárnych kódov  $C(n,m,d)$ . Skript spracúva na vstupe textové súbory generujúcich matíc G. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.10).



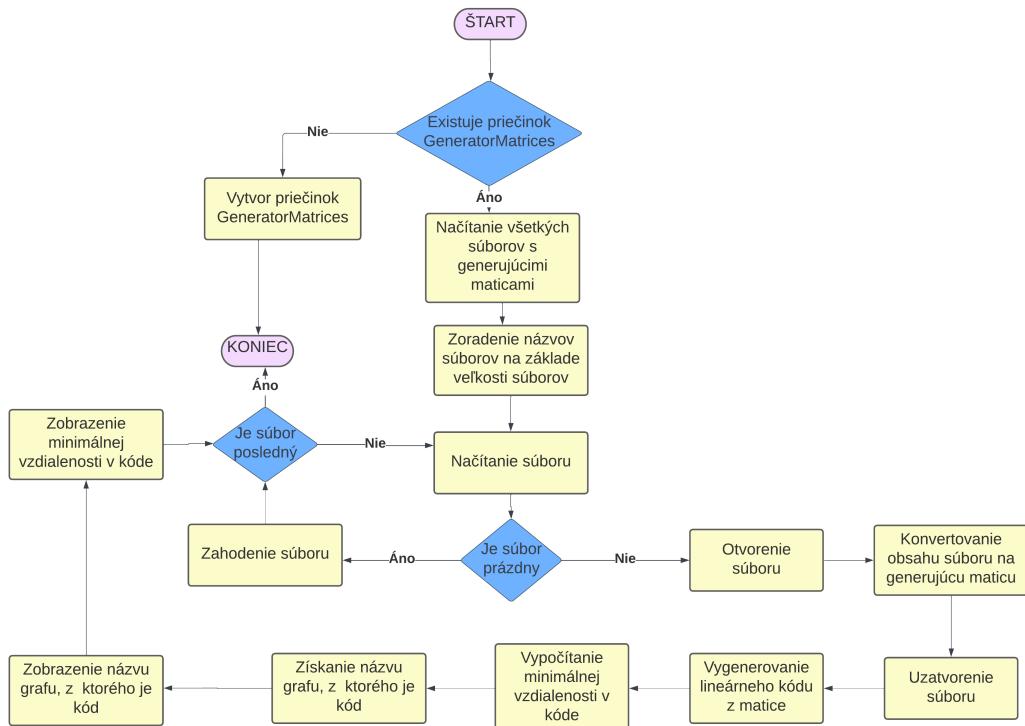
Obr. 5.10: Vývojový diagram - *getLinearCodesParameter* skript

Na obrázku (Obr. 5.10) môžeme vidieť, že skript skúma existenciu priečinku

GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdny, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdny, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu maticu G a súbor sa uzavrie. Z generujúcej matice G sa získa lineárny kód C(n,m,d). Skript následne vypočíta dĺžku kódu n, maximálny počet kódových slov m a minimálnu Hammingovu vzdialenosť d. Následne sa zobrazí názov grafu a postupne sa zobrazujú detailly lineárneho kódu C(n,m,d). Zobrazí sa minimálna Hammingova vzdialenosť v kóde d, maximálny počet kódových slov m a dĺžka lineárneho kódu n. Vypočíta a zobrazí sa parameter perfektného kódu p(n,m,d). Ak je súbor s generujúcou maticou G posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou G. Skript *linearCodesData.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) parametre perfektného kódu spolu s parametrami lineárneho kódu potrebnými na jeho dosiahnutie z textových súborov generujúcich matíc G.

### 5.1.11 Skript minDistanceLinearCodes

V skripte *minDistanceLinearCodes.sagews* zobrazujeme minimálne Hammingove vzdialnosti d. Skript spracúva na vstupe textové súbory generujúcich matíc. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.11).



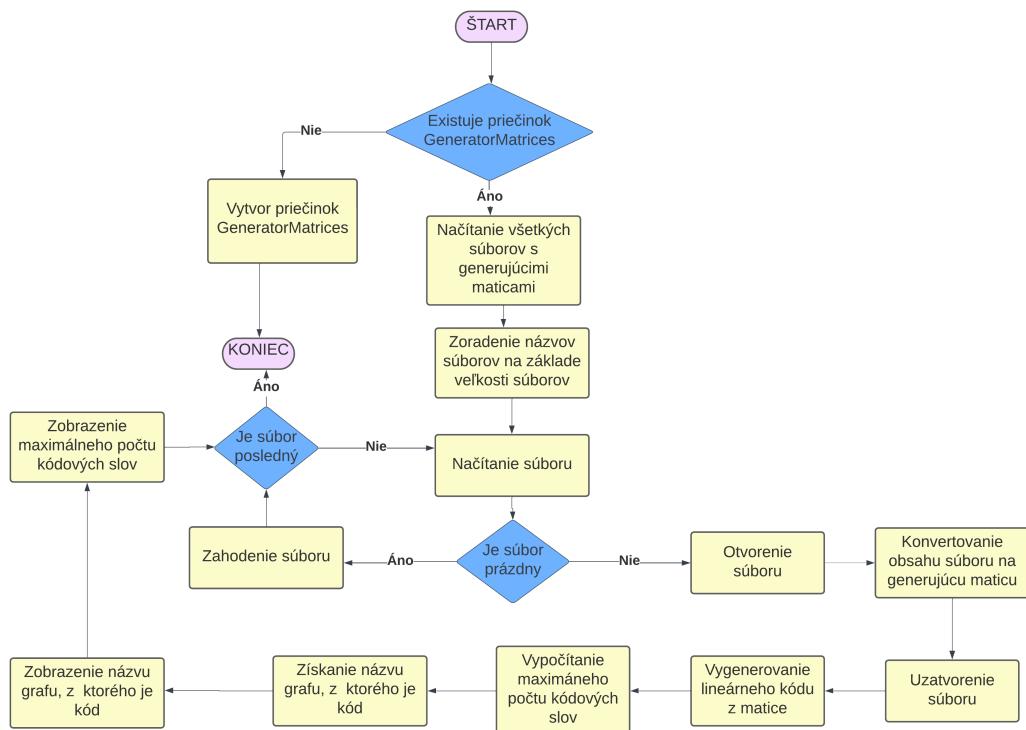
Obr. 5.11: Vývojový diagram - minDistanceLinearCodes skript

Na obrázku (Obr. 5.11) môžeme vidieť, že skript skúma existenciu priečinku GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu maticu G a súbor sa uzavrie. Z generujúcej matice G sa získa lineárny kód  $C(n,m,d)$ . Skript následne vypočíta minimálnu Hammingovu vzdialenosť d. Následne sa získa názov grafu, zobrazí názov grafu a zobrazí sa aj minimálna Hamminova vzdialenosť v kóde.

d. Ak je súbor s generujúcou maticou G posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou G. Skript *minDistanceLinearCodes.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovo náročnosťou možné) minimálne Hammingove vzdialenosťi d z textových súborov generujúcich matíc G.

### 5.1.12 Skript maxWordsLinearCodes

V skripte *maxWordsLinearCodes.sagews* zobrazujeme maximálne počty kódových slov m. Skript spracúva na vstupe textové súbory generujúcich matíc G. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.12).

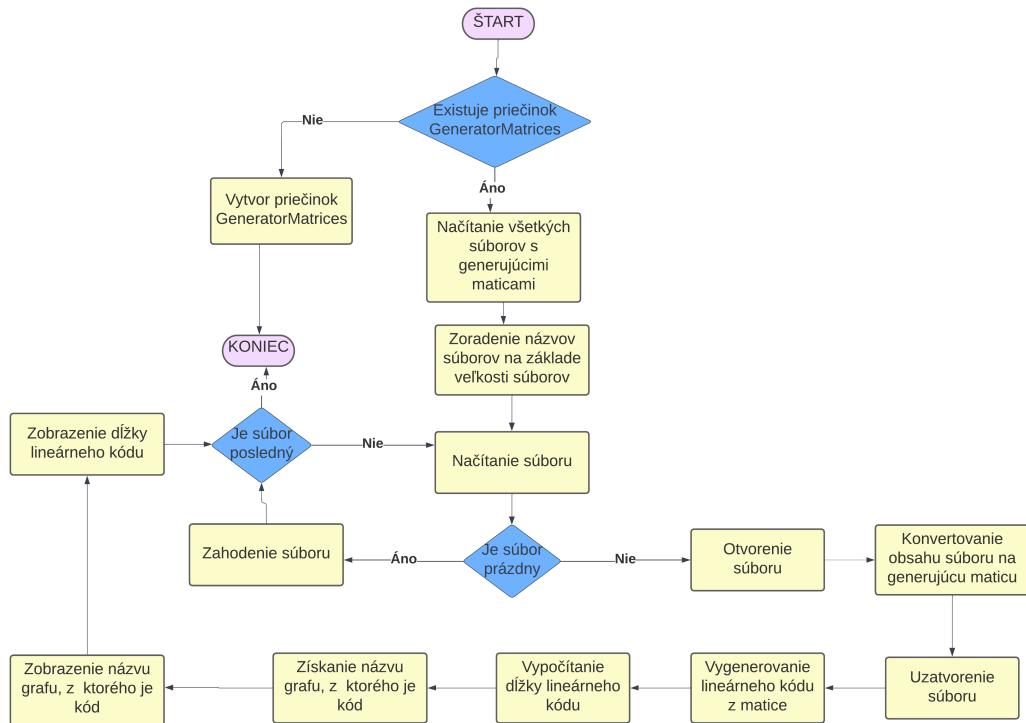


Obr. 5.12: Vývojový diagram - maxWordsLinearCodes skript

Na obrázku (Obr. 5.12) môžeme vidieť, že skript skúma existenciu priečinku GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu maticu G a súbor sa uzavrie. Z generujúcej matice G sa získa lineárny kód. Skript následne vypočíta maximálny počet kódových slov m. Následne sa získa názov grafu, zobrazí názov grafu a zobrazí sa aj maximálny počet kódových slov m. Ak je súbor s generujúcou maticou posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou G. Skript *maxWordsLinearCodes.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) maximálne počty kódových slov m z textových súborov generujúcich matíc G.

### 5.1.13 Skript sizeNLinearCodesData

V skripte *sizeNLinearCodesData.sagews* zobrazujeme dĺžky lineárnych kódov n. Skript spracúva na vstupe textové súbory generujúcich matíc G. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.13).



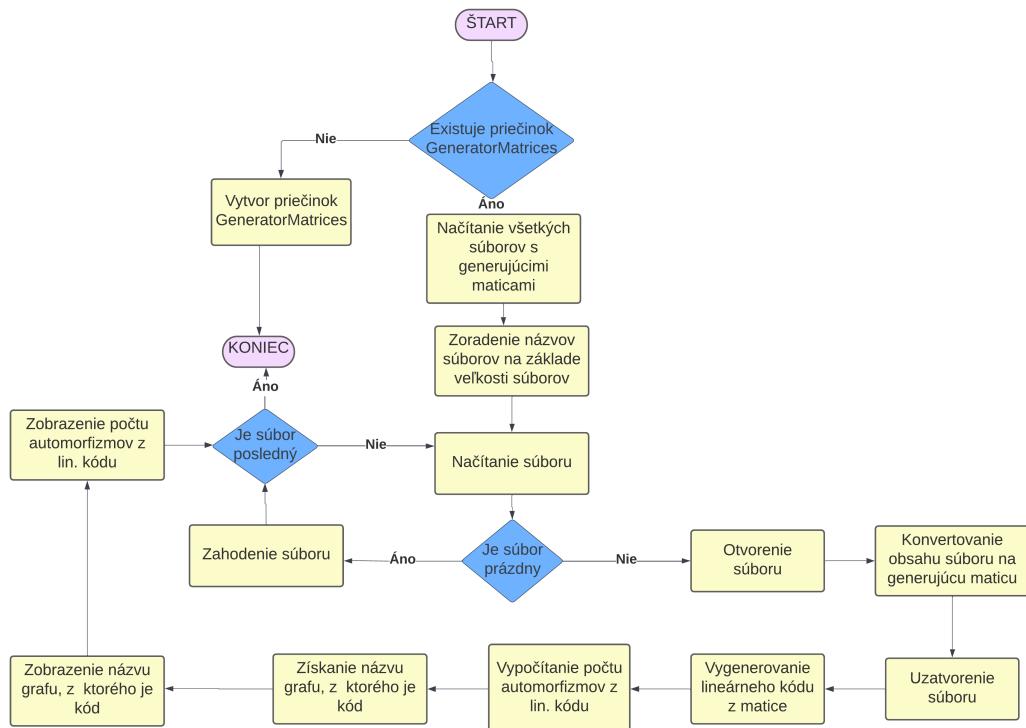
Obr. 5.13: Vývojový diagram - sizeNLinearCodesData skript

Na obrázku (Obr. 5.13) môžeme vidieť, že skript skúma existenciu priečinka GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu maticu G a súbor sa uzavrie. Z generujúcej matice G sa získa lineárny kód C(n,m,d). Skript následne vypočíta dĺžku lineárneho kódu n. Následne sa získava názov grafu, zobrazí sa názov grafu a dĺžka lineárneho kódu n. Ak je súbor s generujúcou

maticou  $G$  posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou  $G$ . Skript *sizeNLinearCodesData.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) dĺžky lineárnych kódov  $n$  z textových súborov generujúcich matíc  $G$ .

### 5.1.14 Skript autLinearCodesData

V skripte *autLinearCodesData.sagews* zobrazujeme veľkosť grupy automorfizmov lineárnych kódov  $|\text{AutGroup}(C)|$ . Skript spracúva na vstupe textové súbory generujúcich matíc. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.14).

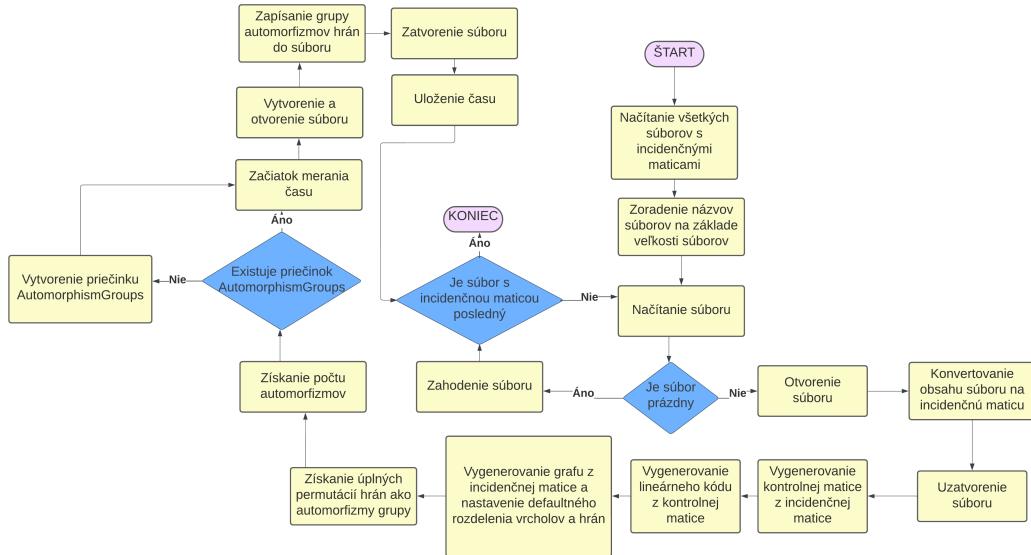


Obr. 5.14: Vývojový diagram - *autLinearCodesData* skript

Na obrázku (Obr. 5.14) môžeme vidieť, že skript skúma existenciu priečinku GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdný, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdný, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu matice G a súbor sa uzavrie. Z generujúcej matice G sa získa lineárny kód  $C(n,m,d)$ . Skript následne vypočíta počet automorfizmov z lineárneho kódu  $|\text{AutGroup}(C)|$ . Následne sa získa názov grafu, zobrazí názov grafu a zobrazí sa aj počet automorfizmov lineárneho kódu  $|\text{AutGroup}(C)|$ . Ak je súbor s generujúcou maticou G posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou G. Skript *autLinearCodesData.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) veľkosť grúp automorfizmov lineárnych kódov z textových súborov generujúcich matíc G.

### 5.1.15 Skript generateAutGroupLCode

V skripte *generateAutGroupLCode.sagews* ukladáme grupy automorfizmov lineárnych kódov  $\text{AutGroup}(C)$ . Skript spracúva na vstupe textové súbory kontrolných matíc H. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.15).



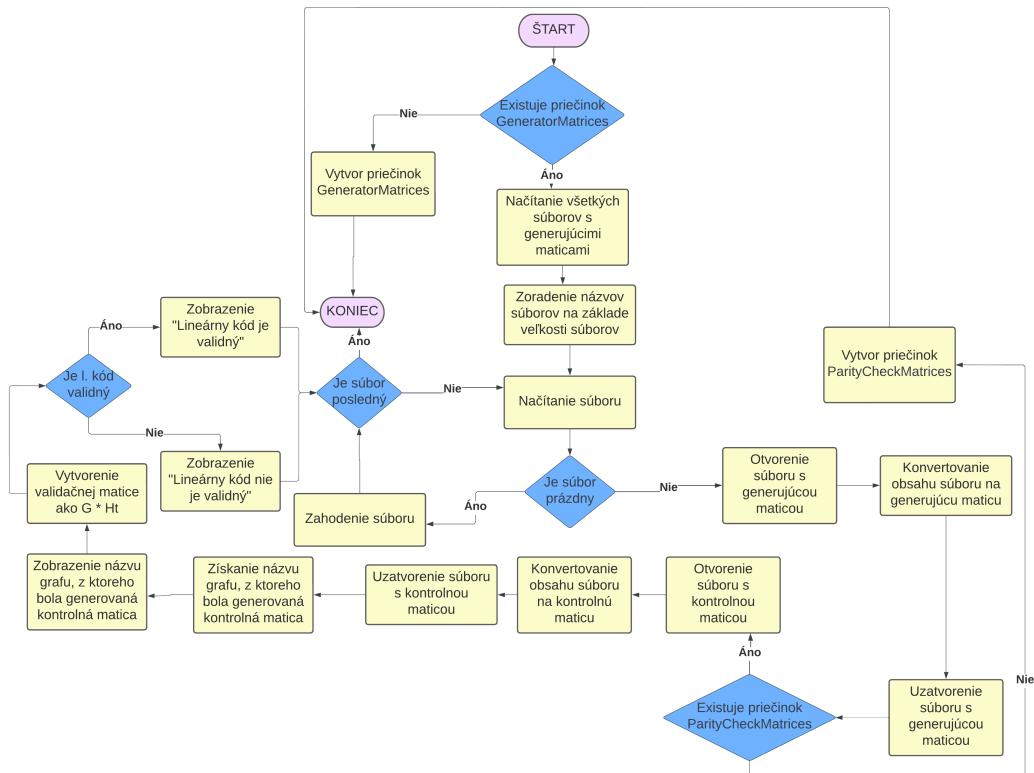
Obr. 5.15: Vývojový diagram - generateAutGroupLCode skript

Na obrázku (Obr. 5.15) môžeme vidieť, že skript najskôr načíta všetky súbory s incidenčnými maticami a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s incidenčnou maticou. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdny, zahodí sa, a ak sa nejedná o posledný súbor v priečinku IncidenceMatrices, zavolá sa ďalší. Ak súbor nie je prázdny, otvorí sa súbor s incidenčnou maticou, konvertuje sa obsah na incidenčnú maticu, z nej sa získa kontrolná matica  $H$  a súbor sa uzavrie. Z incidenčnej matice sa vygeneruje kontrolná matica  $H$  a z nej sa vygeneruje lineárny kód  $C(n,m,d)$ . Následne skript vygeneruje graf z incidenčnej matice a nastaví počiatočné rozdelenie vrcholov a hrán. V ďalších krokoch získame úplné permutácie hrán ako automorfizmy grupy  $\text{AutGroup}(C)$  a počet automorfizmov lineárneho kódu  $|\text{AutGroup}(C)|$ . Skript si potrebuje overiť, či existuje priečinok  $\text{AutomorphismGroups}$  a ak neexistuje tak ho vytvorí a pokračuje ďalej spustením časomieri. Vytvorí sa nový textový súbor, do ktorého sa zapíše

grupa automorfizmov hrán AutGroup(C). Následne sa súbor zavrie a mera-  
nie času sa ukončí. Ak je súbor s incidenčnou maticou posledný, skript skončí.  
V opačnom prípade načíta ďalší textový súbor s incidenčnou maticou. Skript  
*generateAutGroupLCode.sagews* nám umožňuje získať a uložiť naraz (pokiaľ  
je to výpočtovou náročnosťou možné) grupy automorfizmov lineárnych kódov  
AutGroup(C) z textových súborov incidenčných matíc.

### 5.1.16 Skript linearCodeValidation

V skripte *linearCodeValidation.sagews* testujeme, či sú lineárne kódy C(n,m,d)  
validné pomocou ich uložených generujúcich matíc G a kontrolných matíc H.  
Skript spracúva na vstupe textové súbory generujúcich G a kontrolných ma-  
tíc H. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné  
a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.16).



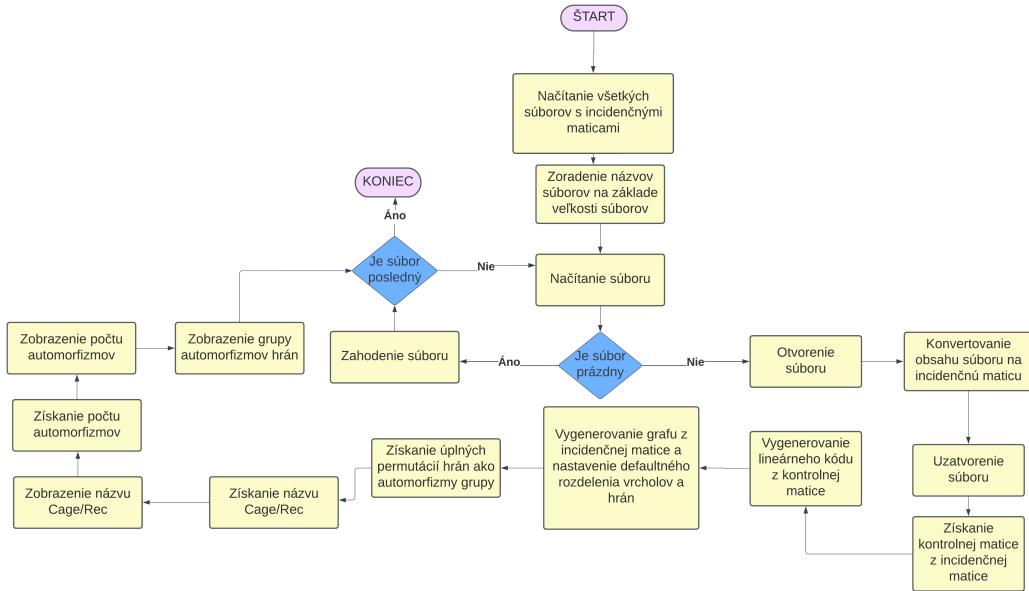
Obr. 5.16: Vývojový diagram - linearCodeValidation skript

Na obrázku (Obr. 5.16) môžeme vidieť, že skript skúma existenciu priečinku GeneratorMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade skript načíta všetky súbory s generujúcimi maticami G a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s generujúcou maticou G. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdny, zahodí sa, a ak sa nejedná o posledný súbor v priečinku GeneratorMatrices, zavolá sa ďalší. Ak súbor nie je prázdny, otvorí sa súbor s generujúcou maticou G, konvertuje sa obsah na generujúcu maticu G a súbor sa uzavrie. Skript následne skúma existenciu priečinku ParityCheckMatrices. Ak by priečinok chýbal, vytvorí ho a skript skončí. V opačnom prípade sa otvorí súbor s kontrolnou maticou H, konvertuje sa obsah na kontrolnú

maticu H a súbor sa uzavrie. Z textového súboru generujúcej matice G sa získa a zobrazí názov grafu. Skript vygeneruje validačnú maticu. Bez ohľadu na to, či je alebo nie je lineárny kód  $C(n,m,d)$  validny, skript zobrazí túto informáciu. Ak je súbor s generujúcou maticou G posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s generujúcou maticou G. Skript *linearCodeValidation.sagews* nám umožňuje získať naraz (pokiaľ je to výpočtovou náročnosťou možné) validácie lineárnych kódov  $C(n,m,d)$  z textových súborov kontrolných H a generujúcich matíc G.

### 5.1.17 Skript autGroupCode

V skripte *autGroupCode.sagews* zobrazujeme grupy automorfizmov hrán lineárnych kódov  $\text{AutGroup}(C)$ . Skript spracúva na vstupe textové súbory kontrolných matíc H. Vo vývojovom diagrame pre náš skript zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.17).



Obr. 5.17: Vývojový diagram - autGroupCode skript

Na obrázku (Obr. 5.17) môžeme vidieť, že skript na jskôr načíta všetky súbory s incidenčnými maticami L a zoradí názvy súborov na základe veľkosti. Zavolá sa textový súbor s incidenčnou maticou L. Skript potrebuje zistiť, či niečo obsahuje. Ak je súbor prázdny, zahodí sa, a ak sa nejedná o posledný súbor v priečinku IncidenceMatrices, zavolá sa ďalší. Ak súbor nie je prázdny, otvorí sa súbor s incidenčnou maticou L, konvertuje sa obsah na incidenčnú maticu L, z nej sa získa kontrolná matica H a súbor sa uzavrie. Z incidenčnej matice L sa vygeneruje kontrolná matica H a z nej sa vygeneruje lineárny kód C(n,m,d). Následne skript vygeneruje graf z incidenčnej matice L a nastaví počiatočné rozdelenie vrcholov a hrán. V ďalších krokoch získame úplné permutácie hrán ako automorfizmy grupy. Názov grafu zistíme zo vstupného textového súboru a zobrazíme ho. Následne získame počet automorfizmov lineárneho kódu  $|\text{AutGroup}(C)|$ . Zobrazíme počet automorfizmov  $|\text{AutGroup}(C)|$  spolu s grupou automorfizmov zloženou z hrán. Ak je súbor s inci-

denčnou maticou  $L$  posledný, skript skončí. V opačnom prípade načíta ďalší textový súbor s incidenčnou maticou  $L$ . Skript *autGroupCode.sagews* nám umožňuje získať (pokiaľ je to výpočtovou náročnosťou možné) grupy automorfizmov lineárnych kódov  $\text{AutGroup}(C)$  z textových súborov incidenčných matíc  $L$ .

Vyššie uvedené skripty môžeme podľa dôležitosti rozdeliť na skripty generujúce vstupné alebo výstupné textové súbory a skripty informačného charakteru zobrazujúce detailly a výpočty.

### 5.1.18 Skripty generujúce vstupné alebo výstupné textové súbory

Do tejto kategórie zaraďujeme všetky skripty, ktoré je potrebné spustiť, aby sme skriptami informačného charakteru dostali relevantné výsledky v nasledujúcim poradí: *generateIncidenceMatrices*, *generateParityCheckMatrices*, *generateAutCodesFromCages*, *generateGeneratorMatrices*, *generateAutGroupLCode*.

### 5.1.19 Skripty informačného charakteru zobrazujúce detailly a výpočty

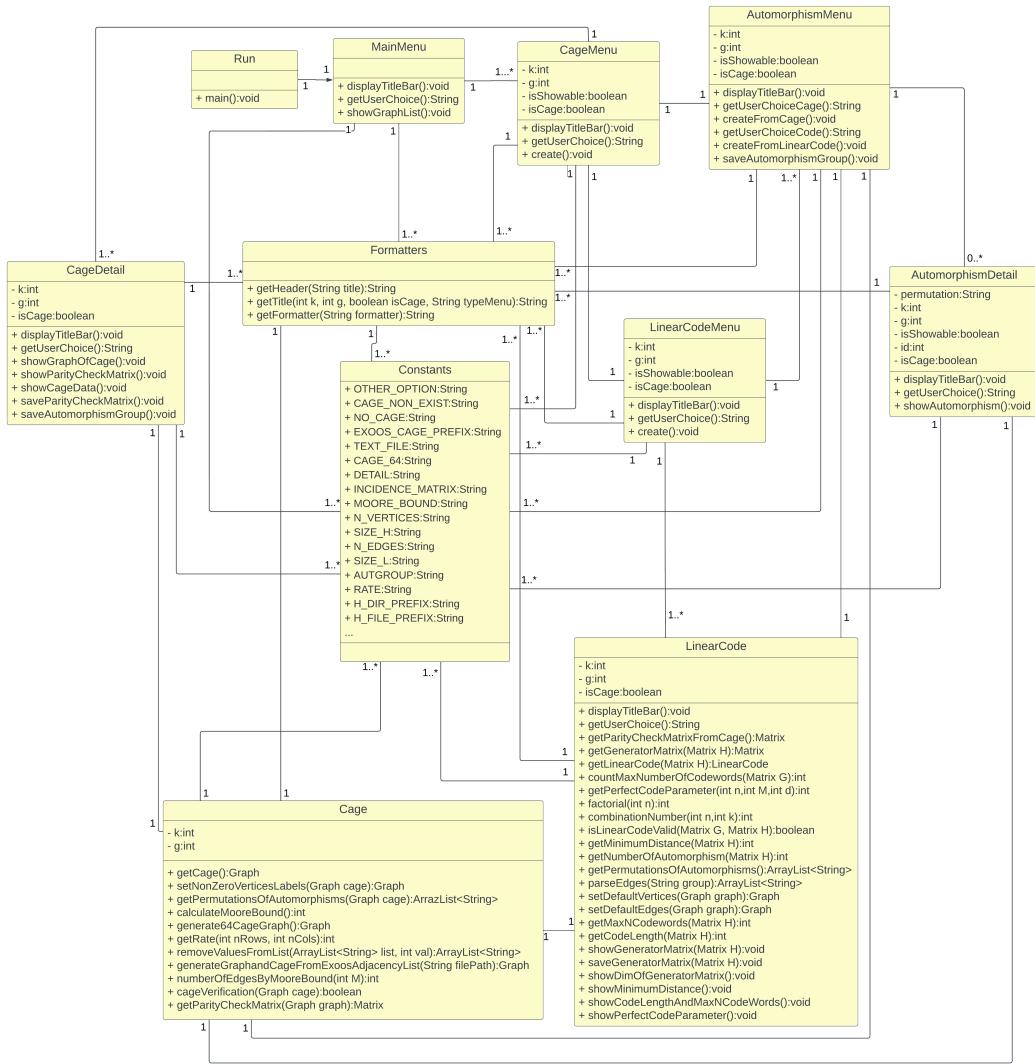
Do tejto kategórie zaraďujeme všetky ostatné skripty potrebné na získanie výsledkov: *generateCageAndLinearCodeByParameters*, *autCodesFromCages*, *mooreBoundsCageValidation*, *cagesData*, *linearCodesData*, *getLinearCodesParameter*, *minDistanceLinearCodes*, *maxWordsLinearCodes*, *sizeNLinearCodesData*, *autLinearCodesData*, *linearCodeValidation*, *autGroupCode*. Nie-

ktoré skripty sú navrhované príliš komplikované s vysokou duplicitou základných funkcií voči ostatným skriptom a ich výpočet je časovo a výpočtovo náročný.

## 5.2 Druhé riešenie - vlastná konzolová aplikácia

Hlavná logika generovania klietok alebo rekordných grafov je implementovaná v *Cage.py*. Používateľ spustí súbor *Run.py*, ktorý ho presmeruje na súbor *MainMenu.py* s hlavným menu. Následne sa naviguje medzi klietkami alebo rekordnými grafmi. Po vybratí konkrétnej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$  vie používateľ zobraziť jej graf (ak je to možné), incidenčnú maticu  $L$ , kontrolnú maticu  $H$ , údaje klietky (Moorove ohraničenie  $M(k,g)$ , počet vrcholov  $|ver|$ , rozmer incidenčnej  $L$  a kontrolnej matice  $H$ , počet hrán  $|edg|$ , počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$ , informačný pomer  $R$  pomocou *CageDetail.py*), grupu automorfizmov pre danú klietku  $\text{AutGroup}(\text{Cage})$ , pričom pri jej zobrazení vie používateľ zvoliť konkrétny automorfizmus. Z konkrétneho automorfizmu vie získať  $L$ ,  $H$  aj  $G$ , a ak  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  patrí do kategórie "graficky zobraziteľných", vie zobraziť aj obraz  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  (*AutomorphismMenu.py* a *Automorphism.py*). Incidenčnú maticu  $L$  získanú z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  vie používateľ aj uložiť, a rovnako tak aj  $\text{AutGroup}(\text{Cage})$  a kontrolnú maticu  $H$ . Používateľ sa dokáže dostať k menu lineárny kód (*LinearCodeMenu.py*). V menu lineárny kód dokáže používateľ zobraziť a uložiť  $G$ , zobraziť jej rozmer,  $|\text{AutGroup}(C)|$  v lineárnom kóde, dĺžku lineárneho kódu  $n$ , počet slov v kóde  $m$ , minimálnu Hammingovu vzdialenosť v kóde  $d$  a parameter perfektného kódu  $p(n,m,d)$  (*LinearCode.py*). Takisto vie zobraziť aj uložiť grupu automorfizmov lineárneho kódu  $\text{AutGroup}(C)$  (*AutomorphismMenu.py* a *Auto-*

*morphismDetail.py*). Textové konštanty sú uložené v *Constants.py*. Na formátovanie výpisov sme navrhli vlastnú triedu *Formatters.py*, kde formátujeme reťazcové výstupy a hlavičku aplikácie spolu s meniacimi sa názvami rôznych menu. Zobrazovanie výsledkov umožňuje používateľovi vyberať si konkrétnie výpočty pre konkrétné Cage(k,g) alebo Rec(k,g) a z nich aj konkrétné automorfizmy. Pythonové súbory, vrátane tried, sú navrhované s minimálnou duplicitou základných funkcií a ich výpočet je menej časovo a výpočtovo náročný, avšak vždy vieme pracovať s grafmi iba jednotlivco. Všetky spomínané triedy znázorňujeme v diagrame tried, ktorý je zobrazený na nasledujúcom obrázku (Obr. 5.17).



Obr. 5.18: Diagram tried - konzolová aplikácia

Spolu s triedami sú zobrazené aj nami navrhnuté funkcie. Okrem predprogramovaných funkcií knižnice SageMath [19] sme pri riešení problematiky programovali aj množstvo vlastných funkcií, niektoré už boli stručne spomenuté v návrhu riešenia. Zobrazíme si tie najdôležitejšie z nich.

### 5.2.1 Výpočet Moorového ohraničenia

```
# vypocita najmensi mozny pocet vrcholov pre potencionalnu klietku (testovacie data)
def calculateMooreBound(self):
    M = 0
    exp = 0
    if self.g % 2 == 1:
        exp = (self.g - 3) / 2
        for i in range(int(exp) + 1):
            M = M + (self.k * ((self.k - 1) ** i))
        M = M + 1
    else:
        exp = (self.g - 2)/2
        for i in range(int(exp) + 1):
            M = M + ((self.k - 1) ** i)
        M = 2 * M
    return M
```

Obr. 5.19: Výpočet Moorového ohraničenia - riešenie

### 5.2.2 Validácia klietok a rekordných grafov

```
# vypocita potencialny pocet hrán
# na zaklade Moorovho ohranicenia (testovacie data)
def numberOfEdgesByMooreBound(self,M):
    return M * self.k / 2

# test vysledkov
def cageVerification(self,cage):
    M = self.calculateMooreBound()
    minPocetHran = self.numberOfEdgesByMooreBound(M)
    # verifikacia poctu vrcholov
    if len(cage.vertices()) < M or len(
        cage.incidence_matrix().column(0)) < M:
        return False
    # verifikacia poctu hrán
    if len(cage.edges()) < minPocetHran or len(
        cage.incidence_matrix().row(0)) < minPocetHran:
        return False
    return True
```

Obr. 5.20: Validácia klietok a rekordných grafov - riešenie

Funkcia `calculateMooreBound()` vypočíta najmenší možný počet vrcholov  $|ver|$  pre graf. V riešeniach je využívaná na validáciu vygenerovaných grafov, ako môžeme vidieť na `cageVerification()`. Na základe funkcie `numberOfEdgesByMooreBound()` sme schopní overiť počet hrán grafu  $|edg|$ .

### 5.2.3 Konštrukcia klietky alebo rekordného grafu zo vstupných dát

```
# sledujeme 3 zoznamy uvažovaných klietok s ktorými porovnavame parametre,
# pre každý zoznam sa grafy generuju iným spôsobom
def getCage(self):
    sageCages = [[3,5], [3,6], [3,7], [3,8], [3,10], [3,11], [4,5], [7,5]]
    exoosCages = [[3,14], [3,16], [3,17], [3,18], [3,20], [3,23], [3,25],
    [4,7], [4,9], [4,10], [5,10], [7,7], [7,8], [10,5], [11,5], [12,5], [13,5]]
    myConstructedCages = [[6,4]]
    cageInput = []
    cageInput.append(self.k)
    cageInput.append(self.g)
    if ((cageInput not in sageCages) and (
        cageInput not in exoosCages) and (cageInput not in myConstructedCages)):
        print(self.formatters.getFormatter(
            1).format(self.const.CAGE_NON_EXIST))
        return
    cage = graphs.EmptyGraph()
    if cageInput in sageCages:
        if self.k == 3:
            if self.g == 5:
                cage = graphs.PetersenGraph()
            if self.g == 6:
                cage = graphs.HeawoodGraph()
            if self.g == 7:
                cage = graphs.McGeeGraph()
            if self.g == 8:
                cage = graphs.TutteCoxeterGraph()
            if self.g == 10:
                cage = graphs.Balaban10Cage()
            if self.g == 11:
                cage = graphs.Balaban11Cage()
        if self.k == 4 and self.g == 5:
            cage = graphs.RobertsonGraph()
        if self.k == 7 and self.g == 5:
            cage = graphs.HoffmanSingletonGraph()
    elif cageInput in exoosCages:
        cage = self.generateGraphandCageFromExoosAdjacencylist(
            self.const.EXOOS_CAGE_PREFIX + str(self.k) + "_" + str(
            self.g) + self.const.TEXT_FILE)
    elif cageInput in myConstructedCages:
        if self.k == 6 and self.g == 4:
            cage = self.generate64CageGraph()
    if (self.cageVerification(cage) == False):
        print(self.const.NO_CAGE + "\n")
        return
    cage = self.setNonZeroVerticesLabels(cage)
    return cage
```

Obr. 5.21: Konštrukcia klietky alebo rekordného grafu - riešenie

Funkcia *getCage()* Spracováva všetky tri spôsoby generovania klietok alebo rekordných grafov. Na vstupe máme parametre pre grafy generované zo SageMath [19], zo zoznamov susedností [21] pomocou parsera a pomocou nášho vlastného návrhu. Výstupom je graf klietky alebo rekordného grafu s počiatocným označením vrcholov.

#### 5.2.4 Generovanie klietky Cage(6,4)

```
# vygeneruje Cage(6,4), bez vstupnych parametrov ako bipartitny graf
def generate64CageGraph(self):
    self.k = 6
    self.g = 4
    M = self.calculateMooreBound()
    s = []
    for i in range(int(M/2)):
        s.append(6)
    graph = graphs.DegreeSequenceBipartite(s,s)
    for i in range (M):
        graph.set_vertex(i, str(i))
    graph.name(self.const.CAGE_64)
    return graph
```

Obr. 5.22: Konštrukcia Cage(6,4) - riešenie

Funkcia *generate64CageGraph()* generuje graf Cage(6,4) na základe údajov, ktoré sú o ňom známe. Napríklad z Moorového ohraničenia  $M(k,g)$  vieme určiť presný počet vrcholov grafu  $|ver|$ .

### 5.2.5 Parser na spracovanie textových súborov zoznamov susedností

```
# odstranenie hodnot zo zoznamu na zaklade hodnoty
def removeValuesFromList(self,list, val):
    return [value for value in list if value != val]

# parsovanie Exoo dokumentov so zoznamami susednych vrcholov na generovanie grafu
def generateGraphandCageFromExoosAdjacencylist(self,filePath):
    file = open(filePath, "r")
    graph = graphs.EmptyGraph()
    mainVertex = 0
    for line in file:
        line = line.replace('\n', '')
        adjacencyVerticies = line.split(" ")
        adjacencyVerticies = self.removeValuesFromList(adjacencyVerticies, '')
        for adjacencyVertex in adjacencyVerticies:
            graph.add_edge(mainVertex, int(adjacencyVertex))
        mainVertex += 1
    file.close()
    return graph
```

Obr. 5.23: Parser zoznamu susedností - riešenie

Funkcia *generateGraphandCageFromExoosAdjacencylist()* spracováva na vstupe textový súbor so zoznamom susedností medzi vrcholmi [21]. Funkcia je parserom, ktorý pre graf definuje hrany medzi vrcholmi tak, aby sme sa z prázdnego grafu dopracovali ku klietke alebo rekordnému grafu. Výstupom funkcie je klietka alebo rekordný graf.

### 5.2.6 Výpočet parametra perfektného kódu

```
# udava vzdialenosť nášho kódu od perfektného
# v intervale 0 - 1, kedy 1 je perfektny kód
def getPerfectCodeParameter(self,n,M,d):
    if d % 2 == 1:
        t = (d-1)/2      # t=(d-1)/2 pre nepárne
    else:
        t = (d-2)/2      # parne (d-1)/2 dostavam desatinne číslo,
    sum = 0.0            # uvádzajem len cele, preto d-2
    for i in range(0,int(t) + 1):
        sum = sum + self.combinationNumber(n,i)
    parameter = (M*sum)/(2**n)
    return parameter
```

Obr. 5.24: Výpočet parametra perfektného kódu - riešenie

Funkcia `getPerfectCodeParameter()` spracováva na vstupe dĺžku lineárneho kódu  $n$ , maximálny počet kódových slov  $m$  a minimálnu Hammingovu vzdialenosť  $d$ . Výstupom funkcie je parameter  $p(n,m,d)$ , ktorý slúži na porovnanie nášho kódu s perfektným.

### 5.2.7 Validácia lineárneho kódu

```
# ak je súčet všetkých elementov matice súčinu G a Ht rovny 0
# a súčasne platí že ma tato matice rovnaky počet riadkov ako G
# a rovnaky počet stĺpcov ako H^t potom je kód validný
def isLinearCodeValid(self,G,H):
    Ht = H.transpose()
    validationMatrix = G * Ht
    elementsSum = sum(sum(validationMatrix))
    if len(G.column(0)) == len(validationMatrix.column(0)) and len(
        Ht.row(0)) == len(validationMatrix.row(0)) and elementsSum == 0:
        return True
    else:
        return False
```

Obr. 5.25: Validácia lineárneho kódu - riešenie

Funkcia *isLinearCodeValid()* spracováva na vstupe kontrolnú H a generujúcu maticu G lineárneho kódu. Jej výstupom je overenie, či je uvažovaný lineárny kód C(n,m,d) validný alebo nie. Funkcia slúži na otestovanie správnosti riešenia.

### 5.2.8 Generovanie incidenčných a kontrolných matíc

```
# vrati kontrolnu maticu lineárneho kodu ako
# incidenčnu maticu klietky
# s lineárne nezávislými riadkami matice
def getParityCheckMatrix(self,graph):
    L = self.getIncidenceMatrix(graph)
    n = len(L.row(0))
    v0 = zero_vector(n)
    Harray = []
    for incidenceRow in L.echelon_form():
        if incidenceRow != v0:
            Harray.append(incidenceRow)
    return matrix(GF(2),Harray)
```

Obr. 5.26: Generovanie incidenčných a kontrolných matíc - riešenie

Funkcia *getParityCheckMatrix()* spracováva na vstupe klietku alebo rekordný graf. Z grafu získame incidenčnú maticu, z ktorej odstránením lineárne závislých riadkov získavame kontrolnú maticu H v echelónovom tvare. Kontrolná matica H je zároveň výstupom funkcie.

### 5.2.9 Ukladanie matice do textového súboru

```
# uloží maticu H
def saveParityCheckMatrix(self):
    graph = Cage(self.k, self.g)
    choice = ''
    cage = graph.getCage()
    if not os.path.exists(self.const.PARITY_CHECK_MATRIX_DIRECTORY):
        os.makedirs(self.const.PARITY_CHECK_MATRIX_DIRECTORY)
    startTime = datetime.datetime.now()
    H = graph.getParityCheckMatrix(cage)
    file = open(self.formatters.getFormatter(9).format(
        self.const.H_DIR_PREFIX, self.k, self.g, self.const.TEXT_FILE), 'w')
    file.write(str(H))
    file.close()
    time = datetime.datetime.now() - startTime
    while choice != self.const.OPTION_X:
        self.displayTitleBar()
        print(self.formatters.getFormatter(24).format([
            self.const.H_FILE_PREFIX, self.k, self.g, self.const.SUCCESS_CREATE, time.total_seconds()]))
        choice = self.getUserChoice()
        self.displayTitleBar()
        if choice == self.const.OPTION_X:
            break
        else:
            print(self.formatters.getFormatter(1).format(self.const.OTHER_OPTION))
```

Obr. 5.27: Ukladanie matice do textoveho suboru - riešenie

Funkcia *getParityCheckMatrix()* (podobne aj *getIncidenceMatrix()* a *getGeneratorMatrix()*) spracuje graf na ďalšie použitie. Vytvorí priečinok Parity-CheckMatrices v prípade, ak neexistuje (prípadne IncidenceMatrices, GeneratorMatrices), vygeneruje potrebnú maticu a uloží ju do textového súboru v spomínanom priečinku. Po vygenerovaní a uložení zobrazí čas trvania tohto procesu a výstupom funkcie je textový súbor s výslednou maticou.

### 5.2.10 Získanie a uloženie grupy automorfizmov z grafu

```
# ulozi grupu automorfizmov klietky alebo rec grafu
def saveAutomorphismGroup(self):
    graph = Cage(self.k, self.g)
    choice = ''
    cage = graph.getCage()
    permutations = graph.getPermutationsOfAutomorphisms(cage)
    if not os.path.exists(self.const.AUTOMORPHISM_GROUP_DIRECTORY):
        os.makedirs(self.const.AUTOMORPHISM_GROUP_DIRECTORY)
    startTime = datetime.datetime.now()
    file = open(self.formatters.getFormatter(9).format(
        self.const.AUT_GROUP_CAGE_DIR_PREFIX, self.k, self.g, self.const.TEXT_FILE), 'w')
    file.write("[\n")
    for i in range(len(permutations)):
        file.write("\t" + str(permutations[i]))
        if i < len(permutations) - 1:
            file.write(",")
        file.write("\n")
    file.write("]")
    file.close()
    time = datetime.datetime.now() - startTime
    while choice != self.const.OPTION_X:
        self.displayTitleBar()
        print(self.formatters.getFormatter(24).format(
            self.const.AUT_GROUP_FILE_PREFIX, self.k, self.g, self.const.SUCCESS_CREATE, time.total_seconds()))
        choice = self.getUserChoice()
        self.displayTitleBar()
        if choice == self.const.OPTION_X:
            break
        else:
            print(self.formatters.getFormatter(1).format(self.const.OTHER_OPTION))
```

Obr. 5.28: Získanie a uloženie grupy automorfizmov z grafu - riešenie

Funkcia `saveAutomorphismGroup()` spracuje graf na ďalšie použitie. Získa úplne permutácie tvorené z vrcholov (v tvare na vytvorenie obrazu grafu). Vytvorí priečinok AutomorphismGroups v prípade, ak neexistuje, uloží jednotlivé permutácie ako automorfizmy do textového súboru v spomínanom priečinku. Po uložení jednotlivých automorfizmov dostávame grupu automorfizmov grafu. Zobrazíme čas trvania tohto procesu a výstupom funkcie je textový súbor s výslednou grupou automorfizmou grafu.

### 5.2.11 Získanie a uloženie grupy automorfizmov z lineárneho kódu

```
# ulozenie grupy automorfizmov z lineárneho kódu
def saveAutomorphismGroup(self):
    linearCode = LinearCode(self.k, self.g, self.isCage)
    choice = ''
    permutations = linearCode.getPermutationsOfAutomorphisms()
    if not os.path.exists(self.const.AUTOMORPHISM_GROUP_DIRECTORY):
        os.makedirs(self.const.AUTOMORPHISM_GROUP_DIRECTORY)
    startTime = datetime.datetime.now()
    file = open(self.formatters.getFormatter(12).format(
        self.const.AUT_GROUP_CAGE_DIR_PREFIX, self.k, self.g,
        self.const.AUT_GROUP_SUFFIX, self.const.TEXT_FILE), 'w')
    file.write("[\n")
    for i in range(len(permutations)):
        file.write("\t{}".format(permutations[i]))
        if i < len(permutations) - 1:
            file.write(",")
        file.write("\n")
    file.write("]")
    file.close()
    time = datetime.datetime.now() - startTime
    while choice != self.const.OPTION_X:
        self.displayTitleBar()
        print(self.formatters.getFormatter(28).format(
            self.const.AUT_GROUP_FILE_PREFIX, self.k, self.g,
            self.const.AUT_GROUP_SUFFIX, self.const.SUCCESS_CREATE, time.total_seconds()))
        choice = self.getUserChoiceCode()
        self.displayTitleBar()
        if choice == self.const.OPTION_X:
            break
        else:
            print(self.formatters.getFormatter(1).format(self.const.OTHER_OPTION))
```

Obr. 5.29: Získanie a uloženie grupy automorfizmov lineárneho kódu - riešenie

Funkcia `saveAutomorphismGroup()` spracuje lineárny kód  $C(n,m,d)$  na ďalšie použitie. Získa úplne permutácie reprezentujúce hrany grafu zodpovedajúcemu príslušnému lineárному kódu  $C(n,m,d)$ . Vytvorí priečinok `AutomorphismGroups` v prípade, ak neexistuje, uloží jednotlivé permutácie ako automorfizmy do textového súboru v spomínanom priečinku. Po uložení jednotlivých automorfizmov dostávame grupu automorfizmov lineárneho kódu

`AutGroup(C)`. Zobrazíme čas trvania tohto procesu a výstupom funkcie je textový súbor s výslednou grupou automorfizmou lineárneho kódu.

### 5.2.12 Inicializácia počiatočných vrcholov a doplnenie grupy automorfizmov grafu

```
# nastavi vrcholy od 1 po pocet vrcholov pre danu klietku
def setNonZeroVerticesLabels(self, cage):
    vertices = []
    for vertex in range(len(cage.vertices(sort=False))):
        vertices.append(vertex + 1)
    cage.relabel(vertices)
    return cage

# vrati doplneny zoznam automorfizmov
def getPermutationsOfAutomorphisms(self, cage):
    vertices = cage.vertices()
    automorphismGroup = cage.automorphism_group().list()
    permutations = []
    for i in range(len(automorphismGroup)):
        if i > 0:
            automorphism = str(automorphismGroup[i])
            for vertex in vertices:
                if str(vertex) not in automorphism:
                    automorphism += '(' + str(vertex) + ')'
            permutations.append(str(automorphism))
    return permutations
```

Obr. 5.30: Doplnenie automorfizmov grafu a inicializácia vrcholov - riešenie

Funkcia `setNonZeroVerticesLabels()` spracováva na vstupe klietku `Cage(k,g)` alebo rekordný graf `Rec(k,g)`. Výstupom funkcie je zoznam vrcholov s označením od jedna po celkový počet vrcholov grafu  $|ver|$ . Funkcia `getPermutationsOfAutomorphisms()` spracováva na vstupe klietku `Cage(k,g)` alebo rekordný graf `Rec(k,g)`, jej výstupom je doplnený zoznam permutácií vrcholov tak, aby obsahovali informáciu o všetkých vrcholoch grafu.

### 5.2.13 Inicializácia počiatočných vrcholov, hrán a doplnenie grupy automorfizmov kódu

```
# vrati zoznam automorfizmov dopnený o hrany
def getPermutationsOfAutomorphisms(self):
    L = self.getIncidenceMatrixFromCage()
    H = self.getParityCheckMatrixFromCage()
    C = self.getLinearCode(H)
    automorphismGroup = C.permutation_automorphism_group().list()
    permutations = []

    graph = Graph(L)
    graph = self.setDefaultVertices(graph)
    graph = self.setDefaultEdges(graph)

    edges=graph.edges()
    automorphism = ""
    for i in range(len(automorphismGroup)):
        if i > 0:
            for group in automorphismGroup[i]:
                automorphism += "("
                for edgesInGroup in group:
                    for edgeInGroup in self.parseEdges(edgesInGroup):
                        for edge in edges:
                            if str(edge[2]) == str(edgeInGroup):
                                automorphism += str(edge)
                                continue
                        automorphism += ")"
                for edge in edges:
                    if str(edge) not in automorphism:
                        automorphism += '(' + str(edge) + ')'
                permutations.append(str(automorphism))
                automorphism = ""
    return permutations

def parseEdges(self, group):
    return str(group).replace('(', '').replace(')', '').split(",")
```

Obr. 5.31: Doplnenie automorfizmov kódu a inicializácia vrcholov, hrán - riešenie

Funkcia `getPermutationsOfAutomorphisms()` spracováva incidenčnú maticu a jej výstupom je doplnený zoznam permutácií hrán reprezentujúcich graf

zodpovedajúci incidenčnej matici tak, aby obsahovali informáciu o všetkých hranách grafu. Dôležitým krokom je označenie hrán grafu pre lepšiu prehľadnosť v permutáciách hrán.

### 5.2.14 Zobrazenie obrazu grafu, jeho incidenčná kontrolná a generujúca matica

```
# zobrazi graf obrazu kliekty pokial je to mozne
# a vygeneruje jeho kontrolnu a gener maticu
def showAutomorphism():
    graph = Cage(self.k, self.g)
    cage = graph.getCage()
    p = Permutation(self.permutation)
    cage.relabel(p)

    # podmienka prehľadneho zobrazenia
    if (self.isShowable):
        cage.show(figsize=10)

    L = graph.getIncidenceMatrix(cage)
    H = graph.getParityCheckMatrix(cage)
    C = codes.from_parity_check_matrix(H)
    G = C.systematic_generator_matrix()
    choice = ''

    while choice != self.const.OPTION_X:
        self.displayTitleBar()
        print(self.formatters.getFormatter(25).format(
            self.const.INCIDENCE_MATRIX, L, self.const.PARITYCHECK_MATRIX,
            H, self.const.GENERATOR_MATRIX, G))
        choice = self.getUserChoice()
        self.displayTitleBar()
        if choice == self.const.OPTION_X:
            break
        else:
            print(self.formatters.getFormatter(1).format(self.const.OTHER_OPTION))
```

Obr. 5.32: Zobrazenie obrazu grafu, incidenčná, kontrolná a generujúca matica - riešenie

Funkcia `showAutomorphism()` spracúva graf a úplnú permutáciu vrcholov z automorfizmu grafu. Na základe permutácie preznačí označenie grafu a získavame obraz pôvodného grafu. Podmienka zobrazenia je nastavená pre

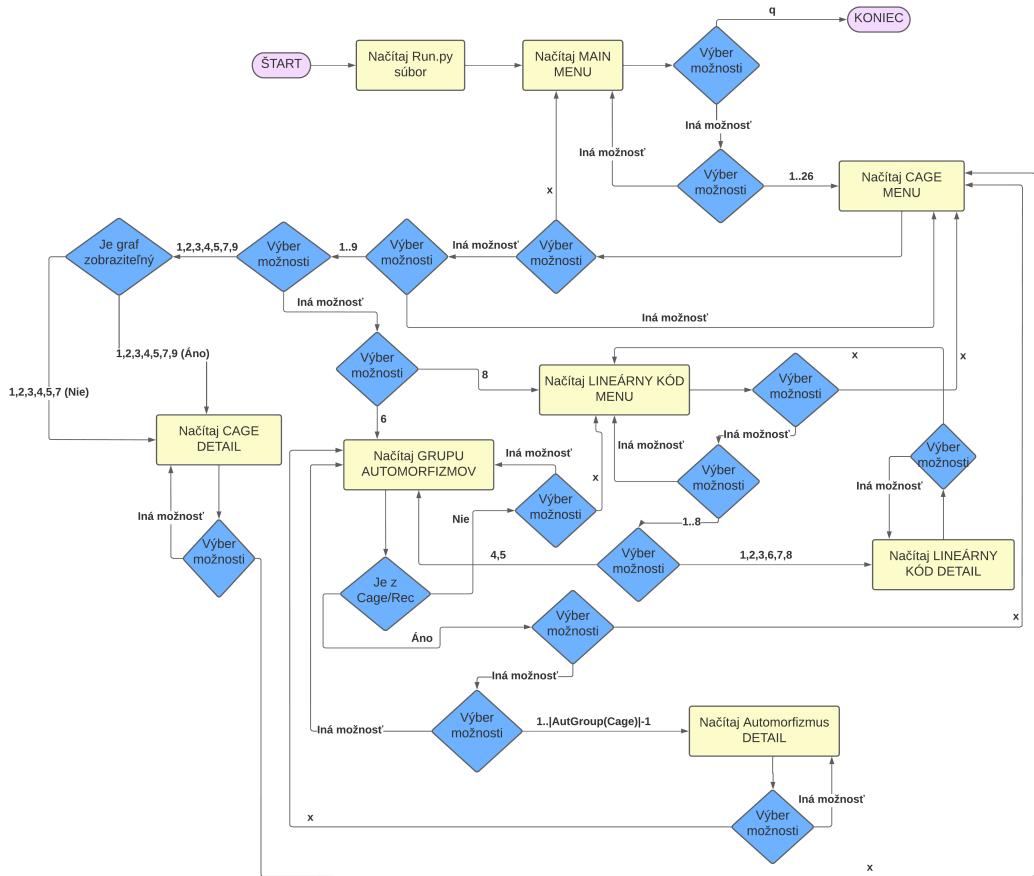
"graficky zobraziteľné klietky". Z grafu vieme získať incidenčnú L, kontrolnú H a generujúcnu maticu G, ktoré spolu s obrazom grafu tvoria výstupy funkcie.

### 5.2.15 Formáter na formátovanie výstupných výpisov

Obr. 5.33: Formátovanie výstupných výpisov - riešenie

Funkcia `getFormatter()` spracováva na vstupe index formátera, na základe ktorého vyberie konkrétny tvar výpisu a ten je vrátený ako výstup funkcie.

Zvyšné funkcie, ako aj obe riešenia sú dostupné v prílohách. Vo vývojovom diagrame pre našu konzolovú aplikáciu zobrazujeme všetky základné a rozhodovacie procesy, ktoré vedú k získaniu výsledkov (Obr. 5.34).



Obr. 5.34: Vývojový diagram - konzolová aplikácia

Na obrázku (Obr. 5.34) zobrazujeme vývojový diagram konzolovej aplikácie. Používateľ sa vie navigovať pomocou výberu možností a dostať sa k potrebným výsledkom (pokiaľ je to výpočtovou náročnosťou možné). Po načítaní súboru Run.py si v MAIN MENU vyberie graf. Následne má deväť možností navigácie. Môže sa vrátiť naspäť pomocou voľby "x", pokračovať načítaním detailu grafu (možnosti "1,2,3,4,5,7,9"), načítaním grupy automorfizmov grafu AutGroup(Cage) (možnosť "6") alebo načítaním menu samotného lineárneho kódu (možnosť "8"). Z každej voľby je možný návrat pomocou možnosti "x" až na MAIN MENU, odtiaľ voľbou "q" ukončíme spustenie

aplikácie. Z MENU LINEÁRNY KÓD sa vieme dostať buď k detailu lineárneho kódu (možnosti "1,2,3,6,7,8") alebo ku grupe automorfizmov (možnosti "4,5"). Z grupy automorfizmov sa vieme dostať k detailu konkrétneho automorfizmu výberom jeho poradového čísla ako možnosti, avšak táto vlastnosť platí iba pre klietky alebo rekordné grafy. Vývojový diagram popisuje prehľadnú navigáciu v konzolovej aplikácii.

### 5.3 Ukladanie výsledkov

Ukladať si budeme incidenčné matice  $L$  vygenerované z klietok  $\text{Cage}(k,g)$  alebo rekordných grafov  $\text{Rec}(k,g)$ , z nich získané kontrolné matice  $H$ , generujúce matice  $G$  vygenerované z kontrolných matíc  $H$ , a takisto automorfizmy grafov  $\text{AutGroup}(\text{Cage})$  a automorfizmy lineárnych kódov  $\text{AutGroup}(C)$ . Všetky výsledky ukladáme vo forme textových súborov. Incidenčné matice  $L$  ukladáme v adresári *IncidenceMatrices*, kontrolné matice  $H$  ukladáme v adresári *ParityCheckMatrices* a generujúce matice  $G$  ukladáme v adresári *GeneratorMatrices*. Všetky sú ukladané v maticovom tvare. Grupy automorfizmov  $\text{AutGroup}(\text{Cage})$  a  $\text{AutGroup}(C)$  ukladáme v adresári *AutomorphismGroups* ako zoznamy úplných permutácií.

# Kapitola 6

## Výstupy navrhovaných riešení

V kapitole výstupy navrhovaných riešení popisujeme výstupy z oboch navrhovaných riešení vrátane vypočítaných výsledkov a ich porovnaní.

### 6.1 Klietky, rekordné grafy a ich reprezentujúce matice

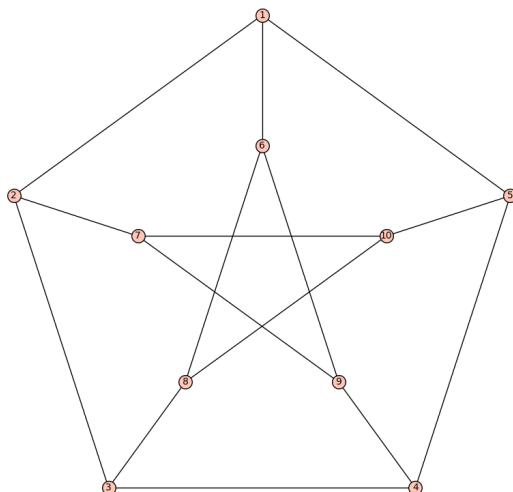
Pre každú klietku  $\text{Cage}(k,g)$  a rekordný graf  $\text{Rec}(k,g)$  uvádzame Moorové ohraničenie  $M(k,g)$ , informačný pomer  $R$ , počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  získaný z  $\text{Cage}(k,g)$ , rozmer incidenčnej matice  $L$ , rozmer kontrolnej matice  $H$  a čas potrebný na ich vygenerovanie a uloženie  $t(L), t(H)$ . Moorové ohraničenie  $M(k,g)$  nám slúži na overenie, či na základe počtu vrcholov môže  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  existovať. Rozmer  $L$  získame pomocou počtu vrcholov  $|ver|$  a hrán  $|edg|$  grafu. Na základe  $|ver|$  a  $|edg|$  sme  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  rozdelili na "graficky zobraziteľné" klietky alebo rekordné grafy a "graficky nezobraziteľné" klietky alebo rekordné grafy. Toto rozdelenie je z dôvodu prehľadnosti obrázkov.  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  s príliš veľkým  $|ver|$  a  $|edg|$  by boli neprehľadné, a preto ich nebudeme graficky zobrazovať. Kon-

trolnú maticu  $H$  generujeme z incidenčnej matice  $L$ . Výsledky popisujeme v tabuľkách, časovú náročnosť zobrazujeme v grafoch  $L$  v závislosti od  $t(L)$  a  $H$  v závislosti od  $t(H)$ . Výsledné  $L$  a  $H$  ukladáme v textových súboroch. V jednotlivých podkapitolách porovnávame výsledky oboch riešení.

### 6.1.1 Graficky zobraziteľné klietky alebo rekordné grafy

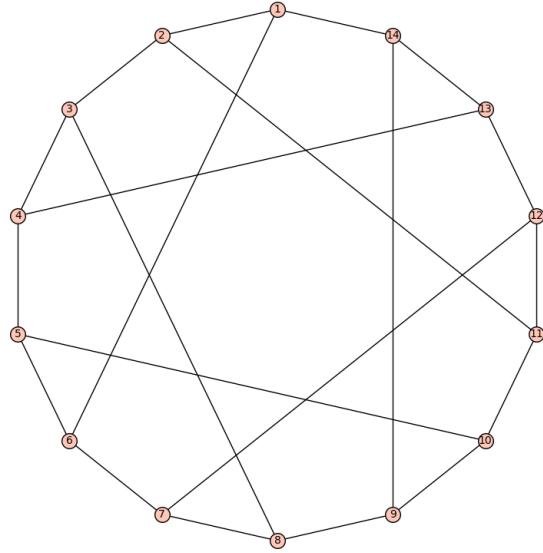
Uvádzame obrázky grafov, ktoré bolo možné prehľadne zobraziť (Obr. 6.1 - Obr. 6.12). Grafy Cage(3,5), Cage(3,6), Cage(3,7), Cage(3,8), Cage(3,10), Cage(3,11), Cage(4,5) a Cage(7,5) sme zobrazili priamo zo SageMath [19]. Grafy Rec(3,14), Cage(4,7), Cage(4,9) a Cage(6,4) sme vykreslili podľa toho, akým spôsobom sa nám ich podarilo vygenerovať a zobraziť.

#### 1. Cage(3,5) - Petersenov graf



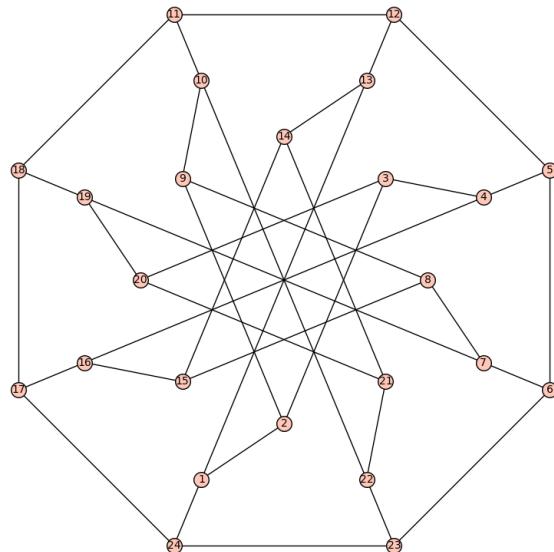
Obr. 6.1: Cage(3,5) - výstupný graf [19]

## 2. Cage(3,6) - Heawoodov graf



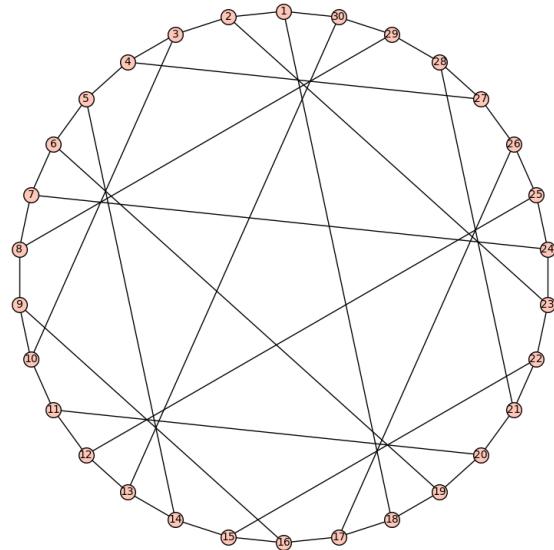
Obr. 6.2: Cage(3,6) - výstupný graf [19]

## 3. Cage(3,7) - McGeeho graf



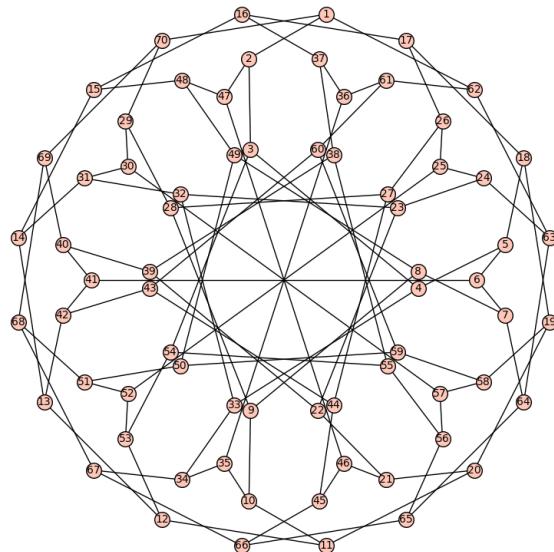
Obr. 6.3: Cage(3,7) - výstupný graf [19]

## 4. Cage(3,8) - Tutteho-Coxeterov graf



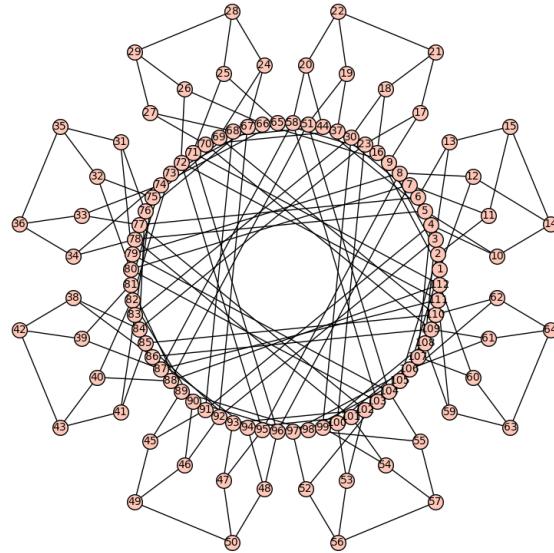
Obr. 6.4: Cage(3,8) - výstupný graf [19]

## 5. Cage(3,10) - Balabanov graf



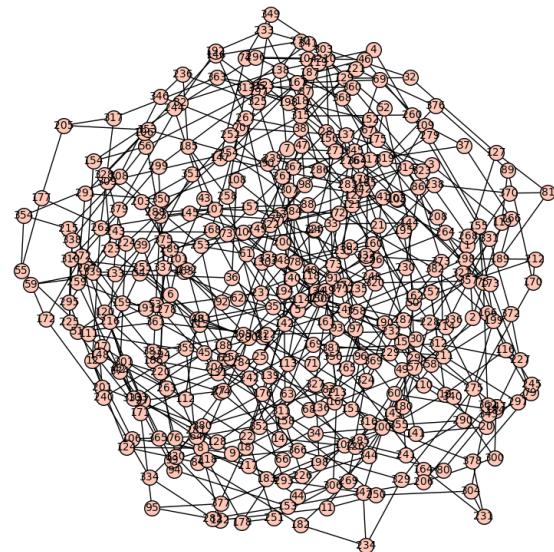
Obr. 6.5: Cage(3,10) - výstupný graf [19]

## 6. Cage(3,11) - Balabanov graf



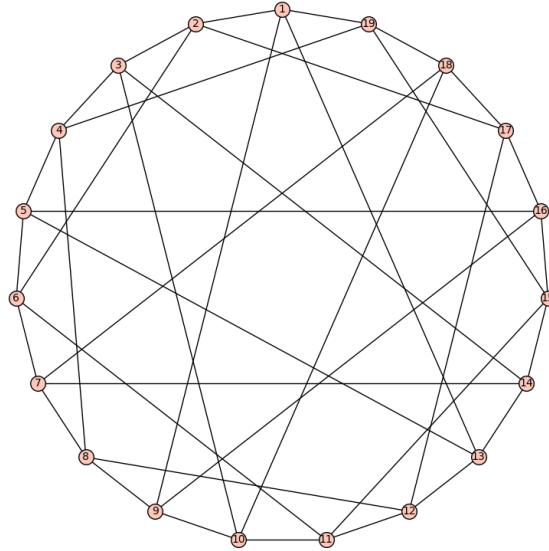
Obr. 6.6: Cage(3,11) - výstupný graf [19]

## 7. Rec(3,14)



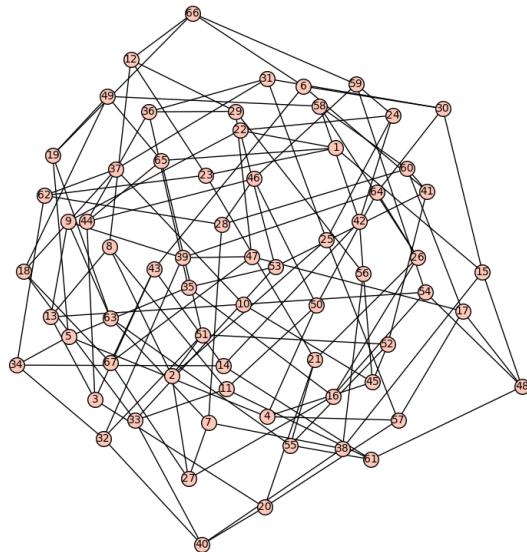
Obr. 6.7: Rec(3,14) - výstupný graf

## 8. Cage(4,5) - Robertsonov graf



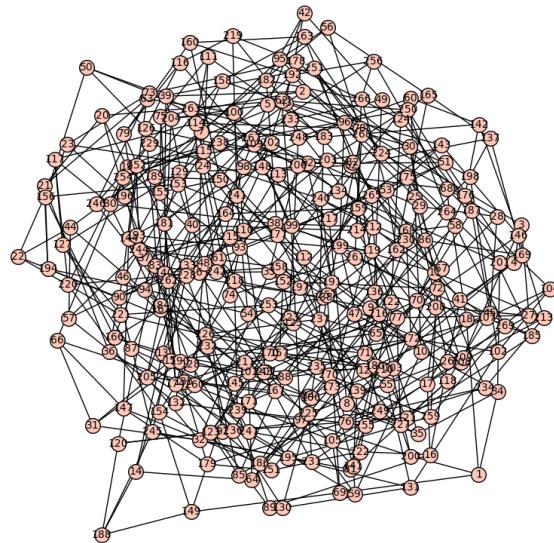
Obr. 6.8: Cage(4,5) - výstupný graf [19]

## 9. Cage(4,7)



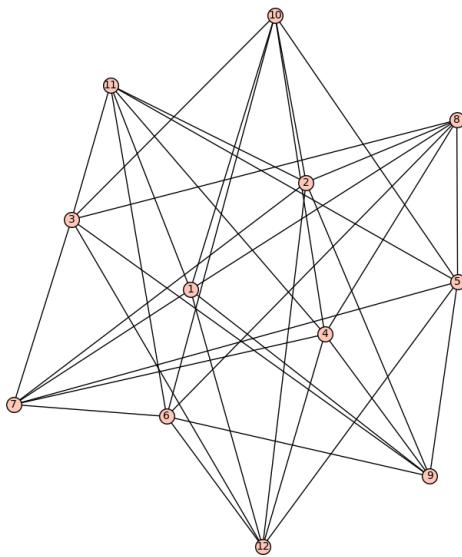
Obr. 6.9: Cage(4,7) - výstupný graf

## 10. Cage(4,9)



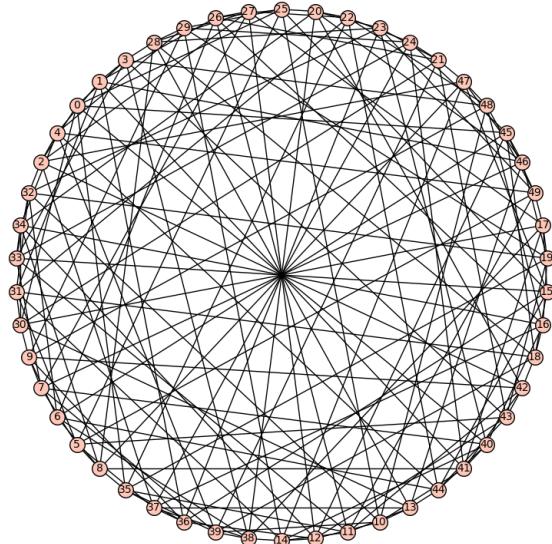
Obr. 6.10: Cage(4,9) - výstupný graf

## 11. Cage(6,4)



Obr. 6.11: Cage(6,4) - výstupný graf

## 12. Cage(7,5)



Obr. 6.12: Cage(7,5) - výstupný graf [19]

### Výstupy riešenia v CoCalc

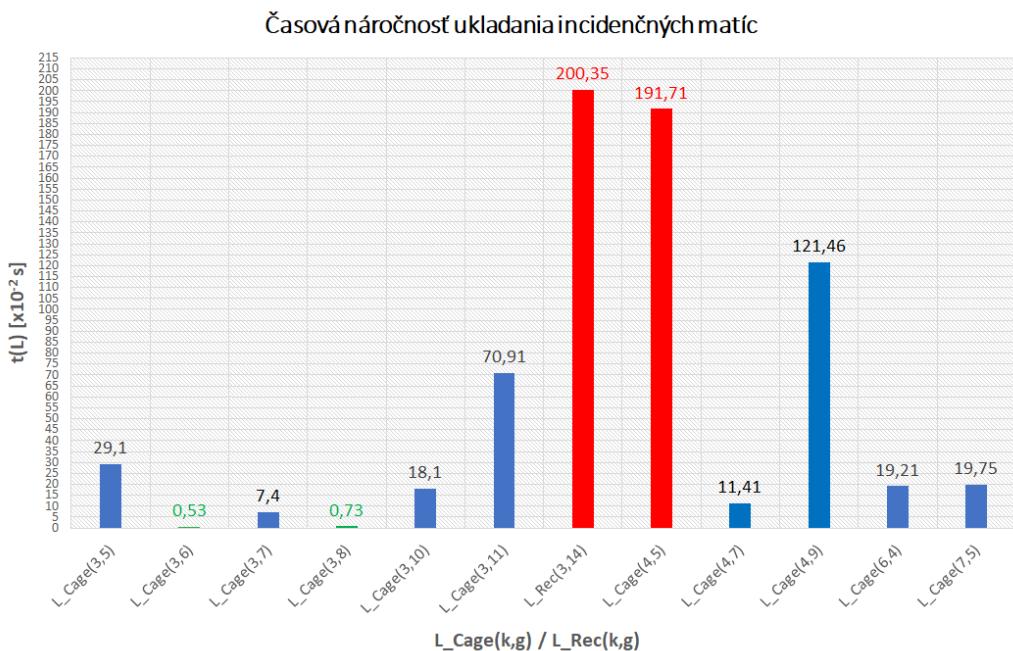
Cage/Rec	M	A(gp)	L	t(L)s	H	t(H)s	R
Cage(3,5)	10	120	$10 \times 15$	0.291	$9 \times 15$	0.0421	0.4
Cage(3,6)	14	336	$14 \times 21$	0.0053	$13 \times 21$	0.0017	0.381
Cage(3,7)	22	32	$24 \times 36$	0.074	$23 \times 36$	0.0023	0.361
Cage(3,8)	30	1440	$30 \times 45$	0.0073	$29 \times 45$	0.0028	0.355
Cage(3,10)	62	80	$70 \times 105$	0.181	$69 \times 105$	0.0106	0.343
Cage(3,11)	94	64	$112 \times 168$	0.7091	$111 \times 168$	0.0267	0.339
Rec(3,14)	254	96	$384 \times 576$	2.0035	$383 \times 576$	1.0082	0.335
Cage(4,5)	17	24	$19 \times 38$	1.9171	$18 \times 38$	0.0038	0.526
Cage(4,7)	53	4	$67 \times 134$	0.1141	$66 \times 134$	0.0633	0.507
Cage(4,9)	161	90	$270 \times 540$	1.2146	$269 \times 540$	0.6059	0.502
Cage(6,4)	12	1036800	$12 \times 36$	0.1921	$11 \times 36$	0.0026	0.694
Cage(7,5)	50	252000	$50 \times 175$	0.1975	$49 \times 175$	0.0114	0.72

Tabuľka 6.1: Tabuľka výstupov graficky zobraziteľných klietok a rekordných grafov v CoCalc

V tabuľke výstupov graficky zobraziteľných klietok a rekordných grafov (Tabuľka 6.1) získanej zo skriptov v CoCalc [18] popisujeme výsledky Cage(k,g).

Na získanie výsledkov bolo potrebné spustiť štyri skripty. Skript *moore-BoundsCageValidation* na zistenie Moorového ohraničenia M(k,g) pre uvažované klietky Cage(k,g) ešte predtým, ako budú zostrojené. Skript *generateIncidenceMatrices* na vygenerovanie incidenčných matíc L a ich uloženie do textových súborov, skript *generateParityCheckMatrices* na vygenerovanie kontrolných matíc H a ich uloženie do textových súborov a skript *cages-Data* na získanie informácií o veľkosti gruppy automorfizmov grafu AutGroup(Cage), rozmeru incidenčných matíc L a rozmeru kontrolných matíc H. Vo všetkých prípadoch bola splnená podmienka Moorového ohraničenia M(k,g) zaručujúca existenciu danej Cage(k,g) alebo Rec(k,g), pretože majú

rovnajúci sa, nanajvýš väčší počet vrcholov  $|ver|$  ako  $M(k,g)$ . Najväčší počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má klietka Cage(6,4) s hodnotou 1036800 a najmenší  $|\text{AutGroup}(\text{Cage})|$  má klietka Cage(4,7) s hodnotou 4. Najväčšia incidenčná matica  $L$  bola vygenerovaná z rekordného grafu Rec(3,14) s rozmerom  $384 \times 576$ , najmenšia  $L$  bola z klietky Cage(3,5) s rozmerom  $10 \times 15$ . Časovú náročnosť generovania a uloženia incidenčných matíc  $t(L)$  popisujeme na obrázku (Obr. 6.13).

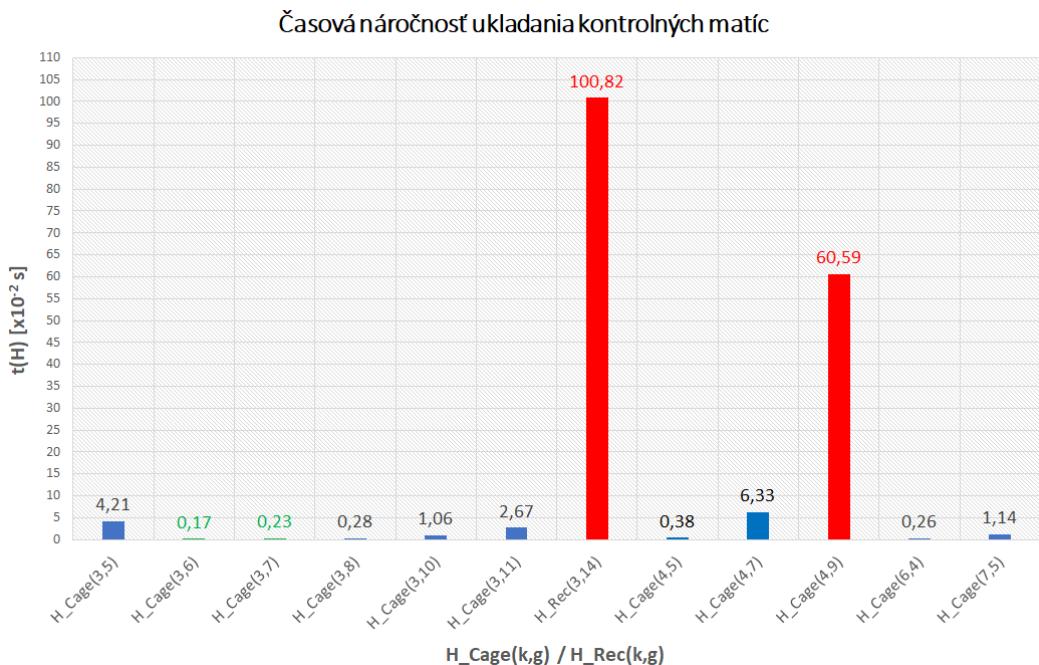


Obr. 6.13: Časová náročnosť ukladania incidenčných matíc z CoCalc č. 1

Na obrázku časová náročnosť ukladania incidenčných matíc z CoCalc č. 1 (Obr. 6.13) zobrazujeme časovú náročnosť  $t(L)$  generovania a ukladania  $L$  graficky zobraziteľných klietok a rekordných grafov. Najdlhšie sa generovala a ukladala  $L$  z  $\text{Rec}(3,14)$ , a to 2,0035 s. Druhá najdlšie generovaná a ukladaná  $L$  bola z  $\text{Cage}(4,5)$ , a to 1,9171 s. Naopak najkratšie bola generovaná a

ukladaná L z klietky Cage(3,6), a to 0,0053 s. Druhá najkratšie generovaná a ukladaná bola L z klietky Cage(3,8), a to 0,0073 s.

Najväčšia kontrolná matica H bola vygenerovaná z incidenčnej matice rekordného grafu Rec(3,14) s rozmerom  $383 \times 576$ , najmenšia H bola vygenerovaná z incidenčnej matice klietky Cage(3,5) s rozmerom  $9 \times 15$ . Časovú náročnosť generovania a uloženia kontrolných matíc  $t(H)$  popisujeme na obrázku (Obr. 6.14).



Obr. 6.14: Časová náročnosť ukladania kontrolných matíc z CoCalc č. 1

Na obrázku časová náročnosť ukladania kontrolných matíc z CoCalc č. 1 (Obr. 6.14) zobrazujeme časovú náročnosť  $t(H)$  generovania a ukladania H graficky zobraziteľných klietok a rekordných grafov. Najdlhšie sa generovala a ukladala H z incidenčnej matice grafu Rec(3,14), a to 1,0082 s. Druhá

najdlšie generovaná a ukladaná H bola z incidenčnej matice grafu Cage(4,9), a to 0,006059 s. Naopak najkratšie bola generovaná a ukladaná H z incidenčnej matice klietky Cage(3,6), a to 0,0017 s. Druhá najkratšie generovaná a ukladaná bola H z incidenčnej matice klietky Cage(3,7), a to 0,0023 s.

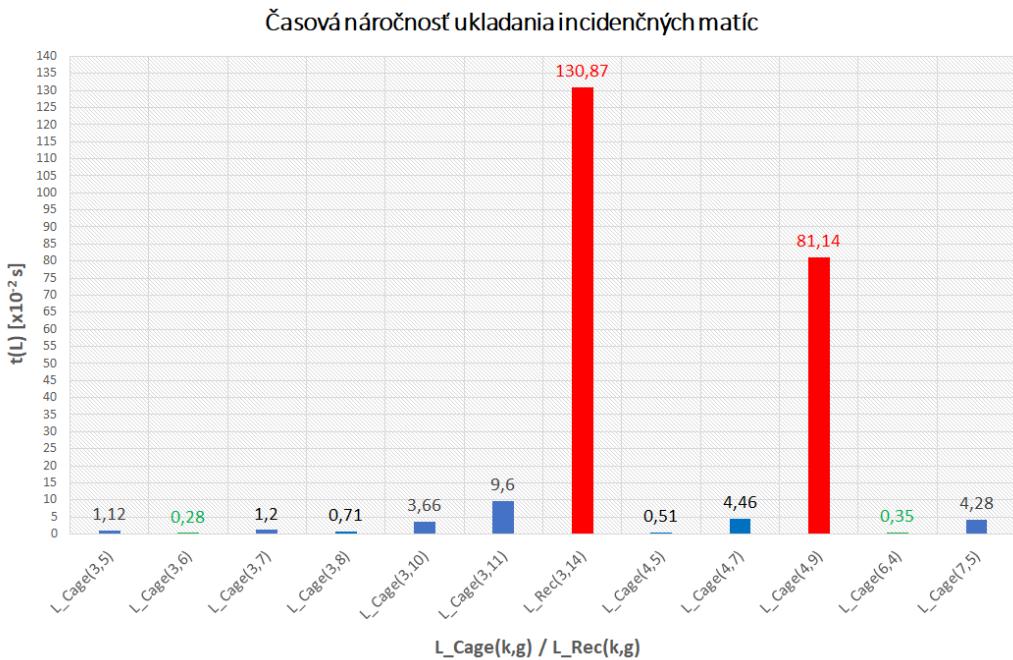
### Výstupy konzolovej aplikácie

Cage/Rec	M	A(gp)	L	t(L)s	H	t(H)s	R
Cage(3,5)	10	120	$10 \times 15$	0.0112	$9 \times 15$	0.013	0.4
Cage(3,6)	14	336	$14 \times 21$	0.0028	$13 \times 21$	0.005	0.381
Cage(3,7)	22	32	$24 \times 36$	0.012	$23 \times 36$	0.0087	0.361
Cage(3,8)	30	1440	$30 \times 45$	0.0071	$29 \times 45$	0.0104	0.355
Cage(3,10)	62	80	$70 \times 105$	0.0366	$69 \times 105$	0.0555	0.343
Cage(3,11)	94	64	$112 \times 168$	0.096	$111 \times 168$	0.1259	0.339
Rec(3,14)	254	96	$384 \times 576$	1.3087	$383 \times 576$	1.588	0.335
Cage(4,5)	17	24	$19 \times 38$	0.0051	$18 \times 38$	0.0078	0.526
Cage(4,7)	53	4	$67 \times 134$	0.0446	$66 \times 134$	0.0628	0.507
Cage(4,9)	161	90	$270 \times 540$	0.8114	$269 \times 540$	1.041	0.502
Cage(6,4)	12	1036800	$12 \times 36$	0.0035	$11 \times 36$	0.0091	0.694
Cage(7,5)	50	252000	$50 \times 175$	0.0428	$49 \times 175$	0.0585	0.72

Tabuľka 6.2: Tabuľka výstupov graficky zobraziteľných klietok a rekordných grafov v konzolovej aplikácii

V tabuľke výstupov graficky zobraziteľných klietok a rekordných grafov konzolovej aplikácie (Tabuľka 6.2) popisujeme výsledky Cage(k,g) a Rec(k,g). Vo všetkých prípadoch bola splnená podmienka Moorového ohraničenia  $M(k,g)$  zaručujúca existenciu danej Cage(k,g) alebo Rec(k,g), pretože majú rovnajúci sa, nanajvýš väčší počet vrcholov  $|ver|$  ako  $M(k,g)$ . Najväčší počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má klietka Cage(6,4) s hodnotou 1036800 a najmenší  $|\text{AutGroup}(\text{Cage})|$  má klietka Cage(4,7) s hodnotou 4. Najväčšia incidenčná matica L bola vygenerovaná z rekordného grafu Rec(3,14) s roz-

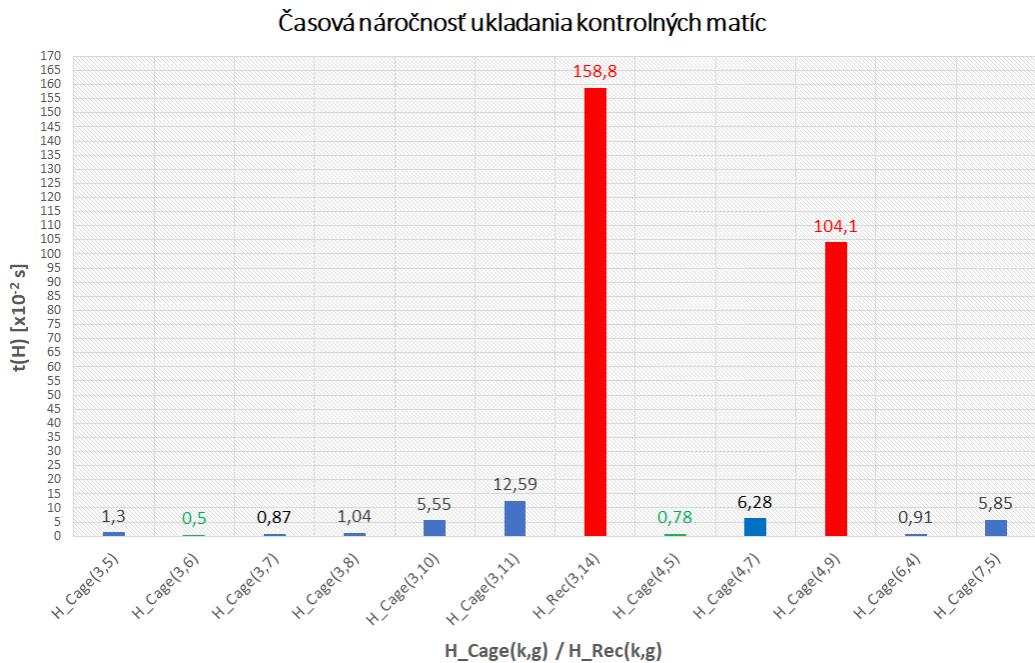
merom  $384 \times 576$ , najmenšia L bola z klietky Cage(3,5) s rozmerom  $10 \times 15$ . Časovú náročnosť generovania a uloženia incidenčných matíc  $t(L)$  popisujeme na obrázku (Obr. 6.15).



Obr. 6.15: Časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 1

Na obrázku časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 1 (Obr. 6.15) zobrazujeme časovú náročnosť  $t(L)$  generovania a ukladania L graficky zobraziteľných klietok a rekordných grafov. Najdlhšie sa generovala a ukladala L z Rec(3,14), a to 1,3087 s. Druhá najdlhšie generovaná a ukladaná L bola z Cage(4,9), a to 0,8114 s. Naopak najkratšie bola generovaná a ukladaná L z klietky Cage(3,6), a to 0,0028 s. Druhá najkratšie generovaná a ukladaná bola L z klietky Cage(6,4), a to 0,0035 s.

Najväčšia kontrolná matica H bola vygenerovaná z incidenčnej matice L re-kordného grafu Rec(3,14) a to  $383 \times 576$ , najmenšia H bola z incidenčnej matice klietky Cage(3,5) s hodnotou  $9 \times 15$ . Časovú náročnosť generovania a uloženia kontrolných matíc t(H) popisujeme na obrázku (Obr. 6.16).



Obr. 6.16: Časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 1

Na obrázku časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 1 (Obr. 6.16) zobrazujeme časovú náročnosť t(H) generovania a ukladania H z incidenčných matíc L graficky zobraziteľných grafov. Najdlhšie sa generovala a ukladala H z incidenčnej matice grafu Rec(3,14), a to 1,5888 s, druhá H v poradi bola z incidenčnej matice L grafu Cage(4,9), a to 1,041 s. Najkratšie bola generovaná a ukladaná H z incidenčnej matice klietky Cage(3,6), a to 0,005 s. Druhá najkratšie generovaná a ukladaná bola H z incidenčnej matice L klietky Cage(4,5), a to 0,0078 s.

### 6.1.2 Graficky nezobraziteľné klietky a rekordné grafy

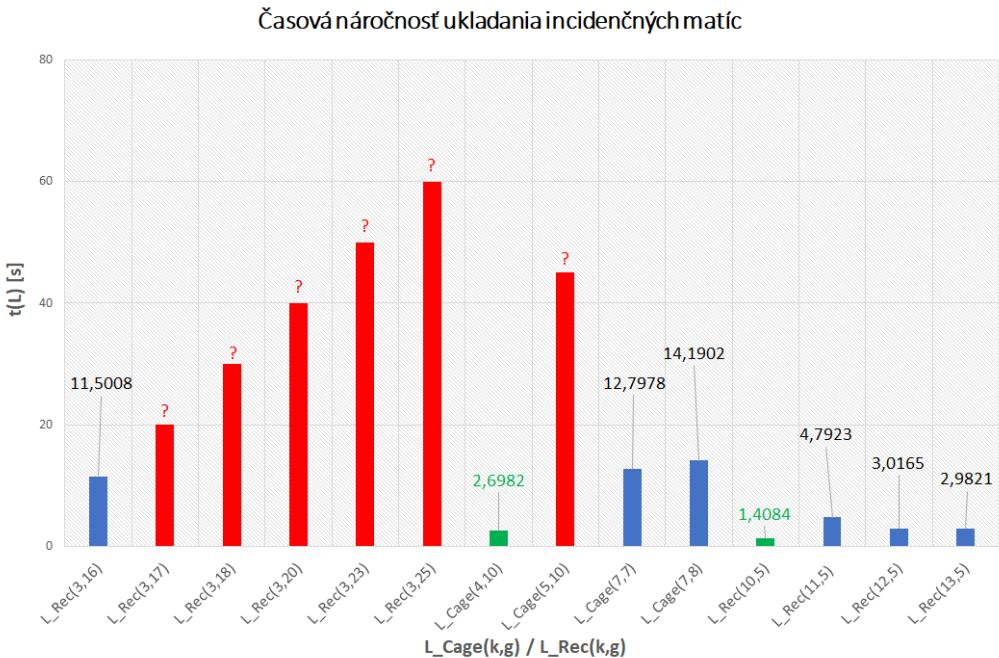
#### Výstupy riešenia v CoCalc

Cage/Rec	M	A(gp)	L	t(L)s	H	t(H)s	R
Rec(3,16)	510	96	960 × 1440	11.5008	959 × 1440	13.5084	0.334
Rec(3,17)	766	?	?	?	?	?	?
Rec(3,18)	1022	?	?	?	?	?	?
Rec(3,20)	2046	?	?	?	?	?	?
Rec(3,23)	6142	?	?	?	?	?	?
Rec(3,25)	12286	?	?	?	?	?	?
Cage(4,10)	242	768	384× 768	2.6982	383× 768	3.0913	0.501
Cage(5,10)	682	3888	?	?	?	?	?
Cage(7,7)	302	320	640 × 2240	12.7978	639 × 2240	11.5972	0.715
Cage(7,8)	518	14112	672 × 2352	14.1902	671 × 2352	16.6917	0.715
Rec(10,5)	101	1	124 × 620	1.4084	123 × 620	0.5767	0.802
Rec(11,5)	122	1	154 × 847	4.7923	153 × 847	1.2235	0.819
Rec(12,5)	145	203	203 × 1218	3.0165	202 × 1218	4.1971	0.834
Rec(13,5)	170	1	230 × 1495	2.9821	229 × 1495	3.2897	0.847

Tabuľka 6.3: Tabuľka výstupov graficky nezobraziteľných klietok a rekordných grafov v CoCalc

V tabuľke výstupov graficky nezobraziteľných klietok a rekordných grafov získanej zo skriptov v CoCalc (Tabuľka 6.3) popisujeme výsledky Cage(k,g) a Rec(k,g). Na získanie výsledkov bolo potrebné spustiť štyri skripty. Skript *mooreBoundsCageValidation* na zistenie Moorového ohraničenia M(k,g) pre

uvažované klietky  $\text{Cage}(k,g)$  ešte predtým, ako budú zstrojené. Skript *generateIncidenceMatrices* na vygenerovanie incidenčných matíc L a ich uloženie do textových súborov, skript *generateParityCheckMatrices* na vygenerovanie kontrolných matíc H a ich uloženie do textových súborov a skript *cagesData* na získanie informácií o veľkosti grupy automorfizmov grafov  $\text{AutGroup}(\text{Cage})$ , rozmeroch incidenčných matíc L, rozmeroch kontrolných matíc H a o informačných pomeroch R. Vo všetkých prípadoch bola splnená podmienka Moorovho ohraničenia  $M(k,g)$  zaručujúca existenciu danej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ , pretože majú rovnajúci sa, nanajvýš väčší počet vrcholov  $|ver|$  ako  $M(k,g)$ . Najväčší počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má klietka  $\text{Cage}(7,8)$  s hodnotou 14112 a najmenší  $|\text{AutGroup}(\text{Cage})|$  majú  $\text{Rec}(10,5)$ ,  $\text{Rec}(11,5)$  a  $\text{Rec}(13,5)$ , všetky majú len jeden. Počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$ , rozmery incidenčných L a kontrolných matíc H ako aj informačný pomer R sa nám nepodarilo získať pre rekordné grafy  $\text{Cage}(5,10)$ ,  $\text{Rec}(3,17)$ ,  $\text{Rec}(3,18)$ ,  $\text{Rec}(3,20)$ ,  $\text{Rec}(3,23)$  a  $\text{Rec}(3,25)$ , pretože pre skript *generateIncidenceMatrices* bolo príliš náročné incidenčné matice L vygenerovať a uložiť do textového súboru. Kontrolné matice H bez vstupných súborov nebolo tým pádom tiež možné vygenerovať a uložiť a bez vstupných textových súborov nebolo ani skriptom *cagesData* možné získať potrebné informácie. Najväčšia L bola vygenerovaná z klietky  $\text{Cage}(7,8)$  s rozmerom  $672 \times 2352$ , najmenšia L z klietky  $\text{Rec}(10,5)$  s rozmerom  $124 \times 620$ . Časovú náročnosť generovania a uloženia incidenčných matíc  $t(L)$  popisujeme na obrázku (Obr. 6.17).

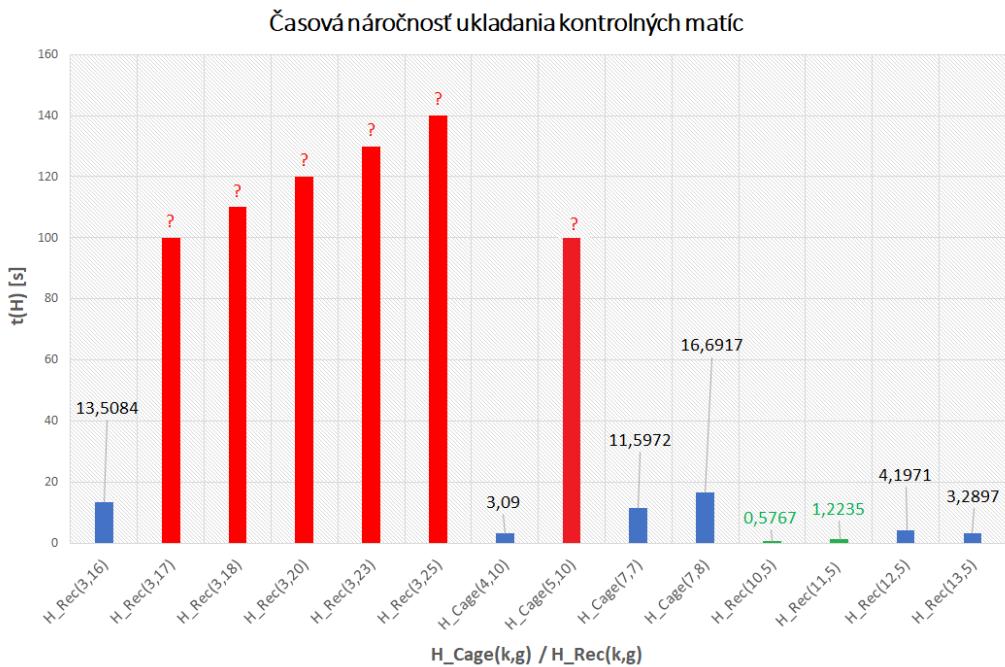


Obr. 6.17: Časová náročnosť ukladania kontrolných matíc z CoCalc č. 2

Na obrázku časová náročnosť ukladania incidenčných matíc z CoCalc č. 2 (Obr. 6.17) zobrazujeme časovú náročnosť generovania a ukladania incidenčných matíc graficky nezobraziteľných Cage( $k,g$ ) a Rec( $k,g$ ). Najdlhšie sa generovala a ukladala  $L$  z Cage(7,8), a to 14,1902 s. Klietku Cage(5,10) a rekordné grafy Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23) a Rec(3,25) bolo príliš náročné vygenerovať a uložiť, preto ich hodnoty neuvádzame. Najkratšie bola generovaná a ukladaná  $L$  z rekordného grafu Rec(10,5), a to 1,4084 s. Druhá najkratšie generovaná a ukladaná bola  $L$  z klietky Cage(4,10), a to 2,6982 s.

Najväčšia  $H$  bola vygenerovaná z incidenčnej matice klietky Cage(7,8), a to  $671 \times 2352$ , najmenšia  $H$  z incidenčnej matice klietky Rec(10,5), a to

$123 \times 620$ . Časovú náročnosť generovania a uloženia kontrolných matíc  $t(H)$  popisujeme na obrázku (Obr. 6.18).



Obr. 6.18: Časová náročnosť ukladania kontrolných matíc z CoCalc č. 2

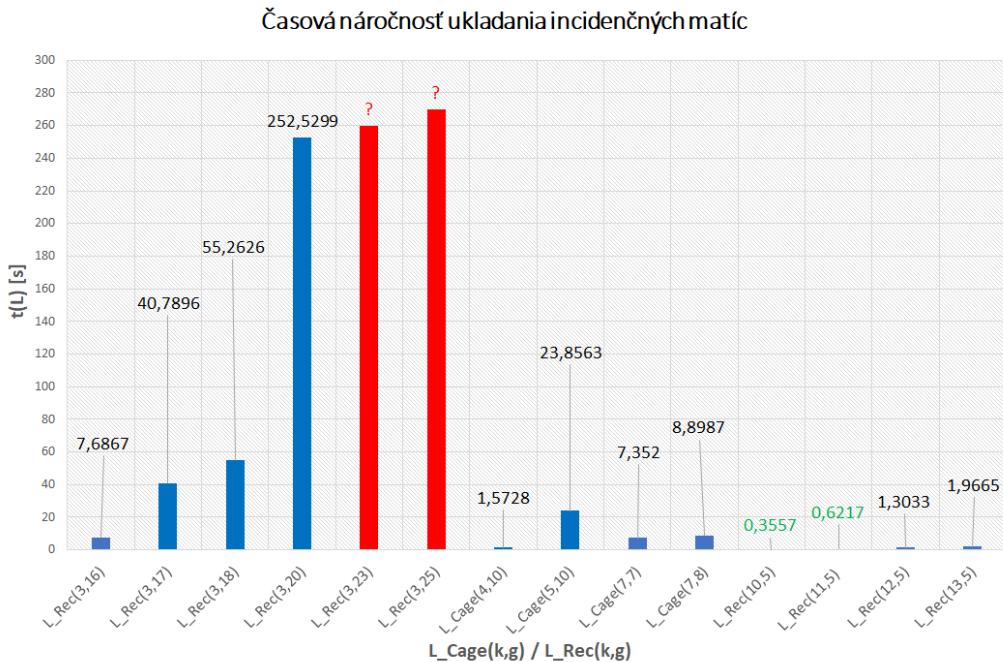
Na obrázku časová náročnosť ukladania kontrolných matíc z CoCalc č. 2 (Obr. 6.18) zobrazujeme časovú náročnosť generovania a ukladania kontrolných matíc  $H$  získaných z incidenčných matíc graficky nezobraziteľných  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$ . Najdlhšie sa generovala a ukladala  $H$  z  $\text{Cage}(7,8)$ , a to 16,6917 s. Rekordné grafy  $\text{Rec}(3,17)$ ,  $\text{Rec}(3,18)$ ,  $\text{Rec}(3,20)$ ,  $\text{Rec}(3,23)$  a  $\text{Rec}(3,25)$  spolu s klietokou  $\text{Cage}(5,10)$  bolo príliš náročné vygenerovať a uložiť. Najkratšie bola generovaná a ukladaná  $H$  z rekordného grafu  $\text{Rec}(10,5)$ , a to 0,5767 s. Druhá najkratšie generovaná a ukladaná bola  $H$  z rekordného grafu  $\text{Rec}(11,5)$ , a to 1,2235 s. Môžeme si všimnúť, že klietky, z ktorých incidenčných matíc  $L$  trvalo generovanie a ukladanie  $H$  najmenej, majú najmenšiu grupu automorfizmov  $\text{AutGroup}(\text{Cage})$ .

## Výstupy konzolovej aplikácie

Cage/Rec	M	A	L	t(L)s	H	t(H)s	R
Rec(3,16)	510	96	960 × 1440	7.6867	959 × 1440	9.9698	0.334
Rec(3,17)	766	544	2176 × 3264	40.7896	2175 × 3264	50.8098	0.334
Rec(3,18)	1022	640	2560 × 3840	55.2626	2559 × 3840	71.05567	0.334
Rec(3,20)	2046	2688	5376 × 8064	252.5299	5375 × 8064	338.2145	0.333
Rec(3,23)	6142	1	49326 × 73989	?	49325 × 73989	?	0.333
Rec(3,25)	12286	1	108906 × 163359	?	108905 × 163359	?	0.333
Cage(4,10)	242	768	384× 768	1.5728	383× 768	1.8724	0.501
Cage(5,10)	682	3888	1296 × 3240	23.8563	1295 × 3240	30.3811	0.6
Cage(7,7)	302	320	640 × 2240	7.352	639 × 2240	9.5586	0.715
Cage(7,8)	518	14112	672 × 2352	8.8987	671 × 2352	10.867	0.715
Rec(10,5)	101	1	124 × 620	0.3557	123 × 620	0.4498	0.802
Rec(11,5)	122	1	154 × 847	0.6217	153 × 847	0.8426	0.819
Rec(12,5)	145	203	203 × 1218	1.3033	202 × 1218	1.6945	0.834
Rec(13,5)	170	1	230 × 1495	1.9665	229 × 1495	2.3199	0.847

Tabuľka 6.4: Tabuľka výstupov graficky nezobraziteľných klietok a rekordných grafov v konzolovej aplikácii

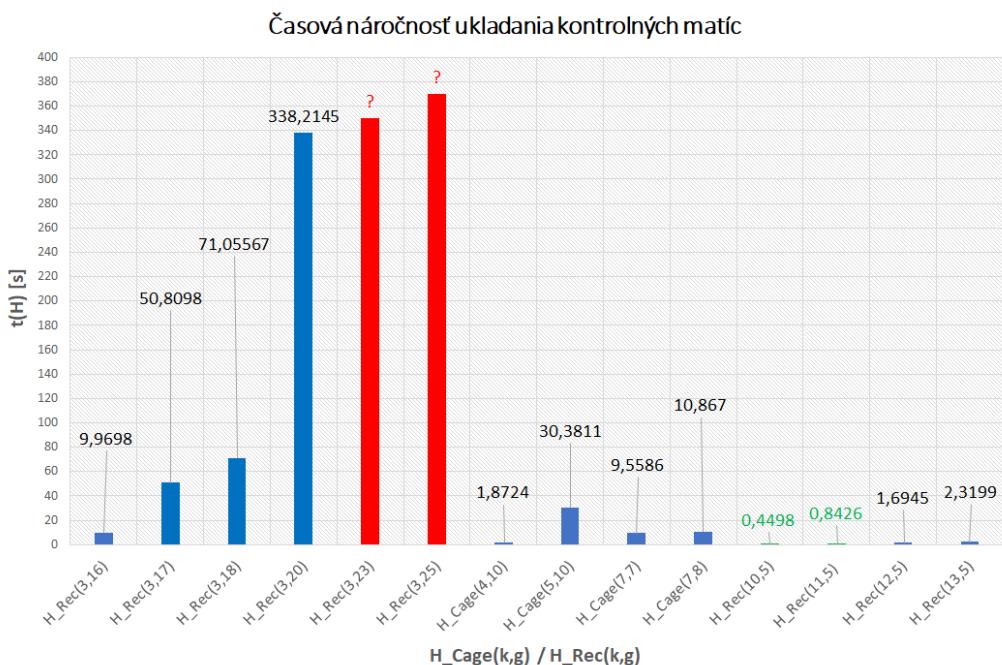
V tabuľke výstupov graficky nezobraziteľných klietok a rekordných grafov konzolovej aplikácie (Tabuľka 6.4) popisujeme výsledky  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$ . Vo všetkých prípadoch bola splnená podmienka Moorového ohraničenia  $M(k,g)$  zaručujúca existenciu danej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ , pretože majú rovnajúci sa, nanajvýš väčší počet vrcholov  $|ver|$  ako  $M(k,g)$ . Najväčší počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  má klietka  $\text{Cage}(7,8)$  s hodnotou 14112 a najmenší  $|\text{AutGroup}(\text{Cage})|$  majú  $\text{Rec}(3,23)$ ,  $\text{Rec}(3,25)$ ,  $\text{Rec}(10,5)$ ,  $\text{Rec}(11,5)$  a  $\text{Rec}(13,5)$ , všetky majú len jeden. Najväčšia L bola vygenerovaná z rekordného grafu  $\text{Rec}(3,25)$  s rozmerom  $108906 \times 163359$ , najmenšia H z klietky  $\text{Rec}(10,5)$  s rozmerom  $124 \times 620$ . Rozmery incidenčných matíc L rekordných grafov  $\text{Rec}(3,23)$  a  $\text{Rec}(3,25)$  sa nám súčasť podarilo vygenerovať, ale nepodarilo sa nám matice uložiť. Časovú náročnosť generovania a uloženia incidenčných matíc t(L) popisujeme na obrázku (Obr. 6.19).



Obr. 6.19: Časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 2

Na obrázku časová náročnosť ukladania incidenčných matíc z konzolovej aplikácie č. 2 (Obr. 6.19) zobrazujeme časovú náročnosť generovania a ukladania incidenčných matíc L graficky nezobraziteľných Cage(k,g) a Rec(k,g). Najdlhšie sa generovala a ukladala L z Rec(3,20), a to 252,5299 s. Rekordné grafy Rec(3,23) a (3,25) bolo príliš náročné vygenerovať a uložiť, preto ich časové hodnoty kontrolných matíc H neuvádzame. Najkratšie bola generovaná a ukladaná L z rekordného grafu Rec(10,5), a to 0,3557 s. Druhá najkratšie generovaná a ukladaná bola L z rekordného grafu Rec(11,5), a to 0,6217 s. Opäť si môžeme všimnúť, že klietky Cage(k,g), z ktorých generovanie a ukladanie L trvalo najmenej, majú najmenšiu grupu automorfizmov AutGroup(Cage).

Najväčšia H bola vygenerovaná a uložená z rekordného grafu Rec(3,20) s rozmerom  $5375 \times 8064$ , najmenšia H z klietky Rec(10,5), a to  $123 \times 620$ . Rozmery kontrolných matíc H z incidenčných matíc L rekordných grafov Rec(3,23) a Rec(3,25) sa nám sice podarilo vygenerovať, ale nepodarilo sa nám matice uložiť. Časovú náročnosť generovania a uloženia kontrolných matíc  $t(H)$  popisujeme na obrázku (Obr. 6.20).



Obr. 6.20: Časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 2

Na obrázku časová náročnosť ukladania kontrolných matíc z konzolovej aplikácie č. 2 (Obr. 6.20) zobrazujeme časovú náročnosť generovania a ukladania kontrolných matíc H z incidenčných matíc L graficky nezobraziteľných Cage(k,g) a Rec(k,g). Najdlhšie sa generovala a ukladala H z incidenčnej matice Rec(3,20), a to 338,2145 s. Incidenčné matice L rekordných grafov

$\text{Rec}(3,23)$  a  $(3,25)$  bolo príliš náročné vygenerovať a uložiť, preto neuvádzame ani hodnoty kontrolných matíc  $H$ . Najkratšie bola generovaná a ukladaná  $H$  z rekordného grafu  $\text{Rec}(10,5)$ , a to 0,4498 s. Druhá najkratšie generovaná a ukladaná bola  $H$  z rekordného grafu  $\text{Rec}(11,5)$ , a to 0,8426 s. Opäť si môžeme všimnúť, že klietky  $\text{Cage}(k,g)$ , z ktorých generovanie a ukladanie  $H$  trvalo najmenej, majú najmenšiu grupu automorfizmov  $\text{AutGroup}(\text{Cage})$ .

## 6.2 Konštrukcia lineárneho kódu z klietky a reordného grafu

Pre každú klietku  $\text{Cage}(k,g)$  a rekordný graf  $\text{Rec}(k,g)$  uvádzame parameter  $r$  vyjadrujúci počet lineárne nezávislých vektorov, lineárny kód  $C(n,m,d)$ , počet automorfizmov  $|\text{AutGroup}(C)|$  získaný z lineárneho kódu  $C(n,m,d)$ , parameter perfektného kódu  $p(n,m,d)$  a optimálneho kódu  $o(n,m,d)$ , rozmer generujúcej matice  $G$  a čas potrebný na jej vygenerovanie a uloženie  $t(G)$ . Parameter  $r$  nám slúži na výpočet maximálneho počtu slov  $m$ . Lineárny kód  $C(n,m,d)$  je definovaný okrem maximálneho počtu slov  $m$  aj jeho dĺžkou  $n$  a minimálnou Hammingovou vzdialenosťou  $d$ , ktoré vieme získať. Parameter  $p(n,m,d)$  vieme vypočítať a určiť, do akej miery je kód perfektný. Parameter  $o(n,m,d)$  vieme vypočítať pomocou tabuľkových hodnôt optimálnych lineárnych kódov. Na základe  $r$  a  $n$  vieme určiť rozmer matice  $G$  a  $t(G)$  potrebný na vygenerovanie a uloženie  $G$ . Výpočet niektorých parametrov je príliš náročný, preto budeme rozlišovať lineárne kódy so všetkými uvažovanými parametrami, lineárne kódy bez parametra optimálneho kódu, lineárne kódy bez minimálnej Hammingovej vzdialenosťi  $d$ , parametrov perfektného a optimálneho kódu, lineárne kódy bez parametrov.

### 6.2.1 Lineárne kódy so všetkými uvažovanými parametrami

#### Výstupy riešenia v CoCalc

Cage	r	$C(n,m,d)$	$p(C)$	$o(C)$	$ A(C) $	G	$t(G)s$
Cage(3,5)	6	$C(15,64,5)$	0.236	0.25	120	$6 \times 15$	0.0025
Cage(3,6)	8	$C(21,256,6)$	0.028	0.1	336	$8 \times 21$	0.0023

Tabuľka 6.5: Tabuľka výstupov lineárnych kódov so všetkými uvažovanými parametrami z CoCalc

V tabuľke výstupov lineárnych kódov  $C(n,m,d)$  so všetkými uvažovanými parametrami z CoCalc (Tabuľka 6.5) popisujeme výsledky Cage(k,g), pre ktoré sa nám podarilo určiť všetky parametre. Z Cage(k,g) sme vedeli určiť  $C(n,m,d)$ , rozmery generujúcich matíc G, ako aj samotné matice vytvoriť a uložiť. Na získanie výsledkov bolo potrebné spustiť 2 skripty. Skript *generateGeneratorMatrices* na vygenerovanie generujúcich matíc G a ich uloženie do textových súborov a skript *linearCodesData* na získanie informácií o počte riadkov generujúcej matice G, dĺžke lineárneho kódu n, parametri lineárneho kódu, maximálneho počtu kódových slov m, minimálnej Hammingovej vzdialosti v kóde d, počte automorfizmov a rozmeru generujúcej matice G.

#### Výstupy konzolovej aplikácie

Cage	r	$C(n,m,d)$	$p(C)$	$o(C)$	$ A(C) $	G	$t(G)s$
Cage(3,5)	6	$C(15,64,5)$	0.236	0.25	120	$6 \times 15$	0.0028
Cage(3,6)	8	$C(21,256,6)$	0.028	0.1	336	$8 \times 21$	0.0065

Tabuľka 6.6: Tabuľka výstupov lineárnych kódov so všetkými uvažovanými parametrami z konzolovej aplikácie

V tabuľke výstupov lineárnych kódov  $C(n,m,d)$  so všetkými uvažovanými parametrami z konzolovej aplikácie (Tabuľka 6.6) popisujeme výsledky  $\text{Cage}(k,g)$ , pre ktoré sa nám podarilo určiť všetky parametre. Z  $\text{Cage}(k,g)$  sme vedeli určiť  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto bolo možné určiť z lineárnych kódov  $C(n,m,d)$  počet automorfizmov  $|\text{AutGroup}(C)|$ , ktoré sú zhodné s počtom automorfizmov z príslušných klietok  $|\text{AutGroup}(\text{Cage})|$ . Určili sme aj parameter perfektného  $p(n,m,d)$  a optimálneho kódu  $o(n,m,d)$ , ktoré sme medzi sebou porovnali a zistili mierny rozdiel. Lineárny kód  $C(15,64,5)$  má parameter optimálneho kódu  $o(n,m,d)$  najbližšie k hodnote 1, preto sa javí ako najviac optimálny v porovnaní s kódom  $C(21,256,6)$ .

### 6.2.2 Lineárne kódy bez parametra optimálneho kódu

#### Výstupy riešenia v CoCalc

<b>Cage</b>	<b>r</b>	<b><math>C(n,m,d)</math></b>	<b><math>p(C)</math></b>	<b><math> A(C) </math></b>	<b><math>G</math></b>	<b><math>t(G)s</math></b>
Cage(3,7)	13	$C(36,8192,7)$	0.000931	32	$13 \times 36$	0.0035
Cage(3,8)	16	$C(45,65536,8)$	0.000028	1440	$16 \times 45$	0.0205
Cage(4,5)	20	$C(38,1048576,5)$	0.002831	24	$20 \times 38$	0.005
Cage(4,7)	68	$C(134,29515 \times 10^{16},7)$	$5.436 \times 10^{-15}$	4	$68 \times 134$	0.1072
Cage(6,4)	25	$C(36,33554432,4)$	0.018066	1036800	$25 \times 36$	0.0053

Tabuľka 6.7: Tabuľka výstupov lineárnych kódov bez parametra optimálneho kódu z CoCalc

V tabuľke výstupov lineárnych kódov  $C(n,m,d)$  z CoCalc (Tabuľka 6.7) bez parametra optimálneho kódu popisujeme výsledky  $\text{Cage}(k,g)$ , pre ktoré sa nám podarilo určiť všetky parametre bez  $o(n,m,d)$ . Z  $\text{Cage}(k,g)$  sme vedeli určiť lineárne kódy  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné

matice vytvoriť a uložiť. Na získanie výsledkov bolo potrebné spustiť niekoľko skriptov. Skript *generateGeneratorMatrices* na vygenerovanie generujúcich matíc G a ich uloženie do textových súborov, skript *linearCodesData* na získanie informácií o počte riadkov generujúcej matice G, dĺžke lineárneho kódu n, parametri lineárneho kódu, maximálneho počtu kódových slov m, minimálnej Hammingovej vzdialenosťi v kóde d, počte automorfizmov a rozmeru generujúcej matice G. Pre časovú náročnosť však tento skript nezvládne získať informácie pre úplne všetky klietky Cage(k,g). Použijeme preto skripty, ktoré získavajú údaje jednotlivo. Pomocou skriptu *maxNWordsLinearCodes* získame maximálne počty kódových slov m. Pomocou skriptu *sizeNlinearCodesData* získame dĺžky lineárnych kódov n a rozmery generujúcich matíc G. Parametre lineárnych kódov získame pomocou skriptu *perfectLinearCodes-Parameter*. Počet grúp automorfizmov získame pomocou skriptu *autLinearCodesData*. Skript *minDistanceLinearCodes* nám pomôže získať minimálnu Hammingovu vzdialenosť d.

### Výstupy konzolovej aplikácie

Cage	r	C(n,m,d)	p(C)	A(C)	G	t(G)s
Cage(3,7)	13	C(36,8192,7)	0.000931	32	$13 \times 36$	0.0105
Cage(3,8)	16	C(45,65536,8)	0.000028	1440	$16 \times 45$	0.0172
Cage(4,5)	20	C(38,1048576,5)	0.002831	24	$20 \times 38$	0.0182
Cage(4,7)	68	C(134,29515 $\times$ $10^{16}$ ,7)	5.436 $\times$ $10^{-15}$	4	$68 \times 134$	0.1471
Cage(6,4)	25	C(36,33554432,4)	0.018066	1036800	$25 \times 36$	0.0157

Tabuľka 6.8: Tabuľka výstupov lineárnych kódov bez parametra optimálneho kódu z konzolovej aplikácie

V tabuľke výsledkov lineárnych kódov C(n,m,d) z konzolovej aplikácie (Tabuľka 6.8) bez parametra optimálneho kódu popisujeme výsledky Cage(k,g),

pre ktoré sa nám podarilo určiť všetky parametre bez  $o(n,m,d)$ . Z  $Cage(k,g)$  sme vedeli určiť lineárny kód  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto bolo možné určiť z lineárnych kódov  $C(n,m,d)$  počet automorfizmov  $|AutGroup(C)|$ , ktoré sú zhodné s počtom automorfizmov z príslušných klietok  $|AutGroup(Cage)|$ . Určili sme aj parameter perfektného kódu  $p(n,m,d)$ . Parameter optimálneho kódu  $o(n,m,d)$  nie je možné určiť, pretože pre získané  $C(n,m,d)$  nie sú známe tabuľkové hodnoty optimálnych kódov. Lineárny kód  $C(15,64,5)$  má parameter perfektného kódu  $p(n,m,d)$  najbližšie k hodnote 1, preto sa javí ako najviac perfektný v porovnaní s ostatnými kódmi kategórie lineárne kódy bez  $o(n,m,d)$  a lineárne kódy so všetkými parametrami. Najväčšiu maticu  $G$  s rozmerom  $68 \times 134$ , najväčší počet kódových slov  $m$  s hodnotou  $29515 \times 10^{16}$  a najmenší parameter  $p(n,m,d)$  s rozmerom  $5.436 \times 10^{-15}$  sme získali z klietky  $Cage(4,7)$ . V tabuľke tiež môžeme pozorovať, že hodnota  $p(n,m,d)$  sa znižuje v závislosti od zvyšujúceho sa obvodu  $g$  v klietkach  $Cage(k,g)$ . Takisto môžeme pozorovať, že zistené  $d$  sa zhodujú s hodnotou obvodov klietok  $Cage(k,g)$ .

### 6.2.3 Lineárne kódy bez minimálnej Hammingovej vzdialenosťi, parametrov perfektného a optimálneho kódu

#### Výstupy riešenia v CoCalc

Cage	r	$C(n,m,d)$	$ A(C) $	G	$t(G)_s$
Cage(3,10)	36	$C(105,68719 \times 10^6,?)$	80	$36 \times 105$	0.037
Cage(3,11)	57	$C(168,14412 \times 10^{13},?)$	64	$57 \times 168$	0.1124
Rec(3,14)	193	$C(576,12554 \times 10^{54},?)$	96	$193 \times 576$	1.4845
Rec(3,16)	481	$C(1440,62435 \times 10^{140},?)$	96	$481 \times 1440$	10.0769
Cage(4,9)	271	$C(540,37943 \times 10^{77},?)$	90	$271 \times 540$	1.6189
Cage(4,10)	385	$C(768,78804 \times 10^{111},?)$	768	$385 \times 768$	3.4982
Cage(7,5)	126	$C(175,85071 \times 10^{33},?)$	252000	$126 \times 175$	0.2828
Cage(7,7)	1601	$C(2240,88925 \times 10^{476},?)$	320	$1601 \times 2240$	38.6138
Cage(7,8)	1681	$C(2352,10750 \times 10^{502},?)$	14112	$1681 \times 2352$	60.9157
Rec(10,5)	497	$C(620,40918 \times 10^{144},?)$	1	$497 \times 620$	3.0941
Rec(11,5)	694	$C(847,82190 \times 10^{204},?)$	1	$694 \times 847$	6.4677
Rec(12,5)	1016	$C(1218,70222 \times 10^{301},?)$	203	$1016 \times 1218$	14.0166
Rec(13,5)	1266	$C(1495,12705 \times 10^{377},?)$	1	$1266 \times 1495$	19.7162

Tabuľka 6.9: Tabuľka výstupov lineárnych kódov bez minimálnej Hammingovej vzdialenosťi v kóde, parametrov perfektného a optimálneho kódu z CoCalc

V tabuľke výstupov lineárnych kódov  $C(n,m,d)$  z CoCalc (Tabuľka 6.9) bez minimálnej Hammingovej vzdialenosťi  $d$ , parametrov perfektného  $p(n,m,d)$  a optimálneho  $o(n,m,d)$  kódu popisujeme výsledky  $Cage(k,g)$  a  $Rec(k,g)$ , pre ktoré sa nám podarilo určiť všetky parametre bez  $d$ ,  $o(n,m,d)$  a  $p(n,m,d)$ . Z  $Cage(k,g)$  a  $Rec(k,g)$  sme vedeli určiť lineárne kódy  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto bolo možné určiť z lineárnych kódov počet automorfizmov  $|AutGroup(C)|$ , ktoré sú zhodné s počtom automorfizmov z príslušných klietok  $|AutGroup(Cage)|$ .

Parametre  $o(n,m,d)$  a  $p(n,m,d)$  nie je možné určiť, pretože pre získané lineárne kódy nie sú známe tabuľkové hodnoty optimálnych kódov a  $d$  je príliš náročné. Najväčšiu maticu  $G$  s rozmerom  $1681 \times 2352$  a najväčší počet kódových slov  $m$  s hodnotou  $10750 \times 10^{502}$  sme získali z klietky Cage(7,8).

### Výstupy konzolovej aplikácie

Cage	r	$C(n,m,d)$	$ A(C) $	$G$	$t(G)s$
Cage(3,10)	36	$C(105,68719 \times 10^6,?)$	80	$36 \times 105$	0.0662
Cage(3,11)	57	$C(168,14412 \times 10^{13},?)$	64	$57 \times 168$	0.163
Rec(3,14)	193	$C(576,12554 \times 10^{54},?)$	96	$193 \times 576$	1.9973
Rec(3,16)	481	$C(1440,62435 \times 10^{140},?)$	96	$481 \times 1440$	15.4061
Rec(3,17)	1089	$C(3264,66323 \times 10^{323},?)$	544	$1089 \times 3264$	71.3121
Rec(3,18)	1281	$C(3840,41632 \times 10^{381},?)$	640	$1281 \times 3840$	93.942
Cage(4,9)	271	$C(540,37943 \times 10^{77},?)$	90	$271 \times 540$	2.4518
Cage(4,10)	385	$C(768,78804 \times 10^{111},?)$	768	$385 \times 768$	4.8246
Cage(5,10)	1945	$C(3240,31867 \times 10^{581},?)$	3888	$1945 \times 3240$	108.6707
Cage(7,5)	126	$C(175,85071 \times 10^{33},?)$	252000	$126 \times 175$	0.3401
Cage(7,7)	1601	$C(2240,88925 \times 10^{476},?)$	320	$1601 \times 2240$	56.0774
Cage(7,8)	1681	$C(2352,10750 \times 10^{502},?)$	14112	$1681 \times 2352$	67.2594
Rec(10,5)	497	$C(620,40918 \times 10^{144},?)$	1	$497 \times 620$	4.5527
Rec(11,5)	694	$C(847,82190 \times 10^{204},?)$	1	$694 \times 847$	8.7172
Rec(12,5)	1016	$C(1218,70222 \times 10^{301},?)$	203	$1016 \times 1218$	18.4707
Rec(13,5)	1266	$C(1495,12705 \times 10^{377},?)$	1	$1266 \times 1495$	28.5862

Tabuľka 6.10: Tabuľka výstupov lineárnych kódov bez minimálnej Hammingovej vzdialenosťi v kode, parametrov perfektného a optimálneho kódu z konzolovej aplikácie

V tabuľke výstupov lineárnych kódov  $C(n,m,d)$  z konzolovej aplikácie (Tabuľka 6.10) bez minimálnej Hammingovej vzdialenosťi  $d$ , parametrov perfektného  $p(n,m,d)$  a optimálneho  $o(n,m,d)$  kódu popisujeme výsledky Cage( $k,g$ ) a Rec( $k,g$ ), pre ktoré sa nám podarilo určiť všetky parametre bez  $d$ ,  $o(n,m,d)$

a  $p(n,m,d)$ . Z  $Cage(k,g)$  a  $Rec(k,g)$  sme vedeli určiť lineárne kódy  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto bolo možné určiť z lineárnych kódov  $C(n,m,d)$  počet automorfizmov  $|AutGroup(C)|$ , ktoré sú zhodné s počtom automorfizmov z príslušných klietkov  $|AutGroup(Cage)|$ . Parameter  $o(n,m,d)$  a parameter  $p(n,m,d)$  nie je možné určiť, pretože pre získané lineárne kódy nie sú známe tabuľkové hodnoty optimálnych kódov a d je príliš náročné. Najväčšiu maticu  $G$  s rozmerom  $1945 \times 3240$  a najväčší počet kódových slov  $m$  s hodnotou  $31867 \times 10^{581}$  sme získali z klietky  $Cage(5,10)$ .

#### 6.2.4 Lineárne kódy bez parametrov

##### Výstupy riešenia v CoCalc

Pre  $Cage(5,10)$ ,  $Rec(3,17)$ ,  $Rec(3,18)$ ,  $Rec(3,20)$ ,  $Rec(3,23)$  a  $Rec(3,25)$  sa nám nepodarilo z online aplikácie CoCalc určiť žiadnen parameter. Generovanie informácií bolo výpočtovo príliš náročné pri použití akéhokoľvek skriptu vychádzajúceho zo vstupných generujúcich matíc  $G$ . Z  $Rec(k,g)$  sme nevedeli určiť ani jeden parameter  $C(n,m,d)$ , rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto nebolo možné určiť z  $C(n,m,d)$  počet automorfizmov  $|AutGroup(C)|$ . Bez d nie sme schopní určiť parameter perfektného  $p(n,m,d)$  ani optimálneho kódu  $o(n,m,d)$ .

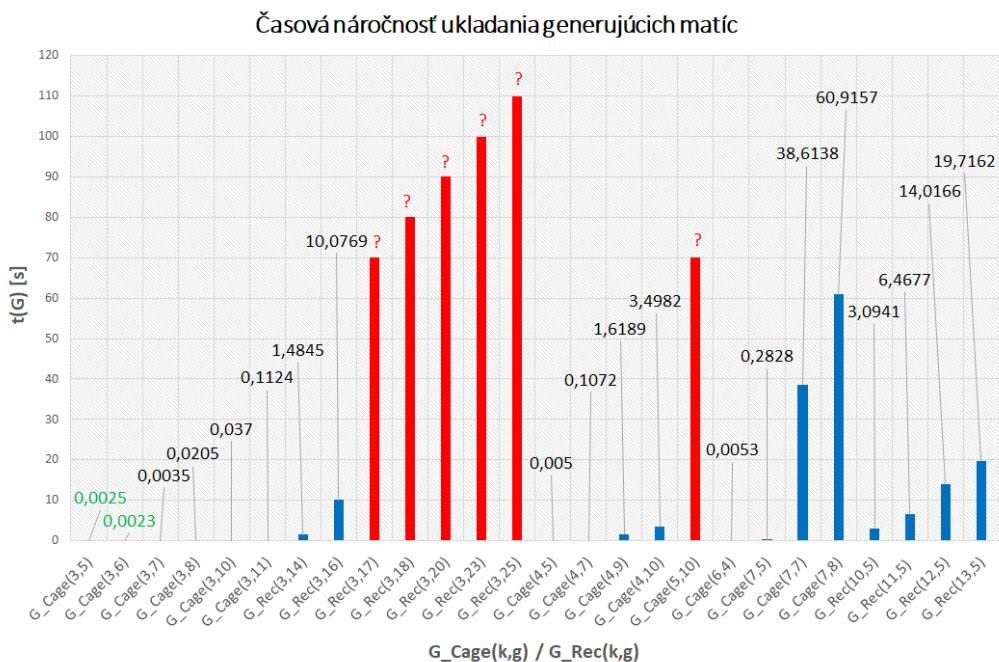
##### Výstupy konzolovej aplikácie

Pre  $Rec(3,20)$ ,  $Rec(3,23)$  a  $Rec(3,25)$  sa nám nepodarilo z konzolovej aplikácie určiť žiadnen parameter, rozmery generujúcich matíc  $G$ , ako aj samotné matice vytvoriť a uložiť. Takisto nebolo možné určiť z  $C(n,m,d)$  počet automorfizmov  $|AutGroup(C)|$ . Bez d nie sme schopní určiť parameter perfekt-

ného  $p(n,m,d)$  ani optimálneho kódu  $o(n,m,d)$ . Určiť všetky parametre je pre naše zariadenie príliš náročné.

### Výstupy riešenia v CoCalc

Časovú náročnosť generovania a uloženia generujúcich matíc  $G$   $t(G)$  z CoCalc popisujeme na obrázku (Obr. 6.21).



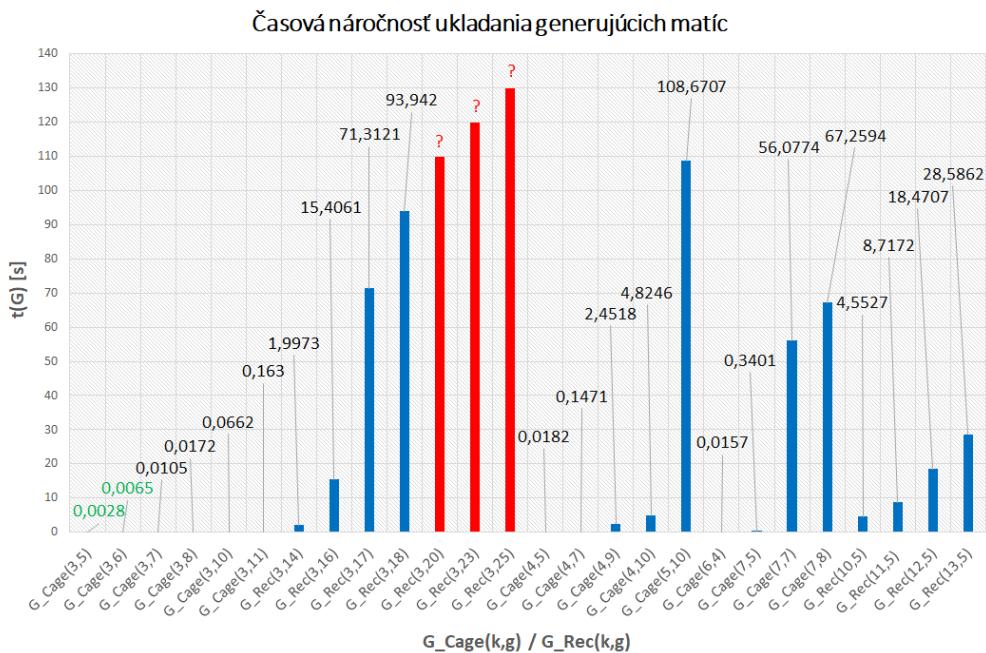
Obr. 6.21: Časová náročnosť ukladania generujúcich matíc z CoCalc

Na obrázku časová náročnosť ukladania generujúcich matíc  $G$  z CoCalc (Obr. 6.21) zobrazujeme časovú náročnosť generovania a ukladania  $G$ . Najdlhšie sa generovala a ukladala  $G$  z klietky Cage(7,8), a to 60,9157 s.  $G$  z rekordných grafov Cage(5,10), Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23) a Rec(3,25)

bolo príliš náročné vygenerovať a uložiť, preto ich  $t(G)$  neuvádzame. Najkratšie bola generovaná a ukladaná  $G$  z klietky Cage(3,6), a to 0,0023 s. Druhá najkratšie generovaná a ukladaná bola  $G$  z klietky Cage(3,5), a to 0,0025 s.

### Výstupy riešenia v konzolovej aplikácii

Časovú náročnosť generovania a uloženia generujúcich matíc  $t(G)$  z konzolovej aplikácie popisujeme na nasledujúcom obrázku (Obr. 6.22).



Obr. 6.22: Časová náročnosť ukladania generujúcich matíc z konzolovej aplikácie

Na obrázku časová náročnosť ukladania generujúcich matíc  $G$  z konzolovej aplikácie (Obr. 6.22) zobrazujeme časovú náročnosť generovania a ukladania  $G$ . Najdlhšie sa generovala a ukladala  $G$  z klietky Cage(5,10), a to 108,6707 s.  $G$  z rekordných grafov Rec(3,20), Rec(3,23) a Rec(3,25) bolo príliš náročné vygenerovať a uložiť, preto ich  $t(G)$  neuvádzame. Najkratšie bola generovaná

a ukladaná G z klietky Cage(3,5), a to 0,0028 s. Druhá najkratšie generovaná a ukladaná bola G z klietky Cage(3,6), a to 0,0065 s.

### 6.3 Konštrukcia grupy automorfizmov z grafu

Pre každú klietku  $\text{Cage}(k,g)$  a rekordný graf  $\text{Rec}(k,g)$  uvádzame počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  získaný z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  a čas potrebný na uloženie  $t(\text{AutG}(\text{Cage}))$ . Z lineárnych kódov  $C(n,m,d)$  vytvorených z klietok a rekordných grafov sme zistili, že veľkosť grupy automorfizmov  $|\text{AutGroup}(C)|$  vygenerovaná z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  je totožná s veľkosťou grupy automorfizmov vygenerovanou z lineárneho kódu  $\text{AutGroup}(C)$ , ktorý sme získali z rovnakej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Tieto  $\text{AutGroup}(\text{Cage})$  ukladáme jednotlivo do textových súborov. Z jednotlivých automorfizmov grupy  $\text{AutGroup}(\text{Cage})$  však vieme zistiť pre každý jeden automorfizmus matice  $L$ ,  $H$ ,  $G$  a porovnať ich s predošlými výsledkami matíc získaných priamo z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Výsledky porovnávame v nasledujúcej tabuľke.

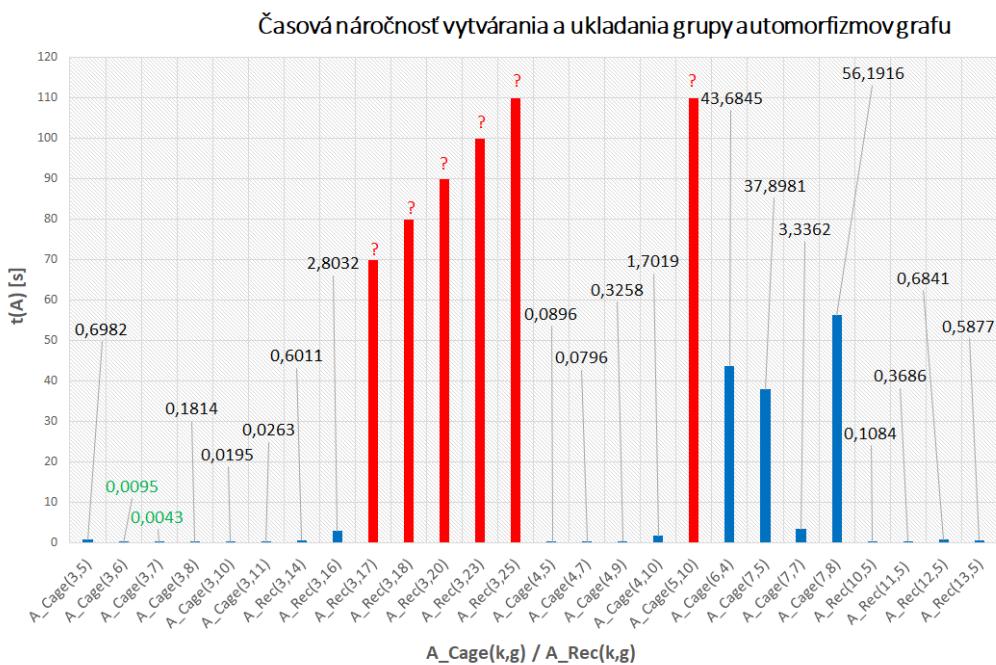
### Výstupy riešenia v CoCalc

Cage	$ \text{AutGroup}(\text{Cage}) $	$t(\mathbf{A})s$
Cage(3,5)	120	0.6982
Cage(3,6)	336	0.0095
Cage(3,7)	32	0.0043
Cage(3,8)	1440	0.1814
Cage(3,10)	80	0.0195
Cage(3,11)	64	0.0263
Rec(3,14)	96	0.6011
Rec(3,16)	96	2.8032
Rec(3,17)	?	?
Rec(3,18)	?	?
Rec(3,20)	?	?
Rec(3,23)	?	?
Rec(3,25)	?	?
Cage(4,5)	24	0.0896
Cage(4,7)	4	0.0796
Cage(4,9)	90	0.3258
Cage(4,10)	768	1.7019
Cage(5,10)	?	?
Cage(6,4)	1036800	43.6845
Cage(7,5)	252000	37.8981
Cage(7,7)	320	3.3362
Cage(7,8)	14112	56.1916
Rec(10,5)	1	0.1084
Rec(11,5)	1	0.3686
Rec(12,5)	203	0.6841
Rec(13,5)	1	0.5877

Tabuľka 6.11: Tabuľka výstupov grúp automorfizmov získaných z klietok alebo rekordných grafov z CoCalc

V tabuľke výstupov grúp automorfizmov získaných z klietok alebo rekordných grafov z CoCalc (Tabuľka 6.11) popisujeme čas generovania a uloženie

nia grupy automorfizmov grafu, ako aj veľkosť samotnej grupy. Pomocou skriptu *generateAutCodesFromCages* vieme získať časy generovania a uloženia jednotlivých grúp automorfizmov. Informácia o počte automorfizmov je už známa z predošlých riešení. Na obrázku (Obr. 6.23) znázorňujeme časovú náročnosť ukladania grúp automorfizmov klietok alebo rekordných grafov z CoCalc.



Obr. 6.23: Časová náročnosť ukladania grúp automorfizmov Cage/Rec z CoCalc

Na obrázku časová náročnosť ukladania grúp automorfizmov klietok alebo rekordných grafov (Obr. 6.23) z CoCalc zobrazujeme časovú náročnosť  $t(A)$ . Z grafov Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23), Rec(3,25) a Cage (5,10) nemáme uložené incidenčné matice, ktoré sú potrebné na vstupe pre skript *generateAutCodesFromCages*, takže pre ne nevieme určiť grupy automorfizmov grafov AutGroup(Cage). Najdlhšie sa generovala a ukladala AutG-

roup(Cage) z klietky Cage(7,8), a to 56,1916 s. Ako druhá sa generovala a ukladala AutGroup(Cage) z klietky Cage(6,4), a to 43,6845 s. Najkratšie bola generovaná a ukladaná AutGroup(Cage) z Cage(3,7), a to 0,0043 s. Druhá najkratšie generovaná a ukladaná bola AutGroup(Cage) z klietky Cage(3,6), a to 0,0095 s.

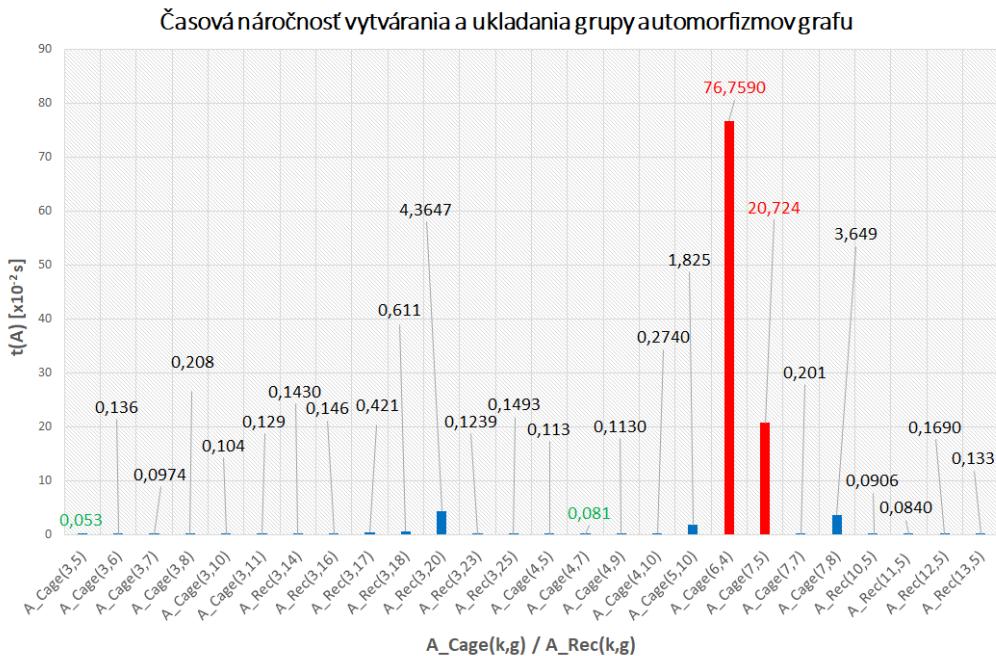
### Výstupy konzolovej aplikácie

Cage	$ \text{AutGroup}(\text{Cage}) $	$t(\mathbf{A})s$
Cage(3,5)	120	0.00053
Cage(3,6)	336	0.00136
Cage(3,7)	32	0.000974
Cage(3,8)	1440	0.00208
Cage(3,10)	80	0.00104
Cage(3,11)	64	0.00129
Rec(3,14)	96	0.00143
Rec(3,16)	96	0.00146
Rec(3,17)	544	0.00421
Rec(3,18)	640	0.00611
Rec(3,20)	2688	0.043647
Rec(3,23)	1	0.001239
Rec(3,25)	1	0.001493
Cage(4,5)	24	0.00113
Cage(4,7)	4	0.00081
Cage(4,9)	90	0.00113
Cage(4,10)	768	0.00274
Cage(5,10)	3888	0.01825
Cage(6,4)	1036800	0.76759
Cage(7,5)	252000	0.20724
Cage(7,7)	320	0.00201
Cage(7,8)	14112	0.03649
Rec(10,5)	1	0.000906
Rec(11,5)	1	0.00084
Rec(12,5)	203	0.00169
Rec(13,5)	1	0.00133

Tabuľka 6.12: Tabuľka výstupov grúp automorfizmov získaných z klietok alebo rekordných grafov z konzolovej aplikácie

Vo všetkých prípadoch majú Cage( $k,g$ ) a Rec( $k,g$ ) totožné matice  $G$  z automorfizmov s  $G$  získanou priamo z lineárneho kódu  $C(n,m,d)$ , ktorý bol

vygenerovaný z totožnej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Rovnako sú totožné aj kontrolné matice  $H$  získané z automorfizmov s kontrolnými maticami  $H$  získanými z incidenčných matíc grafu. V prípade incidenčnej matice  $L$  platí, že  $L$  získané z jednotlivých automorfizmov príslušnej grupy automorfizmov  $\text{AutGroup}(\text{Cage})$  sa medzi sebou líšia, avšak rozmery  $L$  ostávajú zachované pre všetky rovnaké. Tým pádom sa všetky okrem jednej jedinej  $L$  nezhodujú ani s  $L$  získanou priamo z príslušnej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Tieto výsledky neuvádzme v tabuľke, ale je možné si ich porovnať na základe uložených  $L$ ,  $H$ ,  $G$  v textových súboroch a zobrazených matíc  $L$ ,  $H$ ,  $G$  v konzolovej aplikácii, kde je možné pre jednotlivé automorfizmy tieto matice zobraziť. V tabuľke (Tabuľka 6.12) popisujeme časovú náročnosť generovania  $t(A)$  a uloženia  $\text{AutGroup}(\text{Cage})$  získaných z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$   $t(A)$  v konzolovej aplikácii a zobrazíme si ju na obrázku (Obr. 6.24).



Obr. 6.24: Časová náročnosť ukladania grúp automorfizmov Cage/Rec z konzolovej aplikácie

Na obrázku časová náročnosť ukladania grúp automorfizmov Cage/Rec z konzolovej aplikácie (Obr. 6.24) zobrazujeme  $t(A)$ . Najdlhšie sa generovala a ukladala AutGroup(Cage) z klietky Cage(6,4), a to 0,76759 s. Ako druhá sa generovala a ukladala AutGroup(Cage) z klietky Cage(7,5), a to 0,20724 s. Najkratšie bola generovaná a ukladaná AutGroup(Cage) z rekordného grafu Rec(3,5), a to 0,00053 s. Druhá najkratšie generovaná a ukladaná bola AutGroup(Cage) z klietky Cage(4,7), a to 0,00081 s. Môžeme si všimnúť, že Cage(k,g) a Rec(k,g), z ktorých generovanie a ukladanie AutGroup(Cage) trvalo najviac, majú najväčšiu AutGroup(Cage). Generovať lineárne kódy je v našom prípade možné 2 spôsobmi. Prvý spôsob je priamo z klietky a druhý spôsob je z automorfizmu získaného z toho istého grafu. Ak porovnáme lineárne kódy vygenerované z klietky alebo rekordného grafu s lineárnymi kódmi

vygenerovanými z klietkového automorfizmu alebo automorfizmu rekordného grafu tej istej klietky  $\text{Cage}(k,g)$  alebo rekordného grafu  $\text{Rec}(k,g)$ , tak zistíme, že sa jedná o totožné lineárne kódy s rovnakou generujúcou maticou  $G$  a kontrolnou maticou  $H$ , ale rozdielnou incidenčnou maticou  $L$ , avšak rozmery matíc ostávajú takisto zachované. Rozdielnosť incidenčných matíc je spôsobená permutáciou vrcholov.

## 6.4 Konštrukcia grupy automorfizmov z lineárneho kódu

Pre každú klietku  $\text{Cage}(k,g)$  a rekordný graf  $\text{Rec}(k,g)$  uvádzame z nich vygenerovaný lineárny kód, počet automorfizmov  $|\text{AutGroup}(C)|$  získaný z lineárneho kódu a čas potrebný na uloženie grupy automorfizmov  $t(A)$ . Z lineárnych kódov  $C(n,m,d)$  klietok a rekordných grafov sme zistili, že grupa automorfizmov  $\text{AutGroup}(\text{Cage})$  vygenerovaná z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  je totožná s grupou automorfizmov vygenerovanou z lineárneho kódu  $\text{AutGroup}(C)$ , ktorý sme získali z rovnakej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Tým pádom majú aj rovnakú veľkosť  $|\text{AutGroup}(\text{Cage})|$ , avšak lišia sa v tom, aké údaje reprezentujú. Kým  $\text{AutGroup}(\text{Cage})$  sa skladá z permutácií vrcholov,  $\text{AutGroup}(C)$  sa skladá z permutácií hrán.  $\text{AutGroup}(C)$  ukladáme jednotlivo do textových súborov. Výsledné automorfizmy lineárnych kódov je možné porovnať s automorfizmami klietok alebo rekordných grafov. Výsledky zobrazujeme v nasledujúcej tabuľke.

## Výstupy riešenia v CoCalc

Cage	$C(n,m,d)$	$ \text{AutGroup}(C) $	$t(A)s$
Cage(3,5)	$C(15,64,5)$	120	0.000729
Cage(3,6)	$C(21,256,6)$	336	0.001508
Cage(3,7)	$C(36,8192,7)$	32	?
Cage(3,8)	$C(45,65536,8)$	1440	?
Cage(3,10)	$C(105,68719 \times 10^6,?)$	80	?
Cage(3,11)	$C(168,14412 \times 10^{13},?)$	64	?
Rec(3,14)	$C(576,12554 \times 10^{54},?)$	96	?
Rec(3,16)	$C(1440,62435 \times 10^{140},?)$	96	?
Rec(3,17)	?	?	?
Rec(3,18)	?	?	?
Rec(3,20)	?	?	?
Rec(3,23)	?	?	?
Rec(3,25)	?	?	?
Cage(4,5)	$C(38,1048576,5)$	24	?
Cage(4,7)	$C(134,29515 \times 10^{16},7)$	4	?
Cage(4,9)	$C(540,37943 \times 10^{77},?)$	90	?
Cage(4,10)	$C(768,78804 \times 10^{111},?)$	768	?
Cage(5,10)	?	?	?
Cage(6,4)	$C(36,33554432,4)$	1036800	?
Cage(7,5)	$C(175,85071 \times 10^{33},?)$	252000	?
Cage(7,7)	$C(2240,88925 \times 10^{476},?)$	320	?
Cage(7,8)	$C(2352,10750 \times 10^{502},?)$	14112	?
Rec(10,5)	$C(620,40918 \times 10^{144},?)$	1	?
Rec(11,5)	$C(847,82190 \times 10^{204},?)$	1	?
Rec(12,5)	$C(1218,70222 \times 10^{301},?)$	203	?
Rec(13,5)	$C(1495,12705 \times 10^{377},?)$	1	?

Tabuľka 6.13: Tabuľka výstupov grúp automorfizmov získaných z lineárnych kódov v CoCalc

V tabuľke výstupov grúp automorfizmov získaných z lineárnych kódov v CoCalc (Tabuľka 6.13) popisujeme čas generovania a uloženia grupy automor-

fizmov  $\text{AutGroup}(C)$ , ako aj veľkosť samotnej grupy a parametre lineárneho kódu, z ktorého bola grúpa automorfizmov  $\text{AutGroup}(C)$  vygenerovaná. Pomocou skriptu *generateAutGroupLCode* vieme získať časy generovania a uloženia jednotlivých grúp automorfizmov. Informácie o počte automorfizmov a parametroch lineárnych kódov  $C(n,m,d)$  sú už známe z predošlých riešení. Podarilo sa nám vygenerovať grúpy automorfizmov  $\text{AutGroup}(C)$  iba pre 2 lineárne kódy. Pre  $C(15,64,5)$  generovanie a ukladanie trvalo 0,000729 s. a pre  $C(21,256,6)$  generovanie a ukladanie trvalo 0,001508 s. Pre malý počet vygenerovaných a uložených grúp neuvádzame ani obrázok časovej náročnosti ukladania grúp automorfizmov lineárnych kódov  $C(n,m,d)$  z CoCalc.

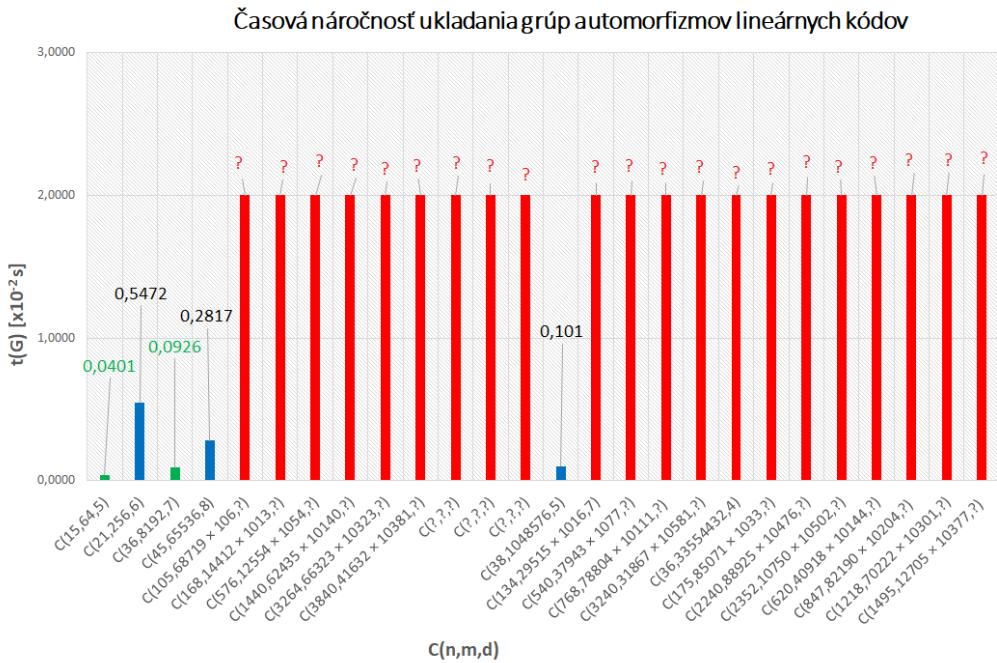
## Výstupy konzolovej aplikácie

Cage	C(n,m,d)	AutGroup(C)	t(A)s
Cage(3,5)	C(15,64,5)	120	0.000401
Cage(3,6)	C(21,256,6)	336	0.005472
Cage(3,7)	C(36,8192,7)	32	0.000926
Cage(3,8)	C(45,65536,8)	1440	0.002817
Cage(3,10)	C( $105,68719 \times 10^6,?$ )	80	?
Cage(3,11)	C( $168,14412 \times 10^{13},?$ )	64	?
Rec(3,14)	C( $576,12554 \times 10^{54},?$ )	96	?
Rec(3,16)	C( $1440,62435 \times 10^{140},?$ )	96	?
Rec(3,17)	C( $3264,66323 \times 10^{323},?$ )	544	?
Rec(3,18)	C( $3840,41632 \times 10^{381},?$ )	640	?
Rec(3,20)	C(?, ?, ?)	2688	?
Rec(3,23)	C(?, ?, ?)	1	?
Rec(3,25)	C(?, ?, ?)	1	?
Cage(4,5)	C(38,1048576,5)	24	0.00101
Cage(4,7)	C( $134,29515 \times 10^{16},7$ )	4	?
Cage(4,9)	C( $540,37943 \times 10^{77},?$ )	90	?
Cage(4,10)	C( $768,78804 \times 10^{111},?$ )	768	?
Cage(5,10)	C( $3240,31867 \times 10^{581},?$ )	3888	?
Cage(6,4)	C(36,33554432,4)	1036800	?
Cage(7,5)	C( $175,85071 \times 10^{33},?$ )	252000	?
Cage(7,7)	C( $2240,88925 \times 10^{476},?$ )	320	?
Cage(7,8)	C( $2352,10750 \times 10^{502},?$ )	14112	?
Rec(10,5)	C( $620,40918 \times 10^{144},?$ )	1	?
Rec(11,5)	C( $847,82190 \times 10^{204},?$ )	1	?
Rec(12,5)	C( $1218,70222 \times 10^{301},?$ )	203	?
Rec(13,5)	C( $1495,12705 \times 10^{377},?$ )	1	?

Tabuľka 6.14: Tabuľka výstupov grúp automorfizmov získaných z lineárnych kódov v konzolovej aplikácii

V tabuľke výsledkov grúp automorfizmov získaných z lineárnych kódov v konzolovej aplikácii (Tabuľka 6.14) popisujeme časovú náročnosť generovania

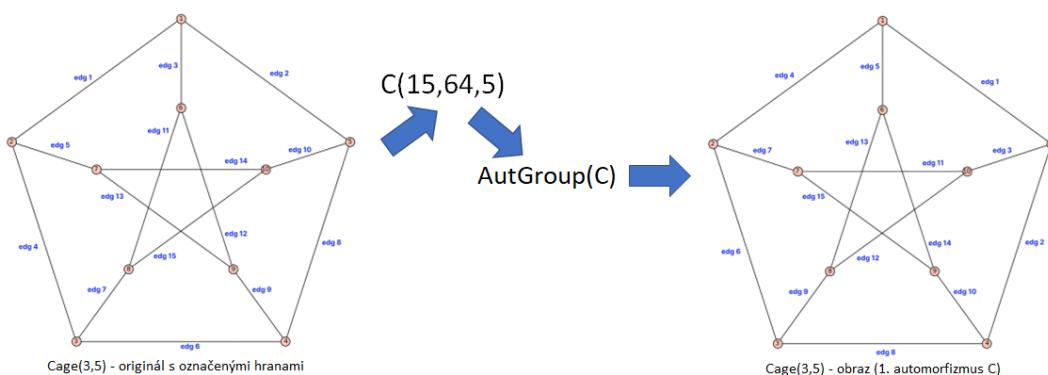
$t(A)$  a uloženia AutGroup(C) lineárnych kódov získaných z Cage(k,g) alebo Rec(k,g)  $t(A)$  a zobrazíme si ju na nasledujúcom obrázku (Obr. 6.25).



Obr. 6.25: Časová náročnosť ukladania grúp automorfizmov lineárnych kódov z konzolovej aplikácie

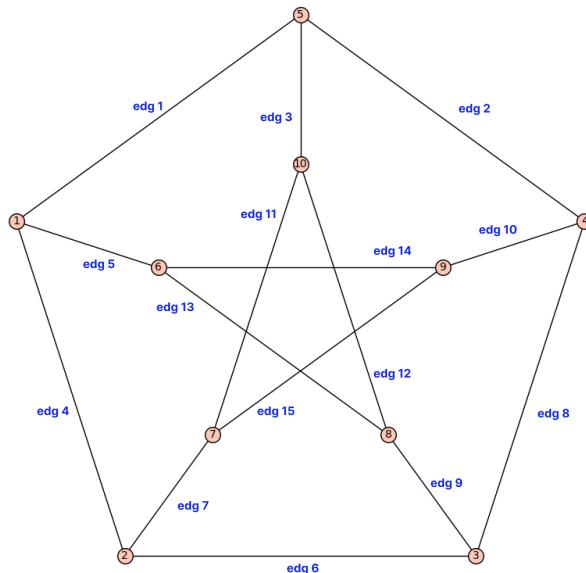
Generovanie výsledných grúp automorfizmov z lineárnych kódov AutGroup(C) v konzolovej aplikácii bolo výpočtovo príliš náročné, pretože sa nám podarilo vygenerovať a uložiť grupy iba pre 5 lineárnych kódov, a to  $C(15,64,5)$ ,  $C(21,256,6)$ ,  $C(36,8192,7)$ ,  $C(45,65536,8)$  a  $C(38,1048576,5)$ . Najkratšie bola generovaná a ukladaná AutGroup(C) z lineárneho kódu  $C(15,64,5)$ , a to 0,000401 s. Druhá najkratšie bola generovaná a ukladaná AutGroup(C) z lineárneho kódu  $C(36,8192,7)$ , a to 0,000926 s. Veľa AutGroup(C) z lineárneho kódu sme nevedeli určiť. Výpočet sa skomplikoval tým, že berieme do úvahy aj vrcholy, medzi ktorými hrana existuje, avšak bez nich nevieme určiť preusporiadanie hrán. Domnievame sa, že automorfizmy z grupy automorfizmov

lineárneho kódu  $\text{AutGroup}(C)$  získaného z klietky alebo rekordného grafu zodpovedajú automorfizmom z grupy automorfizmov samotnej klietky alebo rekordného grafu  $\text{AutGroup}(\text{Cage})$ . Pre overenie tohto tvrdenia sme klietke  $\text{Cage}(3,5)$  pridali označenie hrán a na základe prvého automorfizmu, ktorý sme získali z lineárneho kódu  $C(15,64,5)$  sme nakreslili obraz  $\text{Cage}(3,5)$  (Obr. 6.26).



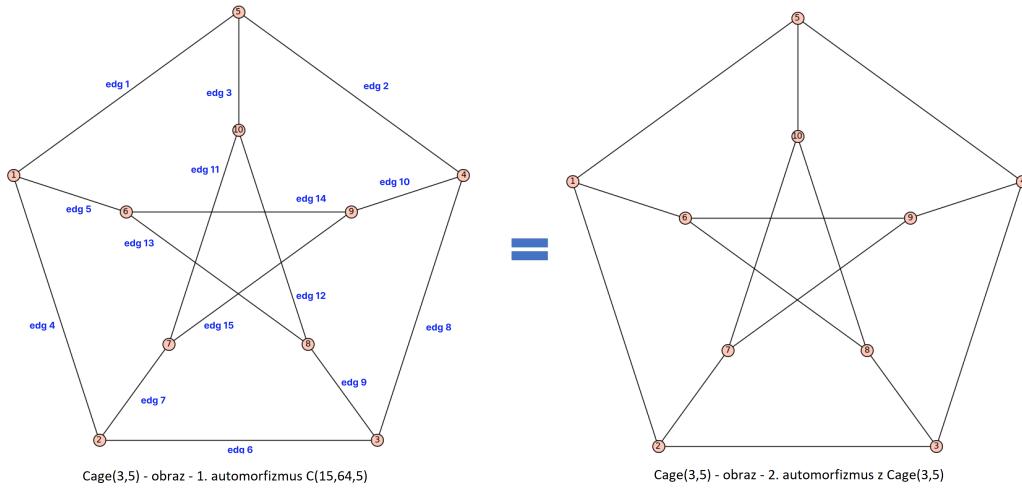
Obr. 6.26: Obraz  $\text{Cage}(3,5)$  zostrojený z prvého automorfizmu  $C(15,64,5)$

Ak by sme získaný obraz  $\text{Cage}(3,5)$  zostrojený z prvého automorfizmu  $C(15,64,5)$  upravili tak, aby sa jednotlivé hrany nachádzali na pôvodných miestach ako má originálny graf  $\text{Cage}(3,5)$ , dostaneme obraz (Obr. 6.27).



Obr. 6.27: Obraz Cage(3,5) zostrojený z prvého automorfizmu  $C(15,64,5)$  s preusporiadanými vrcholmi

Obrazy sme porovnali so získanými obrazmi automorfizmov pre tú istú klietku a zistili sme, že prvý automorfizmus získaný z  $C(15,64,5)$  je totožný s druhým automorfizmom získaným priamo z Cage(3,5) (Obr. 6.28).



Obr. 6.28: Porovnanie obrazov z Cage(3,5)

Grupy automorfizmov grafu a lineárneho kódu  $C(n,m,d)$  reprezentujú rovnaké obrazy. Obe grupy automorfizmov majú rovnakú veľkosť, líšia sa iba v poradí automorfizmov a údajmi, ktoré sú automorfizmami reprezentované (permutáciami vrcholov alebo hrán).

## 6.5 Porovnanie výstupov

Pri konštrukcii klietok  $\text{Cage}(k,g)$ , rekordných grafov  $\text{Rec}(k,g)$  incidenčných matíc  $L$  a z nich aj kontrolných matíc  $H$  sme  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  rozdeľili na základe možnosti zobrazenia ich grafu na graficky zobraziteľné grafy a graficky nezobraziteľné grafy. Vo všetkých prípadoch bola splnená podmienka Moorového ohraničenia  $M(k,g)$  zaručujúca existenciu grafu. Schopnosť získať parametre  $R$ ,  $|\text{AutGroup}(\text{Cage})|$ , rozmery incidenčnej  $L$  a kontrolnej matice  $H$  sa líšila v závislosti od použitého riešenia. Kým v konzolovej aplikácii sme získali všetky údaje, pre riešenie z online aplikácie CoCalc

bolo príliš náročné získať spomínané údaje z Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23), Rec(3,25) a Cage(5,10). Správnosť  $|\text{AutGroup}(\text{Cage})|$  sme overili pomocou hodnôt  $|\text{AutGroup}(\text{Cage})|$  z tabuľky Rec(k,g) tým, že sme výsledok porovnali s očakávanou hodnotou, a takisto nám ho potvrdil aj dokument [12], kde sú niektoré informácie o známych Cage(k,g) alebo Rec(k,g) uvedené. Najväčší počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$  v oboch riešeniach má klietka Cage(6,4) s hodnotou 1036800 a najmenší počet  $|\text{AutGroup}(\text{Cage})|$  majú podľa riešenia konzolovej aplikácie Rec(3,23), Rec(3,25), Rec(10,5), Rec(11,5), Rec(13,5) a to jedna. Podľa online aplikácie CoCalc majú najmenší počet  $|\text{AutGroup}(\text{Cage})|$  Rec(10,5), Rec(11,5) a Rec(13,5). Rekordné grafy Rec(3,23) a Rec(3,25) nebolo možné vygenerovať. Najmenšia incidenčná matica L bola vygenerovaná v oboch riešeniach z klietky Cage(3,5) s rozmerom  $10 \times 15$  a najväčšia L bola vygenerovaná z rekordného grafu Rec(3,25) s hodnotou  $108906 \times 163359$  z konzolovej aplikacie. Najmenšia kontrolná matica H bola vygenerovaná v oboch riešeniach z klietky Cage(3,5) s hodnotou  $9 \times 15$  a najväčšia H bola vygenerovaná z rekordného grafu Rec(3,25) s hodnotou  $108905 \times 163359$  z konzolovej aplikacie. V online aplikácii CoCalc bola najväčšia H aj L vygenerovaná z klietky Cage(7,8). Incidenčná matica L mala rozmer  $672 \times 2352$  a kontrolná matica mala rozmer  $671 \times 2352$ . Incidenčné L aj kontrolné matice H rekordných grafov Rec(3,23) a Rec(3,25) bolo pre konzolovú aplikáciu príliš náročné vygenerovať a uložiť, preto ich hodnoty neuvádzame. Najdlhšie sa generovala a ukladala L z rekordného Rec(3,20), a to 252,5299 s. a z nej sme dostali kontrolnú maticu H a uložili ju za 338,2145 s. Najkratšie bola generovaná a ukladaná L z klietky Cage(3,6), a to 0,0028 s. a z nej sa najrýchlejšie generovala a uložila kontrolná matica (za 0,005 s). Pre online aplikáciu CoCalc bolo príliš náročné vygenerovať a uložiť incidenčné L a kontrolné matice H z rekordných grafov

Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23) a Rec(3,25) a Cage(5,10), preto rovnako ich hodnoty neuvádzame. Najkratšie bola generovaná a ukladaná L z klietky Cage(3,6), a to 0,0053 s. a z nej sa najrýchlejšie generovala a uložila kontrolná matica (za 0,0017 s).

Na základe incidenčnej matice L grafu, z ktorej sme najskôr získali H, sme boli schopní vygenerovať lineárny kód  $C(n,m,d)$ , ako aj jeho generujúcu maticu G. Z generujúcich matíc G sme zistovali r, m, d, a takisto p(n,m,d). Tento parameter sme porovnávali s parametrom optimálneho kódu o(n,m,d) v dvoch prípadoch Cage(3,5), Cage(3,6), kedy to bolo možné pre obe riešenia a zistili sme, že hodnoty sa len mierne líšili. Sledovali sme aj čas generovania kontrolných t(H) a generujúcich matíc t(G), ktorý naznačoval výpočtovú náročnosť. Pri konštrukcii C(n,m,d) sme lineárne kódy rozdelili na základe výpočtovej náročnosti parametrov. Všetky parametre sa nám podarilo určiť pre obe riešenia z klietok Cage(3,5) a Cage(3,6), iba parameter o(n,m,d) sa nám nepodarilo určiť v oboch riešeniach pre Cage(3,7), Cage(3,8), Cage(4,5), Cage(4,7), Cage(6,4). Parametre d, p(n,m,d) a o(n,m,d) sme nevedeli určiť kvôli výpočtovej náročnosti v riešení konzolovej aplikácie consoleAppC- ges pre Cage(3,10), Cage(3,11), Rec(3,14), Rec(3,16), Rec(3,17), Rec(3,18), Cage(4,9), Cage(4,10), Cage(5,10), Cage(7,5), Cage(7,7), Cage(7,8), Rec(10,5), Rec(11,5), Rec(12,5) a Rec(13,5). Pre online aplikáciu CoCalc sú to klietky a rekordné grafy Cage(3,10), Cage(3,11), Rec(3,14), Rec(3,16), Cage(4,9), Cage(4,10), Cage(7,5), Cage(7,7), Cage(7,8), Rec(10,5), Rec(11,5), Rec(12,5) a Rec(13,5). Lineárne kódy, ktorým nebolo možné určiť parametre sme sa pokúšali vygenerovať v konzolovej aplikácii z rekordných grafov Rec(3,20), Rec(3,23), Rec(3,25). Zo všetkých C(n,m,d), okrem tých, ktoré boli vygenerované z rekordných grafov Rec(3,20), Rec(3,23), Rec(3,25), sme doká-

zali zistiť počet automorfizmov  $|\text{AutGroup}(C)|$ , ktorý je zároveň zhodný s  $|\text{AutGroup}(\text{Cage})|$ . V online aplikácii CoCalc sme sa pokúšali lineárne kódy, ktorým nebolo možné určiť parametre, vygenerovať z Cage(5,10), Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23), Rec(3,25). Zo všetkých  $C(n,m,d)$ , okrem tých, ktoré boli vygenerované z rekordných grafov Cage(5,10), Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23), Rec(3,25), sme dokázali zistiť počet automorfizmov  $|\text{AutGroup}(C)|$ , ktorý je zároveň zhodný s  $|\text{AutGroup}(\text{Cage})|$ . Lineárny kód  $C(15,64,5)$  má parametre optimálneho kódu  $o(n,m,d)$  a perfektného kódu  $p(n,m,d)$  pre obe riešenia najbližšie k hodnote 1. Zistili sme, že hodnota  $p(n,m,d)$  sa znižuje v závislosti od zvyšujúceho sa obvodu  $g$  v Cage( $k,g$ ) alebo Rec( $k,g$ ) pre obe riešenia. Minimálne Hammingove vzdialnosti v kódoch  $d$  sa zhodujú s hodnotami obvodov Cage( $k,g$ ) alebo Rec( $k,g$ ) v oboch riešeniach, kedy to výpočtová náročnosť umožňovala. V konzolovej aplikácii sme najväčšiu maticu  $G$  získali s rozmerom  $1945 \times 3240$ , najväčší počet kódových slov  $m$  s hodnotou rádovo  $31867 \times 10^{581}$  sme získali z klietky Cage(5,10). Najdlhšie sa generovala a ukladala  $G$  z klietky Cage(5,10), a to 108,6707 s.  $G$  z rekordných grafov Rec(3,20), Rec(3,23) a (3,25) bolo príliš náročné vygenerovať a uložiť, preto ich  $t(G)$  neuvádzame. Najkratšie bola generovaná a ukladaná  $G$  z klietky Cage(3,5), a to 0,0028 s. Druhá najkratšie generovaná a ukladaná bola  $G$  z klietky Cage(3,6), a to 0,0065 s. V online aplikácii CoCalc sme najväčšiu maticu  $G$  získali s rozmerom  $1681 \times 2352$ , najväčší počet kódových slov  $m$  s hodnotou  $10750 \times 10^{502}$  sme získali z klietky Cage(7,8). Najdlhšie sa generovala a ukladala  $G$  z klietky Cage(7,8) a to 60,9157 s.  $G$  z Cage(5,10), Rec(3,17), Rec(3,18), Rec(3,20), Rec(3,23), Rec(3,25) bolo príliš náročné vygenerovať a uložiť, preto ich  $t(G)$  neuvádzame. Najkratšie bola generovaná a ukladaná  $G$  z klietky Cage(3,6), a to 0,0023 s. Druhá najkratšie generovaná a ukladaná bola  $G$  z klietky Cage(3,5),

a to 0,0025 s.

Z grúp automorfizmov klietok a rekordných grafov sme zistili, že obraz z automorfizmu  $\text{AutGroup}(\text{Cage})$  je rovnaký práve s jedným obrazom z automorfizmu  $\text{AutGroup}(C)$ , ktorý sme získali z rovnakej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Obe grupy automorfizmov majú rovnakú veľkosť  $|\text{AutGroup}(\text{Cage})|$ . V konzolovej aplikácii sa najdlhšie generovala a ukladala  $\text{AutGroup}(\text{Cage})$  z klietky  $\text{Cage}(6,4)$ , a to 0,76759 s. Ako druhá sa generovala a ukladala  $\text{AutGroup}(\text{Cage})$  z klietky  $\text{Cage}(7,5)$ , a to 0,20724 s. Najkratšie bola generovaná a ukladaná  $\text{AutGroup}(\text{Cage})$  z rekordného grafu  $\text{Rec}(3,5)$ , a to 0,00053 s. Druhá najkratšie generovaná a ukladaná bola  $\text{AutGroup}(\text{Cage})$  z klietky  $\text{Cage}(4,7)$ , a to 0,00081 s. V online aplikácii CoCalc sa najdlhšie generovala a ukladala  $\text{AutGroup}(\text{Cage})$  z klietky  $\text{Cage}(7,8)$ , a to až 56,1916 s. Ako druhá sa generovala, a ukladala  $\text{AutGroup}(\text{Cage})$  z klietky  $\text{Cage}(6,4)$ , a to až 43,6845 s. Najkratšie bola generovaná a ukladaná  $\text{AutGroup}(\text{Cage})$  z  $\text{Cage}(3,7)$ , a to 0,0043 s. Druhá najkratšie generovaná a ukladaná bola  $\text{AutGroup}(\text{Cage})$  z  $\text{Rec}(3,6)$ , a to 0,0095 s. Pri generovaní a ukladaní automorfizmov grafu poskytuje online aplikácia CoCalc omnoho pomalšie výsledky ako konzolová aplikácia a pre riešenie z CoCalc nebolo možné vygenerovať a uložiť grupy automorfizmov grafu  $\text{AutGroup}(\text{Cage})$  z rekordných grafov  $\text{Rec}(3,20)$ ,  $\text{Rec}(3,23)$ ,  $\text{Rec}(3,25)$ ,  $\text{Rec}(3,17)$ ,  $\text{Rec}(3,18)$  a klietku  $\text{Cage}(5,10)$ . Vo všetkých prípadoch majú  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  totožnú maticu  $G$  s  $G$  získanou priamo z  $C(n,m,d)$ , ktorý bol vygenerovaný z totožnej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$  a to isté platí aj pre kontrolnú maticu  $H$  získanú z automorfizmu a kontrolnú maticu  $H$  získanú priamo z incidenčnej matice grafu. V prípade incidenčnej matice  $L$  platí, že  $L$  získané z jednotlivých automorfizmov príslušnej grupy automorfizmov  $\text{AutGroup}(\text{Cage})$  sa medzi sebou líšia, avšak rozmery  $L$  ostá-

vajú zachované pre všetky rovnaké.

Grupu automorfizmov lineárneho kódu AutGroup(C) sme schopní vygenerovať v riešení z konzolovej aplikácie iba z  $C(15,64,5)$ ,  $C(21,256,6)$ ,  $C(36, 8192, 7)$ ,  $C(45,65536,8)$  a  $C(38,1048576,5)$ , pretože pre ostatné kódy sa jednalo o výpočtovo príliš náročný problém. Najrýchlejšie sa vygenerovala a uložila grupa automorfizmov z lineárneho kódu  $C(15,64,5)$  AutGroup(C) za 0,000401 s. V online aplikácii CoCalc sme boli schopní vygenerovať grupu automorfizmov lineárneho kódu AutGroup(C) iba z  $C(15,64,5)$ ,  $C(21,256,6)$ , pretože pre ostatné kódy sa jednalo o výpočtovo príliš náročný problém. Najrýchlejšie sa vygenerovala a uložila grupa automorfizmov z lineárneho kódu  $C(15,64,5)$  za 0,000729 s.  $C(21,256,6)$  sa vygenerovala a uložila za 0,001508 s. Ukázali sme, že obrazy z automorfizmov grupy lineárneho kódu AutGroup(C) sú totožné s obrazmi automorfizmov grafu, ktoré zodpovedajú incidenčným maticiam, z ktorých sa vieme prostredníctvom kontrolných matíc H dostať k tomu istému lineárnemu kódu  $C(n,m,d)$ . Dvojice týchto automorfizmov majú vo svojej grupe automorfizmov rôzne poradia.

Po porovnaní oboch riešení sme zistili, že vo väčšine prípadov nám konzolová aplikácia poskytuje rýchlejšie výsledky, prehľadnejší prístup a zvládne výpočtovo náročnejšie problémy, avšak riešenie v Cocalc nám umožňuje získať jeden typ údajov alebo viac jednoducho vypočítateľných údajov pre takmer všetky klietky alebo rekordné grafy, matice a automorfizmy naraz (ak proces nie je príliš výpočtovo náročný), dokonca pre niektoré prípady zvládne výpočet aj rýchlejšie. Výpočtová náročnosť a nižší výkon však majú veľký vplyv na to, že získame menej výsledkov ako z konzolovej aplikácie.

# Kapitola 7

## Záver

Problémom pri komunikácii a prenose informácií cez médium alebo komunikačný kanál je možná strata alebo skreslenie informácie v dôsledku pôsobenia šumu [3]. Sekvencia kódov s rýchlosťou menšou ako je kapacita kanála má schopnosť opraviť všetky chyby spôsobené šumom pridaním redundantných symbolov k pôvodným údajom. Táto realizácia viedla k vývoju kódov na opravu chýb (ECC), ktorých cieľom je zakódovať údaje pridaním určitého množstva redundancie do správy, aby bolo možné pôvodnú správu obnoviť a dekódovať [7].

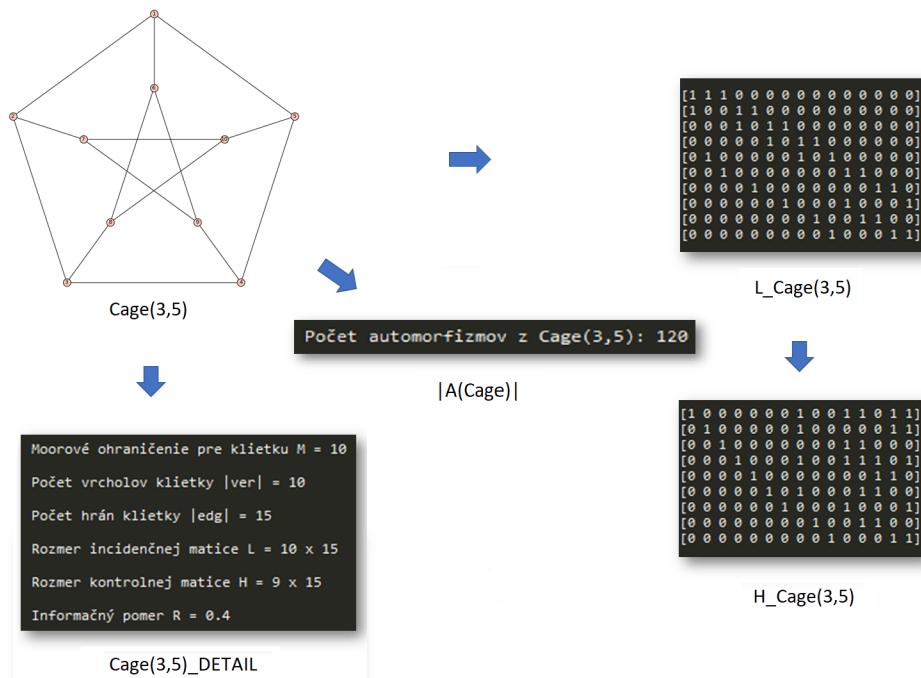
Lineárny kód  $C[n,r]$  je  $r$ -rozmerný lineárny podpriestor priestoru  $F_2^n$  [13, 5], ktorý má bohatú grupu automorfizmov  $\text{AutGroup}(C)$ . Grupy automorfizmov obsahujú množstvo informácií o uvažovanom kóde  $C(n,m,d)$ . Cieľom našej práce bolo zstrojiť  $C(n,m,d)$  s predpísanou  $\text{AutGroup}(\text{Cage})$ , ako aj pokúsiť sa určiť  $\text{AutGroup}(C)$  pre daný kód [17].

Navrhli sme 2 riešenia na skúmanie problematiky. Prvé riešenie bolo implementované v online aplikácii CoCalc [18] vo forme SageMath skriptov

[19]. Nevýhodami SageMath skriptov, ktoré nás limitovali sú pomalší výkon, výpočtovo náročný a neprehľadný prístup k výsledkom. Druhé riešenie bolo implementované ako konzolová aplikácia v programovacom jazyku Python [8], kde bola knižnica SageMath naimportovaná. Riešenie je závislé od výkonu zariadenia používateľa. Využitím vlastného zariadenia čiastočne eliminujeme problém prvého riešenia v pomalšom výkone. Po porovnaní oboch riešení sme zistili, že vo väčšine prípadov nám konzolová aplikácia poskytuje rýchlejšie výsledky a prehľadnejší prístup, avšak riešenie v Cocalc nám umožňuje získať jeden typ údajov alebo viac jednoducho vypočítateľných údajov pre takmer všetky klietky Cage(k,g) alebo rekordné grafy Rec(k,g) matice a automorfizmy naraz, dokonca pre niektoré aj rýchlejšie. Výpočtová náročnosť a nižší výkon majú veľký vplyv na to, že získame menej výsledkov ako z konzolovej aplikácie.

Nami navrhnuté riešenia sme rozdelili do štyroch častí. Ako prvé sme generovali Cage(k,g) a Rec(k,g) využitím „SageMath grafov“, použitím nami navrhnutého parsera zdrojových údajov a použitím vlastného generovania Cage(k,g) na základe známych údajov z literatúry. Existenciu Cage(k,g) a Rec(k,g)sme overovali pomocou Moorového ohraničenia  $M(k,g)$ . Pri konštrukcii Cage(k,g), Rec(k,g), ich incidenčných L a kontrolných matíc H sme Cage(k,g) a Rec(k,g) rozdelili na základe možnosti zobrazenia ich grafu. Z grafov sme získali údaje, ktoré sme vedeli vyhodnotiť. Z údajov sme zistili, že vo všetkých prípadoch bola splnená podmienka Moorového ohraničenia  $M(k,g)$  zaručujúca existenciu danej Cage(k,g) alebo Rec(k,g). Takisto sa nám podarilo zistiť informačný pomer R, počet automorfizmov  $|\text{AutGroup}(\text{Cage})|$ , rozmery incidenčnej L a kontrolnej matice H. Vyhodnotili sme časové náročnosti generovania incidenčných L a kontrolných matíc H. Rekordné grafy

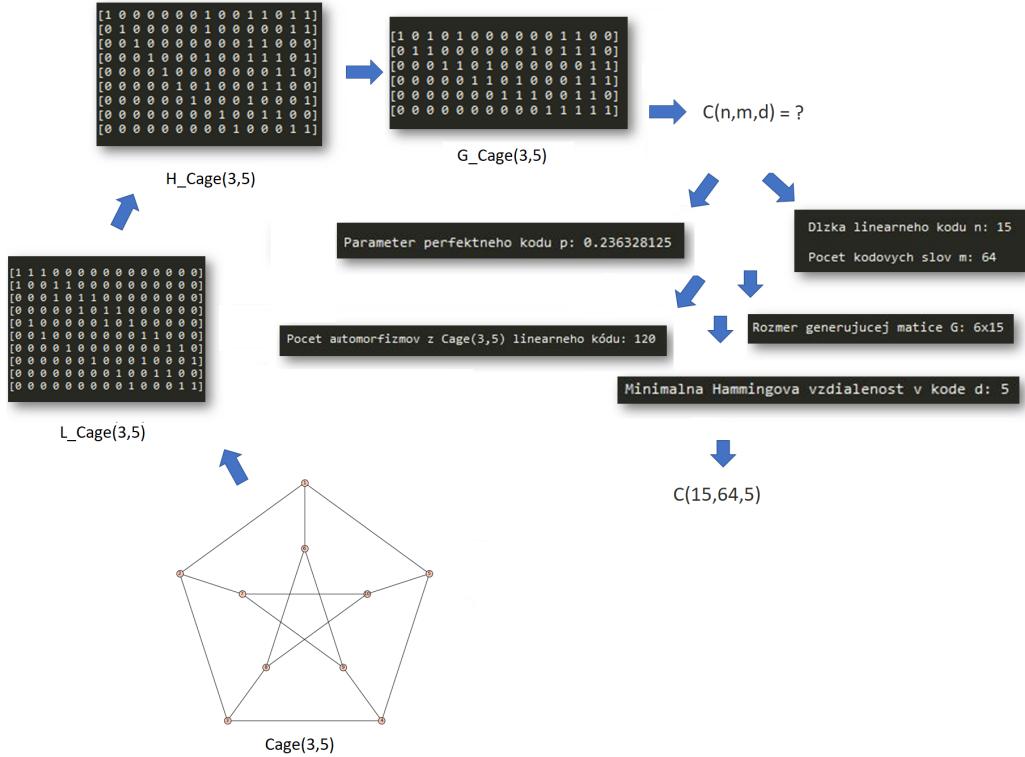
$\text{Rec}(3,23)$  a  $(3,25)$  bolo príliš náročné vygenerovať a uložiť, preto ich hodnoty neuvádzame. Na obrázku (Obr. 7.1) je znázornený spôsob ako sme z grafov získavali informácie.



Obr. 7.1: Cage(3,5) - získavanie údajov

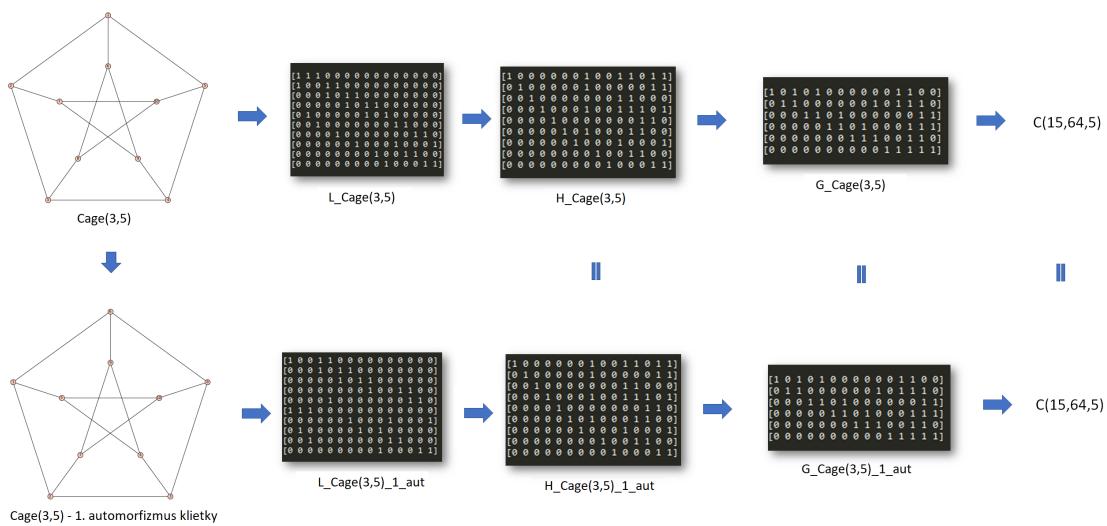
Ako druhé sme generovali lineárny kód  $C(n,m,d)$  a generujúcu maticu  $G$  z  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Lineárny kód  $C(n,m,d)$  sme overili pomocou jeho  $H$  a  $G$ . Z generujúcich matíc  $G$  sme zistovali  $r$ , maximálny počet kódových slov  $m$  v kóde a minimálnu Hammingovu vzdialenosť  $d$ , a takisto parameter perfektného kódu  $p(n,m,d)$ , ktorý vyjadruje vzdialenosť nášho  $C(n,m,d)$  od perfektného kódu  $o(n,m,d)$  v dvoch prípadoch  $\text{Cage}(3,5)$ ,  $\text{Cage}(3,6)$ , kedy to bolo možné a zistili sme, že hodnoty sa mierne líšili. Takisto sme sledovali aj čas generovania a ukladania generujúcich matíc  $t(G)$ . Pri konštrukcii  $C(n,m,d)$  z  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  sme lineárne kódy rozdelili na základe výpočtovej

náročnosti parametrov na lineárne kódy so všetkými uvažovanými parametrami,  $C(n,m,d)$  bez parametra optimálneho kódu,  $C(n,m,d)$  bez minimálnej Hammingovej vzdialenosťi, parametrov perfektného a optimálneho kódu,  $C(n,m,d)$  bez parametrov. Zo všetkých  $C(n,m,d)$  okrem tých, ktoré boli vygenerované z rekordných grafov  $Rec(3,20)$ ,  $Rec(3,23)$ ,  $Rec(3,25)$  sme dokázali zistiť počet automorfizmov  $|AutGroup(C)|$ , a tie sú zároveň zhodné s počtom automorfizmov z príslušných klietok  $|AutGroup(Cage)|$  alebo  $Rec(k,g)$ . Lineárny kód  $C(15,64,5)$  má parametre optimálneho kódu  $o(n,m,d)$  a perfektného kódu  $p(n,m,d)$  najbližšie k hodnote 1. Zistili sme, že hodnota  $p(n,m,d)$  sa znižuje v závislosti od zvyšujúceho sa obvodu  $g$  v klietkach alebo rekordných grafov a minimálne Hammingove vzdialenosťi  $d$  sa zhodujú s hodnotou obvodov grafov. Na obrázku (Obr. 7.2) je znázornený spôsob, ako sme z lineárneho kódu  $C(15,64,5)$  získavali informácie a na základe nich sme ho vedeli pomenovať.

Obr. 7.2:  $C(15,64,5)$  - získavanie údajov

Generovali sme aj  $C(n,m,d)$  z grupy automorfizmov klietky a rekordného grafu  $\text{AutGroup}(\text{Cage})$ , kde je potrebné spracovať jednotlivé automorfizmy z grupy tak, aby sme boli schopní vytvoriť obraz danej  $\text{Cage}(k,g)$  alebo rekordného grafu, vygenerovať z automorfizmov  $C(n,m,d)$  a spracovať výsledky. Vo všetkých prípadoch majú  $\text{Cage}(k,g)$  a  $\text{Rec}(k,g)$  totožnú maticu  $G$  s  $G$  získa-nou priamo z  $C(n,m,d)$ , ktorý bol vygenerovaný z totožnej  $\text{Cage}(k,g)$  alebo  $\text{Rec}(k,g)$ . Kontrolná matica  $H$  získaná z incidenčnej matice automorfizmu sa takisto zhoduje s kontrolnou maticou  $H$  získanou priamo z incidenčnej ma-tice klietky alebo rekordného grafu. V prípade incidenčnej matice  $L$  platí,

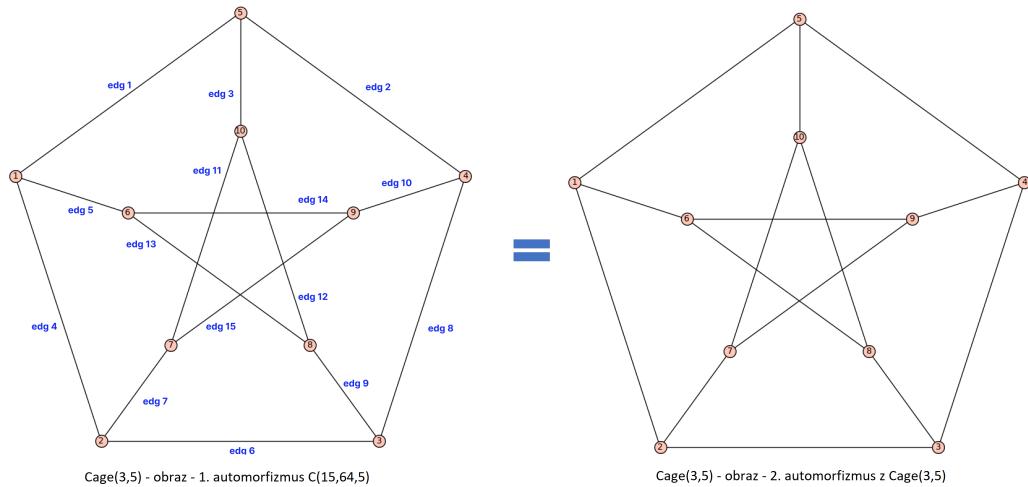
že  $L$  získané z jednotlivých automorfizmov príslušnej grupy automorfizmov sa medzi sebou líšia, avšak rozmery  $L$  ostávajú zachované pre všetky rovnako. Na nasledujúcom obrázku zobrazujeme generovanie lineárnych kódov  $C(n,m,d)$  2 spôsobmi. Prvý spôsob je priamo z klietky a druhý spôsob je z automorfizmu získaného z toho istého grafu. Ilustračný príklad je pre klietku Cage(3,5) a jej prvý automorfizmus (Obr. 7.3).



Obr. 7.3: Porovnanie lineárneho kódu C(15,64,5) zostrojeného z klietky a z automorfizmu klietky

Grupu automorfizmov lineárneho kódu AutGroup(C) sme boli schopní vygenerovať iba z maximálne piatich lineárnych kódov  $C(n,m,d)$  zo všetkých uvažovaných, pretože pre ostatné kódy sa jednalo o príliš výpočtovo náročný problém. Ukázali sme, že obrazy z automorfizmov grupy lineárneho kódu sú totožné s obrazmi automorfizmov grafu, ktoré zodpovedajú incidenčným maticiam, z ktorých prostredníctvom kontrolných matíc H dostávame ten istý lineárny kód  $C(n,m,d)$ . Dvojice týchto automorfizmov majú vo svojej grupe automorfizmov rôzne poradia. Ilustračný príklad je pre 1. automorfizmus li-

neárneho kódu  $C(15,64,5)$  a 2. automorfizmus klietky Cage(3,5) (Obr. 7.4).



Obr. 7.4: Porovnanie obrazov z  $C(15,64,5)$  a Cage(3,5)

Výsledné incidenčné matice L, kontrolné matice H, generujúce matice G a grupy automorfizmov sme uložili do textových súborov. Na uložené výsledky je možné nadviazať a použiť ich ako vstupy pre ďalšie práce, ktoré budú využívať incidenčné matice L, matice H, matice G, grupy automorfizmov alebo sa inšpirujú informáciami z našich tabuliek.

# Príloha A: Používateľská príručka

Táto používateľská príručka slúži ako návod na spúšťanie oboch aplikovaných riešení. V prvom kroku je potrebné si stiahnúť obsah GitHub repozitára (Príloha B) [20] do svojho zariadenia. Na stránke SageMath [19], v časti *Download* je potrebné zvoliť na základe zariadenia buď *macOS binaries*, *Linux/macOS binaries* alebo *Windows binaries* a následne verziu inštalačného súboru, ktorý po stiahnutí a spustení nainštaluje SageMath konzolu do zariadenia používateľa. My sme využili inštalačný súbor *SageMath-9.3-Installer-v0.6.3.exe* pre operačný systém Windows. V ďalšom kroku bude potrebné sa zaregistrovať na webovom prehliadači v online aplikácii CoCalc [18], kde si používateľ založí prázdny projekt. Bude potrebné, aby si do projektu používateľ vložil celý obsah priečinka CoCalc. Po tomto kroku je už používateľ pripravený spustiť prvé navrhované riešenie. V projekte sa vložením vytvorí rovnaká štruktúra priečinka CoCalc ako v GitHub repozitári [20]. Používateľ môže spúšťať jednotlivé skripty prostredníctvom online editora, v ktorom sa skript otvára dvojitým kliknutím. Po spustení skriptu sleduje výpis a výstupné textové súbory v priečinkoch *IncidenceMatrices*, *GeneratorMatrices*, *ParityCheckMatrices*, *AutomorphismGroups*, poprípade modifikuje vstupné parametre k a g. Pre spustenie druhého riešenia používateľ otvorí nainštalovanú SageMath konzolu [19]. Je potrebné sa pomocou príkazu "*cd*" presunúť do priečinka *consoleAppCages*. Konzolová aplikácia sa v SageMath konzole

následne spustí zavolením príkazu "`load('Run.py')`", ktorý načíta základnú triedu. Používateľ sa následne naviguje podľa voľby možností v konzolovej aplikácii a takisto sleduje okrem výpisov aj výstupné textové súbory v priečinkoch *IncidenceMatrices*, *GeneratorMatrices*, *ParityCheckMatrices* a *AutomorphismGroups*.

## Príloha B: Obsah repozitára

Zdrojové kódy sú dostupné v GitHub repozitári [20], ktorý má nasledujúcu štruktúru:

- priečinok **CoCalc** so zdrojovými súbormi a výsledkami .
  - priečinok **IncidenceMatrices** s textovými súbormi incidenčných matíc
  - priečinok **GeneratorMatrices** s textovými súbormi generujúcich matíc
  - priečinok **ParityCheckMatrices** s textovými súbormi kontrolných matíc
  - priečinok **AutomorphismGroups** s textovými súbormi grúp automorfizmov
  - priečinok **ExoosCages** s textovými súbormi vstupných zoznamov susedností
  - skript **autCodesFromCages.sagews**
  - skript **autGroupCode.sagews**
  - skript **autLinearCodesData.sagews**
  - skript **cagesData.sagews**

- skript **generateAutCodesFromCages.sagews**
- skript **generateAutGroupLCode.sagews**
- skript **generateCageAndLinearCodeByParameters.sagews**
- skript **generateGeneratorMatrices.sagews**
- skript **generateIncidenceMatrices.sagews**
- skript **generateParityCheckMatrices.sagews**
- skript **linearCodesData.sagews**
- skript **linearCodeValidation.sagews**
- skript **maxnWordsLinearCodes.sagews**
- skript **minDistanceLinearCodes.sagews**
- skript **mooreBoundsCageValidation.sagews**
- skript **perfectLinearCodesParameter.sagews**
- skript **sizeNlinearCodesData.sagews**
- priečinok **consoleAppCages** so zdrojovými súbormi a výsledkami .
  - priečinok **IncidenceMatrices** s textovými súbormi incidenčných matíc
  - priečinok **GeneratorMatrices** s textovými súbormi generujúcich matíc
  - priečinok **ParityCheckMatrices** s textovými súbormi kontrolných matíc
  - priečinok **AutomorphismGroups** s textovými súbormi grúp automorfizmov
  - priečinok **ExoosCages** s textovými súbormi vstupných zoznamov susedností

- súbor **AutomorphismDetail.py**
  - súbor **AutomorphismMenu.py**
  - súbor **Cage.py**
  - súbor **CageDetail.py**
  - súbor **CageMenu.py**
  - súbor **Constants.py**
  - súbor **Formatters.py**
  - súbor **LinearCode.py**
  - súbor **LinearCodeMenu.py**
  - súbor **MainMenu.py**
  - súbor **Run.py**
- priečinok **sourceMaterials** s použitou literatúrou
  - súbor **Diplomova\_Praca.pdf**

# Zoznam použitej literatúry

- [1] Claude Elwood Shannon. “A Mathematical Theory of Communication”. In: *The Bell System Technical Journal* 27 (1948), s. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [2] D. R. Fulkerson a O. A. Gross. “Incidence matrices and interval graphs.” In: *Pacific Journal of Mathematics* 15.3 (1965), s. 835–855. DOI: pjm/1102995572.
- [3] R. Hill. *A First Course in Coding Theory*. Zv. 72-74. Oxford applied mathematics and computing science series 459. Oxford University Press, 1988, s. 251. ISBN: 0-19-853803-0. DOI: 10.2307/3618021.
- [4] Stephan Wesemeyer. *On the automorphism group of various Goppa codes. Dissertation Thesis*. University of Exeter in the Faculty of Science, 1997.
- [5] W Cary Huffman a Vera Pless. *Fundamentals of Error-Correcting Codes, Basic concepts of linear codes*. Cambridge Univ. Press, 2003, s. 1–20. ISBN: 9780511807077. DOI: <https://doi.org/10.1017/CBO9780511807077>.
- [6] Gabofetswe Malema a Michael Liebelt. “High Girth Column-Weight Two LDPC Codes Based on Distance Graphs”. In: *EURASIP Journal on Wireless Communications and Networking* 2007 (2006), s. 5. DOI: doi:10.1155/2007/48158.

- [7] G.A. Malema. *Low-Density Parity-Check Codes: Construction and Implementation. Dissertation Thesis*. School of Electrical, Electronic Engineering, Faculty of Engineering, Computer a Mathematical Sciences The University of Adelaide, Australia, 2007.
- [8] Guido Van Rossum a Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.
- [9] Peter Vandendriessche. “LDPC codes associated with linear representations of geometries”. In: *American Institute of Mathematical Sciences’ Journals* 4.3 (2010), s. 405–417.
- [10] R.A. Beezer. *A First Course in Linear Algebra*. Open Textbook Library. Congruent Press, 2012, s. 630. ISBN: 9780984417551.
- [11] Sharma Rakesh Goswami Ashish. “Low Complexity, High Performance LDPC Codes Based on Defected Fullerene Graphs”. In: *International Journal of Mathematical and Computational Sciences* 6.2 (2012), s. 136–138. DOI: 10.5281.
- [12] Geoffrey Exoo a Robert Jajcay. “Dynamic cage survey”. In: *The Electronic Journal of Combinatorics* DS16.3 (2013), s. 55. DOI: 10.37236/37.
- [13] Rafael Misoczki et al. *MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes*. IEEE, 2013, s. 1–21. ISBN: 978-1-4799-0446-4. DOI: 10.1109/ISIT.2013.6620590.
- [14] Microsoft. *Visual Studio Code*. Available at: <https://code.visualstudio.com/docs>. 2015.
- [15] Branislav Boráň. *Hustota inverzií riedkych cyklických matíc. Bachelor’s Thesis*. Slovenská technická univerzita v Bratislave, Fakulta elektrotechniky a informatiky, Slovenská republika, 2018.

- [16] Andries Brouwer. *Small binary codes*. Available at: <https://www.win.tue.nl/~aeb/codes/binary-1.html>. 2018.
- [17] Nisreen Mohammad Nashash. *On The Automorphism Groups of Some Linear Codes. Master's Thesis*. Faculty of GraduateStudies, Hebron University, Palestine, 2019.
- [18] Inc. Sagemath. *CoCalc – Collaborative Calculation and Data Science*. Available at: <https://cocalc.com>. 2020.
- [19] W. Stein The Sage Developers et al. *SageMath, version 9.3*. Available at: <http://www.sagemath.org>. 2020.
- [20] Branislav Boráň. *VLASTNOSTI-A-SYMETRIE-LINEARNYCH-KODOV*. Available at: <https://github.com/boran2/VLASTNOSTI-A-SYMETRIE-LINEARNYCH-KODOV>. 2022.
- [21] Geoffrey Exoo. *Regular Graphs of Given Degree and Girth*. Available at: <http://cs.indstate.edu/ge/CAGES/index.html>.