Sveučilište u Zagrebu
Prirodoslovno–matematički fakultet
Matematički odjel

Vedran Krčadinac

Klasifikacija konačnih projektivnih ravnina reda 7 i 8

Diplomski rad

Zagreb, prosinac 1996.

Predgovor

Cilj ovog rada je dokazati jedinstvenost konačnih projektivnih ravnina reda 7 i 8.

U prvom poglavlju iznesene su neke činjenice iz teorije konačnih projektivnih ravnina. Navedene su ravnine reda 7 i 8 konstruirane pomoću konačnih polja, da bi ih se moglo usporediti s ravninama iz četvrtog poglavlja.

U drugom poglavlju objašnjene su osnovne ideje dokaza jedinstvenosti ravnina reda 7 i 8. Ukratko je opisana klasifikacija za ostale redove za koje je provedena.

Dokaz jedinstvenosti uključuje generiranje predstavnika klasa izotopije i glavnih klasa latinskih kvadrata. To je tema trećeg poglavlja. Dobio sam 564 predstavnika klasa izotopije reda 7, što se ne slaže s brojem objavljenim u literaturi. Provjeri tog rezultata posvećena je treća točka. Smatram da je moj rezultat ispravan.

Četvrto poglavlje posvećeno je samoj klasifikaciji. Sve ravnine dobivene kao rezultat potrage izomorfne su onoj klasičnoj, čime je dokazana jedinstvenost za redove 7 i 8.

U petom poglavlju objašnjeno je zašto korišteni programi nisu primjenjivi na red 9. Navedeno je kratko razmišljanje o primjeni računala u matematici.

Radu su priložena dva dodatka. U prvom su navedeni predstavnici glavnih klasa latinskih kvadrata, a u drugom kod korištenih programa.

Želio bih izraziti zahvalnost svima koji su na bilo koji način doprinjeli nastanku ovog rada, a posebno voditelju, prof. dr. Jurju Šiftaru, te dr. Mariu Pavčeviću s FER-a.

Sadržaj

1 Konačne projektivne ravnine

U ovom poglavlju navedene su osnovne činjenice o konačnim projektivnim ravninama. Izlaganje je ograničeno na tvrdnje kasnije korištene u radu. Detaljan pregled teorije, kao i dokazi navedenih tvrdnji nalaze se u knjigama [1] i [2].

Definicija 1.1. **Incidencijska struktura** *je uređena trojka* $(\mathcal{P}, \mathcal{L}, I)$*, gdje su* $\mathcal{P}$ *i* $\mathcal{L}$ *neprazni skupovi, a* $I \subseteq \mathcal{P} \times \mathcal{L}$ *relacija, koju zovemo* **incidencija**.

Elemente skupa $\mathcal{P}$ zovemo *točke* i označavamo velikim slovima $A, B, C \ldots$, a elemente skupa $\mathcal{L}$ *pravci* i označavamo malim slovima $a, b, c \ldots$. O incidenciji često govorimo geometrijskim rječnikom, npr. pravac prolazi točkom, točka leži na pravcu.

Definicija 1.2. **Projektivna ravnina** *je incidencijska struktura koja zadovoljava aksiome:*

(P1) *Svake dvije točke incidentne su s točno jednim pravcem.*

(P2) *Svaka dva pravca sijeku se u jedinstvenoj točki.*

(P3) *Postoje četiri točke od kojih nikoje tri ne leže na istom pravcu.*

Za projektivnu ravninu $\mathbf{P}' = (\mathcal{P}', \mathcal{L}', I')$ *kažemo da je* **podravnina** *od* $\mathbf{P}$ *ako je* $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{L}' \subseteq \mathcal{L}$, $I' = I|_{\mathcal{P}' \times \mathcal{L}'}$.

Jedinstvenu spojnicu točaka A i B označavamo s $A \vee B$, a jedinstveno sjecište pravaca c i d s $c \wedge d$. U slučaju kad su skupovi $\mathcal{P}$ i $\mathcal{L}$ konačni govorimo o *konačnoj projektivnoj ravnini.*

Propozicija 1.3. *Neka je* $\mathbf{P} = (\mathcal{P}, \mathcal{L}, I)$ *konačna projektivna ravnina. Tada postoji prirodni broj* $n \geq 2$ *takav da:*

- *Svaki pravac sadrži točno* $n + 1$ *točaka.*

- *Svaka točka leži na točno* $n+1$ *pravaca.*
- *Ukupan broj pravaca je* n^2+n+1.
- *Ukupan broj točaka je* n^2+n+1.

Za ravninu **P** iz prethodne propozicije kažemo da je *reda* n.

Projektivnu ravninu reda n možemo konstruirati iz konačnog polja od n elemenata $(F,+,\cdot)$. Neka je $V = F \times F \times F$ trodimenzionalni vektorski prostor nad F. Neka je $\mathcal{P}$ skup svih jednodimenzionalnih, a $\mathcal{L}$ skup svih dvodimenzionalnih potprostora od V. Za relaciju incidencije I uzimamo skupovnu inkluziju $\subseteq$. Pokazuje se da je $(\mathcal{P}, \mathcal{L}, \subseteq)$ projektivna ravnina istog reda kao i polje F. Ovako konstruiranu projektivnu ravninu zovemo *klasična ravnina reda* n.

Navodim tablice zbrajanja i množenja konačnih polja reda 7 i 8 i klasične ravnine konstruirane pomoću njih na opisani način. Ravnine su radi uštede prostora prikazane u sljedećem obliku: točke su brojevi $1, 2, \dots, v$, a pravci $(n+1)$–člani skupovi točaka. Incidencija je relacija "biti element" $\in$. Da bismo opisali cijelu ravninu dovoljno je navesti skup pravaca $\mathcal{L}$.

Definicija 1.4. *Uvedimo poredak na skupove* $\mathcal{P}$ *i* $\mathcal{L}$, $\mathcal{P} = (P_1, P_2, \dots, P_v)$, $\mathcal{L} = (\ell_1, \ell_2, \dots, \ell_v)$, *gdje je* $v = n^2+n+1$. **Incidencijska matrica** *ravnine* **P** *je* $v \times v$ *matrica* $A = [a_{ij}]_{i,j=1}^{v}$ *definirana s*

$$a_{ij} = \begin{cases} 1, & \textit{ako je } P_i \textit{ incidentan s } \ell_j \\ 0, & \textit{ako } P_i \textit{ nije incidentan s } \ell_j \end{cases}$$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Primjer 1. Konačno polje reda 7.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Primjer 2. Konačno polje reda 8.

```
{ {2, 9, 16, 23, 30, 37, 44, 51},  {1, 9, 10, 11, 12, 13, 14, 15},
  {8, 9, 22, 28, 34, 40, 46, 52},  {5, 9, 19, 29, 32, 42, 45, 55},
  {4, 9, 18, 27, 36, 38, 47, 56},  {7, 9, 21, 26, 31, 43, 48, 53},
  {6, 9, 20, 24, 35, 39, 50, 54},  {3, 9, 17, 25, 33, 41, 49, 57},
  {1, 2, 3, 4, 5, 6, 7, 8},        {2, 15, 22, 29, 36, 43, 50, 57},
  {2, 12, 19, 26, 33, 40, 47, 54}, {2, 11, 18, 25, 32, 39, 46, 53},
  {2, 14, 21, 28, 35, 42, 49, 56}, {2, 13, 20, 27, 34, 41, 48, 55},
  {2, 10, 17, 24, 31, 38, 45, 52}, {1, 51, 52, 53, 54, 55, 56, 57},
  {8, 15, 21, 27, 33, 39, 45, 51}, {5, 12, 22, 25, 35, 38, 48, 51},
  {4, 11, 20, 29, 31, 40, 49, 51}, {7, 14, 19, 24, 36, 41, 46, 51},
  {6, 13, 17, 28, 32, 43, 47, 51}, {3, 10, 18, 26, 34, 42, 50, 51},
  {1, 30, 31, 32, 33, 34, 35, 36}, {7, 15, 20, 25, 30, 42, 47, 52},
  {8, 12, 18, 24, 30, 43, 49, 55}, {6, 11, 22, 26, 30, 41, 45, 56},
  {5, 14, 17, 27, 30, 40, 50, 53}, {3, 13, 21, 29, 30, 38, 46, 54},
  {4, 10, 19, 28, 30, 39, 48, 57}, {1, 23, 24, 25, 26, 27, 28, 29},
  {6, 15, 19, 23, 34, 38, 49, 53}, {4, 12, 21, 23, 32, 41, 50, 52},
  {8, 11, 17, 23, 36, 42, 48, 54}, {3, 14, 22, 23, 31, 39, 47, 55},
  {7, 13, 18, 23, 35, 40, 45, 57}, {5, 10, 20, 23, 33, 43, 46, 56},
  {1, 44, 45, 46, 47, 48, 49, 50}, {5, 15, 18, 28, 31, 41, 44, 54},
  {7, 12, 17, 29, 34, 39, 44, 56}, {3, 11, 19, 27, 35, 43, 44, 52},
  {8, 14, 20, 26, 32, 38, 44, 57}, {4, 13, 22, 24, 33, 42, 44, 53},
  {6, 10, 21, 25, 36, 40, 44, 55}, {1, 37, 38, 39, 40, 41, 42, 43},
  {4, 15, 17, 26, 35, 37, 46, 55}, {3, 12, 20, 28, 36, 37, 45, 53},
  {5, 11, 21, 24, 34, 37, 47, 57}, {6, 14, 18, 29, 33, 37, 48, 52},
  {8, 13, 19, 25, 31, 37, 50, 56}, {7, 10, 22, 27, 32, 37, 49, 54},
  {1, 16, 17, 18, 19, 20, 21, 22}, {3, 15, 16, 24, 32, 40, 48, 56},
  {6, 12, 16, 27, 31, 42, 46, 57}, {7, 11, 16, 28, 33, 38, 50, 55},
  {4, 14, 16, 25, 34, 43, 45, 54}, {5, 13, 16, 26, 36, 39, 49, 52},
  {8, 10, 16, 29, 35, 41, 47, 53} }
```

Primjer 3. Klasična projektivna ravnina reda 7.

```
{ {2, 10, 18, 26, 34, 42, 50, 58, 66}, {1, 10, 11, 12, 13, 14, 15, 16, 17},
  {3, 10, 19, 28, 37, 46, 55, 64, 73}, {7, 10, 23, 27, 38, 44, 57, 61, 72},
  {8, 10, 24, 33, 35, 47, 53, 60, 70}, {9, 10, 25, 31, 36, 43, 56, 62, 69},
  {4, 10, 20, 30, 40, 45, 51, 65, 71}, {5, 10, 21, 32, 39, 49, 54, 59, 68},
  {6, 10, 22, 29, 41, 48, 52, 63, 67}, {1, 2, 3, 4, 5, 6, 7, 8, 9},
  {2, 11, 19, 27, 35, 43, 51, 59, 67}, {2, 15, 23, 31, 39, 47, 55, 63, 71},
  {2, 16, 24, 32, 40, 48, 56, 64, 72}, {2, 17, 25, 33, 41, 49, 57, 65, 73},
  {2, 12, 20, 28, 36, 44, 52, 60, 68}, {2, 13, 21, 29, 37, 45, 53, 61, 69},
  {2, 14, 22, 30, 38, 46, 54, 62, 70}, {1, 18, 19, 20, 21, 22, 23, 24, 25},
  {3, 11, 18, 29, 36, 47, 54, 65, 72}, {7, 15, 18, 30, 35, 49, 52, 64, 69},
  {8, 16, 18, 27, 41, 45, 55, 62, 68}, {9, 17, 18, 28, 39, 48, 51, 61, 70},
  {4, 12, 18, 32, 38, 43, 53, 63, 73}, {5, 13, 18, 31, 40, 46, 57, 60, 67},
  {6, 14, 18, 33, 37, 44, 56, 59, 71}, {1, 50, 51, 52, 53, 54, 55, 56, 57},
  {4, 11, 21, 31, 41, 44, 50, 64, 70}, {3, 15, 22, 33, 40, 43, 50, 61, 68},
  {9, 16, 19, 29, 38, 49, 50, 60, 71}, {7, 17, 20, 32, 37, 47, 50, 62, 67},
  {6, 12, 24, 27, 39, 46, 50, 65, 69}, {8, 13, 23, 30, 36, 48, 50, 59, 73},
  {5, 14, 25, 28, 35, 45, 50, 63, 72}, {1, 58, 59, 60, 61, 62, 63, 64, 65},
  {5, 11, 20, 33, 38, 48, 55, 58, 69}, {6, 15, 19, 32, 36, 45, 57, 58, 70},
  {3, 16, 25, 30, 39, 44, 53, 58, 67}, {4, 17, 23, 29, 35, 46, 56, 58, 68},
  {8, 12, 22, 31, 37, 49, 51, 58, 72}, {7, 13, 24, 28, 41, 43, 54, 58, 71},
  {9, 14, 21, 27, 40, 47, 52, 58, 73}, {1, 66, 67, 68, 69, 70, 71, 72, 73},
  {6, 11, 23, 28, 40, 49, 53, 62, 66}, {4, 15, 25, 27, 37, 48, 54, 60, 66},
  {7, 16, 21, 33, 36, 46, 51, 63, 66}, {3, 17, 24, 31, 38, 45, 52, 59, 66},
  {5, 12, 19, 30, 41, 47, 56, 61, 66}, {9, 13, 22, 32, 35, 44, 55, 65, 66},
  {8, 14, 20, 29, 39, 43, 57, 64, 66}, {1, 26, 27, 28, 29, 30, 31, 32, 33},
  {7, 11, 22, 26, 39, 45, 56, 60, 73}, {9, 15, 20, 26, 41, 46, 53, 59, 72},
  {5, 16, 23, 26, 37, 43, 52, 65, 70}, {8, 17, 19, 26, 40, 44, 54, 63, 69},
  {3, 12, 21, 26, 35, 48, 57, 62, 71}, {6, 13, 25, 26, 38, 47, 51, 64, 68},
  {4, 14, 24, 26, 36, 49, 55, 61, 67}, {1, 34, 35, 36, 37, 38, 39, 40, 41},
  {8, 11, 25, 32, 34, 46, 52, 61, 71}, {5, 15, 24, 29, 34, 44, 51, 62, 73},
  {4, 16, 22, 28, 34, 47, 57, 59, 69}, {6, 17, 21, 30, 34, 43, 55, 60, 72},
  {9, 12, 23, 33, 34, 45, 54, 64, 67}, {3, 13, 20, 27, 34, 49, 56, 63, 70},
  {7, 14, 19, 31, 34, 48, 53, 65, 68}, {1, 42, 43, 44, 45, 46, 47, 48, 49},
  {9, 11, 24, 30, 37, 42, 57, 63, 68}, {8, 15, 21, 28, 38, 42, 56, 65, 67},
  {6, 16, 20, 31, 35, 42, 54, 61, 73}, {5, 17, 22, 27, 36, 42, 53, 64, 71},
  {7, 12, 25, 29, 40, 42, 55, 59, 70}, {4, 13, 19, 33, 39, 42, 52, 62, 72},
  {3, 14, 23, 32, 41, 42, 51, 60, 69} }
```

Primjer 4. Klasična projektivna ravnina reda 8.

Propozicija 1.5. *Neka je $n \geq 2$, $v = n^2 + n + 1$, A $v \times v$ matrica čiji su elementi 0 ili 1. Označimo s I jediničnu matricu reda v, a s J kvadratnu matricu reda n čiji su svi elementi jedinice. Sljedeće tri tvrdnje su ekvivalentne:*

- $A \cdot A^\tau = nI + J$
- $A^\tau \cdot A = nI + J$
- *A je incidencijska matrica projektivne ravnine reda n.*

Definicija 1.6. Latinski kvadrat *reda n je $n \times n$ matrica popunjena brojevima $1, 2, \ldots, n$ tako da se u svakom retku i stupcu svaki od tih brojeva javlja točno jednom. Kažemo da su latinski kvadrati $L = [l_{ij}]_{i,j=1}^n$, $M = [m_{ij}]_{i,j=1}^n$* **ortogonalni** *ako se u skupu $\{(l_{ij}, m_{ij}) \mid i, j = 1 \ldots n\}$ javlja svih n^2 uređenih parova iz $\{1, \ldots, n\} \times \{1, \ldots, n\}$.*

Propozicija 1.7. *Neka je $\{L_1, \cdots, L_m\}$ skup međusobno ortogonalnih latinskih kvadrata reda n. Onda je $m \leq n - 1$.*

Propozicija tvrdi da skup međusobno ortogonalnih latinskih kvadrata reda n ima najviše $n - 1$ članova. Ako je taj skup točno $(n - 1)$–član, kažemo da je *potpun*.

Teorem 1.8. *Konačna projektivna ravnina reda n postoji ako i samo ako postoji potpun skup međusobno ortogonalnih latinskih kvadrata reda n.*

Štoviše, od konačne projektivne ravnine efektivno možemo konstruirati potpun skup latinskih kvadrata odgovarajućeg reda, i obrnuto.

Teorem 1.9. (Bruck–Ryser) *Neka je $n \equiv 1 \pmod 4$ ili $n \equiv 2 \pmod 4$. Ako postoji projektivna ravnina reda n, onda se n može prikazati kao zbroj kvadrata dva cijela broja, $n = a^2 + b^2$, $a, b \in \mathbb{Z}$.*

2 Izomorfizam i klasifikacija

2.1 Izomorfizam i povijest klasifikacije

Definicija 2.1. *Neka su* $\mathbf{P} = (\mathcal{P}, \mathcal{L}, I)$, $\mathbf{P}' = (\mathcal{P}', \mathcal{L}', I')$ *projektivne ravnine. Kažemo da su* $\mathbf{P}$ *i* $\mathbf{P}'$ **izomorfne** *ako postoje bijekcije* $\alpha : \mathcal{P} \to \mathcal{P}'$, $\beta : \mathcal{L} \to \mathcal{L}'$ *takve da* $\forall P \in \mathcal{P}$, $\forall \ell \in \mathcal{L}$ *vrijedi* $PI\ell \iff \alpha(P)I'\beta(\ell)$. *Uređen par* (α, β) *zovemo* **izomorfizam** *projektivnih ravnina.*

Jasno je da izomorfne ravnine imaju isti red. Pogledajmo kako se pojam izomorfizma prenosi na incidencijske matrice ravnina reda n. Uvedimo na skupove $\mathcal{P}$, $\mathcal{P}'$, $\mathcal{L}$, $\mathcal{L}'$ uređaj: $\mathcal{P} = (P_1, P_2, \ldots, P_v)$, $\mathcal{P}' = (P_1', P_2', \ldots, P_v')$, $\mathcal{L} = (\ell_1, \ell_2, \ldots, \ell_v)$, $\mathcal{L}' = (\ell_1', \ell_2', \ldots, \ell_v')$. Neka su $A = [a_{ij}]_{i,j=1}^{v}$, $A' = [a_{ij}']_{i,j=1}^{v}$ incidencijske matrice od $\mathbf{P}$ i $\mathbf{P}'$ redom. Neka je (α, β) izomorfizam. Istim slovima označavat ćemo i inducirane permutacije skupa $\{1, 2, \ldots, v\}$, tj. smatrat ćemo da je $\alpha(P_i) = P_{\alpha(i)}'$, $\beta(\ell_j) = \ell_{\beta(j)}'$. U tom slučaju vrijedi $a_{ij} = 1 \iff P_i I \ell_j \iff P_{\alpha(i)}' I' \ell_{\beta(j)}' \iff a_{\alpha(i)\beta(j)}' = 1$. Slijedi da je i $a_{ij} = 0 \iff a_{\alpha(i)\beta(j)}' = 0$, pa je $a_{ij} = a_{\alpha(i)\beta(j)}'$, $\forall i, j = 1 \ldots v$. Dakle, A se dobiva permutacijom redaka i stupaca matrice A'.

Vrijedi i obrat, ako se permutacijom redaka i stupaca matricu A' može transformirati u A, onda su ravnine $\mathbf{P}$ i $\mathbf{P}'$ izomorfne.

Osnovni problem kojim se bavimo je određivanje broja međusobno neizomorfnih projektivnih ravnina zadanog reda.

Za niže redove problem klasifikacije može se riješiti preko veze s ortogonalnim latinskim kvadratima, opisane u teoremu 1.8. Pokazuje se da su do na izomorfizam jedine projektivne ravnine redova 2, 3, 4 i 5 one klasične. Iz Bruck–Ryserovog teorema slijedi da ne postoji projektivna ravnina reda 6.

Dokaz da je klasična ravnina reda 7 jedinstvena objavljen je 1953. u dva članka. U [3] je dokazano da ravnina reda 7 ne može imati podravninu reda 2. Ta je činjenica iskorištena u [4] da bi se dokazalo kako je svaka ravnina reda 7 izomorfna klasičnoj. Ispravka manje greške objavljena je u [5].

Za ravninu reda 8 do danas nije poznat dokaz jedinstvenosti u klasičnom smislu. U članku [6], objavljenom 1956., dan je izvještaj o potrazi za ravninama reda 8 pomoću računala. Pronađena je samo jedna ravnina, izomorfna klasičnoj. Kako je potraga bila iscrpljujuća, tj. ispitane su sve mogućnosti, zaključeno je da je ravnina reda 8 jedinstvena.

U to doba bile su poznate četiri međusobno neizomorfne ravnine reda 9. Pitanje da li ih ima još bilo je otvoreno. U nizu članaka [7], [8], [9] objavljenih u razdoblju od 1959. do 1964. opisane su potrage za ravninama reda 9 pomoću računala. Niti jedna nije dala novu ravninu. Međutim, potrage nisu bile iscrpljujuće, jer to mogućnosti računala iz tog doba nisu dozvoljavale. Tako je problem klasifikacije za red 9 ostao otvoren.

U idućem razdoblju više se radilo na redu 10 nego 9. Naime, 10 je najmanji red za koji je čak i pitanje egzistencije projektivne ravnine bilo otvoreno. Nije bila poznata niti jedna ravnina reda 10, niti je bilo dokazano da je ona nemoguća. Pristup je bio potpuno drugačiji nego u [6], preko binarnih linearnih kodova. Samo neki od članaka iz tog razdoblja su [11]–[13]. Konačni rezultat da ne postoji ravnina reda 10 dobiven je 1989. Time je riješen i problem klasifikacije za red 10, dok je za 9 još uvijek bio otvoren.

Problem je za red 9 riješen tek u članku [10] iz 1991. Ideja za potragu bila je u biti ista kao u [6]. Napredak je omogućen bržim računalima i programskim tehnikama razvijenim u sklopu potrage za ravninom reda 10.

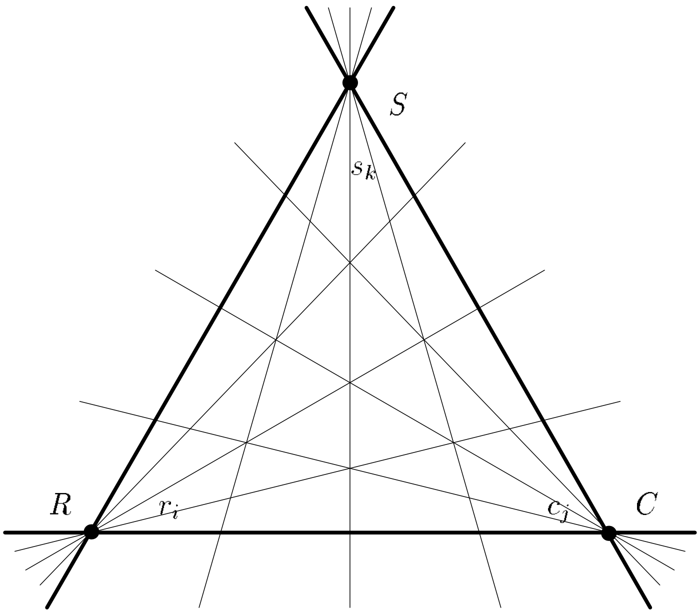
U svom radu koristim tehnike vrlo slične onima iz [10] da bih dokazao jedinstvenost ravnina reda 7 i 8. U nastavku ću skicirati osnovne ideje za klasifikaciju, zajedničke radovima [6] i [10]. Detaljniji opis algoritama, korištenih programa i rezultata potrage nalazi se u poglavljima 3 i 4.

2.2 Ideja za klasifikaciju

Postoje mnoge duboke veze između konačnih projektivnih ravnina i latinskih kvadrata. Jedan način da od ravnine reda n dobijemo latinske kvadrate spomenut je u teoremu 1.8. Dobivamo $n-1$ latinskih kvadrata reda n, koji su međusobno ortogonalni. Ideja za klasifikaciju ravnina većih redova temelji se na drugoj konstrukciji, pomoću koje od ravnine reda n dobivamo jedan latinski kvadrat reda $n-1$.

Neka su R, C, S tri nekolinearne točke ravnine $\mathbf{P}$. Označimo pravce kroz točku R različite od $R \vee C$ i $R \vee S$ simbolima $r_1, r_2, \ldots, r_{n-1}$. Pravce kroz C različite od $C \vee R$ i $C \vee S$ označimo s $c_1, c_2, \ldots, c_{n-1}$, a pravce kroz S različite od $S \vee R$ i $S \vee C$ sa $s_1, s_2, \ldots, s_{n-1}$. Latinski kvadrat $L = [l_{ij}]_{i,j=1}^{n-1}$ definiramo na sljedeći način: $l_{ij} = k$ ako i samo ako je $(r_i \wedge c_j) \vee S = s_k$, dakle ako je spojnica točke S sa sjecištem pravaca r_i i c_j označena simbolom s_k.

Nije teško pokazati da je L zaista latinski kvadrat. Za različite indekse j, j' pravci $c_j, c_{j'}$ su različiti, pa su i točke $r_i \wedge c_j, r_i \wedge c_{j'}$ različite. Kada bi se spojnice $(r_i \wedge c_j) \vee S, (r_i \wedge c_{j'}) \vee S$ podudarale, to bi značilo da sve tri točke R, C, S leže na pravcu $(r_i \wedge c_j) \vee (r_i \wedge c_{j'})$, što nije istina. Tako smo vidjeli da se u svakom retku od L svaki od simbola $1, 2, \ldots, n-1$ javlja najviše jednom, a ista tvrdnja vrijedi i za stupce.



Slika 1.

Temeljna ideja za klasifikaciju ravnina je da obrnemo tu konstrukciju. Ako imamo latinski kvadrat L možemo bez smanjenja općenitosti popuniti $2n+1$ redaka i $3n$ stupaca incidencijske matrice ravnine $\mathbf{P}$, kako će biti opisano u četvrtom poglavlju. To je dovoljno da računalu prepustimo popunjavanje preostalog dijela matrice na sve moguće načine. Dobit ćemo dovoljno mali broj potpunih incidencijskih matrica da ih sve možemo usporediti.

Ako želimo dobiti sve ravnine reda n morat ćemo konstrukciju provesti nad svim latinskim kvadratima reda $n-1$. Zato je važno potpunu listu latinskih kvadrata od koje krećemo što je moguće više reducirati.

To ćemo napraviti identifikacijom kvadrata čije razlike ovise samo o proizvoljnim elementima konstrukcije od L : označavanju pravaca kroz R, C i S i označavanju samih točaka R, C, S. Neka su L, L' dobiveni od iste ravnine $\mathbf{P}$ uz isti izbor točaka R, C, S, a razlikuju se npr. samo zbog drugačijeg označavanja pravaca kroz R ($r_1, r_2, \ldots, r_{n-1}$ i $r'_1, r'_2, \ldots, r'_{n-1}$). Onda ćemo popunjavajući na sve dozvoljene načine incidencijske matrice dobivene od L i L' dobiti istu ravninu (odn. ravnine), pa iz početne liste kvadrata možemo izbaciti L'.

Različitim označavanjima pravaca kroz R odgovaraju permutacije redaka kvadrata L, različitim označavanjima pravaca kroz C permutacije stupaca, a pravaca kroz S permutacije simbola kvadrata L. Kvadrate koji se ovim operacijama mogu pretvoriti jedan u drugi smatrat ćemo ekvivalentnim. Naša će početna lista sadržati samo po jednog predstavnika iz svake klase ekvivalencije.

Ali i sama uloga točaka R, C, S u biti je proizvoljna. Kad jednom odaberemo tri nekolinearne točke u ravnini $\mathbf{P}$, svaka od njih može poslužiti za definiranje redaka, stupaca ili simbola kvadrata L. Zamjena uloga točaka R i C odgovara zamjeni redaka i stupaca kvadrata L, transpoziciji. Zamjena uloga točaka R i S odgovara zamjeni redaka i simbola kvadrata L. Smisao toga bit će opisan u trećem poglavlju. Slična tvrdnja vrijedi i za zamjenu C i S. Ukupni broj permutacija na simbolima R, C i S je $3! = 6$. Mi ćemo identificirati kvadrate koji se mogu pretvoriti jedan u drugi bilo kojom od tih transformacija, takozvanim operacijama *konjugiranja*.

Potraga za svim mogućim ravninama reda n podijeljena je u sljedeće korake:

1. Generiranje potpune liste latinskih kvadrata reda $n-1$ koji se ne mogu pretvoriti jedan u drugi permutacijama redaka, stupaca i simbola, niti operacijama konjugiranja.

2. Prevođenje svakog od tih kvadrata u prvih $2n+1$ redaka i $3n$ stupaca incidencijske matrice od $\mathbf{P}$ i popunjavanje preostalog dijela na sve moguće načine.

3. Uspoređivanje svih tako dobivenih potpunih incidencijskih matrica i izbacivanje onih među njima koje predstavljaju izomorfne ravnine.

Korak 1 opisan je u trećem, a koraci 2 i 3 u četvrtom poglavlju.

3 Latinski kvadrati

3.1 Klase izotopije i glavne klase

Definicija 3.1. *Za latinski kvadrat $L = [l_{ij}]_{i,j=1}^n$ kažemo da je* **reduciran** *ako su elementi u prvom retku i stupcu poredani u prirodnom poretku, tj. ako je $l_{1i} = l_{i1} = i$, $\forall i = 1, 2, \ldots, n$. Broj latinskih kvadrata reda n označavamo s U_n, a broj reduciranih kvadrata s T_n.*

Propozicija 3.2. $U_n = n!(n-1)!\ T_n$

Dokaz. Neka je $n \in \mathbb{N}$ zadan. Skup svih latinskih kvadrata reda n označimo s $\mathcal{U}$, a skup reduciranih kvadrata s $\mathcal{T} = \{R_1, R_2, \ldots, R_k\}$, gdje je $k = T_n$. Neka je $R_l = [r_{ij}^{(l)}]_{i,j=1}^n$, $l = 1, 2, \ldots, k$. Definiramo skupove

$$\mathcal{T}_l = \{\ [r_{\alpha(i)\beta(j)}^{(l)}]_{i,j=1}^n \mid \alpha, \beta \in S_n, \alpha(1) = 1\ \}, \qquad l = 1, 2, \ldots, k$$

(S_n je grupa permutacija skupa $\{1, 2, \ldots, n\}$). To su skupovi latinskih kvadrata od kojih svaki ima $n!(n-1)!$ elemenata. Formula će vrijediti ako su oni disjunktni i u uniji daju cijeli $\mathcal{U}$.

Druga tvrdnja se lako provjerava. Za proizvoljni $L = [l_{ij}]_{i,j=1}^n \in \mathcal{U}$ permutacijom svih stupaca i redaka od drugog do n-tog možemo postići da bude reduciran (uzlazno sortiramo stupce, zatim retke). Točnije, postoje $\gamma, \delta \in S_n$, $\gamma(1) = 1$ takvi da je $[l_{\gamma(i)\delta(j)}]_{i,j=1}^n = R_l$ za neki $l \in \{1, 2, \ldots k\}$. Očito za $\alpha = \gamma^{-1}$, $\beta = \delta^{-1}$ vrijedi $L = [r_{\alpha(i)\beta(j)}^{(l)}]_{i,j=1}^n$, dakle $L \in \mathcal{T}_l$. Tako smo vidjeli da je $\mathcal{U} = \cup_{i=1}^k \mathcal{T}_i$.

Još treba provjeriti da su $\mathcal{T}_1, \ldots, \mathcal{T}_k$ međusobno disjunktni. Neka su $l, l' \in \{1, 2, \ldots, k\}$ različiti. Za proizvoljne kvadrate

$$A = [r_{\alpha(i)\beta(j)}^{(l)}]_{i,j=1}^n \in \mathcal{T}_l, \quad A' = [r_{\alpha'(i)\beta'(j)}^{(l')}]_{i,j=1}^n \in \mathcal{T}_{l'}$$

($\alpha, \alpha', \beta, \beta' \in S_n$, $\alpha(1) = \alpha'(1) = 1$) pokazat ću da su različiti. Imamo tri slučaja:

(1) $\beta \neq \beta'$, recimo $\beta(j) \neq \beta'(j)$ za $j \in \{1, 2, \ldots, n\}$. Onda je

$$r_{\alpha(1)\beta(j)}^{(l)} = r_{1\beta(j)}^{(l)} = \beta(j) \neq \beta'(j) = r_{1\beta'(j)}^{(l')} = r_{\alpha'(1)\beta'(j)}^{(l')},$$

pa se A i A' razlikuju na mjestu $(1, j)$.

(2) $\beta = \beta'$, $\alpha \neq \alpha'$, recimo $\alpha(i) \neq \alpha'(i)$ za $i \in \{2, 3, \ldots, n\}$. Ako je $q = \beta^{-1}(1)$ imamo

$$r^{(l)}_{\alpha(i)\beta(q)} = r^{(l)}_{\alpha(i)1} = \alpha(i) \neq \alpha'(i) = r^{(l')}_{\alpha'(i)1} = r^{(l')}_{\alpha'(i)\beta'(q)}.$$

U ovom slučaju A i A' razlikuju se na mjestu $(i, q) = (i, \beta^{-1}(1))$.

(3) $\beta = \beta'$, $\alpha = \alpha'$. Zbog $l \neq l'$ kvadrati R_l, $R_{l'}$ su različiti, recimo na mjestu (i, j), $i, j \in \{2, 3, \ldots, n\}$, tj. $r^{(l)}_{ij} \neq r^{(l')}_{ij}$. Uz oznake $p = \alpha^{-1}(i)$, $q = \beta^{-1}(j)$ vrijedi

$$r^{(l)}_{\alpha(p)\beta(q)} = r^{(l)}_{ij} \neq r^{(l')}_{ij} = r^{(l')}_{\alpha'(p)\beta'(q)}.$$

Dakle, A i A' se razlikuju na mjestu $(p, q) = (\alpha^{-1}(i), \beta^{-1}(j))$.

■

Problem određivanja veličine T_n je jako težak. Danas su poznate samo vrijednosti za $n \leq 10$, navedene u tablici.

n	T_n	Ω_n	M_n
1	1	1	1
2	1	1	1
3	1	1	1
4	4	2	2
5	56	2	2
6	9408	22	12
7	16942080	564	147
8	535281401856	1676267	283657
9	377597570964258816	?	?
10	7580721483160132811489280	?	?

Tablica 1.

Za $n \leq 9$ podaci se nalaze u knjizi [14]. T_{10} je izračunat 1995. godine, u radu [16], a može ga se naći i u knjizi [15]. Pomoću prvog programa iz dodatka 2 (`reduced`) u razumnom se vremenu mogu dobiti brojevi $T_1, T_2, \ldots, T_7$. Program je, kao i svi ostali korišteni u radu, pisan u programskom jeziku C. Definicija programskog jezika C nalazi se u [20]. Koristio sam i knjigu [21].

Prije nego što nastavim izlaganje uvest ću nekoliko oznaka. Ako je $A = [a_{ij}]_{i,j=1}^n$ latinski kvadrat, element na mjestu (i,j) označavat ćemo s $A(i,j) = a_{ij}$. $A(i)$ će označavati i–ti redak matrice A shvaćen kao permutacija, tj.

$$A(i) = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \end{pmatrix} \in S_n$$

Definicija 3.3. **Latinski pravokutnik** *tipa* $l \times n$ $(l, n \in \mathbb{N},\ l \leq n)$ *je matrica* $L = [l_{ij}]_{i=1}^{l}{}_{j=1}^{n}$ *čiji su elementi brojevi* $1, 2, \ldots, n$, *takva da se u svakom retku i stupcu svaki od tih brojeva javlja najviše jednom.*

Za latinske pravokutnike upotrebljavat ćemo iste oznake:

$$L(i,j) = l_{ij}, \qquad L(i) = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ l_{i1} & l_{i2} & l_{i3} & \cdots & l_{in} \end{pmatrix} \in S_n$$

Definicija 3.4. *Neka su* L, M *latinski pravokutnici tipa* $l \times n$. *Kažemo da su* L *i* M **izotopni** *ako postoje permutacije* $r \in S_l$, $c, s \in S_n$ *takve da je* $M(i,j) = s(\,L(r(i), c(j))\,)$, $\forall i = 1 \ldots l$, $\forall j = 1 \ldots n$. *Uređenu trojku* (r, c, s) *zovemo* **izotopija** *pravokutnika* L *i* M. *Ako je* $L = M$ *kažemo da je* (r, c, s) **autotopija** *od* L.

Pokazuje se da je "biti izotopan" relacija ekvivalencije na skupu svih latinskih pravokutnika zadanog tipa. Pripadne klase ekvivalencije zovemo *klase izotopije*. Broj klasa izotopije latinskih kvadrata reda n (pravokutnika tipa $n \times n$) označavat ćemo s Ω_n. Skup svih autotopija danog latinskog pravokutnika je grupa, uz prirodnu operaciju "kompozicije po komponentama",

$$(r, c, s) \circ (r', c', s') = (r \circ r', c \circ c', s \circ s').$$

Na latinski pravokutnik tipa $l \times n$ možemo gledati kao na uređenu l–torku permutacija iz S_n : $L = (\,L(1), L(2), \ldots, L(l)\,)$. U toj interpretaciji izotopiju (r, c, s) pravokutnika L i M zapisujemo kao $M(i) = s \circ L(r(i)) \circ c$, $\forall i = 1 \ldots l$.

Sad ćemo preciznije objasniti operacije konjugiranja latinskog kvadrata, spomenute u prethodnom poglavlju. Latinskom kvadratu A reda n možemo pridružiti ternarnu relaciju nad $\{1, 2, \ldots, n\}$, tj. podskup od $\{1, 2, \ldots, n\}^3$:

$$A \longmapsto \tilde{A} = \{\ (i, j, A(i,j)) \mid i, j = 1 \ldots n\ \}$$

Djelovanje grupe S_3 na uređenim trojkama definira se s

$$(x_1, x_2, x_3)^\alpha = (x_{\alpha(1)}, x_{\alpha(2)}, x_{\alpha(3)}), \quad \alpha \in S_3.$$

Definicija se proširuje na skupove uređenih trojki:

$$X = \{\, (x_1^i, x_2^i, x_3^i) \mid i \in I \,\}, \quad X^\alpha = \{\, (x_1^i, x_2^i, x_3^i)^\alpha \mid i \in I \,\}$$

Tako od ternarne relacije $\tilde{A}$ možemo napraviti šest ternarnih relacija $\tilde{A}^\alpha$, $\alpha \in S_3$. Svakoj od njih odgovara latinski kvadrat $\tilde{A}^\alpha \mapsto A^\alpha$, koji zovemo *konjugat* kvadrata A. Dokaz da zaista dobivamo šest latinskih kvadrata nalazi se u [17].

Ako za α uzmemo permutaciju $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$, dobit ćemo transponirani kvadrat, $A^\alpha = A^\tau$. Pogledajmo kako se na kvadrat odražava konjugiranje permutacijom $\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$, "zamjena stupaca i simbola". Konjugirani kvadrat ima svojstvo $A^\alpha(i, A(i,j)) = j$, $\forall i, j$. U terminima permutacija to zapisujemo kao $[A^\alpha(i) \circ A(i)]\,(j) = j$, $\forall i, j \iff A^\alpha(i) \circ A(i) = id$, $\forall i \iff A^\alpha(i) = A^{-1}(i)$, $\forall i$. Vidimo da A^α dobivamo invertiranjem redaka matrice A (shvaćenih kao permutacije). Slično bi vidjeli da je konjugiranje s $\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$ ("zamjena redaka i simbola") zapravo invertiranje stupaca od A.

Definicija 3.5. *Za latinske kvadrate A, B reda n kažemo da* **pripadaju istoj glavnoj klasi** *ako postoji $\beta \in S_3$ takav da je A izotopan konjugiranom kvadratu B^β.*

"Pripadati istoj glavnoj klasi" je relacija ekvivalencije, čije klase nazivamo *glavne klase* latinskih kvadrata. Očito vrijedi implikacija : A izotopan s $B \Longrightarrow A$ i B pripadaju istoj glavnoj klasi. Iz toga slijedi da se svaka glavna klasa sastoji od jedne ili više klasa izotopije.

U drugom poglavlju objašnjeno je da za klasifikaciju projektivnih ravnina reda $n+1$ trebamo po jedan predstavnik svake glavne klase latinskih kvadrata reda n. Prvo ćemo generirati predstavnike klasa izotopije, koje ćemo reducirati izbacivanjem kvadrata koji pripadaju istoj glavnoj klasi.

3.2 Invarijante izotopije

Ideja za dobivanje predstavnika klasa izotopije preuzeta je iz članka J.W.Browna [18]. Da bih objasnio algoritam, uvest ću nekoliko invarijanti izotopije latinskih pravokutnika.

Definicija 3.6. *Neka je L latinski pravokutnik tipa $l \times n$. Označimo s k broj redaka od l koji su parni (kao permutacije), a s h broj neparnih redaka ($k + h = l$). Uređen par* $(\max\{h,k\}, \min\{h,k\})$ *zovemo* **vrsta** *pravokutnika L.*

Do na izotopiju možemo pretpostaviti da je $k \geq h$ i da su prvih k redaka od L parni. U slučaju da je $k < h$ djelujemo na L izotopijom $(id, (1\ 2), id)$ (zamijenimo prvi i drugi stupac). To je ekvivalentno množenju zdesna redaka od L transpozicijom $(1\ 2)$, nakon čega oni mijenjaju predznak i dobivamo $k > h$. Permutacijom redaka od L postižemo da parni reci dolaze prvi.

Jasno je da se izotopni pravokutnici podudaraju u vrsti, tj. vrsta je invarijanta izotopije.

Definicija 3.7. *Neka je L latinski pravokutnik tipa $l \times n$. Označimo s $[i,j]_L = [i,j]$ permutaciju koja i–ti redak od L prevodi u j–ti, $[i,j] \circ L(i) = L(j)$ ($\implies$ $[i,j] = L(j) \circ L(i)^{-1}$). Niz*

$$([1,2], [1,3], [2,3], [1,4], [2,4], [3,4], \ldots, [l-1, l])$$

zovemo **potpuna prezentacija** *pravokutnika L.*

Definicija 3.8. *Neka permutacija $p \in S_n$ u svom cikličkom rastavu ima k_1 ciklusa duljine 1, k_2 ciklusa duljine 2, …, k_n ciklusa duljine n. Vektor $(k_1, k_2, \ldots, k_n)$ zovemo* **ciklički tip** *permutacije p (u Brownovoj terminologiji cycle pattern).*

Cikličkim tipovima permutacija iz S_n odgovaraju neuređene particije broja n. Nas će zanimati ciklički tipovi permutacija bez fiksnih točaka (točnije, permutacija $[i,j]$ iz potpune prezentacije latinskog pravokutnika), za koje je $k_1 = 0$. Njima odgovaraju particije broja n na pribrojnike ≥ 2.

Lema 3.9. *Za svake dvije permutacije $p, s \in S_n$ ciklički tipovi od p i $s \circ p \circ s^{-1}$ su jednaki, tj. ciklički tip se ne mijenja konjugiranjem permutacije.*

Dokaz. Ako je $p = c_1 \circ c_2 \circ \cdots \circ c_m$ ciklički rastav od p, onda je $s \circ p \circ s^{-1} = (s \circ c_1 \circ s^{-1}) \circ (s \circ c_2 \circ s^{-1}) \circ \cdots \circ (s \circ c_m \circ s^{-1})$. Zato je dovoljno dokazati da je za svaki ciklus c duljine k permutacija $s \circ c \circ s^{-1}$ također ciklus duljine k. Neka je $c = (a_1\ a_2 \cdots a_k)$. Označimo s $b_i = s(a_i)$, $i = 1 \ldots k$. Onda je $(s \circ c \circ s^{-1})(b_i) = s(c(s^{-1}(b_i))) = s(c(a_i)) = s(a_{i+1}) = b_{i+1}$, $i = 1 \ldots k-1$, te $(s \circ c \circ s^{-1})(b_k) = b_1$. Dakle, $s \circ c \circ s^{-1} = (b_1\ b_2 \cdots b_k)$. Treba još primijetiti da iz disjunktnosti ciklusa $c_1, \ldots, c_m$ slijedi da su ciklusi $s \circ c_1 \circ s^{-1}, \ldots, s \circ c_m \circ s^{-1}$ disjunktni, pa se zaista radi o cikličkom rastavu od $s \circ p \circ s^{-1}$.

■

Lema 3.10. *Za svaku permutaciju $p \in S_n$ ciklički tipovi od p i p^{-1} su jednaki, tj. ciklički tip se ne mijenja invertiranjem permutacije.*

Dokaz. Dovoljno je vidjeti da je za ciklus c duljine k permutacija c^{-1} također ciklus duljine k. Ako je $c = (a_1\ a_2 \cdots a_k)$, očito je $c^{-1} = (a_k\ a_{k-1} \cdots a_1)$.

■

Definicija 3.11. *Neka je L latinski pravokutnik tipa $l \times n$. Označimo s $1, 2, \ldots, m$ različite cikličke tipove permutacija bez fiksnih točaka iz S_n. Za svaki $i \in \{1, 2, \ldots, m\}$ neka je c_i broj permutacija u potpunoj prezentaciji od L koje imaju i–ti ciklički tip. Vektor $(c_1, c_2, \ldots, c_m)$ zovemo* **vektor cikličkih tipova** *pravokutnika L (cycle pattern count vector, CPC).*

Propozicija 3.12. *Izotopni latinski pravokutnici imaju isti vektor cikličkih tipova, tj. vektor cikličkih tipova je invarijanta izotopije.*

Dokaz. Neka su pravokutnici L, M izotopni, $M(i) = s \circ L(r(i)) \circ c$, $\forall i = 1 \ldots l$, za $r \in S_l$, $c, s \in S_n$. Ako je $([1,2]_L, [1,3]_L, [2,3]_L, \ldots, [l-1, l]_L)$ potpuna prezentacija od L, za potpunu prezentaciju od M vrijedi $[i,j]_M = M(j) \circ M(i)^{-1} = (s \circ L(r(j)) \circ c) \circ (s \circ L(r(i)) \circ c)^{-1} = s \circ L(r(j)) \circ L(r(i))^{-1} \circ s^{-1} = s \circ [r(i), r(j)]_L \circ s^{-1}$. Ako identificiramo permutaciju i njoj inverznu, što smijemo zbog leme 10, nizovi $(\, [i,j]_L\,)_{i<j}$ i $(\, [r(i), r(j)]_L\,)_{i<j}$ razlikuju se samo u poretku. Permutacije $[r(i), r(j)]_L$ i $[i,j]_M = s \circ [r(i), r(j)]_L \circ s^{-1}$ imaju po lemi 9 isti ciklički tip, pa je broj permutacija danog cikličkog tipa

u nizovima $([i,j]_L)_{i<j}$ i $([i,j]_M)_{i<j}$ isti.

■

Definicija 3.13. *Neka je L latinski pravokutnik tipa $l \times n$, $k = \binom{l}{2}$, $(p_1, p_2, \ldots, p_k)$ njegova potpuna prezentacija. Neka je $p_i = c_{i1} \circ c_{i2} \circ \cdots \circ c_{im_i}$ ciklički rastav permutacije p_i, $i = 1 \ldots k$. Za $1 \le i \le n$, $2 \le l \le n$ označimo s a_{ij} broj ciklusa duljine j među $c_{11}, c_{12}, \ldots, c_{1m_1}$, $c_{21}, c_{22}, \ldots, c_{2m_2}$, $\ldots$, $c_{k1}, c_{k2}, \ldots, c_{km_k}$ koji sadrže simbol i. Vektor $a_i = (a_{i2}, a_{i3}, \ldots, a_{in})$ zovemo* **vektor ciklusa** *simbola i (digit cycle vector, DCV).*

Propozicija 3.14. *Multiskup vektora ciklusa $\{a_1, a_2, \ldots, a_n\}$ je invarijanta izotopije, tj. podudara se za izotopne pravokutnike.*

Dokaz. Neka su L, M izotopni pravokutnici, $M(i) = s \circ L(r(i)) \circ c$, $i = 1 \ldots l$, $r \in S_l$, $c, s \in S_n$. Ako su $c_{11}, c_{12}, \ldots, c_{1m_1}, \ldots, c_{k1}, c_{k2}, \ldots, c_{km_k}$ ciklusi u potpunoj prezentaciji od L, tada iz leme 9, leme 10 i prethodnog dokaza slijedi da su $s \circ \tilde{c}_{11} \circ s^{-1}, s \circ \tilde{c}_{12} \circ s^{-1}, \ldots, s \circ \tilde{c}_{1m_1} \circ s^{-1}, \ldots, s \circ \tilde{c}_{k1} \circ s^{-1}, s \circ \tilde{c}_{k2} \circ s^{-1}, \ldots, s \circ \tilde{c}_{km_k} \circ s^{-1}$ ciklusi u potpunoj prezentaciji od M, gdje je $\tilde{c}_{ij} = c_{ij}$ ili c_{ij}^{-1}. U dokazu leme 10 vidjeli smo da c_{ij} i $\tilde{c}_{ij}$ sadrže iste simbole, jedino možda u drugom poretku. Za proizvoljni simbol $i \in \{1, 2, \ldots, n\}$ i ciklus $\tilde{c}$ koji ga sadrži iz dokaza leme 9 znamo da ciklus $s \circ \tilde{c} \circ s^{-1}$ sadrži simbol $s(i)$. Iz rečenoga slijedi da, ako su $a_1, a_2, \ldots a_n$ vektori ciklusa u pravokutniku L, a $b_1, b_2, \ldots b_n$ u M, vrijedi $a_i = b_{s(i)}$, $\forall i = 1 \ldots n$. Prema tome, multiskupovi $\{a_1, a_2, \ldots, a_n\}$ i $\{b_1, b_2, \ldots, b_n\}$ su jednaki.

■

Napomena 3.15. Iz dokaza se vidi i više, da treća komponenta izotopije (r, c, s) pridružuje simbole pravokutnika L i M s istim vektorima ciklusa. To se može iskoristiti da bi ubrzali ispitivanje izotopnosti dva pravokutnika. Treba provjeriti samo one permutacije $s \in S_n$ koje pridružuju simbole s odgovarajućim vektorima ciklusa.

3.3 Generiranje predstavnika

Ideja Brownovog članka je proširivanje pravokutnika redak po redak i izbacivanje izotopnih u svakom koraku. Krenemo od pravokutnika tipa $1 \times n$, proširimo ga na sve moguće načine do pravokutnika tipa $2 \times n$ i izbacimo međusobno izotopne. Zatim preostale proširimo do $3 \times n$ pravokutnika, izbacimo međusobno izotopne itd. Nakon n koraka dobijemo predstavnike klasa izotopije latinskih kvadrata reda n.

Posao se može smanjiti korištenjem prve invarijante, vrste pravokutnika. U pravokutniku vrste (k, h), $k \geq h$ do na izotopiju prvih k redaka su parni. To znači da u prvih $\left\lceil \frac{n}{2} \right\rceil$ koraka možemo generirati samo parne pravokutnike, takve kojima su svi reci parni ($\lceil\ \rceil$ je funkcija "strop", $\lceil x \rceil$ = najmanji cijeli broj veći ili jednak x). Algoritam za $n = 7$ izgleda ovako:

1. Generiraj parne pravokutnike tipa 1×7. Do na izotopiju to je samo redak $[1\ 2 \cdots 7]$.

2. Proširi taj pravokutnik na sve dozvoljene načine parnim retkom. Izbaci izotopne među dobivenim 2×7 pravokutnicima.

3. Proširi te pravokutnike parnim retkom i izbaci izotopne. Dobiju se svi parni 3×7 pravokutnici (do na izotopiju).

4. Na analogan način generiraj sve međusobno neizotopne parne pravokutnike tipa 4×7.

5. Proširi pravokutnike iz prethodnog koraka na sve dozvoljene načine s tri neparna retka, tj. nadopuni ih do kvadrata vrste $(4, 3)$. Izbaci izotopne među njima.

6. Proširi pravokutnike iz četvrtog koraka jednim parnim retkom. U ovom koraku ne vrši se izbacivanje izotopnih pravokutnika.

7. Proširi sve dobivene pravokutnike s dva proizvoljna retka (ne nužno neparna) do kvadrata i izbaci izotopne. Dobijemo sve kvadrate vrsta $(5, 2)$, $(6, 1)$ i $(7, 0)$.

8. Spoji rezultate petog i sedmog koraka u potpunu listu predstavnika klasa izotopije.

U šestom koraku više se ne isplati izbacivati izotopne pravokutnike. Svaki od 5×7 pravokutnika može se samo na nekoliko načina dopuniti do latinskog kvadrata. U sedmom koraku eliminiraju se izotopi među tako dobivenim kvadratima. Algoritam je za kvadrate reda 6 vrlo sličan. Proširivanje parnim retkom vrši se do dimenzije 4×6, a nakon toga se nadopuni do kvadrata.

Deveti korak algoritma je reduciranje predstavnika klasa izotopije na predstavnike glavnih klasa. Algoritam je realiziran u obliku shell skripta iz drugog odjeljka dodatka 2. Izvođenjem skripta dobivamo predstavnike klasa izotopije u datotekama `iso6` i `iso7`, te predstavnike glavnih klasa u datotekama `main6` i `main7`. Skripte pozivaju niz programa, također navedenih u dodatku. U nastavku ću objasniti te programe.

Za proširivanje parnim retkom zadužen je program 7.4, `evenexpand`. On koristi neke od funkcija iz biblioteke `permutations`, navedene u 7.3. Prilično je jednostavan i neću ga šire obrazlagati.

Program 7.5 (`oddcomplete`) nadopunjuje latinske pravokutnike do kvadrata neparnim recima, a program 7.6 (`complete`) proizvoljnim recima. Oba programa koriste biblioteku `permutations`. Vrlo su slični, a zasnivaju se na proširivanju pravokutnika redak po redak i backtrackingu.

Najsloženiji program u ovoj grupi je sedmi po redu, `isofilter`. On uzima niz latinskih pravokutnika s ulaza i vraća neizotopne među njima. Program se zasniva na brzoj funkciji za provjeru izotopnosti dva pravokutnika, `isotest`. Ipak, ključni faktor koji ga čini dovoljno brzim je korištenje invarijanti izotopije. Pravokutnici na ulazu dijele se u skupine s istim invarijantama. Testiranje izotopnosti vrši se samo unutar skupina, jer pravokutnici s različitim invarijantama sigurno nisu izotopni.

Ako pretpostavimo da je vrijeme potrebno za uspoređivanje dva kvadrata konstantno, time smo postigli da je vrijeme izvođenja proporcionalno količini pravokutnika na ulazu. Svaki novi pravokutnik treba usporediti samo s nekoliko zapamćenih, ne sa svima. Bez korištenja invarijanti vrijeme bi raslo otprilike kvadratno.

Faktor proporcionalnosti ovisi o veličini najveće skupine pravokutnika s istim invarijantama. Naravno, s povećanjem količine pravokutnika na ulazu rast će i taj faktor, ali se to može riješiti uvođenjem novih invarijanti. Za $n \leq 7$ bile su dovoljne dvije opisane, CPC (vektor cikličkih tipova) i DCV (multiskup vektora ciklusa). Brown je za prebrojavanje kvadrata reda 8 koristio još jednu invarijantu.

Prvo ću objasniti kako radi funkcija `isotest`. Uvest ću još jedan naziv: za pravokutnik L tipa $l \times n$ niz $([1,1]_L, [1,2]_L, \ldots, [1,l]_L) = (\,L(i) \circ L(1)^{-1}\,)_{i=1}^{l}$ zovemo *glavna prezentacija* (principle presentation). Radi se zapravo o izotopu pravokutnika L kojem smo normalizirali prvi redak.

Provjera izotopnosti pravokutnika L i M "grubom silom" sastoji se od uspoređivanja L sa svih $l! \cdot (n!)^2$ slika od M pod djelovanjem transformacija iz $S_l \times S_n \times S_n$. Za to današnja računala nisu dovoljno brza. Nasreću, provjeru možemo ubrzati.

Propozicija 3.16. *Pravokutnici L i M su izotopni ako i samo ako postoje permutacije $r \in S_l$, $s \in S_n$ takve da za njihove glavne prezentacije vrijedi*

$$M(i) \circ M(1)^{-1} = s \circ L(r(i)) \circ L(r(1))^{-1} \circ s^{-1}, \quad \forall i = 1 \ldots l.$$

Dokaz. L i M su izotopni $\iff \exists\, r \in S_l,\ c, s, \in S_n$ takvi da je $M(i) = s \circ L(r(i)) \circ c$, $\forall i = 1 \ldots l \iff M(i) \circ M(1)^{-1} = s \circ L(r(i)) \circ c \circ (s \circ L(r(1)) \circ c)^{-1} = s \circ L(r(i)) \circ L(r(1))^{-1} \circ s^{-1}$, $\forall i = 1 \ldots l$ $(c = A(r(1))^{-1} \circ s^{-1} \circ B(1))$. $\blacksquare$

Ako kvadrate prebacimo u glavne prezentacije, treba ispitati $l! \cdot n!$ slučajeva umjesto $l! \cdot (n!)^2$. To se može još ubrzati. Sljedeći algoritam umjesto faktora $l!$ ima faktor koji polinomijalno ovisi o l.

Napravi glavnu prezentaciju od L, $\tilde{L}(i) = L(i) \circ L(1)^{-1}$, $i = 1 \ldots l$.

<u>Za</u> $r = 1, 2, \ldots, l$

Napravi glavnu prezentaciju od M s retkom r kao prvim retkom, $\tilde{M}(i) = M(i) \circ M(r)^{-1}$, $i = 1 \ldots l$.

<u>Za</u> sve $s \in S_n$

Označi sve $\tilde{M}(j)$–ove kao neiskorištene.

<u>Za</u> $i = 1, 2, \ldots, n$

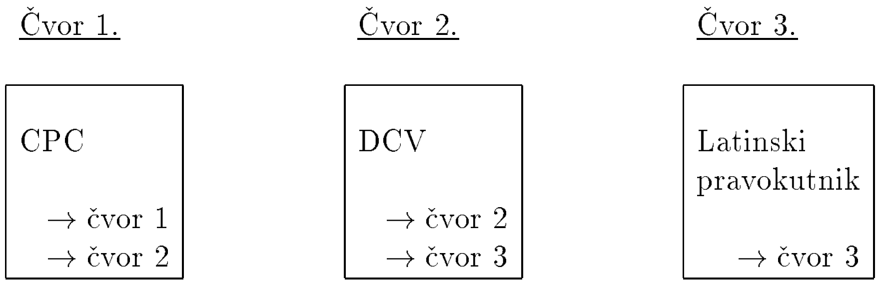
Traži među neiskorištenim $\tilde{M}(j)$–ovima permutaciju $s \circ \tilde{L}(i) \circ s^{-1}$. Ako je nađeš, označi taj $\tilde{B}(j)$ kao iskorišten i nastavi s nutarnjom petljom. Ako je ne nađeš, izađi iz nutarnje petlje i pokušaj sa sljedećom permutacijom s.

<u>kraj</u> kada se u nutarnjoj petlji dođe do kraja ($i = n$; u tom slučaju pronađena je izotopija) ili kada se pređu sve permutacije $s \in S_n$.

<u>kraj</u> kada se pronađe izotopija ili kada r dođe do l. U tom slučaju pravokutnici nisu izotopni.

Algoritam bi se mogao još ubrzati. U skladu s napomenom 15, nije nužno provjeravati sve permutacije $s \in S_n$, nego samo one koje pridružuju simbole s istim vektorima ciklusa. To za $n \leq 7$ nije bilo potrebno, pa sam funkciju `isotest` napisao na osnovi gornjeg algoritma.

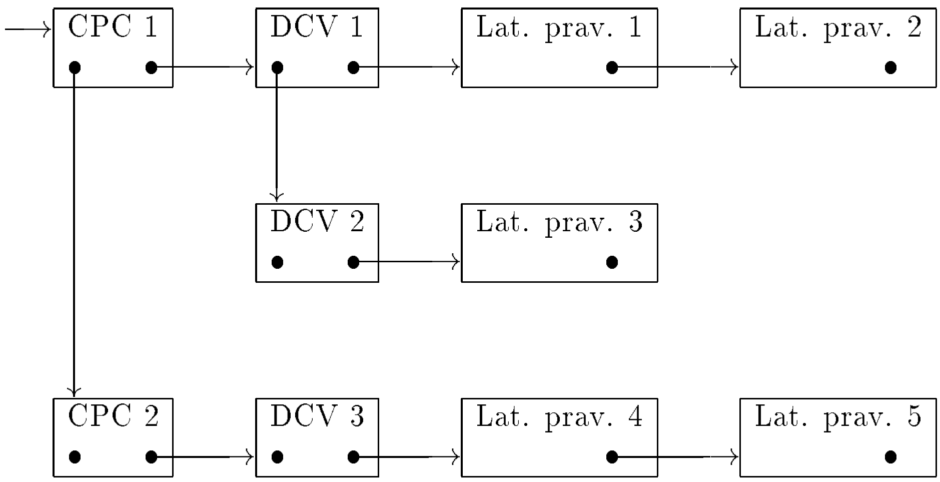
Ideju o dijeljenju pravokutnika u skupine s istim invarijantama ostvario sam korištenjem jedne specijalne strukture podataka. Radi se o stablu s tri vrste čvorova.



Slika 2.

Čvor prvog tipa sadrži invarijantu CPC i pokazivače na čvor prvog tipa i čvor drugog tipa. Čvor drugog tipa sadrži invarijantu DCV, pokazivač na čvor drugog tipa i pokazivač na čvor trećeg tipa. Treći tip čvora sadrži latinski pravokutnik i pokazivač na idući čvor istog tipa.

Tipično stablo izgleda ovako:



Slika 3.

Kad `isofilter` učita novi latinski pravokutnik, on mu najprije izračuna invarijante. Zatim uspoređuje njegov CPC s već zapamćenim (pomiče se vertikalno po stablu). Ako ga ne nađe, na dno stabla dodaje novi CPC, pripadni

DCV i latinski pravokutnik. Ako ga nađe, pomiče se "desno", na pripadnu skupinu DCV-ova. Zatim uspoređuje DCV-ove iz te skupine s DCV-om učitanog kvadrata (pomiče se vertikalno po stablu u drugom stupcu). Ako ne nađe novi DCV, dodaje ga na kraj skupine, zajedno s novim pravokutnikom. Ako ga nađe, pomiče se desno, na skupinu latinskih pravokutnika s istim invarijantama. Tek tu se vrši provjera izotopnosti s kvadratima iz skupine. Ako je novi pravokutnik izotopan nekom iz skupine, obrađuje se sljedeći pravokutnik na ulazu. Ako nije, nadodaje se na kraj skupine.

Ovo su bile osnovne ideje šestog programa iz dodatka, `isofilter`. Sljedeći program `mainfilter` služi da bi od liste predstavnika klasa izotopije dobili predstavnike glavnih klasa. Vrlo je sličan `isofilteru`. Jedina bitna razlika je što se za latinski kvadrat na ulazu provjerava izotopnost njega i njegovih konjugata sa zapamćenim kvadratima. Ako je bar jedan konjugat izotopan nekom od zapamćenih kvadrata, prelazi se na novi kvadrat s ulaza. Ako nije, stablo se proširuje novim predstavnikom. Program je oko šest puta sporiji od `isofiltera`, ali to nije bitno, jer kao ulaz uzima već znatno reduciranu listu predstavnika klasa izotopije.

3.4 Rezultati i provjera

Danas poznati brojevi klasa izotopije i glavnih klasa navedeni su u tablici 1. Vrijednosti Ω_8 i M_8 preuzete su iz članka [19]. Za $n \leq 7$ podaci su dobiveni pomoću opisanih programa.

Broj klasa izotopije reda 7, $\Omega_7 = 564$ ne slaže se s brojem objavljenim u literaturi. U knjizi [14], a i u znatno novijoj knjizi [15] nalazi se podatak $\Omega_7 = 563$. Zanimljivo je da se broj glavnih klasa koji sam dobio slaže s podatkom iz knjiga, $M_7 = 147$.

U [14] iznesena je povijest prebrojavanja latinskih kvadrata. Glavne klase latinskih kvadrata reda 7 prvi je prebrojavao H.W.Norton, 1939. godine. On je dobio vrijednost $M_7 = 146$. Nezavisno od njega isti je posao napravio A.Sade, 1951. godine. Dobio je jednu klasu više, $M_7 = 147$. Nažalost u knjizi se ne spominje porijeklo broja $\Omega_7 = 563$. Do originalnih članaka Nortona i Sadea nisam uspio doći, a niti u [15] ne spominje se odakle je preuzet taj broj.

Slična greška se dogodila za red 8. Rezultat objavljen u Brownovom članku je $\Omega_8 = 1676257$. Autori članka [10] ponovo su prebrojavali klase izotopije i dobili $\Omega_8 = 1676267$, deset više nego Brown. Tu su vrijednost, zajedno s brojem glavnih klasa $M_8 = 283657$ objavili u [19].

Sve to ukazuje na težinu problema prebrojavanja latinskih kvadrata i važnost nezavisne provjere rezultata. Prva ideja za provjeru sporne vrijednosti $\Omega_7 = 564$ je usporedba s brojem glavnih klasa $M_7 = 147$, u kojem se slažu svi članci nakon Sadea. Napisao sam program koji uzima listu od 147 predstavnika glavnih klasa, od svakog pravi šest konjugata i računa koliko je među njima neizotopnih. Kao rezultat dobivamo distribuciju glavnih klasa prema broju klasa izotopije koje sadrže.

Broj neizotopnih konjugata	Glavne klase	Dobiveno klasa izotopije
1	23	23
2	5	10
3	61	183
6	58	384
Ukupno:	147	564

Tablica 2.

Potvrđena je vrijednost $\Omega_7 = 564$, ali se program kojim je dobivena tablica također zasniva na funkciji `isotest`. Ako `isofilter` daje krive rezultate, greška je najvjerojatnije upravo u toj funkciji, pa provjera nije nezavisna. Moglo bi se prigovoriti i pretpostavci o ispravnosti liste predstavnika glavnih klasa, dobivene pomoću spornih programa.

Ideju za nezavisnu provjeru našao sam u [19]. Radi se o provjeri usklađenosti brojeva Ω_7 i T_7. Prednost je što se ne koristi funkcija za testiranje izotopnosti, a vrijednost $T_7 = 16942080$ zajednička je svim člancima koje sam susreo (i osobno sam je potvrdio programom `reduced`).

Označimo s X skup svih latinskih kvadrata reda n, a s G direktni produkt simetričnih grupa $G = S_n \times S_n \times S_n$. Definiramo djelovanje grupe G na skupu X:

$$A^{(r,c,s)} = [\, s(\, a_{r(i)c(j)} \,) \,]_{i,j=1}^{n} \quad \in X,$$

gdje je $A = [a_{ij}]_{i,j=1}^{n} \in X$, $(r, c, s) \in G$. Očito je stabilizator G_A grupa autotopije, a orbita A^G klasa izotopije latinskog kvadrata $A \in X$. Vrijedi poznata formula koja povezuje red stabilizatora i kardinalitet orbite :

$$|G| = \mathrm{card}(A^G) \cdot |G_A|.$$

Budući da se skup X razlaže na disjunktne orbite (klase izotopije), vrijedi

$$U_n = \mathrm{card}(X) = \sum_A \frac{|G|}{|G_A|},$$

gdje se suma uzima po predstavnicima klasa izotopije. Korištenjem propozicije 3.2 i $|G| = (n!)^3$ dobivamo formulu

$$T_n = n! \cdot n \cdot \sum_A \frac{1}{|G_A|}.$$

Vidimo da treba izračunati red grupe autotopija $|G_A|$ svakog predstavnika. Problem sam riješio tako da sam latinske pravokutnike "vjerno" modelirao obojenim grafovima. Zatim sam programom `nauty` (autor B.D.McKay), opisanim u [22], računao grupe automorfizama tih grafova.

Definicija 3.17. **Graf** *je uređena trojka* $\mathcal{G} = (V, E, \psi)$, *gdje je*

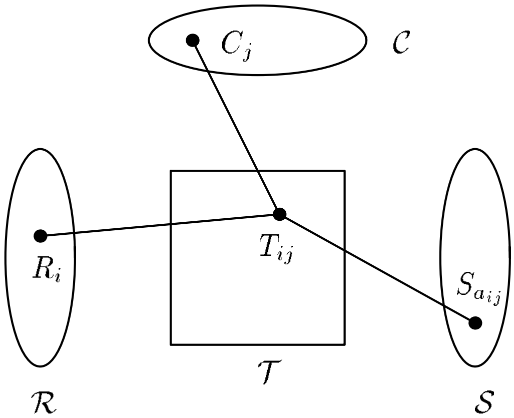
- V *neprazan konačan skup, čije elemente zovemo* **vrhovi**
- E *neprazan konačan skup, čije elemente zovemo* **bridovi**
- $\psi: E \to \mathcal{P}_2(V)$ **funkcija incidencije**, *koja svakom bridu* $e \in E$ *pridružuje dvočlani skup vrhova* $\{u, v\} \in \mathcal{P}_2(V)$.

Vršno k**–bojanje** *grafa* $\mathcal{G}$ *je svaka funkcija* $\varphi: V \to \{1, 2, \ldots, k\}$. *Uređenu četvorku* (V, E, ψ, φ) *zovemo* **obojani graf**.

Definicija 3.18. **Automorfizam** *obojanog grafa* $\mathcal{G} = (V, E, \psi, \varphi)$ *je uređen par* (θ, η) *bijekcija* $\theta: V \to V$, $\eta: E \to E$ *takvih da vrijedi*

- $(\forall u, v \in V)\ (\forall e \in E) \quad \psi(e) = \{u, v\} \iff \phi(\eta(e)) = \{\theta(u), \theta(v)\}$
- $(\forall u \in V) \quad \varphi(u) = \varphi(\theta(u))$

Skup svih automorfizama obojanog grafa $\mathcal{G}$ tvori grupu obzirom na kompoziciju, koju zovemo *grupa automorfizama* od $\mathcal{G}$.



Slika 4.

Definicija 3.19. *Neka je $A = [a_{ij}]_{i,j=1}^n$ latinski kvadrat. Pridružujemo mu obojani graf $\mathcal{G}(A)$, definiran na sljedeći način :*

- *Neka su $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$, $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, $\mathcal{T} = \{\, T_{ij} \mid i, j = 1 \ldots n \,\}$ međusobno disjunktni skupovi. Za skup vrhova uzimamo njihovu uniju, $V = \mathcal{R} \cup \mathcal{C} \cup \mathcal{S} \cup \mathcal{T}$. Skup V ima $n^2 + 3n$ elemenata.*

- *Za skup bridova uzimamo skup $E = \{\, e_{ij}, f_{ij}, g_{ij} \mid i, j = 1 \ldots n \,\}$ od $3n^2$ elemenata, disjunktan s V.*

- *Definiramo funkciju incidencije*

$$\psi(e_{ij}) = \{R_i, T_{ij}\},\ \psi(f_{ij}) = \{C_j, T_{ij}\},\ \psi(g_{ij}) = \{S_{a_{ij}}, T_{ij}\}, \quad \forall i, j = 1 \ldots n$$

- *Definiramo bojanje $\varphi \colon V \to \{1, 2, 3, 4\}$*

$$\varphi(R_i) = 1,\ \varphi(C_i) = 2,\ \varphi(S_i) = 3,\ \varphi(T_{ij}) = 4, \quad \forall i, j = 1 \ldots n$$

Propozicija 3.20. *Grupa automorfizama grafa $\mathcal{G}(A)$ izomorfna je grupi autotopija latinskog kvadrata A.*

Redove grupa autotopija latinskih kvadrata računao sam pomoću programa 7.9, `autcount`. Program uzima s ulaza niz latinskih pravokutnika i svakog od njih transformira u graf $\mathcal{G}(A)$, u formatu prikladnom za `nauty`. Zatim poziva `nauty`, koji vraća red grupe automorfizama grafa $\mathcal{G}(A)$. Prema propoziciji on je jednak redu grupe autotopija kvadrata A, $|G_A|$. Kao rezultat dobivamo ovu tablicu.

$\lvert G_A \rvert$	Broj predstavnika s tim redom
1	415
2	118
3	7
4	6
5	5
6	3
8	6
12	1
24	1
168	1
294	1

Tablica 3.

Pomoću tablice možemo izračunati

$$\begin{aligned} n! \cdot n \cdot \sum \tfrac{1}{|G_A|} &= 7! \cdot 7 \cdot \Big(\; 415 + 118 \cdot \tfrac{1}{2} + 7 \cdot \tfrac{1}{3} + 6 \cdot \tfrac{1}{4} + 5 \cdot \tfrac{1}{5} + \\ &\qquad + 3 \cdot \tfrac{1}{6} + 6 \cdot \tfrac{1}{8} + \tfrac{1}{12} + \tfrac{1}{24} + \tfrac{1}{168} + \tfrac{1}{294} \; \Big) = \\ &= 16942080 = T_7 \end{aligned}$$

Time smo provjerili usklađenost brojeva T_7 i Ω_7.

U prvom dodatku navedeni su predstavnici glavnih klasa kvadrata reda 6 i 7, dobiveni izvođenjem skripta 7.2. To je ulaz za program iz sljedećeg poglavlja, koji konstruira sve moguće ravnine reda 7 i 8. Radi potpunosti navedeni su i predstavnici glavnih klasa nižih redova.

4 Klasifikacija ravnina reda 7 i 8

4.1 Prevođenje kvadrata u parcijalnu matricu

U drugom poglavlju objašnjeno je kako se u ravnini **P** reda n konstruira latinski kvadrat reda $n-1$. Rečeno je da poznavanje kvadrata L omogućuje da bez smanjenja općenitosti popunimo $2n+1$ redaka i $3n$ stupaca incidencijske matrice A ravnine **P**. U ovoj točki objašnjavam kako.

Prva tri retka matrice A neka odgovaraju točkama R, C i S redom, a prva tri stupca pravcima $C \vee S$, $R \vee C$, $R \vee S$. U gornjem lijevom kutu od A pojavljuje se 3×3 podmatrica kao na slici.

	$C \vee S$	$R \vee C$	$R \vee S$	$r_1\, r_2 \cdots r_{n-1}$	$c_1\, c_2 \cdots c_{n-1}$	$s_1\, s_2 \cdots s_{n-1}$	$3n{+}1 \cdots n^2{+}n{+}1$
R	0	1	1	$1\ 1 \cdots\cdots 1$	$0\ 0 \cdots\cdots 0$	$0\ 0 \cdots\cdots 0$	
C	1	1	0	$0\ 0 \cdots\cdots 0$	$1\ 1 \cdots\cdots 1$	$0\ 0 \cdots\cdots 0$	0
S	1	0	1	$0\ 0 \cdots\cdots 0$	$0\ 0 \cdots\cdots 0$	$1\ 1 \cdots\cdots 1$	
4	1	0	0				
5	1	0	0				
$\vdots$	$\vdots$	$\vdots$	$\vdots$				
$n+2$	1	0	0				
$n+3$	0	1	0				
$n+4$	0	1	0				
$\vdots$	$\vdots$	$\vdots$	$\vdots$				
$2n+1$	0	1	0				
$n+3$	0	1	0				
$n+4$	0	1	0				
$\vdots$	$\vdots$	$\vdots$	$\vdots$				
$2n+1$	0	1	0				
$3n+1$							
$\vdots$							
$\vdots$		0					
$\vdots$							
n^2+n+1							

Slika 5.

Stupci 4 do $n+2$ neka odgovaraju preostalim pravcima kroz R, $r_1, r_2, \ldots,$ r_{n-1}. Stupci $n+3$ do $2n+3$ odgovaraju pravcima kroz C, $c_1, c_2, \ldots c_{n-1}$, a stupci $2n+2$ do $3n$ pravcima kroz S, $s_1, s_2, \ldots, s_{n-1}$. Stoga možemo popuniti prva tri retka matrice, kao na slici 5.

Retke matrice označimo dualno. Recima 4 do $n+2$ odgovaraju preostale točke na pravcu $C \vee S$, recima $n+3$ do $2n+1$ preostale točke na $R \vee C$, a recima $2n+2$ do $3n$ preostale točke na $R \vee S$. Time su potpuno određena i prva tri stupca matrice.

Sad ćemo odrediti preostale incidencije u gornjoj lijevoj $3n \times 3n$ podmatrici od A. Točke kojima odgovaraju reci 4 do $n+2$ spojene su sa C i S pravcem $C \vee S$. Zato one ne leže niti na jednom od pravaca $c_1, \ldots c_{n-1}, s_1, \ldots, s_{n-1}$. Slijedi da su blokovi $(2,3)$ i $(2,4)$ sa slike 6 popunjeni nulama.

Promotrimo spojnice tih točaka s R. U obzir dolaze jedino pravci $r_1, \ldots,$ r_{n-1}. Svaki od njih sadrži najviše jednu od točaka 4 do $n+2$, pa je blok $(2,2)$ permutacijska $(n-1) \times (n-1)$ matrica (ima u svakom retku i stupcu po jednu jedinicu, a ostalo su nule). Permutiranjem redaka 4 do $n+2$ možemo postići da to bude jedinična matrica.

Sličnim zaključivanjem možemo odrediti incidenciju točaka $n+3$ do $3n$ s pravcima r_i, c_j, s_k i tako popuniti cijelu $3n \times 3n$ podmatricu, kao na slici 6.

Idući korak je sortiranje stupaca $3n+1$ do n^2+n+1 u leksikografskom poretku. Prvo dolaze preostali pravci kroz točku 4, zatim 5 i tako dalje. Time smo dio matrice desno od $3n \times 3n$ bloka podijelili na $n-1$ blok–stupaca širine $n-1$. U bloku $(2,5)$ imamo jedinice u prvom retku, u bloku $(2,6)$ u drugom retku itd.

Blokovi $(3,5)$, $(3,6), \ldots, (3, n+3)$ su permutacijske matrice. Siječenjem pravaca $3n+1$ do $4n-1$ s pravcem 2 zaključujemo da je u svakom retku bloka $(3,5)$ točno jedna jedinica. Spajanjem točaka $n+3$ do $2n+1$ s točkom 4 slijedi da je u svakom stupcu točno jedna jedinica. Analogno se vidi da su blokovi $(3,6), \ldots, (3, n+3)$ permutacijske matrice, a zbog leksikografskog sortiranja to su jedinične matrice reda $n-1$.

Ponovo vrijede dualni zaključci za retke. Sortiranje redaka $3n+1$ do n^2+n+1 u leksikografskom poretku daje blok–strukturu sa slike 6.

Ono što još možemo zaključiti o matrici bez poznavanja latinskog kvadrata L je da su blokovi na dijagonali $(5,5)$, $(6,6), \ldots, (n+3, n+3)$ nulmatrice, a svi ostali blokovi su permutacijske matrice. Na primjer, ovako se vidi da je blok (5,5) popunjen nulama: kada bi se na mjestu (i, j) pojavila jedinica, pravac $3n + j$ sijekao bi se s pravcem 4 u dvije točke, 4 i $3n + i$, što nije moguće.

Na primjeru bloka $(5, 6)$ demonstrirat ću da su ostali blokovi permutacijske matrice. Pravac $4n$ siječe se s pravcem 4 točno u jednoj točki. Jedina mogućnost za to su točke $3n + 1$ do $4n - 1$, pa u prvom stupcu bloka $(5, 6)$ imamo jednu jedinicu, a ostalo su nule. Ista tvrdnja vrijedi za ostale stupce tog bloka, što zaključujemo skalarnim množenjem stupaca $4n + 1$ do $5n - 2$ sa stupcem 4. Množenjem redaka $3n + 1$ do $4n - 1$ s retkom 5 vidimo da u svakom retku bloka $(5, 6)$ imamo točno po jednu jedinicu.

	1	2	3 ···							
1	0	1	1	1 ··· 1						
2	1	1	0		1 ··· 1		0	0	· · ·	0
3	1	0	1			1 ··· 1				
4	1			1			1 ··· 1			
⋮	⋮			⋱	0	0		1 ··· 1	· · ·	
n+2	1			1						1 ··· 1
n+3		1				1	1	1		1
⋮		⋮		0	0	⋱	⋱	⋱	· · ·	⋱
2n+1		1				1	1	1		1
2n+2			1		1					
⋮			⋮	0	⋱	0			· · ·	
3n			1		1					
3n+1				1	1					
⋮		0		⋮	⋱	B_1	0		· · ·	
4n–1				1	1					
4n				1	1					
⋮		0		⋮	⋱	B_2		0	· · ·	
5n–2				1	1					
⋮		⋮		⋮	⋮	⋮	⋮	⋮	⋱	⋮
n^2+3				1	1					
⋮		0		⋮	⋱	B_{n-1}			· · ·	0
n^2+n+1				1	1					

Slika 6.

Ako znamo latinski kvadrat $L = [l_{ij}]_{i,j=1}^{n}$, možemo popuniti još i blokove $(5,4), (6,4), \ldots, (n+3,4)$. Označimo ih redom s $B_1, B_2, \ldots, B_{n-1}$. Promotrimo npr. blok B_1. Njegov prvi redak odgovara točki $r_1 \wedge c_1$. Ta je točka zbog definicije kvadrata L spojena s točkom S pravcem $s_{l_{11}}$. To znači da na mjestu l_{11} u prvom retku bloka B_1 imamo jedinicu, a na ostalim mjestima nule. U drugom retku od B_1 jedinica je na mjestu l_{12}, u trećem na l_{13} itd. Analogno popunjavamo blokove $B_2, \ldots, B_{n-1}$.

Time smo, kao što je i bilo najavljeno, popunili $3n$ stupaca i $2n+1$ redaka incidencijske matrice ravnine $\mathbf{P}$. Preostali dio matrice podijeljen je na blokove dimenzija $(n-1) \times (n-1)$. Blokovi na dijagonali su nulmatrice, a ostali su permutacijske matrice.

4.2 Nadopunjavanje do incidencijske matrice

Prevođenje latinskog kvadrata u parcijalnu incidencijsku matricu realizirao sam u sklopu programa 7.10. Program je organiziran prema sljedećem generalnom algoritmu.

Dok ima neobrađenih latinskih kvadrata na ulazu

Korak 1. Učitaj kvadrat L.

Korak 2. Popuni prvih $2n+1$ redaka i $3n$ stupaca matrice A.

Korak 3. Sistematski pokušavaj nadopuniti A do incidencijske matrice projektivne ravnine. Kad god to uspiješ, ispiši potpunu incidencijsku matricu.

Drugi korak sam isprogramirao u skladu s konstrukcijom iz prethodne točke. Taj dio programa ne sadrži nove ideje, pa ga neću obrazlagati. U ovoj točki tumačim kako sam realizirao treći korak algoritma.

Preostali dio matrice popunjava se stupac po stupac, počevši od stupca $3n+1$. Svaki od stupaca sadrži $n-1$ nepopunjena dijela duljine $n-1$.

Nepopunjeni blokovi sa slike 6 su permutacijske matrice, pa je u svakom od dijelova točno jedna jedinica. Dakle, imamo $(n-1)^{n-1}$ kandidata za svaki stupac. Npr. za $n=8$ to iznosi $7^7 = 823543$.

Nepopunjenih stupaca ima $n^2+n+1-3n=(n-1)^2$. U prvi mah se čini da treba ispitati $(n-1)^{(n-1)^3}$ slučajeva, što nadilazi mogućnosti današnjih računala već za $n=4$. Ta se "nemoguća potraga" ipak može realizirati i za $n=8$, korištenjem algoritma poznatog kao backtracking.

Opći problem koji se rješava backtrackingom je pronalaženje svih uređenih m–torki $(x_1,x_2,\ldots,x_m)$ iz skupa $A_1\times A_2\times\cdots\times A_m$ ($A_1\ldots A_m$ su neki konačni skupovi) koje zadovoljavaju svojstvo P_m. Kod nas su A_k–ovi skupovi kandidata za stupce ($\mathrm{card}A_k=(n-1)^{n-1}$), a za svojstvo P_m možemo uzeti drugi kriterij iz propozicije 1.5. On zapravo kaže da je skalarni produkt svaka dva različita stupca 1, a stupca sa samim sobom $n+1$. Drugi uvjet je ispunjen već zbog izbora kandidata za stupce, pa treba provjeravati samo prvi.

Osnovna ideja backtrackinga je izbjegavanje ispitivanja svih članova iz $A_1\times A_2\times\cdots\times A_m$. Treba smisliti svojstva $P_1,P_2,\ldots,P_{m-1}$ za parcijalna rješenja takva da $P_{k+1}(x_1,\ldots,x_k,x_{k+1})$ implicira $P_k(x_1,\ldots,x_k)$, za $k=1\ldots m-1$. Drugim riječima, ako $(x_1,\ldots,x_k)$ ne zadovoljava svojstvo P_k, onda niti jedno proširenje $(x_1,\ldots,x_k,x_{k+1})$ ne zadovoljava P_{k+1}. Po indukciji, niti jedno proširenje do cjelovitog rješenja $(x_1,\ldots,x_k,\ldots,x_m)$ neće zadovoljavati svojstvo P_m.

Algoritam sistematski pronalazi sva rješenja $(x_1,\ldots,x_m)$, razmatrajući samo ona parcijalna rješenja $(x_1,\ldots,x_k)$ koja zadovoljavaju svojstvo P_k.

<u>Korak 1.</u> [Započni potragu] Postavi k na 0.

<u>Korak 2.</u> [Nađi nasljednike] U B_k spremi skup svih $y_{k+1}\in A_{k+1}$ takvih da $(x_1,\ldots,x_k,y_{k+1})$ zadovoljava svojstvo P_{k+1}.

<u>Korak 3.</u> [Ima li još nasljednika?] Ako je B_{k+1} prazan, idi na korak 6.

<u>Korak 4.</u> [Idi naprijed] Odaberi bilo koji x_{k+1} iz B_{k+1}, izbriši ga iz skupa B_{k+1} i povećaj k za jedan.

<u>Korak 5.</u> [Pronađeno rješenje?] Ako je $k=m$, ispiši rješenje $(x_1,\ldots,x_m)$ i idi na korak 6. Ako je $k<m$, idi na korak 2.

<u>Korak 6.</u> [Vrati se natrag, "backtrack"] Smanji k za jedan. Ako je $k\geq 0$ idi na korak 3, inače algoritam završava.

Međusvojstva $P_1, \ldots, P_{m-1}$ možemo birati na razne načine, ali kod nas se nameću sama po sebi. Svojstvo P_k je ispunjeno ako je skalarni produkt k–tog nadodanog stupca sa stupcima prije njega 1. Za prvih $n+2$ stupaca to vrijedi uvijek, pa provjeravamo samo da li je skalarni produkt stupaca $n+3, \ldots, 3n+k-1$ sa stupcem $3n+k$ jednak 1.

Algoritam za backtracking možemo si predočiti preko "stabla potrage". Na prvom nivou stabla su elementi iz A_1 koji zadovoljavaju svojstvo P_1. Na drugom su uređeni parovi (x_1, x_2) koji zadovoljavaju P_2. Element x_1 s prvog nivoa spojen je sa svim parovima na drugom nivou kojima je x_1 prva komponenta. Na trećemo nivou stabla su uređene trojke (x_1, x_2, x_3) koje zadovoljavaju P_3. Elementi na drugom nivou spojeni su sa svojim nasljednicima na trećem, i tako dalje. Ako pretpostavimo da je vrijeme potrebno za obrađivanje jednog čvora konstantno, vrijeme izvršavanja algoritma proporcionalno je ukupnom broju čvorova u stablu.

Obrađivanje jednog čvora stabla kod nas uključuje ispitivanje $(n-1)^{n-1}$ kandidata za pripadni stupac. Za $n = 8$ to na računalima koja sam koristio traje oko dvije sekunde. Prosječno stablo ima oko 2000 čvorova, pa obrađivanje jednog latinskog kvadrata reda 7 traje otprilike sat vremena. To znači da bi ispitivanje 147 predstavnika glavnih klasa trajalo oko tjedan dana.

Kad sam to izračunao, odlučio sam ubrzati program. Ideja je smanjiti skupove kandidata A_k. Kod nadodavanja drugog stupca znamo i više nego da je u svakom nepopunjenom dijelu duljine $n-1$ jedna jedinica. Blokovi na slici 6 su permutacijske matrice, pa jedinice ne mogu stajati na mjestima na kojima su u prvom nadodanom stupcu. To znači da za drugi stupac imamo samo $(n-2)^{n-1}$ kandidata. Za treći stupac skup kandidata iz istog razloga ima $(n-3)^{n-1}$ članova. Za stupac $n-1$ imamo samo jedan kandidat. Kad prijeđemo na novi blok (nadodani stupac n), skup kandidata opet ima $(n-1)^{n-1}$ članova, zatim se opet smanjuje itd.

Time sam smanjio vrijeme potrebno za obrađivanje čvorova na svim nivoima stabla, osim na onima koji odgovaraju prvim stupcima u blokovima. Ukupno vrijeme izvršavanja smanjilo se na oko 15 sati, što je prihvatljivo.

Više o backtrackingu može se naći u člancima [23], [24] i [12].

4.3 Rezultati

Nakon izvođenja skripta 7.2 u datotekama `main6` i `main7` dobijemo predstavnike glavnih klasa. One služe kao ulaz programu 7.10.

Samo jedan kvadrat reda 6 (od njih 12) i jedan kvadrat reda 7 (od njih 147) daju projektivne ravnine. Radi se o kvadratima 6.5 i 7.145 iz dodatka 1. Oba pripadaju glavnim klasama u kojima su Cayleyeve tablice cikličkih grupa.

Djelovanjem izotopije

$$r = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 5 & 3 & 4 & 2 & 6 \end{pmatrix}, \quad c = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 6 & 2 & 4 & 3 & 5 \end{pmatrix}, \quad s = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 5 & 4 & 6 & 2 \end{pmatrix}$$

na kvadrat 6.5 dobivamo ciklički kvadrat reda 6,

```
1 2 3 4 5 6
2 3 4 5 6 1
3 4 5 6 1 2
4 5 6 1 2 3
5 6 1 2 3 4
6 1 2 3 4 5
```

Izotopija $r = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 7 & 6 & 5 & 4 & 3 & 2 \end{pmatrix}$, $c = s = id$ primjenjena na kvadrat 7.145 daje ciklički kvadrat reda 7.

Kvadrati 6.5 i 7.145 daju po dvije potpune incidencijske matrice. Navodim samo jednu matricu reda 6, na kojoj je vidljiva blok struktura opisana u točki 4.1.

011	111111	000000	000000	000000	000000	000000	000000	000000	000000
110	000000	111111	000000	000000	000000	000000	000000	000000	000000
101	000000	000000	111111	000000	000000	000000	000000	000000	000000
100	100000	000000	000000	111111	000000	000000	000000	000000	000000
100	010000	000000	000000	000000	111111	000000	000000	000000	000000
100	001000	000000	000000	000000	000000	111111	000000	000000	000000
100	000100	000000	000000	000000	000000	000000	111111	000000	000000
100	000010	000000	000000	000000	000000	000000	000000	111111	000000
100	000001	000000	000000	000000	000000	000000	000000	000000	111111
010	000000	000000	100000	100000	100000	100000	100000	100000	100000
010	000000	000000	010000	010000	010000	010000	010000	010000	010000
010	000000	000000	001000	001000	001000	001000	001000	001000	001000
010	000000	000000	000100	000100	000100	000100	000100	000100	000100
010	000000	000000	000010	000010	000010	000010	000010	000010	000010
010	000000	000000	000001	000001	000001	000001	000001	000001	000001
001	000000	100000	000000	000100	000001	000010	100000	001000	010000
001	000000	010000	000000	000010	000100	000001	010000	100000	001000
001	000000	001000	000000	000001	000010	000100	001000	010000	100000
001	000000	000100	000000	100000	001000	010000	000100	000001	000010
001	000000	000010	000000	010000	100000	001000	000010	000100	000001
001	000000	000001	000000	001000	010000	100000	000001	000010	000100
000	100000	100000	100000	000000	000100	010000	001000	000010	000001
000	100000	010000	010000	000000	000010	001000	100000	000001	000100
000	100000	001000	001000	000000	000001	100000	010000	000100	000010
000	100000	000100	000100	000000	100000	000010	000001	010000	001000
000	100000	000010	000010	000000	010000	000001	000100	001000	100000
000	100000	000001	000001	000000	001000	000100	000010	100000	010000
000	010000	100000	001000	100000	000000	000001	000010	010000	000100
000	010000	010000	100000	010000	000000	000100	000001	001000	000010
000	010000	001000	010000	001000	000000	000010	000100	100000	000001
000	010000	000100	000001	000100	000000	001000	010000	000010	100000
000	010000	000010	000100	000010	000000	100000	001000	000001	010000
000	010000	000001	000010	000001	000000	010000	100000	000100	001000
000	001000	100000	010000	000010	001000	000000	000001	000100	100000
000	001000	010000	001000	000001	100000	000000	000100	000010	010000
000	001000	001000	100000	000100	010000	000000	000010	000001	001000
000	001000	000100	000010	010000	000001	000000	001000	100000	000100
000	001000	000010	000001	001000	000100	000000	100000	010000	000010
000	001000	000001	000100	100000	000010	000000	010000	001000	000001
000	000100	100000	000100	000001	010000	001000	000000	100000	000010
000	000100	010000	000010	000100	001000	100000	000000	010000	000001
000	000100	001000	000001	000010	100000	010000	000000	001000	000100
000	000100	000100	100000	001000	000010	000001	000000	000100	010000
000	000100	000010	010000	100000	000001	000100	000000	000010	001000
000	000100	000001	001000	010000	000100	000010	000000	000001	100000
000	000010	100000	000001	010000	000010	100000	000100	000000	001000
000	000010	010000	000100	001000	000001	010000	000010	000000	100000
000	000010	001000	000010	100000	000100	001000	000001	000000	010000
000	000010	000100	001000	000010	010000	000100	100000	000000	000001
000	000010	000010	100000	000001	001000	000010	010000	000000	000100
000	000010	000001	010000	000100	100000	000001	001000	000000	000010
000	000001	100000	000010	001000	100000	000100	010000	000001	000000
000	000001	010000	000001	100000	010000	000010	001000	000100	000000
000	000001	001000	000100	010000	001000	000001	100000	000010	000000
000	000001	000100	010000	000001	000100	100000	000010	001000	000000
000	000001	000010	001000	000100	000010	010000	000001	100000	000000
000	000001	000001	100000	000010	000001	001000	000100	010000	000000

Incidencijska matrica dobivena od kvadrata 6.5.

Rezultate potrage navodim u formatu u kojem su primjeri 3 i 4. Rezultati 1 i 2 su projektivne ravnine reda 7 dobivene od latinskog kvadrata 6.5. Rezultati 3 i 4 su projektivne ravnine reda 8 dobivene od kvadrata 7.145. Incidencijska matrica pripada projektivnoj ravnini navedenoj kao rezultat 1.

Pomoću programa `nauty` mogu se naći izomorfizmi između ovih ravnina i primjera 3 i 4. Djelovanjem bijekcijom

```
1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1  2  9  16 23 37 51 44 30 3  4  6  8  7  5  10 11 13 15 14 12 17 18 20 22 21 19 24 25 27

   31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
   29 28 26 38 39 41 43 42 40 52 53 55 57 56 54 45 46 48 50 49 47 31 32 34 36 35 33
```

na točke rezultata 1 dobivamo primjer 3. Izomorfizam između rezultata 2 i primjera 3 je bijekcija

```
1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1  2  9  16 37 23 51 30 44 3  6  4  8  5  7  10 13 11 15 12 14 17 20 18 22 19 21 38 41 39

   31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
   43 40 42 24 27 25 29 26 28 52 55 53 57 54 56 31 34 32 36 33 35 45 48 46 50 47 49
```

Izomorfizam između rezultata 3 i primjera 4 zadan je s

```
1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1  2  10 18 26 42 34 58 66 50 3  4  6  5  8  9  7  11 12 14 13 16 17 15 19 20 22 21 24 25

31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
23 27 28 30 29 32 33 31 43 44 46 45 48 49 47 35 36 38 37 40 41 39 59 60 62 61 64 65 63 67

   61 62 63 64 65 66 67 68 69 70 71 72 73
   68 70 69 72 73 71 51 52 54 53 56 57 55
```

Konačno, izomorfizam između rezultata 4 i primjera 4 je bijekcija

```
1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1  2  10 18 34 50 42 66 26 58 3  5  7  6  9  4  8  11 13 15 14 17 12 16 19 21 23 22 25 20

31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
24 35 37 39 38 41 36 40 51 53 55 54 57 52 56 43 45 47 46 49 44 48 67 69 71 70 73 68 72 27

   61 62 63 64 65 66 67 68 69 70 71 72 73
   29 31 30 33 28 32 59 61 63 62 65 60 64
```

Dakle, do na izomorfizam jedine projektivne ravnine reda 7 i 8 su one klasične.

```
{ {2, 3, 4, 5, 6, 7, 8, 9},          {1, 2, 10, 11, 12, 13, 14, 15},
  {1, 3, 16, 17, 18, 19, 20, 21},  {1, 4, 22, 23, 24, 25, 26, 27},
  {1, 5, 28, 29, 30, 31, 32, 33},  {1, 6, 34, 35, 36, 37, 38, 39},
  {1, 7, 40, 41, 42, 43, 44, 45},  {1, 8, 46, 47, 48, 49, 50, 51},
  {1, 9, 52, 53, 54, 55, 56, 57},  {2, 16, 22, 28, 34, 40, 46, 52},
  {2, 17, 23, 29, 35, 41, 47, 53}, {2, 18, 24, 30, 36, 42, 48, 54},
  {2, 19, 25, 31, 37, 43, 49, 55}, {2, 20, 26, 32, 38, 44, 50, 56},
  {2, 21, 27, 33, 39, 45, 51, 57}, {3, 10, 22, 29, 36, 43, 50, 57},
  {3, 11, 23, 30, 34, 44, 51, 55}, {3, 12, 24, 28, 35, 45, 49, 56},
  {3, 13, 25, 32, 39, 40, 47, 54}, {3, 14, 26, 33, 37, 41, 48, 52},
  {3, 15, 27, 31, 38, 42, 46, 53}, {4, 10, 19, 28, 39, 44, 48, 53},
  {4, 11, 20, 29, 37, 45, 46, 54}, {4, 12, 21, 30, 38, 43, 47, 52},
  {4, 13, 16, 31, 36, 41, 51, 56}, {4, 14, 17, 32, 34, 42, 49, 57},
  {4, 15, 18, 33, 35, 40, 50, 55}, {5, 10, 20, 25, 35, 42, 51, 52},
  {5, 11, 21, 26, 36, 40, 49, 53}, {5, 12, 19, 27, 34, 41, 50, 54},
  {5, 13, 17, 22, 38, 45, 48, 55}, {5, 14, 18, 23, 39, 43, 46, 56},
  {5, 15, 16, 24, 37, 44, 47, 57}, {6, 10, 21, 24, 32, 41, 46, 55},
  {6, 11, 19, 22, 33, 42, 47, 56}, {6, 12, 20, 23, 31, 40, 48, 57},
  {6, 13, 18, 27, 29, 44, 49, 52}, {6, 14, 16, 25, 30, 45, 50, 53},
  {6, 15, 17, 26, 28, 43, 51, 54}, {7, 10, 16, 23, 33, 38, 49, 54},
  {7, 11, 17, 24, 31, 39, 50, 52}, {7, 12, 18, 22, 32, 37, 51, 53},
  {7, 13, 19, 26, 30, 35, 46, 57}, {7, 14, 20, 27, 28, 36, 47, 55},
  {7, 15, 21, 25, 29, 34, 48, 56}, {8, 10, 17, 27, 30, 37, 40, 56},
  {8, 11, 18, 25, 28, 38, 41, 57}, {8, 12, 16, 26, 29, 39, 42, 55},
  {8, 13, 20, 24, 33, 34, 43, 53}, {8, 14, 21, 22, 31, 35, 44, 54},
  {8, 15, 19, 23, 32, 36, 45, 52}, {9, 10, 18, 26, 31, 34, 45, 47},
  {9, 11, 16, 27, 32, 35, 43, 48}, {9, 12, 17, 25, 33, 36, 44, 46},
  {9, 13, 21, 23, 28, 37, 42, 50}, {9, 14, 19, 24, 29, 38, 40, 51},
  {9, 15, 20, 22, 30, 39, 41, 49} }
```

Rezultat 1.

```
{ {2, 3, 4, 5, 6, 7, 8, 9},        {1, 2, 10, 11, 12, 13, 14, 15},
  {1, 3, 16, 17, 18, 19, 20, 21},  {1, 4, 22, 23, 24, 25, 26, 27},
  {1, 5, 28, 29, 30, 31, 32, 33},  {1, 6, 34, 35, 36, 37, 38, 39},
  {1, 7, 40, 41, 42, 43, 44, 45},  {1, 8, 46, 47, 48, 49, 50, 51},
  {1, 9, 52, 53, 54, 55, 56, 57},  {2, 16, 22, 28, 34, 40, 46, 52},
  {2, 17, 23, 29, 35, 41, 47, 53}, {2, 18, 24, 30, 36, 42, 48, 54},
  {2, 19, 25, 31, 37, 43, 49, 55}, {2, 20, 26, 32, 38, 44, 50, 56},
  {2, 21, 27, 33, 39, 45, 51, 57}, {3, 10, 22, 29, 36, 43, 50, 57},
  {3, 11, 23, 30, 34, 44, 51, 55}, {3, 12, 24, 28, 35, 45, 49, 56},
  {3, 13, 25, 32, 39, 40, 47, 54}, {3, 14, 26, 33, 37, 41, 48, 52},
  {3, 15, 27, 31, 38, 42, 46, 53}, {4, 10, 19, 32, 34, 45, 48, 53},
  {4, 11, 20, 33, 35, 43, 46, 54}, {4, 12, 21, 31, 36, 44, 47, 52},
  {4, 13, 16, 29, 37, 42, 51, 56}, {4, 14, 17, 30, 38, 40, 49, 57},
  {4, 15, 18, 28, 39, 41, 50, 55}, {5, 10, 20, 23, 39, 42, 49, 52},
  {5, 11, 21, 24, 37, 40, 50, 53}, {5, 12, 19, 22, 38, 41, 51, 54},
  {5, 13, 17, 26, 36, 45, 46, 55}, {5, 14, 18, 27, 34, 43, 47, 56},
  {5, 15, 16, 25, 35, 44, 48, 57}, {6, 10, 21, 25, 30, 41, 46, 56},
  {6, 11, 19, 26, 28, 42, 47, 57}, {6, 12, 20, 27, 29, 40, 48, 55},
  {6, 13, 18, 22, 33, 44, 49, 53}, {6, 14, 16, 23, 31, 45, 50, 54},
  {6, 15, 17, 24, 32, 43, 51, 52}, {7, 10, 16, 24, 33, 38, 47, 55},
  {7, 11, 17, 22, 31, 39, 48, 56}, {7, 12, 18, 23, 32, 37, 46, 57},
  {7, 13, 19, 27, 30, 35, 50, 52}, {7, 14, 20, 25, 28, 36, 51, 53},
  {7, 15, 21, 26, 29, 34, 49, 54}, {8, 10, 17, 27, 28, 37, 44, 54},
  {8, 11, 18, 25, 29, 38, 45, 52}, {8, 12, 16, 26, 30, 39, 43, 53},
  {8, 13, 20, 24, 31, 34, 41, 57}, {8, 14, 21, 22, 32, 35, 42, 55},
  {8, 15, 19, 23, 33, 36, 40, 56}, {9, 10, 18, 26, 31, 35, 40, 51},
  {9, 11, 16, 27, 32, 36, 41, 49}, {9, 12, 17, 25, 33, 34, 42, 50},
  {9, 13, 21, 23, 28, 38, 43, 48}, {9, 14, 19, 24, 29, 39, 44, 46},
  {9, 15, 20, 22, 30, 37, 45, 47}}
```

Rezultat 2.

```
{ {2, 3, 4, 5, 6, 7, 8, 9, 10},          {1, 2, 11, 12, 13, 14, 15, 16, 17},
  {1, 3, 18, 19, 20, 21, 22, 23, 24},   {1, 4, 25, 26, 27, 28, 29, 30, 31},
  {1, 5, 32, 33, 34, 35, 36, 37, 38},   {1, 6, 39, 40, 41, 42, 43, 44, 45},
  {1, 7, 46, 47, 48, 49, 50, 51, 52},   {1, 8, 53, 54, 55, 56, 57, 58, 59},
  {1, 9, 60, 61, 62, 63, 64, 65, 66},   {1, 10, 67, 68, 69, 70, 71, 72, 73},
  {2, 18, 25, 32, 39, 46, 53, 60, 67},  {2, 19, 26, 33, 40, 47, 54, 61, 68},
  {2, 20, 27, 34, 41, 48, 55, 62, 69},  {2, 21, 28, 35, 42, 49, 56, 63, 70},
  {2, 22, 29, 36, 43, 50, 57, 64, 71},  {2, 23, 30, 37, 44, 51, 58, 65, 72},
  {2, 24, 31, 38, 45, 52, 59, 66, 73},  {3, 11, 25, 33, 41, 49, 57, 65, 73},
  {3, 12, 26, 34, 42, 50, 58, 66, 67},  {3, 13, 27, 35, 43, 51, 59, 60, 68},
  {3, 14, 28, 36, 44, 52, 53, 61, 69},  {3, 15, 29, 37, 45, 46, 54, 62, 70},
  {3, 16, 30, 38, 39, 47, 55, 63, 71},  {3, 17, 31, 32, 40, 48, 56, 64, 72},
  {4, 11, 18, 35, 45, 47, 58, 64, 69},  {4, 12, 19, 36, 39, 48, 59, 65, 70},
  {4, 13, 20, 37, 40, 49, 53, 66, 71},  {4, 14, 21, 38, 41, 50, 54, 60, 72},
  {4, 15, 22, 32, 42, 51, 55, 61, 73},  {4, 16, 23, 33, 43, 52, 56, 62, 67},
  {4, 17, 24, 34, 44, 46, 57, 63, 68},  {5, 11, 19, 28, 43, 46, 55, 66, 72},
  {5, 12, 20, 29, 44, 47, 56, 60, 73},  {5, 13, 21, 30, 45, 48, 57, 61, 67},
  {5, 14, 22, 31, 39, 49, 58, 62, 68},  {5, 15, 23, 25, 40, 50, 59, 63, 69},
  {5, 16, 24, 26, 41, 51, 53, 64, 70},  {5, 17, 18, 27, 42, 52, 54, 65, 71},
  {6, 11, 20, 31, 36, 51, 54, 63, 67},  {6, 12, 21, 25, 37, 52, 55, 64, 68},
  {6, 13, 22, 26, 38, 46, 56, 65, 69},  {6, 14, 23, 27, 32, 47, 57, 66, 70},
  {6, 15, 24, 28, 33, 48, 58, 60, 71},  {6, 16, 18, 29, 34, 49, 59, 61, 72},
  {6, 17, 19, 30, 35, 50, 53, 62, 73},  {7, 11, 21, 26, 32, 44, 59, 62, 71},
  {7, 12, 22, 27, 33, 45, 53, 63, 72},  {7, 13, 23, 28, 34, 39, 54, 64, 73},
  {7, 14, 24, 29, 35, 40, 55, 65, 67},  {7, 15, 18, 30, 36, 41, 56, 66, 68},
  {7, 16, 19, 31, 37, 42, 57, 60, 69},  {7, 17, 20, 25, 38, 43, 58, 61, 70},
  {8, 11, 22, 30, 34, 40, 52, 60, 70},  {8, 12, 23, 31, 35, 41, 46, 61, 71},
  {8, 13, 24, 25, 36, 42, 47, 62, 72},  {8, 14, 18, 26, 37, 43, 48, 63, 73},
  {8, 15, 19, 27, 38, 44, 49, 64, 67},  {8, 16, 20, 28, 32, 45, 50, 65, 68},
  {8, 17, 21, 29, 33, 39, 51, 66, 69},  {9, 11, 23, 29, 38, 42, 48, 53, 68},
  {9, 12, 24, 30, 32, 43, 49, 54, 69},  {9, 13, 18, 31, 33, 44, 50, 55, 70},
  {9, 14, 19, 25, 34, 45, 51, 56, 71},  {9, 15, 20, 26, 35, 39, 52, 57, 72},
  {9, 16, 21, 27, 36, 40, 46, 58, 73},  {9, 17, 22, 28, 37, 41, 47, 59, 67},
  {10, 11, 24, 27, 37, 39, 50, 56, 61}, {10, 12, 18, 28, 38, 40, 51, 57, 62},
  {10, 13, 19, 29, 32, 41, 52, 58, 63}, {10, 14, 20, 30, 33, 42, 46, 59, 64},
  {10, 15, 21, 31, 34, 43, 47, 53, 65}, {10, 16, 22, 25, 35, 44, 48, 54, 66},
  {10, 17, 23, 26, 36, 45, 49, 55, 60} }
```

Rezultat 3.

```
{ {2, 3, 4, 5, 6, 7, 8, 9, 10},         {1, 2, 11, 12, 13, 14, 15, 16, 17},
  {1, 3, 18, 19, 20, 21, 22, 23, 24},   {1, 4, 25, 26, 27, 28, 29, 30, 31},
  {1, 5, 32, 33, 34, 35, 36, 37, 38},   {1, 6, 39, 40, 41, 42, 43, 44, 45},
  {1, 7, 46, 47, 48, 49, 50, 51, 52},   {1, 8, 53, 54, 55, 56, 57, 58, 59},
  {1, 9, 60, 61, 62, 63, 64, 65, 66},   {1, 10, 67, 68, 69, 70, 71, 72, 73},
  {2, 18, 25, 32, 39, 46, 53, 60, 67},  {2, 19, 26, 33, 40, 47, 54, 61, 68},
  {2, 20, 27, 34, 41, 48, 55, 62, 69},  {2, 21, 28, 35, 42, 49, 56, 63, 70},
  {2, 22, 29, 36, 43, 50, 57, 64, 71},  {2, 23, 30, 37, 44, 51, 58, 65, 72},
  {2, 24, 31, 38, 45, 52, 59, 66, 73},  {3, 11, 25, 33, 41, 49, 57, 65, 73},
  {3, 12, 26, 34, 42, 50, 58, 66, 67},  {3, 13, 27, 35, 43, 51, 59, 60, 68},
  {3, 14, 28, 36, 44, 52, 53, 61, 69},  {3, 15, 29, 37, 45, 46, 54, 62, 70},
  {3, 16, 30, 38, 39, 47, 55, 63, 71},  {3, 17, 31, 32, 40, 48, 56, 64, 72},
  {4, 11, 18, 37, 42, 48, 59, 61, 71},  {4, 12, 19, 38, 43, 49, 53, 62, 72},
  {4, 13, 20, 32, 44, 50, 54, 63, 73},  {4, 14, 21, 33, 45, 51, 55, 64, 67},
  {4, 15, 22, 34, 39, 52, 56, 65, 68},  {4, 16, 23, 35, 40, 46, 57, 66, 69},
  {4, 17, 24, 36, 41, 47, 58, 60, 70},  {5, 11, 19, 30, 45, 50, 56, 60, 69},
  {5, 12, 20, 31, 39, 51, 57, 61, 70},  {5, 13, 21, 25, 40, 52, 58, 62, 71},
  {5, 14, 22, 26, 41, 46, 59, 63, 72},  {5, 15, 23, 27, 42, 47, 53, 64, 73},
  {5, 16, 24, 28, 43, 48, 54, 65, 67},  {5, 17, 18, 29, 44, 49, 55, 66, 68},
  {6, 11, 20, 28, 38, 46, 58, 64, 68},  {6, 12, 21, 29, 32, 47, 59, 65, 69},
  {6, 13, 22, 30, 33, 48, 53, 66, 70},  {6, 14, 23, 31, 34, 49, 54, 60, 71},
  {6, 15, 24, 25, 35, 50, 55, 61, 72},  {6, 16, 18, 26, 36, 51, 56, 62, 73},
  {6, 17, 19, 27, 37, 52, 57, 63, 67},  {7, 11, 21, 27, 36, 39, 54, 66, 72},
  {7, 12, 22, 28, 37, 40, 55, 60, 73},  {7, 13, 23, 29, 38, 41, 56, 61, 67},
  {7, 14, 24, 30, 32, 42, 57, 62, 68},  {7, 15, 18, 31, 33, 43, 58, 63, 69},
  {7, 16, 19, 25, 34, 44, 59, 64, 70},  {7, 17, 20, 26, 35, 45, 53, 65, 71},
  {8, 11, 22, 31, 35, 44, 47, 62, 67},  {8, 12, 23, 25, 36, 45, 48, 63, 68},
  {8, 13, 24, 26, 37, 39, 49, 64, 69},  {8, 14, 18, 27, 38, 40, 50, 65, 70},
  {8, 15, 19, 28, 32, 41, 51, 66, 71},  {8, 16, 20, 29, 33, 42, 52, 60, 72},
  {8, 17, 21, 30, 34, 43, 46, 61, 73},  {9, 11, 23, 26, 32, 43, 52, 55, 70},
  {9, 12, 24, 27, 33, 44, 46, 56, 71},  {9, 13, 18, 28, 34, 45, 47, 57, 72},
  {9, 14, 19, 29, 35, 39, 48, 58, 73},  {9, 15, 20, 30, 36, 40, 49, 59, 67},
  {9, 16, 21, 31, 37, 41, 50, 53, 68},  {9, 17, 22, 25, 38, 42, 51, 54, 69},
  {10, 11, 24, 29, 34, 40, 51, 53, 63}, {10, 12, 18, 30, 35, 41, 52, 54, 64},
  {10, 13, 19, 31, 36, 42, 46, 55, 65}, {10, 14, 20, 25, 37, 43, 47, 56, 66},
  {10, 15, 21, 26, 38, 44, 48, 57, 60}, {10, 16, 22, 27, 32, 45, 49, 58, 61},
  {10, 17, 23, 28, 33, 39, 50, 59, 62} }
```

Rezultat 4.

5 Osvrt na veće redove i zaključak

Klasifikacija ravnina reda 9 napravljena je na isti način kao u ovom radu. Objasnit ću zašto moji programi (uz današnja računala) ipak nisu primjenjivi na red 9.

Prvo mjesto na kojem nisam prešao na viši red je generiranje latinskih kvadrata. Za klasifikaciju ravnina reda 9 trebalo bi generirati listu predstavnika glavnih klasa reda 8. Moji programi su za to prespori. Razlog nije sporije računalo, nego to što nisam koristio nekoliko važnih faktora za ubrzanje koje su Brown i autori članka [10] koristili. Funkciju za ispitivanje izotopnosti dva kvadrata mogao sam napisati efikasnije, koristeći ideju iz napomene 3.15. Najvažniji razlog zbog kojeg je moj program sporiji je premali broj invarijanti. Skupovi latinskih pravokutnika s istim invarijantama su zbog toga veliki, pa se vrši previše testiranja izotopnosti.

Kad bih programu dodao još nekoliko invarijanti, najvjerojatnije bih bio u stanju generirati i predstavnike reda 8. To nisam napravio, jer je prepreka koju treba svladati u idućem koraku (generiranju incidencijskih matrica od latinskih kvadrata) puno veća. Prije nego što se osvrnem na taj problem, dat ću još nekoliko napomena o latinskim kvadratima.

Broj klasa izotopije i glavnih klasa kvadrata reda 9 i danas je otvoren problem. Ni programi autora članka [10] nisu dovoljni za $n = 9$. Razlog prema mojoj procjeni ne leži u procesorskoj snazi današnjih računala, nego u prevelikom broju klasa. Današnja računala nemaju dovoljno radne memorije da pohrane sve predstavnike reda 9. Prelazak na vanjsku memoriju, koja je bitno većeg kapaciteta, ne predstavlja rješenje, jer je ona ujedno i puno sporija. Problem bi se za $n = 9$ u skorijoj budućnosti možda mogao riješiti korištenjem paralelnih distribuiranih računala. Svaki čvor paralelnog računala pamtio bi jednu ili nekoliko grupa latinskih pravokutnika s istim invarijantama i bio bi zadužen za testiranje izotopnosti novih pravokutnika s tim grupama. Na taj bi način problem pamćenja velike količine kvadrata bio raspoređen na više računala, a dobili bi i dodatno ubrzanje procesorske snage.

Međutim, čak i kad bi imali predstavnike glavnih klasa reda 9, pristupom iz ovog rada ne bismo mogli riješiti problem klasifikacije za red 10. Konstruiranje incidencijskih matrica na opisani način predstavlja neusporedivo veći izazov nego što je generiranje latinskih kvadrata. Ilustrirat ću to na primjeru prelaska s reda 8 na red 9, koji nisam ostvario.

Pretpostavimo da program 7.10 treba isto toliko vremena za obrađivanje kvadrata reda 8 kao za kvadrat reda 7, oko 5 minuta. To je vrlo gruba procjena, jer se vrijeme prelaskom na viši red produljuje za nekoliko redova veličine. U tom slučaju obrada svih 283657 predstavnika glavnih klasa reda 8 trajala bi oko dvije i pol godine.

Iz toga proizlazi da je program korišten u [10] bitno brži od programa 7.10. Klasični backtracking uz današnja računala nije dovoljan za $n = 9$. Autori članka [10] koristili su modifikaciju algoritma, takozvanu matricu kompatibilnosti. Pomoću te metode izbjegnuto je generiranje predstavnika na nižim nivoima stabla, što dovodi do znatnog ubrzanja. Tehnika je razvijena u sklopu potrage za ravninom reda 10, a detaljan opis nalazi se u [12].

Prelazak na red 10 čini metodu ovog rada potpuno neupotrebljivom. Taj problem je riješen preko teorije binarnih linearnih kodova.

Na kraju ću iznijeti moje razmišljanje o primjeni računala u matematici. Nagli rast brzine računala omogućio je rješavanje matematičkih problema na nov način. "Dokazi" pomoću računala mogu sadržavati milijune slučajeva, koje ljudi nikada neće biti u stanju pojedinačno ispitati. To donosi mogućnost rješavanja problema koji su iznad ljudskih mogućnosti, ali ima i neke negativne posljedice. "Kompjuterski" dokazi redovito se vrlo teško provjeravaju. Čovjek gubi mogućnost provjere svakog pojedinog zaključka ili računa. Može se provjeriti jedino mehanizam koji vrši računanje, kompjuterski program. Primjer s latinskim kvadratima pokazuje da je vrlo lako napraviti grešku.

Često se krivo smatra da su svi rezultati dobiveni pomoću računala "eksperimentalne prirode", da nisu egzaktni. Taj prigovor nije uvijek na mjestu, jer je u mnogim slučajevima vjerojatnost da hardverska greška utječe na rezultat zanemarivo mala.

U članku [24] dan je pregled većine kombinatornih rezultata dobivenih pomoću računala do sredine šezdesetih godina. Sličan bi pothvat danas zahtijevao mnogo više od jednog članka. Neki od najvećih do jučer otvorenih problema riješeni su u zadnje vrijeme pomoću računala. Najpoznatiji primjeri su problem četiri boje i nepostojanje projektivne ravnine reda 10. Kako god se čovjek odnosio prema upotrebi računala, neosporno je da ona mijenjaju način rada i samo područje interesa u matematici. Na čitatelju ostaje prosudba jesu li te promjene pozitivne ili negativne.

6 DODATAK 1. Predstavnici glavnih klasa latinskih kvadrata.

$$\begin{matrix}1\end{matrix}$$

1.1

$$\begin{matrix}1&2\\2&1\end{matrix}$$

2.1

$$\begin{matrix}1&2&3\\2&3&1\\3&1&2\end{matrix}$$

3.1

$$\begin{matrix}1&2&3&4\\2&3&4&1\\3&4&1&2\\4&1&2&3\end{matrix}$$

4.1

$$\begin{matrix}1&2&3&4\\2&1&4&3\\3&4&1&2\\4&3&2&1\end{matrix}$$

4.2

$$\begin{matrix}1&2&3&4&5\\2&3&4&5&1\\3&4&5&1&2\\4&5&1&2&3\\5&1&2&3&4\end{matrix}$$

5.1

$$\begin{matrix}1&2&3&4&5\\2&1&4&5&3\\3&5&1&2&4\\4&3&5&1&2\\5&4&2&3&1\end{matrix}$$

5.2

$$\begin{matrix}1&2&3&4&5&6\\2&1&5&6&3&4\\3&6&1&5&4&2\\4&3&2&1&6&5\\5&4&6&3&2&1\\6&5&4&2&1&3\end{matrix}$$

6.1

$$\begin{matrix}1&2&3&4&5&6\\2&1&5&6&4&3\\5&3&1&2&6&4\\3&4&6&1&2&5\\6&5&4&3&1&2\\4&6&2&5&3&1\end{matrix}$$

6.2

$$\begin{matrix}1&2&3&4&5&6\\2&1&6&5&3&4\\5&6&1&2&4&3\\4&3&2&1&6&5\\6&5&4&3&1&2\\3&4&5&6&2&1\end{matrix}$$

6.3

$$\begin{matrix}1&2&3&4&5&6\\2&1&6&5&3&4\\5&4&1&6&2&3\\4&3&2&1&6&5\\6&5&4&3&1&2\\3&6&5&2&4&1\end{matrix}$$

6.4

$$\begin{matrix}1&2&3&4&5&6\\3&1&2&6&4&5\\2&3&1&5&6&4\\4&5&6&1&2&3\\6&4&5&3&1&2\\5&6&4&2&3&1\end{matrix}$$

6.5

$$\begin{matrix}1&2&3&4&5&6\\3&1&2&5&6&4\\2&3&1&6&4&5\\4&6&5&1&3&2\\5&4&6&2&1&3\\6&5&4&3&2&1\end{matrix}$$

6.6

$$\begin{matrix}1&2&3&4&5&6\\3&1&5&6&4&2\\5&6&1&2&3&4\\4&5&6&1&2&3\\6&4&2&3&1&5\\2&3&4&5&6&1\end{matrix}$$

6.7

$$\begin{matrix}1&2&3&4&5&6\\3&1&2&5&6&4\\2&3&1&6&4&5\\4&5&6&1&2&3\\5&6&4&3&1&2\\6&4&5&2&3&1\end{matrix}$$

6.8

$$\begin{matrix}1&2&3&4&5&6\\3&1&2&5&6&4\\2&3&1&6&4&5\\5&6&4&1&2&3\\6&4&5&3&1&2\\4&5&6&2&3&1\end{matrix}$$

6.9

$$\begin{matrix}1&2&3&4&5&6\\2&1&6&3&4&5\\3&5&1&2&6&4\\4&6&5&1&2&3\\5&3&4&6&1&2\\6&4&2&5&3&1\end{matrix}$$

6.10

$$\begin{matrix}1&2&3&4&5&6\\2&1&5&3&6&4\\3&6&1&2&4&5\\4&5&6&1&3&2\\6&4&2&5&1&3\\5&3&4&6&2&1\end{matrix}$$

6.11

$$\begin{matrix}1&2&3&4&5&6\\2&1&5&6&4&3\\3&4&1&5&6&2\\6&5&2&1&3&4\\4&3&6&2&1&5\\5&6&4&3&2&1\end{matrix}$$

6.12

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 6 7 4 3 5	2 1 4 6 7 3 5	2 1 7 6 4 5 3	2 1 5 6 7 4 3	2 1 7 6 3 4 5	2 1 4 3 6 7 5	2 1 4 3 6 7 5	2 1 7 6 4 5 3	2 1 4 5 3 7 6
3 4 2 6 7 5 1	7 5 1 3 6 2 4	7 3 1 2 6 4 5	6 3 1 5 4 7 2	6 7 1 3 4 5 2	3 6 1 7 2 5 4	3 7 1 5 4 2 6	7 5 1 2 6 3 4	6 5 1 7 2 4 3
4 3 1 5 2 7 6	5 6 7 1 2 4 3	3 4 5 1 2 7 6	4 7 6 1 2 3 5	4 3 5 1 2 7 6	4 5 2 1 7 3 6	4 6 7 1 2 5 3	3 4 5 1 2 7 6	7 3 2 1 6 5 4
5 7 4 3 6 1 2	4 3 6 7 1 5 2	6 5 2 7 1 3 4	7 4 2 3 1 5 6	3 5 6 7 1 2 4	5 4 7 6 1 2 3	6 4 5 7 1 3 2	6 3 2 7 1 4 5	4 7 5 6 1 3 2
6 5 7 1 3 2 4	3 7 2 5 4 1 6	4 7 6 5 3 1 2	5 6 7 2 3 1 4	5 6 4 2 7 1 3	7 3 6 5 4 1 2	5 3 2 6 7 1 4	4 7 6 5 3 1 2	3 4 6 2 7 1 5
7 6 5 2 1 4 3	6 4 5 2 3 7 1	5 6 4 3 7 2 1	3 5 4 7 6 2 1	7 4 2 5 6 3 1	6 7 5 2 3 4 1	7 5 6 2 3 4 1	5 6 4 3 7 2 1	5 6 7 3 4 2 1
7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	7.9

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 7 5 6 4 3	2 1 5 3 4 7 6	2 1 4 5 3 7 6	2 1 6 5 7 3 4	2 1 4 3 7 5 6	2 1 5 6 7 4 3	2 1 5 3 4 7 6	2 1 5 3 4 7 6	2 1 5 3 4 7 6
3 5 1 2 4 7 6	6 3 1 7 2 5 4	6 4 1 2 7 5 3	6 5 1 7 2 4 3	3 6 1 5 4 7 2	6 3 1 5 4 7 2	6 3 1 7 2 5 4	6 4 1 5 7 3 2	6 3 1 2 7 4 5
7 3 6 1 2 5 4	7 5 4 1 6 2 3	7 3 2 1 6 4 5	3 7 4 1 6 5 2	4 7 6 1 2 3 5	4 7 6 1 2 3 5	7 5 4 1 6 2 3	7 3 4 1 6 2 5	7 4 2 1 6 5 3
4 7 5 6 1 3 2	3 7 2 6 1 4 5	3 7 5 6 1 2 4	4 3 2 6 1 7 5	7 4 5 6 1 2 3	7 5 4 3 1 2 6	3 7 6 2 1 4 5	4 7 2 6 1 5 3	5 7 4 6 1 3 2
6 4 2 7 3 1 5	4 6 7 5 3 1 2	5 6 7 3 4 1 2	7 4 5 2 3 1 6	5 3 7 2 6 1 4	5 6 7 2 3 1 4	4 6 7 5 3 1 2	5 6 7 2 3 1 4	3 6 7 5 2 1 4
5 6 4 3 7 2 1	5 4 6 2 7 3 1	4 5 6 7 2 3 1	5 6 7 3 4 2 1	6 5 2 7 3 4 1	3 4 2 7 6 5 1	5 4 2 6 7 3 1	3 5 6 7 2 4 1	4 5 6 7 3 2 1
7.10	7.11	7.12	7.13	7.14	7.15	7.16	7.17	7.18

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 4 7 6 5 3	2 1 4 5 3 7 6	2 1 7 6 4 5 3	2 1 7 3 6 5 4	2 1 4 7 6 5 3	2 1 4 7 6 5 3	2 1 4 7 6 5 3	2 1 6 5 4 7 3	2 1 4 5 3 7 6
6 4 1 5 7 3 2	6 3 1 7 4 2 5	3 6 1 2 7 4 5	5 4 1 6 7 2 3	5 7 1 3 4 2 6	5 6 1 3 7 2 4	5 4 1 6 3 7 2	5 4 1 2 7 3 6	6 3 1 7 2 5 4
5 3 7 1 4 2 6	7 5 6 1 2 4 3	7 3 5 1 6 2 4	6 7 4 1 2 3 5	6 5 7 1 2 3 4	6 4 5 1 3 7 2	6 5 7 1 2 3 4	4 6 7 1 3 2 5	7 6 5 1 4 2 3
3 7 6 2 1 4 5	3 4 7 6 1 5 2	4 7 2 5 1 3 6	7 3 6 5 1 4 2	4 3 6 2 1 7 5	3 7 2 5 1 4 6	3 7 2 5 1 4 6	3 7 2 6 1 5 4	3 7 2 6 1 4 5
7 6 5 3 2 1 4	5 6 2 3 7 1 4	5 4 6 7 3 1 2	3 5 2 7 4 1 6	3 4 5 6 7 1 2	7 3 6 2 4 1 5	7 3 6 2 4 1 5	7 5 4 3 6 1 2	5 4 6 3 7 1 2
4 5 2 6 3 7 1	4 7 5 2 6 3 1	6 5 4 3 2 7 1	4 6 5 2 3 7 1	7 6 2 5 3 4 1	4 5 7 6 2 3 1	4 6 5 3 7 2 1	6 3 5 7 2 4 1	4 5 7 2 6 3 1
7.19	7.20	7.21	7.22	7.23	7.24	7.25	7.26	7.27

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 4 7 6 5 3	2 1 7 3 6 5 4	2 1 6 7 4 3 5	2 1 4 5 3 7 6	2 1 4 5 3 7 6	2 1 6 7 4 3 5	2 1 7 6 4 5 3	2 1 4 6 7 3 5	2 1 6 5 4 7 3
6 4 1 3 7 2 5	6 3 1 2 7 4 5	3 4 1 5 2 7 6	6 4 1 2 7 5 3	6 4 1 7 2 5 3	3 4 1 5 2 7 6	7 3 1 5 6 4 2	7 5 1 3 2 4 6	5 3 1 7 2 4 6
5 3 2 1 4 7 6	5 7 2 1 4 3 6	6 7 5 1 3 2 4	7 3 5 1 6 2 4	7 3 5 1 6 2 4	6 3 2 1 7 5 4	3 6 5 1 7 2 4	5 6 7 1 3 2 4	4 6 7 1 3 2 5
4 6 7 5 1 3 2	3 4 6 5 1 7 2	5 6 7 2 1 4 3	3 7 2 6 1 4 5	3 7 2 6 1 4 5	5 6 7 2 1 4 3	5 4 2 3 1 7 6	3 4 6 5 1 7 2	3 7 2 6 1 5 4
3 7 5 6 2 1 4	4 6 5 7 2 1 3	7 5 4 3 6 1 2	5 6 7 3 4 1 2	5 6 7 3 4 1 2	7 5 4 3 6 1 2	6 7 4 2 3 1 5	6 7 5 2 4 1 3	7 5 4 3 6 1 2
7 5 6 2 3 4 1	7 5 4 6 3 2 1	4 3 2 6 7 5 1	4 5 6 7 2 3 1	4 5 6 2 7 3 1	4 7 5 6 3 2 1	4 5 6 7 2 3 1	4 3 2 7 6 5 1	6 4 5 2 7 3 1
7.28	7.29	7.30	7.31	7.32	7.33	7.34	7.35	7.36

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 5 6 3 7 4	2 1 4 6 7 3 5	2 1 5 6 7 4 3	2 1 7 6 3 4 5	2 1 5 3 4 7 6	2 1 5 3 4 7 6	2 1 7 6 3 4 5	2 1 4 5 3 7 6	2 1 6 7 4 3 5
5 3 1 7 2 4 6	7 5 1 3 6 2 4	6 7 1 3 2 5 4	6 3 1 5 7 2 4	6 5 1 2 7 3 4	6 3 1 2 7 4 5	6 5 1 7 4 3 2	6 5 1 3 7 4 2	6 7 1 5 2 4 3
3 6 7 1 4 2 5	5 7 6 1 2 4 3	4 5 7 1 6 3 2	4 7 5 1 6 3 2	7 3 6 1 2 4 5	7 5 6 1 2 3 4	4 6 2 1 7 5 3	7 3 6 1 4 2 5	3 4 7 1 6 5 2
7 4 6 3 1 5 2	4 3 2 7 1 5 6	3 4 2 5 1 7 6	3 5 4 2 1 7 6	4 7 2 6 1 5 3	4 7 2 6 1 5 3	5 7 4 3 1 2 6	5 7 2 6 1 3 4	4 3 5 2 1 7 6
6 5 4 2 7 1 3	3 6 7 5 4 1 2	7 6 4 2 3 1 5	5 6 2 7 4 1 3	5 6 4 7 3 1 2	5 6 4 7 3 1 2	7 3 5 2 6 1 4	4 6 5 7 2 1 3	5 6 2 3 7 1 4
4 7 2 5 6 3 1	6 4 5 2 3 7 1	5 3 6 7 4 2 1	7 4 6 3 2 5 1	3 4 7 5 6 2 1	3 4 7 5 6 2 1	3 4 6 5 2 7 1	3 4 7 2 6 5 1	7 5 4 6 3 2 1
7.37	7.38	7.39	7.40	7.41	7.42	7.43	7.44	7.45

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 4 5 3 7 6	2 1 6 5 7 3 4	2 1 7 3 6 5 4	2 1 7 3 6 5 4	2 1 6 5 7 3 4	2 1 4 5 3 7 6	7 1 4 6 3 2 5	7 1 5 6 4 2 3	7 1 5 3 6 2 4
6 3 1 7 2 4 5	6 4 1 2 3 7 5	6 3 1 2 7 4 5	6 3 1 2 7 4 5	6 4 1 7 3 5 2	6 5 1 7 2 4 3	2 3 1 5 7 4 6	2 5 1 7 3 4 6	2 6 1 5 4 7 3
7 4 2 1 6 5 3	3 7 5 1 6 4 2	5 6 2 1 4 7 3	5 7 6 1 4 2 3	3 6 7 1 2 4 5	7 3 6 1 4 2 5	5 6 2 1 4 7 3	3 6 4 1 2 7 5	4 5 2 1 7 3 6
3 7 5 6 1 2 4	7 5 4 3 1 2 6	7 5 4 6 1 3 2	7 5 4 6 1 3 2	4 3 5 2 1 7 6	3 4 7 6 1 5 2	6 5 7 2 1 3 4	6 4 7 3 1 5 2	6 3 4 7 1 5 2
5 6 7 3 4 1 2	5 6 2 7 4 1 3	3 4 5 7 2 1 6	3 4 5 7 2 1 6	7 5 2 6 4 1 3	5 6 2 3 7 1 4	3 4 5 7 6 1 2	5 3 6 2 7 1 4	3 4 7 6 2 1 5
4 5 6 2 7 3 1	4 3 7 6 2 5 1	4 7 6 5 3 2 1	4 6 2 5 3 7 1	5 7 4 3 6 2 1	4 7 5 2 6 3 1	4 7 6 3 2 5 1	4 7 2 5 6 3 1	5 7 6 2 3 4 1
7.46	7.47	7.48	7.49	7.50	7.51	7.52	7.53	7.54

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
7 1 6 5 3 2 4	7 1 4 6 2 3 5	2 1 7 3 6 5 4	2 1 7 6 3 4 5	2 1 7 6 3 4 5	2 1 6 5 7 3 4	2 1 7 6 3 4 5	2 1 5 6 7 4 3	2 1 7 6 3 4 5
2 6 1 7 4 3 5	6 4 1 7 3 5 2	6 3 1 2 7 4 5	6 7 1 3 4 5 2	6 3 1 7 2 5 4	6 7 1 2 3 4 5	6 3 1 5 4 7 2	6 3 1 7 4 5 2	6 3 1 7 2 5 4
4 3 7 1 2 5 6	5 3 7 1 4 2 6	5 7 6 1 4 2 3	4 5 2 1 7 3 6	4 5 2 1 6 7 3	3 4 7 1 6 5 2	4 7 6 1 2 5 3	4 6 7 1 2 3 5	4 7 5 1 6 2 3
6 5 4 3 1 7 2	2 7 6 5 1 4 3	7 6 4 5 1 3 2	5 6 4 7 1 2 3	7 4 5 2 1 3 6	7 5 4 3 1 2 6	7 5 4 3 1 2 6	3 4 2 5 1 7 6	7 4 6 5 1 3 2
5 4 2 6 7 1 3	3 6 5 2 7 1 4	3 4 5 7 2 1 6	7 3 5 2 6 1 4	3 6 4 5 7 1 2	5 6 2 7 4 1 3	3 6 5 2 7 1 4	7 5 6 2 3 1 4	3 5 4 2 7 1 6
3 7 5 2 6 4 1	4 5 2 3 6 7 1	4 5 2 6 3 7 1	3 4 6 5 2 7 1	5 7 6 3 4 2 1	4 3 5 6 2 7 1	5 4 2 7 6 3 1	5 7 4 3 6 2 1	5 6 2 3 4 7 1
7.55	7.56	7.57	7.58	7.59	7.60	7.61	7.62	7.63

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 5 3 4 7 6	7 1 4 2 6 3 5	7 1 4 2 6 3 5	2 1 5 6 7 4 3	2 1 4 5 3 7 6	2 1 6 5 7 3 4	2 1 7 6 3 4 5	2 1 4 5 3 7 6	2 1 7 3 6 5 4
6 5 1 7 3 2 4	2 3 1 7 4 5 6	2 4 1 3 7 5 6	6 4 1 7 2 3 5	6 4 1 7 2 5 3	6 5 1 7 2 4 3	6 5 1 2 4 7 3	6 3 1 7 2 5 4	6 4 1 5 2 7 3
7 6 4 1 2 3 5	5 6 7 1 3 2 4	5 6 7 1 3 2 4	4 5 2 1 3 7 6	7 3 2 1 6 4 5	3 7 4 1 6 5 2	4 3 6 1 7 5 2	7 4 2 1 6 3 5	5 3 2 1 7 4 6
5 7 6 2 1 4 3	4 5 2 6 1 7 3	4 5 2 6 1 7 3	7 3 6 5 1 2 4	3 7 5 6 1 2 4	4 3 5 2 1 7 6	5 4 2 7 1 3 6	5 7 6 2 1 4 3	4 5 6 7 1 3 2
3 4 7 5 6 1 2	6 4 5 3 7 1 2	6 3 5 7 4 1 2	5 6 7 3 4 1 2	5 6 7 3 4 1 2	7 6 2 3 4 1 5	7 6 5 3 2 1 4	4 6 5 3 7 1 2	7 6 4 2 3 1 5
4 3 2 6 7 5 1	3 7 6 5 2 4 1	3 7 6 5 2 4 1	3 7 4 2 6 5 1	4 5 6 2 7 3 1	5 4 7 6 3 2 1	3 7 4 5 6 2 1	3 5 7 6 4 2 1	3 7 5 6 4 2 1
7.64	7.65	7.66	7.67	7.68	7.69	7.70	7.71	7.72

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 7 3 6 5 4	2 1 5 3 4 7 6	2 1 4 7 6 5 3	2 1 4 5 3 7 6	2 1 7 3 6 5 4	2 1 4 7 6 5 3	7 1 6 3 4 2 5	2 1 7 3 6 5 4	2 1 4 7 6 5 3
6 7 1 2 4 3 5	6 5 1 2 7 3 4	6 7 1 5 3 4 2	6 5 1 7 2 4 3	6 4 1 5 7 3 2	6 3 1 5 4 7 2	2 6 1 7 3 5 4	6 3 1 2 7 4 5	6 3 1 2 7 4 5
5 3 6 1 7 4 2	7 3 6 1 2 4 5	5 3 6 1 7 2 4	7 3 2 1 6 5 4	5 7 2 1 3 4 6	5 4 2 1 7 3 6	4 5 2 1 7 3 6	5 7 4 1 3 2 6	5 4 6 1 3 7 2
7 6 4 5 1 2 3	5 7 4 6 1 2 3	3 4 5 2 1 7 6	4 7 5 6 1 3 2	4 5 6 2 1 7 3	4 6 7 3 1 2 5	6 3 4 5 1 7 2	3 4 6 5 1 7 2	4 7 5 3 1 2 6
3 4 5 7 2 1 6	3 4 7 5 6 1 2	7 6 2 3 4 1 5	3 6 7 2 4 1 5	7 3 4 6 2 1 5	3 7 5 6 2 1 4	5 4 7 6 2 1 3	4 6 5 7 2 1 3	3 6 7 5 2 1 4
4 5 2 6 3 7 1	4 6 2 7 3 5 1	4 5 7 6 2 3 1	5 4 6 3 7 2 1	3 6 5 7 4 2 1	7 5 6 2 3 4 1	3 7 5 2 6 4 1	7 5 2 6 4 3 1	7 5 2 6 4 3 1
7.73	7.74	7.75	7.76	7.77	7.78	7.79	7.80	7.81

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 4 7 6 5 3	2 1 5 3 4 7 6	2 1 6 5 7 3 4	2 1 4 5 3 7 6	2 1 7 3 6 5 4	2 1 7 3 6 5 4	2 1 7 5 4 3 6	7 1 5 2 6 4 3	7 1 4 6 2 5 3
6 4 1 2 3 7 5	6 3 1 5 7 2 4	6 4 1 7 2 5 3	6 5 1 7 4 2 3	6 4 1 5 2 7 3	6 3 1 7 4 2 5	5 6 1 2 3 7 4	2 6 1 5 3 7 4	2 6 1 3 4 7 5
5 3 6 1 7 4 2	7 5 2 1 6 4 3	3 6 5 1 4 7 2	7 6 5 1 2 3 4	5 7 4 1 3 2 6	5 7 4 1 2 3 6	4 7 5 1 6 2 3	3 4 6 1 7 2 5	3 5 6 1 7 4 2
4 7 5 3 1 2 6	4 7 6 2 1 3 5	5 3 7 2 1 4 6	3 7 2 6 1 4 5	7 3 6 2 1 4 5	3 4 5 6 1 7 2	7 3 4 6 1 5 2	6 3 4 7 1 5 2	6 3 5 7 1 2 4
3 6 7 5 2 1 4	5 6 4 7 3 1 2	4 7 2 3 6 1 5	5 4 6 3 7 1 2	3 6 5 7 4 1 2	4 6 2 5 7 1 3	3 4 6 7 2 1 5	4 5 7 3 2 1 6	5 4 7 2 3 1 6
7 5 2 6 4 3 1	3 4 7 6 2 5 1	7 5 4 6 3 2 1	4 3 7 2 6 5 1	4 5 2 6 7 3 1	7 5 6 2 3 4 1	6 5 2 3 7 4 1	5 7 2 6 4 3 1	4 7 2 5 6 3 1
7.82	7.83	7.84	7.85	7.86	7.87	7.88	7.89	7.90

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 6 5 7 3 4	2 1 5 6 7 4 3	2 1 7 3 6 5 4	2 1 5 7 3 4 6	7 1 2 3 4 5 6	2 1 7 6 3 4 5	7 1 6 2 3 4 5	2 1 6 5 7 3 4	2 1 5 7 3 4 6
6 4 1 7 3 5 2	6 4 1 7 2 3 5	6 7 1 5 4 2 3	5 3 1 6 2 7 4	2 5 1 7 6 3 4	6 3 1 5 4 7 2	2 3 1 5 6 7 4	6 4 1 3 2 7 5	5 3 1 6 2 7 4
3 7 5 1 2 4 6	4 7 2 1 3 5 6	5 6 4 1 3 7 2	3 6 7 1 4 2 5	6 3 7 1 2 4 5	4 7 6 1 2 5 3	3 5 4 1 7 2 6	3 7 5 1 4 2 6	3 6 4 1 7 5 2
4 5 7 6 1 2 3	7 3 6 5 1 2 4	7 3 6 2 1 4 5	7 4 6 3 1 5 2	3 4 5 6 1 7 2	7 6 5 3 1 2 4	4 6 5 7 1 3 2	7 5 2 6 1 4 3	6 4 7 3 1 2 5
7 6 2 3 4 1 5	5 6 7 3 4 1 2	3 4 5 7 2 1 6	4 7 2 5 6 1 3	5 6 4 2 7 1 3	3 5 4 2 7 1 6	5 4 7 6 2 1 3	5 6 4 7 3 1 2	7 5 6 2 4 1 3
5 3 4 2 6 7 1	3 5 4 2 6 7 1	4 5 2 6 7 3 1	6 5 4 2 7 3 1	4 7 6 5 3 2 1	5 4 2 7 6 3 1	6 7 2 3 4 5 1	4 3 7 2 6 5 1	4 7 2 5 6 3 1
7.91	7.92	7.93	7.94	7.95	7.96	7.97	7.98	7.99

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 7 5 4 3 6	7 1 4 6 3 2 5	2 1 6 3 7 4 5	2 1 7 3 6 5 4	2 1 5 6 3 7 4	7 1 5 3 2 4 6	7 1 2 3 4 5 6	7 1 5 2 6 4 3	7 1 4 2 6 3 5
5 4 1 6 3 7 2	2 5 1 7 6 4 3	5 4 1 6 3 7 2	5 6 1 7 3 4 2	5 4 1 7 6 3 2	2 6 1 7 4 5 3	2 4 1 5 6 7 3	2 4 1 6 3 7 5	2 4 1 3 7 5 6
4 6 2 1 7 5 3	5 6 7 1 4 3 2	7 6 4 1 2 5 3	6 3 4 1 7 2 5	3 6 2 1 7 4 5	6 4 2 1 7 3 5	5 3 6 1 7 2 4	3 5 4 1 7 2 6	5 6 2 1 3 7 4
6 3 4 7 1 2 5	6 3 2 5 1 7 4	6 3 7 5 1 2 4	4 7 5 2 1 3 6	4 7 6 2 1 5 3	5 3 6 2 1 7 4	3 6 7 2 1 4 5	4 6 7 5 1 3 2	4 5 7 6 1 2 3
7 5 6 3 2 1 4	3 4 5 2 7 1 6	3 5 2 7 4 1 6	7 4 6 5 2 1 3	7 3 4 5 2 1 6	4 5 7 6 3 1 2	6 5 4 7 3 1 2	5 3 6 7 2 1 4	6 3 5 7 4 1 2
3 7 5 2 6 4 1	4 7 6 3 2 5 1	4 7 5 2 6 3 1	3 5 2 6 4 7 1	6 5 7 3 4 2 1	3 7 4 5 6 2 1	4 7 5 6 2 3 1	6 7 2 3 4 5 1	3 7 6 5 2 4 1
7.100	7.101	7.102	7.103	7.104	7.105	7.106	7.107	7.108

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
7 1 6 5 3 2 4	7 1 6 5 2 4 3	7 1 5 3 6 2 4	7 1 5 2 6 4 3	7 1 6 3 2 5 4	7 1 6 2 4 5 3	7 1 2 3 6 4 5	7 1 2 6 3 5 4	7 1 5 6 2 3 4
2 4 1 6 7 5 3	2 6 1 7 3 5 4	2 4 1 7 3 5 6	2 6 1 7 3 5 4	2 3 1 6 4 7 5	2 3 1 5 7 4 6	6 3 1 5 7 2 4	2 4 1 5 6 7 3	2 4 1 3 7 5 6
4 6 7 1 2 3 5	6 3 2 1 4 7 5	3 5 6 1 7 4 2	5 3 6 1 4 7 2	4 5 2 1 7 3 6	6 5 7 1 2 3 4	2 4 7 1 3 5 6	6 5 4 1 7 3 2	6 5 2 1 4 7 3
6 5 4 3 1 7 2	5 4 7 6 1 3 2	6 3 4 2 1 7 5	4 5 7 3 1 2 6	6 4 7 5 1 2 3	4 6 5 3 1 7 2	4 5 6 7 1 3 2	3 6 7 2 1 4 5	5 3 6 7 1 4 2
5 3 2 7 4 1 6	3 5 4 2 7 1 6	4 6 7 5 2 1 3	3 4 2 6 7 1 5	5 6 4 7 3 1 2	3 4 2 7 6 1 5	5 7 4 6 2 1 3	4 3 5 7 2 1 6	4 6 7 2 3 1 5
3 7 5 2 6 4 1	4 7 5 3 6 2 1	5 7 2 6 4 3 1	6 7 4 5 2 3 1	3 7 5 2 6 4 1	5 7 4 6 3 2 1	3 6 5 2 4 7 1	5 7 6 3 4 2 1	3 7 4 5 6 2 1
7.109	7.110	7.111	7.112	7.113	7.114	7.115	7.116	7.117

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
7 1 4 6 2 3 5	2 1 5 7 3 4 6	2 1 5 7 3 4 6	2 1 7 5 6 4 3	2 1 7 5 6 4 3	2 1 7 5 6 4 3	2 1 6 3 7 4 5	2 1 6 3 7 4 5	2 1 4 6 7 3 5
6 7 1 2 3 5 4	3 5 1 6 2 7 4	3 5 1 6 2 7 4	7 3 1 6 4 2 5	7 5 1 6 3 2 4	7 5 1 6 4 3 2	7 5 1 2 6 3 4	7 5 1 6 3 2 4	7 5 1 3 4 2 6
5 3 7 1 4 2 6	5 4 7 1 6 3 2	5 4 6 1 7 3 2	3 7 4 1 2 5 6	3 7 4 1 2 5 6	3 7 4 1 2 5 6	5 7 4 1 3 2 6	5 3 2 1 4 7 6	5 6 7 1 2 4 3
3 5 6 7 1 4 2	6 7 4 5 1 2 3	6 7 4 5 1 2 3	4 6 5 3 1 7 2	4 6 5 3 1 7 2	6 4 2 3 1 7 5	3 4 7 6 1 5 2	4 6 5 7 1 3 2	4 3 6 7 1 5 2
4 6 2 5 7 1 3	7 6 2 3 4 1 5	7 6 2 3 4 1 5	6 5 2 7 3 1 4	6 3 2 7 4 1 5	5 3 6 2 7 1 4	4 6 5 7 2 1 3	6 7 4 5 2 1 3	3 7 2 5 6 1 4
2 4 5 3 6 7 1	4 3 6 2 7 5 1	4 3 7 2 6 5 1	5 4 6 2 7 3 1	5 4 6 2 7 3 1	4 6 5 7 3 2 1	6 3 2 5 4 7 1	3 4 7 2 6 5 1	6 4 5 2 3 7 1
7.118	7.119	7.120	7.121	7.122	7.123	7.124	7.125	7.126

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 7 6 4 5 3	2 1 5 6 3 7 4	2 1 4 7 6 5 3	2 1 7 3 6 5 4	2 1 5 3 4 7 6	2 1 5 7 3 4 6	2 1 7 6 4 5 3	2 1 7 5 6 4 3	2 1 7 3 6 5 4
7 3 1 5 6 2 4	3 5 1 7 2 4 6	6 7 1 2 3 4 5	6 7 1 2 4 3 5	6 4 1 7 3 5 2	5 4 1 6 2 7 3	3 4 1 5 2 7 6	3 6 1 7 2 5 4	5 4 1 6 2 7 3
3 6 5 1 7 4 2	5 6 4 1 7 2 3	5 3 6 1 4 7 2	5 4 6 1 3 7 2	7 5 6 1 2 3 4	3 5 7 1 6 2 4	7 5 2 1 6 3 4	7 5 6 1 4 3 2	6 7 4 1 3 2 5
5 4 2 3 1 7 6	7 4 6 3 1 5 2	3 4 7 5 1 2 6	4 3 5 7 1 2 6	4 3 7 6 1 2 5	7 6 4 5 1 3 2	5 7 6 3 1 4 2	5 7 4 3 1 2 6	7 3 6 5 1 4 2
6 7 4 2 3 1 5	6 3 7 2 4 1 5	7 6 5 3 2 1 4	7 6 4 5 2 1 3	5 6 4 2 7 1 3	4 3 6 2 7 1 5	6 3 4 2 7 1 5	4 3 2 6 7 1 5	3 5 2 7 4 1 6
4 5 6 7 2 3 1	4 7 2 5 6 3 1	4 5 2 6 7 3 1	3 5 2 6 7 4 1	3 7 2 5 6 4 1	6 7 2 3 4 5 1	4 6 5 7 3 2 1	6 4 5 2 3 7 1	4 6 5 2 7 3 1
7.127	7.128	7.129	7.130	7.131	7.132	7.133	7.134	7.135

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
2 1 7 6 4 5 3	2 1 7 6 4 5 3	2 1 4 5 3 7 6	2 1 4 5 3 7 6	2 1 5 7 3 4 6	2 1 7 6 3 4 5	2 1 6 5 7 3 4	2 1 6 5 7 3 4	7 1 6 3 4 2 5
3 7 1 5 6 4 2	3 7 1 5 6 4 2	6 5 1 3 7 4 2	6 3 1 7 2 4 5	5 3 1 6 2 7 4	6 5 1 3 7 2 4	6 5 1 2 4 7 3	6 4 1 7 3 5 2	2 4 1 5 7 3 6
7 6 4 1 3 2 5	7 6 5 1 3 2 4	7 4 2 1 6 3 5	7 6 2 1 4 5 3	3 4 7 1 6 2 5	4 3 5 1 6 7 2	3 6 7 1 2 4 5	3 7 2 1 6 4 5	5 6 7 1 3 4 2
5 4 2 3 1 7 6	5 4 2 3 1 7 6	5 3 7 6 1 2 4	3 7 5 6 1 2 4	7 6 4 5 1 3 2	3 4 2 7 1 5 6	7 3 4 6 1 5 2	4 5 7 3 1 2 6	6 3 5 2 1 7 4
6 3 5 2 7 1 4	6 3 4 2 7 1 5	4 6 5 7 2 1 3	5 4 6 3 7 1 2	4 5 6 2 7 1 3	5 7 6 2 4 1 3	5 4 2 7 3 1 6	7 6 5 2 4 1 3	4 5 2 7 6 1 3
4 5 6 7 2 3 1	4 5 6 7 2 3 1	3 7 6 2 4 5 1	4 5 7 2 6 3 1	6 7 2 3 4 5 1	7 6 4 5 2 3 1	4 7 5 3 6 2 1	5 3 4 6 2 7 1	3 7 4 6 2 5 1
7.136	7.137	7.138	7.139	7.140	7.141	7.142	7.143	7.144

1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
7 1 2 3 4 5 6	7 1 2 3 4 5 6	7 1 4 6 3 2 5
6 7 1 2 3 4 5	2 4 1 7 6 3 5	2 6 1 7 4 5 3
5 6 7 1 2 3 4	6 3 5 1 2 7 4	3 4 5 1 2 7 6
4 5 6 7 1 2 3	3 5 7 6 1 4 2	6 5 7 3 1 4 2
3 4 5 6 7 1 2	5 6 4 2 7 1 3	5 3 6 2 7 1 4
2 3 4 5 6 7 1	4 7 6 5 3 2 1	4 7 2 5 6 3 1
7.145	7.146	7.147

7 DODATAK 2. Programi.

7.1 `reduced.c`

Program uzima prirodni broj n s ulaza i generira sve reducirane latinske kvadrate reda n.

```
/*

  REDUCED.C
  A program that generates all reduced
  latin squares of a given order.
  Vedran Krcadinac (krcko@student.math.hr)
  Department of Mathematics, U-Zagreb, Croatia

*/

#include <stdlib.h>
#include <stdio.h>

#define MAXN 9      /* Maximal order of the latin square */
#define BUFFSIZE 10 /* Size of the buffer for condensed output */
#define COLS 80     /* Width of the screen */


static int n=0;            /* Actual order of the latin square */
static int ls[MAXN][MAXN]; /* The latin square */
static double counter=0.0; /* The number of squares found */

static int mask=3;         /* An integer masking the output format */
/* Meaning of the bits and options to put them on/off are as following:
   1  When the search is finished the total number of latin squares is
      reported. -c for yes, -C for no (the default is yes).
   2  Squares are printed to the standard output. -o for yes, -O for no
      (the default is yes).
   4  Condensed output. -f for yes, -F for no (the default is no)
   8  The order n is printed at the beginning. This is useful if the
      output is redirected to 'isofilter' or some other programs.
      -l for yes, -L for no (the default is no)
*/


/* The output function */

static int buffer[BUFFSIZE][MAXN][MAXN]; /* A buffer to store a few
                                            latin squares            */
static int stored=0;                     /* Number of stored squares */

void output(flush)
int flush; /* A boolean variable. If it is set the squares stored in
      buffer are printed, otherwise ls is entered in buffer   */
{ int i,j,k,m;
  if (flush)
  { if (mask & 4)
```

```
    { for(i=0; i<n; ++i)
      { for(j=0; j<stored; ++j)
{ for(k=0; k<n; ++k) printf("%2i",buffer[j][i][k]);
  printf("  ");
}
printf("\n");
      }
      printf("\n");
    }
  }
  else
  { counter = counter + 1.0;
    if (mask & 2)
    { if (mask & 4)
      { for(i=0; i<n; ++i)
 for(j=0; j<n; ++j) buffer[stored][i][j] = ls[i][j];
        ++stored;
m = (COLS - 1)/(2*n + 2); /* Number of squares printed horizontally */
if (stored == m) { output(1); stored = 0; }
      }
      else
      { for(i=0; i<n; ++i)
{ for(j=0; j<n; ++j) printf("%d ",ls[i][j]);
          printf("\n");
        }
        printf("\n");
      }
    }
  }
}


/* The main function */

void main(argc,argv)
int argc;
char *argv[];
{ int i,j;
  int latin;            /* A boolean variable */
  int pos[MAXN][MAXN];  /* Positions of the symbols. pos[i][j] holds the
                           position of the i-th symbol in the j-th row    */
  int row[MAXN];        /* The i-th symbol is being put on row number
                           row[i]                                          */

  /* Command line arguments */
  if (argc > 1)
  { n = argv[1][0]-'0';
    for(i=1; i<argc; ++i)
    { j=0;
      if (argv[i][0] == '-') j=1;
      if (argv[i][j] == 'c') mask |= 1;
      if (argv[i][j] == 'C') mask &= ~1;
      if (argv[i][j] == 'o') mask |= 2;
      if (argv[i][j] == 'O') mask &= ~2;
      if (argv[i][j] == 'f') mask |= 4;
```

```
      if (argv[i][j] == 'F') mask &= ~4;
      if (argv[i][j] == 'l') mask |= 8;
      if (argv[i][j] == 'L') mask &= ~8;
      if ((argv[i][j] == 'h') || (argv[i][j] == 'H') ||
          (argv[i][j] == '?'))
      { printf("Usage: reduced [order] [-cCoOfFlL]\n");
        printf("Options:\n");
        printf("  Report number of squares: -c yes, -C no (default yes)\n");
        printf("  Print squares: -o yes, -O no (default yes)\n");
        printf("  Condensed output: -f yes, -F no (default no)\n");
printf("  Print order N: -l yes, -L no (default no)\n");
        exit(1);
      }
    }
  }

  /* The requested order should be the first argument. If this is not
     the case, it will have to be entered interactively                */
  while ((n < 1) || (n >= MAXN))
  { printf("\nOrder of latin squares (1-%d)? ",MAXN-1);
    scanf("%i",&n);
  }

  if ((mask & 8) == 8) printf("%d\n\n",n);
  else  printf("\n");


  /* Initial setup */
  for(i=0; i<n; ++i)
    for(j=0; j<n; ++j) { pos[i][j] = 1;  ls[i][j] = 0;  }
  for(i=0; i<n; ++i)
  { pos[i][0]=i;  pos[i][i]=0;  ls[i][0]=i+1;
    ls[0][i]=i+1;  row[i]=1;
  }
  row[1]=2;

  i=0;  /* The i-th symbol is being put on the square */

  /* The backtrack algorythm */
  while (i >= 0)
  { while ((pos[i][row[i]] < n) && (ls[row[i]][pos[i][row[i]]] != 0))
      ++(pos[i][row[i]]);
    if (pos[i][row[i]] == n)
    { pos[i][row[i]] = 1;
      --(row[i]);
      if (row[i] == i) --(row[i]);
      if (row[i] > 0) { ls[row[i]][pos[i][row[i]]] = 0;
++(pos[i][row[i]]);
      }
      else  { if (i == 1) row[i] = 2;
              else row[i] = 1;
              --i;
      if (i >= 0) { row[i]=n-1;
                      ls[row[i]][pos[i][row[i]]] = 0;
    ++(pos[i][row[i]]);
```

```
  }
    }
    }
    else
    { latin = 1;
      j = 0;
      while ((j < row[i]) && latin)
      { if (pos[i][row[i]] == pos[i][j]) latin = 0;
        ++j;
      }
      if (latin)
      { ls[row[i]][pos[i][row[i]]] = i+1;
++(row[i]);
if (row[i] == i) ++(row[i]);
if (row[i] >= n)
        { if (i == n-1) { output(0);
                          row[n-1] = n-2;
                          ls[n-2][pos[n-1][n-2]] = 0;
  ++(pos[n-1][n-2]);
}
  else ++i;
}
      }
      else ++(pos[i][row[i]]);
    }
  }

  if (n == 1) output(0);
  if (n == 2) { ls[1][1] = 1; output(0); }

  if (stored) output(1);
  if (mask & 1)
  printf("Number of order %d reduced latin squares: %.0f\n\n",
         n,counter);
}
```

7.2 Shell skripte za generiranje kvadrata

Izvođenjem ovih datoteka dobivamo predstavnike klasa izotopije i glavnih klasa kvadrata reda 6 i 7. Prije pokretanja u tekućem direktoriju treba imati izvršne verzije programa `evenexpand`, `oddcomplete`, `complete`, `isofilter` i `mainfilter`, kompajlirane s odgovarajućim parametrom N ($N = 6$ za `make6`, $N = 7$ za `make7`). U operativnom sustavu MS–DOS imena datoteka ograničena su na 8 znakova. Imena programa treba prilagoditi skraćenom obliku iz skripti. Međurezultati se spremaju u datoteke `r1`, `r2`,. . ., `f1`, `f2`,. . .. Konačni rezultat sprema se u `iso6` i `main6`, odnosno `iso7` i `main7`.

7.2.1 `make6.bat` (verzija za MS-DOS)

```
echo 1 > r1
echo 1 2 3 4 5 6 >> r1
evenexpa < r1 > f2
isofilte -l < f2 > r2
evenexpa < r2 > f3
isofilte -l < f3 > r3
oddcompl < r3 > f3_3
isofilte < f3_3 > r3_3
evenexpa < r3 > f4
complete < f4 > f4_
isofilte < f4_ > r4_
copy r3_3+r4_ iso6
mainfilt < iso6 > main6
```

7.2.2 `make6.scr` (verzija za UNIX)

```
echo 1 > r1
echo 1 2 3 4 5 6 >> r1
evenexpand < r1 > f2
isofilter -l < f2 > r2
evenexpand < r2 > f3
isofilter -l < f3 > r3
oddcomplete < r3 > f3_3
isofilter < f3_3 > r3_3
evenexpand < r3 > f4
complete < f4 > f4_
isofilter < f4_ > r4_
cat r3_3 r4_ > iso6
mainfilter < iso6 > main6
```

7.2.3 `make7.bat` (verzija za MS-DOS)

```
echo 1 > r1
echo 1 2 3 4 5 6 7 >> r1
evenexpa < r1 > f2
isofilte -l < f2 > r2
evenexpa < r2 > f3
isofilte -l < f3 > r3
evenexpa < r3 > f4
isofilte -l < f4 > r4
oddcompl < r4 > f4_3
isofilte < f4_3 > r4_3
evenexpa < r4 > f5
complete < f5 > f5_
isofilte < f5_ > r5_
copy r4_3+r5_ iso7
mainfilt < iso7 > main7
```

7.2.4 make7.scr (verzija za UNIX)

```
echo 1 > r1
echo 1 2 3 4 5 6 7 >> r1
evenexpand < r1 > f2
isofilter -l < f2 > r2
evenexpand < r2 > f3
isofilter -l < f3 > r3
evenexpand < r3 > f4
isofilter -l < f4 > r4
oddcomplete < r4 > f4_3
isofilter < f4_3 > r4_3
evenexpand < r4 > f5
complete < f5 > f5_
isofilter < f5_ > r5_
cat r4_3 r5_ > iso7
mainfilter < iso7 > main7
```

7.3 permutations.c

Zbirka potpograma za rad s permutacijama.

```
/*
     PERMUTATIONS.C
     A library of functions for permutations
     of an N-element set.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

/*
    Permutations are stored as character arrays of length N.
    Entries are different elements from the set 0, 1, ... , N-1.
    You should define the macro N in your main program (0 <= N <= 255).
*/

/*  PRINTPERM
    Prints permutation *in to the standard output. Options:
     bit      value      meaning

     0-1        0        No brackets
                1        Brackets       (     )
                2        Square brackets [     ]
                3        Braces         {     }

      2         0        Spaces between symbols
                1        Commas between symbols

      3         1        Symbols are 0, 1, ... , N-1
                0        Symbols are 1, 2, ... , N

      4         0        Newline
                1        No newline
```

```
*/
  void printperm(char *in, int opts)
  {  int i;
     switch (opts & 3)
     { case 0: break;
       case 1: printf("("); break;
       case 2: printf("["); break;
       case 3: printf("{");
     }
     for (i=0; i < N-1; ++i)
       if (opts & 4) printf("%d,",in[i] + ((opts & 8) != 8));
       else printf("%d ",in[i] + ((opts & 8) != 8));
     switch (opts & 3)
     { case 0: printf("%d",in[N-1] + ((opts & 8) != 8)); break;
       case 1: printf("%d)",in[N-1] + ((opts & 8) != 8)); break;
       case 2: printf("%d]",in[N-1] + ((opts & 8) != 8)); break;
       case 3: printf("%d}",in[N-1] + ((opts & 8) != 8));
     }
     if (!(opts & 16)) printf("\n");
   }


/*  LEXSUCC
    Stores in *out the permutation lexicographically succeeding *in.
    If *in is the last permutation (N-1,N-2,...,0), the result is
    (0,1,...,N-1) and the function returns 0. Otherwise it returns 1.
*/
  int lexsucc(char *in,char *out)
  { int i,j,found;
    char temp[N];

    i = N-1;
    found = 1;
    while ((i > 0) && (in[i] < in[i-1])) --i;
    --i;
    if (i == -1)
    { for (i=0; i < N; ++i) out[i] = i;
      return 0;
    }
    else
    { j = i;
      while (i > 0) temp[i] = in[--i];
      found = 0;
      for (i = N-1; i > j; --i)
      if (found) temp[j+N-i] = in[i];
      else { if (in[i] > in[j]) { temp[j] = in[i];
                                  temp[j+N-i] = in[j];
                                  found=1;
                                }
             else { temp[j+N-i] = in[i]; }
           }
      for(i=0; i < N; ++i) out[i] = temp[i];
      return 1;
    }
  }
```

```
/*  COMPOSE
    Composition of permutations
    *out = *in1 o *in2
*/
  void compose(char *in1,char *in2,char *out)
  { int i;
    char temp[N];

    for (i=0; i<N; ++i) temp[i] = in1[in2[i]];
    for (i=0; i<N; ++i) out[i] = temp[i];
  }


/* INVERSE
   Stores in *out the inverse of *in.
*/
  void inverse(char *in, char *out)
  { int i,j;
    char temp, temp1[N], temp2[N];

    for (i=0; i<N; ++i)
      { temp1[i] = in[i]; temp2[i] = i; }
    for (i=0; i<N-1; ++i)
      for (j=i+1; j<N; ++j)
        if (temp1[i] > temp1[j])
        { temp = temp1[i];
          temp1[i] = temp1[j];
          temp1[j] = temp;
          temp = temp2[i];
          temp2[i] = temp2[j];
          temp2[j] = temp;
        }
    for (i=0; i<N; ++i) out[i] = temp2[i];
  }


/*  SIGN
    Sign of permutation *in: 0 for even, 1 for odd.
*/
  int sign(char *in)
  { int i,j,sgn;
    sgn = 0;
    for (i=0; i<N-1; ++i)
      for(j=i+1; j<N; ++j)
        if (in[i]>in[j]) sgn = (!sgn);
    return sgn;
  }


/*  CMP
    Compare memory segments starting at *in1, *in2, of 'length' bytes.
    Returns 0 if they are equal, an integer >0 if *in1 > *in2
    (lexicographically), otherwise an integer <0.
```

```
*/
  int cmp(char *in1, char *in2, int length)
  { int i=0,out=0;
    while ((i < length) && (out == 0))
    { out = in1[i] - in2[i];
      ++i;
    }
    return out;
  }


/*  COPY
    Copy memory segments starting at *in of 'length' bytes to
    location *out.
*/
  void copy(char *in, char *out, int length)
  { int i;
    for (i=0; i<length; ++i) out[i] = in[i];
  }
```

7.4 evenexpand.c

Program uzima prirodni broj $l < N$ i niz latinskih pravokutnika tipa $l \times N$ s ulaza i proširuje ih na sve moguće načine parnim retkom do $(l+1) \times N$ pravokutnika.

```
/*
     EVENEXPAND.C
     A program that takes as its input an integer l<N
     and a sequence of latin l*N rectangles. It returns
     all latin rectangles obtained by expanding them
     with one even row.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define N 7

#include <stdio.h>
#include "permutations.c"

char ls[N][N];

void output(int l)
{ int i;
  for (i=0; i<=l; ++i) printperm(ls[i],0);
  printf("\n");
}


void main()
{ int l,i,j,ok,x;
```

```
  char pos[N];

  /* Read integer l */

  scanf("%i",&l);
  printf("\n%d\n\n",l+1);

  /* Main loop */

  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N; ++j)
    { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<l; ++i)
      for (j=0; j<N; ++j)
      { scanf("%i",&x); ls[i][j] = x-1; }

    for (i=0; i<N; ++i)
      { pos[i] = 0; ls[l][i] = 255; }

    i = 0;
    while (i >= 0)
    { if (pos[i] == N)
      { pos[i] = 0;
        --i;
        if (i >= 0) { ls[l][pos[i]] = 255; ++pos[i]; }
      }
      else
      { while ((ls[l][pos[i]] != -1) && (++pos[i] < N)) ;
        if (pos[i] < N)
        { ok = 1;
          for (j=0; (j<l) && ok; ++j) ok &= ( ls[j][pos[i]] != i );
          if (ok)
          { ls[l][pos[i]] = i;
            ++i;
            if (i == N)
    { if (!sign(ls[l])) output(l);
              --i;
              ls[l][pos[i]] = 255;
              ++pos[i];
            }
          }
          else ++pos[i];
        }
      }
    }
  }
}
```

7.5 oddcomplete.c

Program uzima prirodni broj $l < N$ i niz latinskih pravokutnika tipa $l \times N$ s ulaza i nadopunjuje ih na sve moguće načine neparnim recima do latinskog kvadrata.

```
/*
     ODDCOMPLETE.C
     A program that takes as its input an integer l<N
     and a sequence of latin l*N rectangles and returns
     all latin squares obtained by expanding them
     with rows of odd signature.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define N 7
#define COUNT 0 /* Only count completitions */

#include <stdio.h>
#include "permutations.c"

char ls[N][N];

void output()
{ int i;
  for (i=0; i<N; ++i) printperm(ls[i],0);
  printf("\n");
}


void main()
{ int l,i,j,ok,r,x;
  char pos[N][N];
  int symb[N];
  double counter=0.0;


  /* Read integer l */

  scanf("%i",&l);
#if COUNT
#else
  printf("\n%d\n\n",N);
#endif

  /* Main loop */

  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N; ++j)
    { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<l; ++i)
      for (j=0; j<N; ++j)
      { scanf("%i",&x); ls[i][j] = x-1; }
```

```
    for (r=1; r<N; ++r)
      for (i=0; i<N; ++i)
      { pos[r][i] = 0; ls[r][i] = 255; }
    for (r=1; r<N; ++r) symb[r] = 0;

    r = 1;
    while (r >= 1)
    { if (pos[r][symb[r]] == N)
      { pos[r][symb[r]] = 0;
--symb[r];
if (symb[r] >= 0) { ls[r][pos[r][symb[r]]] = 255; ++pos[r][symb[r]]; }
else { symb[r] = 0; --r; }
      }
      else
      { while ((ls[r][pos[r][symb[r]]] != -1) && (++pos[r][symb[r]] < N)) ;
if (pos[r][symb[r]] < N)
{ ok = 1;
  for (j=0; (j<r) && ok; ++j)
    ok &= ( ls[j][pos[r][symb[r]]] != symb[r] );
  if (ok)
  { ls[r][pos[r][symb[r]]] = symb[r];
    ++symb[r];
    if (symb[r] == N)
    { if (sign(ls[r])) ++r;
      else
      { --symb[r];
ls[r][pos[r][symb[r]]] = 255;
++pos[r][symb[r]];
      }
      if (r == N)
      {
#if COUNT
counter = counter + 1.0;
#else
output();
#endif
--r;
--symb[r];
ls[r][pos[r][symb[r]]] = 255;
++pos[r][symb[r]];
      }
    }
  }
  else ++pos[r][symb[r]];
}
      }
    }
  }

#if COUNT
  printf("\nTotal number of completitions: %.0f\n\n",counter);
#endif

}
```

7.6 complete.c

Program uzima prirodni broj $l < N$ i niz latinskih pravokutnika tipa $l \times N$ s ulaza i nadopunjuje ih na sve moguće načine proizvoljnim recima do latinskog kvadrata.

```
/*
     COMPLETE.C
     A program that takes as its input an integer l<N
     and a sequence of latin l*N rectangles. It returns
     all latin squares obtained by expanding them.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define N 7
#define COUNT 0 /* Only count completitions */

#include <stdio.h>
#include "permutations.c"

char ls[N][N];

void output()
{ int i;
  for (i=0; i<N; ++i) printperm(ls[i],0);
  printf("\n");
}


void main()
{ int l,i,j,ok,r,x;
  char pos[N][N];
  int symb[N];
  double counter=0.0;


  /* Read integer l */

  scanf("%i",&l);
#if COUNT
#else
  printf("\n%d\n\n",N);
#endif

  /* Main loop */

  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N; ++j)
    { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<l; ++i)
      for (j=0; j<N; ++j)
      { scanf("%i",&x); ls[i][j] = x-1; }
```

```
    for (r=1; r<N; ++r)
      for (i=0; i<N; ++i)
      { pos[r][i] = 0; ls[r][i] = 255; }
    for (r=1; r<N; ++r) symb[r] = 0;

    r = 1;
    while (r >= 1)
    { if (pos[r][symb[r]] == N)
      { pos[r][symb[r]] = 0;
--symb[r];
if (symb[r] >= 0) { ls[r][pos[r][symb[r]]] = 255; ++pos[r][symb[r]]; }
else { symb[r] = 0; --r; }
      }
      else
      { while ((ls[r][pos[r][symb[r]]] != -1) && (++pos[r][symb[r]] < N)) ;
if (pos[r][symb[r]] < N)
{ ok = 1;
  for (j=0; (j<r) && ok; ++j)
    ok &= ( ls[j][pos[r][symb[r]]] != symb[r] );
  if (ok)
  { ls[r][pos[r][symb[r]]] = symb[r];
    ++symb[r];
    if (symb[r] == N)
    { ++r;
      if (r == N)
      {
#if COUNT
counter = counter + 1.0;
#else
output();
#endif
--r;
--symb[r];
ls[r][pos[r][symb[r]]] = 255;
++pos[r][symb[r]];
      }
    }
  }
  else ++pos[r][symb[r]];
}
      }
    }
  }

#if COUNT
  printf("\nTotal number of completitions: %.0f\n\n",counter);
#endif

}
```

7.7 isofilter.c

Program uzima s ulaza prirodni broj $l \leq N$ i niz latinskih pravokutnika tipa $l \times N$. Vraća neizotopne među njima. Pri tome je bitno da svi pravokutnici na ulazu imaju normalizirani prvi redak, $[1\ 2 \cdots N]$. Inače je moguće da program da krivi rezultat.

```
/*
     ISOFILTER.C
     A program that takes as its input an integer l<=N
     and a sequence of latin l*N rectangles with the first
     row (1,2,..,N) and returns the non-isotopic among them.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define DEBUG 0 /* Print various information for debugging */

#define MEMBLOCK 1024 /* Allocate memory in units of MEMBLOCK bytes */

#define NULL 0 /* The null pointer */

/* Exactly one of the following macros must be 1 and all other 0 */

#define N4 0  /* N = 4 */
#define N5 0  /* N = 5 */
#define N6 0  /* N = 6 */
#define N7 1  /* N = 7 */
#define N8 0  /* N = 8 */


#if N8
#define N 8
#define M 28  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
         { case 4:  ++cpc[0]; break;\
   case 34: ++cpc[1]; break;\
   case 17: ++cpc[2]; break;\
   case 513: ++cpc[3]; break;\
   case 136: ++cpc[4]; break;\
   case 64: ++cpc[5]; break;\
   case 8192: ++cpc[6];\
 }
#define NTYPES 7
#endif

#if N7
#define N 7
#define M 21  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 10:  ++cpc[0]; break;\
   case 129: ++cpc[1]; break;\
   case 40: ++cpc[2]; break;\
   case 2048: ++cpc[3];\
 }
```

```
#define NTYPES 4
#endif

#if N6
#define N 6
#define M 15  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 3:  ++cpc[0]; break;\
   case 33: ++cpc[1]; break;\
   case 16: ++cpc[2]; break;\
   case 512: ++cpc[3];\
 }
#define NTYPES 4
#endif

#if N5
#define N 5
#define M 10  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 9:  ++cpc[0]; break;\
   case 128: ++cpc[1];\
 }
#define NTYPES 2
#endif

#if N4
#define N 4
#define M 6  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 2:  ++cpc[0]; break;\
   case 32: ++cpc[1];\
 }
#define NTYPES 2
#endif


static int cvalue[7]={1,8,32,128,512,2048,8192};

#include <stdio.h>
#include "permutations.c"

/* Test two reduced rectangles for isotopy */

int isotest(int l, char *A, char *B)
{ char symb[N], invsymb[N], perm[N];
  char pp[N][N];
  int i,j,r,isotopes;
  int used[N];

  isotopes = 0;
  r = 0;
  do
  {
    /* Make principle presentation of B with r as first row */
```

```
    inverse(B+r*N,perm);
    for (i=0; i<l; ++i) compose(B+i*N,perm,pp[i]);

    /* Check all symbol permutations */

    for (i=0; i<N; ++i) symb[i] = i;
    do
    { for (i=0; i<N; ++i) used[i] = 0;
      inverse(symb,invsymb);
      i=0;
      do
      {
compose(invsymb,pp[i],perm);
compose(perm,symb,perm);
j = 0;
while ((j<l) && used[j]) ++j;
while ((j<l) && (cmp(perm,A+j*N,N) != 0))
{ ++j;
  while ((j<l) && used[j]) ++j;
}
if (j<l) { used[j] = 1; ++i; }
      } while ((i < l) && (j < l));
      if (i == l) isotopes = 1;
    } while ((!isotopes) && lexsucc(symb,symb));
    if (!isotopes) ++r;
  } while ((r<l) && (!isotopes));

  return isotopes;
}

void main(int argc,char *argv[])
{ int i,j,r,x;
  char ls[N][N];
  char perm[N];
  int l,m;

  int ctype;
  int start,now,length;
  int used[N];
  char inv[N][N], cp[M][N];
  char cpc[NTYPES], dcv[N][N-1];
  int sort[N],tmp;

  typedef struct node3 {
      char ls[N][N];
      struct node3 *down;
      } N3;
  typedef struct node2 {
      char dcv[N][N-1];
      struct node3 *right;
      struct node2 *down;
      } N2;
  typedef struct node1 {
      char cpc[NTYPES];
      struct node2 *right;
```

```
      struct node1 *down;
      } N1;
  N1 *n1_start, *n1_now, *n1_prev;
  N2 *n2_now, *n2_prev;
  N3 *n3_now, *n3_prev;
  char *free, *end, *malloc();
  int blockcount=1;
  int found1,found2,found3;

  double count=0.0;

  int mask = 0;  /* An integer masking the output format */
  /* Meaning of the bits and options to put them on/off are as following:
     1  Print the invariants as well as the resulting latin rectangles.
-i for yes, -I for no (the default is no)
     2  Before the output the number of rows l is printed. -l for yes,
-L for no (the default is no)
     4  Count the rectangles as they are processed and report their
number. -c for yes, -C for no (the default is no)
     8  Report the memory allocation. -m for yes, -M for no (the
        default is no)
  */

  /* Command line arguments */

  if (argc > 1)
  { for(i=1; i<argc; ++i)
    { j=0;
      if (argv[i][0] == '-') j=1;
      if (argv[i][j] == 'i') mask |= 1;
      if (argv[i][j] == 'I') mask &= ~1;
      if (argv[i][j] == 'l') mask |= 2;
      if (argv[i][j] == 'L') mask &= ~2;
      if (argv[i][j] == 'c') mask |= 4;
      if (argv[i][j] == 'C') mask &= ~4;
      if (argv[i][j] == 'm') mask |= 8;
      if (argv[i][j] == 'M') mask &= ~8;
      if ((argv[i][j] == 'h') || (argv[i][j] == 'H') ||
  (argv[i][j] == '?'))
      { printf("Usage: isofilter [-iIlLcCmM] sequence of latin rectangles\n");
printf("Options:\n");
printf("  Print invariants: -i yes, -I no (default no)\n");
printf("  Print number of rows: -l yes, -L no (default no)\n");
printf("  Count incoming rectangles: -c yes, -C no (default no)\n");
printf("  Report memory allocation: -m yes, -M no (default no)\n");
        exit(1);
      }
    }
  }

  /* Allocate first block of memory and set up the pointers */

  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
```

```
    exit(0);
  }
  ++blockcount;
  end = free + MEMBLOCK-1;
  n1_start = (N1 *) free;
  n2_now = NULL; n2_prev = NULL;
  n3_now = NULL; n3_prev = NULL;
  n1_now = NULL; n1_prev = NULL;


  /* Read integer l */

  scanf("%i",&l);


  /*************/
  /* Main loop */
  /*************/

  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N; ++j)
      { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<l; ++i)
      for (j=0; j<N; ++j)
       { scanf("%i",&x); ls[i][j] = x-1; }

    count = count + 1.0;
    if ((mask & 4) == 4) printf("%.0f\n",count);

#if DEBUG
    /* Print latin rectangle */

    for (i=0; i<l; ++i) printperm(ls[i],0);
#endif

    /* Make complete presentation */

    for (i=0; i<l; ++i) inverse(ls[i],inv[i]);
    m = 0;
    for (i=1; i<l; ++i)
      for(j=0; j<i; ++j) compose(inv[j],ls[i],cp[m++]);
    /*  m = l(l-1)/2  */

    /* Compute sign2 and sign3 invariant */

    for (i=0; i<N-1; ++i) cpc[i]= 0;
    for (i=0; i<N; ++i)
      for (j=0; j<N-1; ++j) dcv[i][j] = 0;

    for (r=0; r<m; ++r)
    { for (i=0; i<N; ++i) used[i] = 0;
      ctype = 0;
      j = 0;
      while (j<N)
```

```
      { while ((j<N) && (used[j])) ++j;
if (j<N)
{ start = j;
  length = -1;
  used[j] = 1;
  now = cp[r][start];
  while (now != start)
  { ++length;
    used[now] = 1;
    now = cp[r][now];
  }
  ctype += cvalue[length];
  ++dcv[cp[r][start]][length];
  now = cp[r][start];
  while (now != start)
  { ++dcv[cp[r][now]][length];
    now = cp[r][now];
  }
}
      }
      INCRCPC
    }

    /* Sort digit cycle vectors */

    for (i=0; i<N; ++i) sort[i] = i;

    for (i=0; i<N-1; ++i)
      for (j=i+1; j<N; ++j)
if (cmp(dcv[sort[i]],dcv[sort[j]],N-1) > 0)
{ tmp = sort[i];
  sort[i] = sort[j];
  sort[j] = tmp;
}

#if DEBUG
    /* Print cycle pattern count */

    printf("  cpc = ");
    for (i=0; i<NTYPES; ++i) printf("%d ",cpc[i]);
    printf("\n");

    /* Print digit cycle vectors */

    for (i=0; i<N; ++i)
    { printf("  dcv(%d) = ",sort[i]+1);
      for (j=0; j<N-1; ++j) printf("%d ",dcv[sort[i]][j]);
      printf("\n");
    }
#endif

    /* Search for isotopes */

    /* First rectangle */
```

```
    if (n1_prev == 0)
    { n1_now = (N1 *) free;
      n1_prev = (N1 *) free;
      free = free + sizeof(N1);
      n2_now = (N2 *) free;
      n2_prev = (N2 *) free;
      free = free + sizeof(N2);
      n3_now = (N3 *) free;
      n3_prev = (N3 *) free;
      free = free + sizeof(N3);
      if (free > end) { printf("\nMEMBLOCK must be bigger!\n"); exit(0); }
      else
      { copy(cpc,n1_now->cpc,NTYPES);
n1_now->down = NULL;
n1_now->right = n2_now;
for (i=0; i<N; ++i)
  copy(dcv[sort[i]],n2_now->dcv[i],N-1);
n2_now->down = NULL;
n2_now->right = n3_now;
copy((char *)ls,(char *)n3_now->ls,N*N);
n3_now->down = NULL;
      }

#if DEBUG
      printf("First rectangle memorized!\n\n");
#endif

    }

    /* Not the first rectangle */

    else
    {
#if DEBUG
      printf("I'm at the not-first rectangle search part.\n");
#endif
      found1 = 0;
      n1_now = n1_start;
      while ((n1_now != NULL) && !found1)
      { if (cmp(cpc,n1_now->cpc,NTYPES)==0)
{ found1 = 1;
  found2 = 0;
#if DEBUG
          printf("CPC part completed. Preforming DCV part.\n");
#endif
  n2_now = n1_now->right;
  while ((n2_now != NULL) && !found2)
  { found2 = 1;
    i = 0;
    while ((i<N) &&
   (found2 = (cmp(dcv[sort[i]],n2_now->dcv[i],N-1) == 0)))
  ++i;
    if (found2)
    { found3 = 0;
      n3_now = n2_now->right;
```

```
      while ((n3_now != NULL) && !found3)
      {
#if DEBUG
                printf("Calling isotest...");
#endif
                found3 = isotest(1,(char *)ls,(char *)n3_now->ls);
#if DEBUG
                printf("OK.\n");
#endif
n3_prev = n3_now;
n3_now = n3_now->down;
      }
    }
    n2_prev = n2_now;
    n2_now = n2_now->down;
  }
}
n1_prev = n1_now;
n1_now = n1_now->down;
      }

#if DEBUG
      printf("found1=%d, found2=%d, found3 =%d\n\n",found1,found2,found3);
#endif

      /* Memorize cycle pattern count, if necessary */

      if (!found1)
      {

/* Allocate next block of memory */

if ((free+sizeof(N1)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
          free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
  end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memorizing new cpc.\n");
#endif

n1_now = (N1 *) free;
n1_prev->down = n1_now;
free = free + sizeof(N1);
```

```
copy(cpc,n1_now->cpc,NTYPES);
n1_now->down = NULL;

found2 = 0;
      }

      /* Memorize digit cycle vector, if necessary */

      if (!found2)
      {

/* Allocate next block of memory */

        if ((free+sizeof(N2)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
          end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memoriziong new dcv.\n");
#endif

n2_now = (N2 *) free;
if (found1) n2_prev->down = n2_now;
else n1_now->right = n2_now;
free = free + sizeof(N2);

for (i=0; i<N; ++i)
  copy(dcv[sort[i]],n2_now->dcv[i],N-1);
n2_now->down = NULL;

found3 = 0;
      }

      /* Memorize latin rectangle, if necessary */

      if (!found3)
      {

/* Allocate next block of memory */

if ((free+sizeof(N3)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
```

```
  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
  end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memorizing new ls.\n");
#endif

n3_now = (N3 *) free;
if (found2) n3_prev->down = n3_now;
else n2_now->right = n3_now;
free = free + sizeof(N3);

copy((char *)ls,(char *)n3_now->ls,N*N);
n3_now->down = NULL;
      }

    }

  } /* End of main loop */


  /**********/
  /* Output */
  /**********/

#if DEBUG
  printf("\nOutput:\n");
#endif

  printf("\n");
  if ((mask & 2) == 2) printf("%d\n\n",l);

  n1_now = n1_start;
  while (n1_now != NULL)
  {
    /* Print cycle pattern count */

    if (mask & 1)
    { printf("cpc = ");
      for (i=0; i<NTYPES; ++i) printf("%d ",n1_now->cpc[i]);
      printf("\n\n");
    }

    n2_now = n1_now->right;
    while (n2_now != NULL)
    {
      /* Print digit cycle vectors */
```

```
      if (mask & 1)
      { for (i=0; i<N; ++i)
{ printf("  dcv(%d) = ",i+1);
  for (j=0; j<N-1; ++j) printf("%d ",n2_now->dcv[i][j]);
  printf("\n");
}
printf("\n");
      }

      n3_now = n2_now->right;
      while (n3_now != NULL)
      {
/* Print latin rectangle */

for (i=0; i<l; ++i)
{ if (mask & 1) printf("    ");
  printperm(n3_now->ls[i],0);
}
printf("\n");
n3_now = n3_now->down;
      }
      n2_now = n2_now->down;
    }
    n1_now = n1_now->down;
  }

}
```

7.8 mainfilter.c

Program uzima s ulaza niz latinskih kvadrata reda N i vraća one koji ne pripadaju istim glavnim klasama.

```
/*
     MAINFILTER.C
     A program that takes a sequence of latin squares
     of order N from the standard input and returns the main
     class representatives among them.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define DEBUG 0 /* Print various information for debugging */

#define MEMBLOCK 1024 /* Allocate memory in units of MEMBLOCK bytes */

#define NULL 0 /* The null pointer */

/* Exactly one of the following macros must be 1 and all other 0 */

#define N5 0  /* N = 5 */
#define N6 0  /* N = 6 */
```

```
#define N7 1  /* N = 7 */
#define N8 0  /* N = 8 */


#if N8
#define N 8
#define M 28  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 4:  ++cpc[0]; break;\
   case 34: ++cpc[1]; break;\
   case 17: ++cpc[2]; break;\
   case 513: ++cpc[3]; break;\
   case 136: ++cpc[4]; break;\
   case 64: ++cpc[5]; break;\
   case 8192: ++cpc[6];\
 }
#define NTYPES 7
#endif

#if N7
#define N 7
#define M 21  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 10:  ++cpc[0]; break;\
   case 129: ++cpc[1]; break;\
   case 40: ++cpc[2]; break;\
   case 2048: ++cpc[3];\
 }
#define NTYPES 4
#endif

#if N6
#define N 6
#define M 15  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 3:  ++cpc[0]; break;\
   case 33: ++cpc[1]; break;\
   case 16: ++cpc[2]; break;\
   case 512: ++cpc[3];\
 }
#define NTYPES 4
#endif

#if N5
#define N 5
#define M 10  /*  M = N(N-1)/2  */
#define INCRCPC switch (ctype)\
 { case 9:  ++cpc[0]; break;\
   case 128: ++cpc[1];\
 }
#define NTYPES 2
#endif


static int cvalue[7]={1,8,32,128,512,2048,8192};
```

```
#include <stdio.h>
#include "permutations.c"

/* Conjugate operations */

void rowinv(char *ls)
{ int i;
  for (i=0; i<N; ++i) inverse(ls+i*N,ls+i*N);
}

void transpose(char *ls)
{ int i,j;
  char trn[N][N];

  for (i=0; i<N; ++i)
    for (j=0; j<N; ++j) trn[i][j] = *(ls+i+j*N);
  for (i=0; i<N; ++i)
    for (j=0; j<N; ++j) *(ls+j+i*N) = trn[i][j];
}


/* Test two reduced rectangles for isotopy */

int isotest(int l, char *A, char *B)
{ char symb[N], invsymb[N], perm[N];
  char ppB[N][N];
  int i,j,r,isotopes;
  int used[N];

  isotopes = 0;
  r = 0;
  do
  {
    /* Make principle presentation of B with r as first row */

    inverse(B+r*N,perm);
    for (i=0; i<l; ++i) compose(B+i*N,perm,ppB[i]);

    /* Check all symbol permutations */

    for (i=0; i<N; ++i) symb[i] = i;
    do
    { for (i=0; i<N; ++i) used[i] = 0;
      inverse(symb,invsymb);
      i=0;
      do
      {
compose(invsymb,ppB[i],perm);
compose(perm,symb,perm);
j = 0;
while ((j<l) && used[j]) ++j;
while ((j<l) && (cmp(perm,A+j*N,N) != 0))
{ ++j;
  while ((j<l) && used[j]) ++j;
```

```
}
if (j<l) { used[j] = 1; ++i; }
      } while ((i < l) && (j < l));
      if (i == l) isotopes = 1;
    } while ((!isotopes) && lexsucc(symb,symb));
    if (!isotopes) ++r;
  } while ((r<l) && (!isotopes));

  return isotopes;
}


void main(int argc,char *argv[])
{ int c,i,j,r,x;
  char ls[N][N];
  char perm[N];
  int l,m;

  int ctype;
  int start,now,length;
  int used[N];
  char inv[N][N], cp[M][N];
  char cpc[NTYPES], dcv[N][N-1];
  int sort[N],tmp;

  typedef struct node3 {
      char ls[N][N];
      struct node3 *down;
      } N3;
  typedef struct node2 {
      char dcv[N][N-1];
      struct node3 *right;
      struct node2 *down;
      } N2;
  typedef struct node1 {
      char cpc[NTYPES];
      struct node2 *right;
      struct node1 *down;
      } N1;
  N1 *n1_start, *n1_now, *n1_prev;
  N2 *n2_now, *n2_prev;
  N3 *n3_now, *n3_prev;
  char *free, *end, *malloc();
  int blockcount=1;
  int found1,found2,found3;

  double count=0.0;

  int mask = 0;  /* An integer masking the output format */
  /* Meaning of the bits and options to put them on/off are as following:
     1  Print the invariants as well as the resulting latin rectangles.
-i for yes, -I for no (the default is no)
     2  Before the output the number of rows l is printed. -l for yes,
-L for no (the default is no)
     4  Count the rectangles as they are processed and report their
```

```
number. -c for yes, -C for no (the default is no)
     8  Report memory allocation. -m for yes, -M for no (the
        default is no)
  */

  /* Command line arguments */

  if (argc > 1)
  { for(i=1; i<argc; ++i)
    { j=0;
      if (argv[i][0] == '-') j=1;
      if (argv[i][j] == 'i') mask |= 1;
      if (argv[i][j] == 'I') mask &= ~1;
      if (argv[i][j] == 'l') mask |= 2;
      if (argv[i][j] == 'L') mask &= ~2;
      if (argv[i][j] == 'c') mask |= 4;
      if (argv[i][j] == 'C') mask &= ~4;
      if (argv[i][j] == 'm') mask |= 8;
      if (argv[i][j] == 'M') mask &= ~8;
      if ((argv[i][j] == 'h') || (argv[i][j] == 'H') ||
  (argv[i][j] == '?'))
      { printf("Usage: mainfilter [-iIlLcCmM] sequence of latin rectangles\n");
printf("Options:\n");
printf("  Print invariants: -i yes, -I no (default no)\n");
printf("  Print number of rows: -l yes, -L no (default no)\n");
printf("  Count incoming rectangles: -c yes, -C no (default no)\n");
printf("  Report memory allocation: -m yes, -M no (default no)\n");
        exit(1);
      }
    }
  }

  /* Allocate first block of memory and set up the pointers */

  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  end = free + MEMBLOCK-1;
  n1_start = (N1 *) free;
  n2_now = NULL; n2_prev = NULL;
  n3_now = NULL; n3_prev = NULL;
  n1_now = NULL; n1_prev = NULL;


  /* Set integer l (number of rows) to N */

  l = N;


  /*************/
  /* Main loop */
  /*************/
```

```
  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N; ++j)
      { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<l; ++i)
      for (j=0; j<N; ++j)
       { scanf("%i",&x); ls[i][j] = x-1; }

    count = count + 1.0;
    if ((mask & 4) == 4) printf("%.0f\n",count);

#if DEBUG
    /* Print latin rectangle */

    for (i=0; i<l; ++i) printperm(ls[i],0);
#endif

    found3 = 0;

  for (c=0; (c<6) && !found3; ++c)
  {

    /* Apply conjugate operation */

    if (c % 2) rowinv((char *)ls);
    else transpose((char *)ls);
#if DEBUG
    if (c % 2) printf("Applying row inverse.\n");
    else printf("Applying transposition.\n");
#endif

    /* Make principle presentation */

    inverse(ls[0],perm);
    for(i=0; i<l; ++i) compose(ls[i],perm,ls[i]);

    /* Make complete presentation */

    for (i=0; i<l; ++i) inverse(ls[i],inv[i]);
    m = 0;
    for (i=1; i<l; ++i)
      for(j=0; j<i; ++j) compose(inv[j],ls[i],cp[m++]);
    /*  m = l(l-1)/2  */

    /* Compute sign2 and sign3 invariant */

    for (i=0; i<N-1; ++i) cpc[i]= 0;
    for (i=0; i<N; ++i)
      for (j=0; j<N-1; ++j) dcv[i][j] = 0;

    for (r=0; r<m; ++r)
    { for (i=0; i<N; ++i) used[i] = 0;
      ctype = 0;
      j = 0;
```

```
      while (j<N)
      { while ((j<N) && (used[j])) ++j;
if (j<N)
{ start = j;
  length = -1;
  used[j] = 1;
  now = cp[r][start];
  while (now != start)
  { ++length;
    used[now] = 1;
    now = cp[r][now];
  }
  ctype += cvalue[length];
  ++dcv[cp[r][start]][length];
  now = cp[r][start];
  while (now != start)
  { ++dcv[cp[r][now]][length];
    now = cp[r][now];
  }
}
      }
      INCRCPC
    }

    /* Sort digit cycle vectors */

    for (i=0; i<N; ++i) sort[i] = i;

    for (i=0; i<N-1; ++i)
      for (j=i+1; j<N; ++j)
if (cmp(dcv[sort[i]],dcv[sort[j]],N-1) > 0)
{ tmp = sort[i];
  sort[i] = sort[j];
  sort[j] = tmp;
}

#if DEBUG
    /* Print cycle pattern count */

    printf("  cpc = ");
    for (i=0; i<NTYPES; ++i) printf("%d ",cpc[i]);
    printf("\n");

    /* Print digit cycle vectors */

    for (i=0; i<N; ++i)
    { printf("  dcv(%d) = ",sort[i]+1);
      for (j=0; j<N-1; ++j) printf("%d ",dcv[sort[i]][j]);
      printf("\n");
    }
#endif

    /* Search for isotopes */

    /* First rectangle */
```

```
    if (n1_prev == 0)
    { n1_now = (N1 *) free;
      n1_prev = (N1 *) free;
      free = free + sizeof(N1);
      n2_now = (N2 *) free;
      n2_prev = (N2 *) free;
      free = free + sizeof(N2);
      n3_now = (N3 *) free;
      n3_prev = (N3 *) free;
      free = free + sizeof(N3);
      if (free > end) { printf("\nMEMBLOCK must be bigger!\n"); exit(0); }
      else
      { copy(cpc,n1_now->cpc,NTYPES);
n1_now->down = NULL;
n1_now->right = n2_now;
for (i=0; i<N; ++i)
  copy(dcv[sort[i]],n2_now->dcv[i],N-1);
n2_now->down = NULL;
n2_now->right = n3_now;
copy((char *)ls,(char *)n3_now->ls,N*N);
n3_now->down = NULL;
      }

      found1 = 1;
      found2 = 1;
      found3 = 1;

#if DEBUG
      printf("First rectangle memorized!\n\n");
#endif

    }

    /* Not the first rectangle */

    else
    {
      found1 = 0;
      n1_now = n1_start;
      while ((n1_now != NULL) && !found1)
      { if (cmp(cpc,n1_now->cpc,NTYPES)==0)
{ found1 = 1;
  found2 = 0;
  n2_now = n1_now->right;
  while ((n2_now != NULL) && !found2)
  { found2 = 1;
    i = 0;
    while ((i<N) &&
   (found2 = (cmp(dcv[sort[i]],n2_now->dcv[i],N-1) == 0)))
  ++i;
    if (found2)
    { found3 = 0;
      n3_now = n2_now->right;
      while ((n3_now != NULL) && !found3)
```

```
      { found3 = isotest(l,(char *)ls,(char *)n3_now->ls);
n3_prev = n3_now;
n3_now = n3_now->down;
      }
    }
    n2_prev = n2_now;
    n2_now = n2_now->down;
  }
}
n1_prev = n1_now;
n1_now = n1_now->down;
      }

#if DEBUG
      printf("found1=%d, found2=%d, found3 =%d\n\n",found1,found2,found3);
#endif

    }
  }

      /* Memorize cycle pattern count, if necessary */

      if (!found1)
      {

/* Allocate next block of memory */

if ((free+sizeof(N1)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
          free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
  end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memorizing new cpc.\n");
#endif

n1_now = (N1 *) free;
n1_prev->down = n1_now;
free = free + sizeof(N1);

copy(cpc,n1_now->cpc,NTYPES);
n1_now->down = NULL;

found2 = 0;
      }
```

```
      /* Memorize digit cycle vector, if necessary */

      if (!found2)
      {

/* Allocate next block of memory */

        if ((free+sizeof(N2)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
          end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memoriziong new dcv.\n");
#endif

n2_now = (N2 *) free;
if (found1) n2_prev->down = n2_now;
else n1_now->right = n2_now;
free = free + sizeof(N2);

for (i=0; i<N; ++i)
  copy(dcv[sort[i]],n2_now->dcv[i],N-1);
n2_now->down = NULL;

found3 = 0;
      }

      /* Memorize latin rectangle, if necessary */

      if (!found3)
      {

/* Allocate next block of memory */

if ((free+sizeof(N3)) > end)
{
  if ((mask & 8) == 8)
    printf("Allocating %d. block of memory....",blockcount);
  free = malloc(MEMBLOCK);
  if (free == NULL)
  { printf("\nOut of memory on %d. malloc() call!\n",blockcount);
    exit(0);
  }
```

```
  ++blockcount;
  if ((mask & 8) == 8)
    printf("OK.\n");
  end = free + MEMBLOCK-1;
}

#if DEBUG
printf("Memorizing new ls.\n");
#endif

n3_now = (N3 *) free;
if (found2) n3_prev->down = n3_now;
else n2_now->right = n3_now;
free = free + sizeof(N3);

copy((char *)ls,(char *)n3_now->ls,N*N);
n3_now->down = NULL;
      }



  } /* End of main loop */


  /**********/
  /* Output */
  /**********/

#if DEBUG
  printf("\nOutput:\n");
#endif

  printf("\n");
  if ((mask & 2) == 2) printf("%d\n\n",l);

  n1_now = n1_start;
  while (n1_now != NULL)
  {
    /* Print cycle pattern count */

    if (mask & 1)
    { printf("cpc = ");
      for (i=0; i<NTYPES; ++i) printf("%d ",n1_now->cpc[i]);
      printf("\n\n");
    }

    n2_now = n1_now->right;
    while (n2_now != NULL)
    {
      /* Print digit cycle vectors */

      if (mask & 1)
      { for (i=0; i<N; ++i)
{ printf("  dcv(%d) = ",i+1);
  for (j=0; j<N-1; ++j) printf("%d ",n2_now->dcv[i][j]);
```

```
  printf("\n");
}
printf("\n");
      }

      n3_now = n2_now->right;
      while (n3_now != NULL)
      {
/* Print latin rectangle */

for (i=0; i<l; ++i)
{ if (mask & 1) printf("    ");
  printperm(n3_now->ls[i],0);
}
printf("\n");
n3_now = n3_now->down;
      }
      n2_now = n2_now->down;
    }
    n1_now = n1_now->down;
  }

}
```

7.9 autcount.c

Program uzima s ulaza broj n i niz latinskih kvadrata reda n, računa i ispisuje red grupe autotopija svakog od njih. Na kraju ispisuje distribuciju kvadrata prema redovima njihovih grupa.

```
/*
     AUTCOUNT.C
     A program that takes as its input an integer 'ord' (order)
     and a sequence of latin squares and computes their autotopy
     group orders.
     The program uses nauty by Brendan D. McKay. It needs to
     be linked with nauty.c and nautil.c.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#include <stdio.h>

#define MAXORD 20
#define MAXN 460  /* 3*MAXORD + MAXORD^2 */
#include "nauty.h"

void main()
{
  graph g[MAXN*MAXM];
  nvector lab[MAXN],ptn[MAXN],orbits[MAXN];
  static DEFAULTOPTIONS(options);
  statsblk(stats);
```

```
  setword workspace[50*MAXM];

  int ord,n,m,v,i,j,x;
  set *gv, *gv2;
  char ls[MAXORD][MAXORD];
  int grporders[50][2];

  for (i=0; i<50; ++i)
  { grporders[i][0] = 0;
    grporders[i][1] = 0; }

  /* Options for nauty */

  options.writemarkers = FALSE;
  options.writeautoms = FALSE;
  options.defaultptn = FALSE;
  options.getcanon = FALSE;
  options.tc_level = 4;

  /* Read order and define n,m (for nauty) */

  scanf("%i",&ord);
  if ((ord < 1) || (ord > MAXORD))
  { printf("Order must be in the range 1..%d\n",MAXORD);
    exit(1);
  }

  n = 3*ord+ord*ord;
  m = (n + WORDSIZE - 1) / WORDSIZE;

  /* Main loop */

  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<ord; ++j)
      { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<ord; ++i)
      for (j=0; j<ord; ++j)
{ scanf("%i",&x); ls[i][j] = x-1; }

    /* Clear graph */

    for (v=0; v<n; ++v)
    { gv = GRAPHROW(g,v,m);
      EMPTYSET(gv,m);
    }

    /* Join rows */

    for (v=0; v<ord; ++v)
    {  gv = GRAPHROW(g,v,m);
       for (i=0; i<ord; ++i) ADDELEMENT(gv,3*ord+v*ord+i);
       for (i=0; i<ord; ++i)
       { gv = GRAPHROW(g,3*ord+v*ord+i,m);
 ADDELEMENT(gv,v);
```

```
      }
   }

   /* Join columns */

   for (v=0; v<ord; ++v)
   {  gv = GRAPHROW(g,v+ord,m);
      for (i=0; i<ord; ++i) ADDELEMENT(gv,3*ord+v+i*ord);
      for (i=0; i<ord; ++i)
      { gv = GRAPHROW(g,3*ord+v+i*ord,m);
ADDELEMENT(gv,v+ord);
      }
   }

   /* Join symbols */

   for (v=0; v<ord; ++v)
   {  gv = GRAPHROW(g,v+2*ord,m);
      for (i=0; i<ord; ++i)
      { j=0;
while (ls[i][j] != v) ++j;
ADDELEMENT(gv,3*ord+i*ord+j);
gv2 = GRAPHROW(g,3*ord+i*ord+j,m);
ADDELEMENT(gv2,v+2*ord);
      }
   }

   /* Define initial colouring */

   for (i=0; i<n; ++i) { lab[i] = i; ptn[i] = 1; }
   ptn[ord-1] = 0;
   ptn[2*ord-1] = 0;
   ptn[3*ord-1] = 0;
   ptn[n-1] = 0;

   /* Call nauty */

   nauty(g,lab,ptn,NILSET,orbits,&options,&stats,
 workspace,50*MAXM,m,n,NILGRAPH);

   /* Print order of autotopy group and update grporders */

   printf("%.0f\n",stats.grpsize1);
   x = (int) stats.grpsize1;
   i = 0;
   while ((grporders[i][0] != 0) && (grporders[i][0] != x)) ++i;
   if (grporders[i][0] == 0)
   { grporders[i][0] = x; grporders[i][1] = 1; }
   else ++grporders[i][1];

 }

 /* Print statistics */

 printf("\nDistribution of squares with respect to autotopy group order:");
```

```
  printf("\n\n  |Aut| Number of squares\n\n");
  i = 0;
  while (grporders[i][0] != 0)
  { printf("%5i %7i\n",grporders[i][0],grporders[i][1]);
    ++i;
  }
}
```

7.10 incmat.c

Program uzima niz latinskih kvadrata reda $N-1$ s ulaza. Svakog prevodi u prvih $3N$ redaka i $2N+1$ stupaca parcijalne incidencijske matrice. Zatim je pokušava dopuniti do incidencijske matrice projektivne ravnine reda N. Svaki puta kada mu to uspije ispisuje rezultat.

```
/*
     INCMAT.C
     This program takes as its input a sequence of latin
     squares of order N-1. It translates each to the first
     2N+1 rows and 3N columns of a (N^2+N+1)x(N^2+N+1) matrix.
     The program attempts to complete this matrix to a projective
     plane incidence matrix of order N.
     Vedran Krcadinac (krcko@student.math.hr)
     Department of Mathematics, U-Zagreb, Croatia
*/

#define N 8

#include <stdlib.h>
#include <stdio.h>

static char mat[N*N+N+1][N+3];  /* The incidence matrix */

static bit[8] = {1,2,4,8,16,32,64,128};
static bitsum[256]; /* bitsum[i] = number of set bits in i */
static int skip[(N-1)*(N-1)]; /* Zero blocks on the diagonal */


/***********************
 * The output function *
 ***********************/

void output(int format)
/* Formats: 0 = condensed
    1 = block
    2 = Mathematica */
{ int i,j,k;

  printf("\n");
  if (format==2) printf("{");
  for (i=0; i<3; ++i)
  { if (format==2)
    { if (i==0) printf("{");
```

```
      else printf(" {");
    }
    for(j=0; j<N*N+N+1; ++j)
    { if ((mat[j][0] & bit[i]) == 0) printf("0");
      else printf("1");
      if ((format==1) && (((j-2)%(N-1))==0)) printf(" ");
      if (format==2)
      { if (j<N*N+N) printf(",");
else printf("},");
      }
    }
    printf("\n");
  }

  if (format==1) printf("\n");

  for (i=1; i<N+3; ++i)
  { for(j=0; j<N-1; ++j)
    { if (format==2) printf(" {");
      for(k=0; k<N*N+N+1; ++k)
      { if ((mat[k][i] & bit[j]) == 0) printf("0");
else printf("1");
if ((format==1) && (((k-2)%(N-1))==0)) printf(" ");
if (format==2)
{ if (k<N*N+N) printf(",");
  else
  { if ((i==N+2) && (j==N-2)) printf("}}");
    else printf("},");
  }
        }
      }
      printf("\n");
    }
    if (format==1) printf("\n");
  }
}


/* Inner product of rows c1, c2 */

int dot(char *c1, char *c2)
{ int i,sum;

  sum = 0;
  for (i=0; i<N+3; ++i)
  { sum += bitsum[*c1 & *c2];
    ++c1; ++c2;
  }
  return sum;
}

/* Compatibility test for row n */

int test(int n)
{ int ok,j;
```

```
  char *col;

  col = mat[3*N+n];
  ok = 1;
  for (j=3*N+n-1; ok && (j>2); --j) ok = dot(col,mat[j]) == 1;
  return ok;
}

/* Initialize column n */

void setcol(int n)
{ int i;
  mat[3*N+n][3] = 1;
  for(i=4; i<N+3; ++i) mat[3*N+n][i] = 1;
  mat[3*N+n][skip[n]] = 0;
}


void main()
{ int i,j,k,x,ok;
  int full;
  char endbit;
  char ls[N-1][N-1];  /* The latin square */
  int col,backtrack,depth,count;
  double nodes[(N-1)*(N-1)];

  /* Set up array bitsum & other stuff */

  for (i=0; i<256; ++i)
  { bitsum[i] = 0;
    j = i;
    while (j != 0)
    { if (j & 1) ++bitsum[i];
      j >>= 1;
    }
  }

  for (i=0; i<(N-1); ++i)
    for (j=0; j<(N-1); ++j) skip[i*(N-1)+j] = 4+i;


  count = 0;
  full = 0;
  for (i=0; i<N-1; ++i) full += bit[i];
  endbit = bit[N-1];

  depth = (N-1)*(N-1);  /* Depth of the backtrack tree */


  /*************/
  /* Main loop */
  /*************/

  /* Read a latin square */
```

```
  while (scanf("%i",&x) == 1)
  { ls[0][0] = x-1;
    for (j=1; j<N-1; ++j)
      { scanf("%i",&x); ls[0][j] = x-1; }
    for (i=1; i<N-1; ++i)
      for (j=0; j<N-1; ++j)
{ scanf("%i",&x); ls[i][j] = x-1; }

    ++count;
    printf("\nProcessing latin square number %i.\n",count);

    /*******************/
    /* Make the matrix */
    /*******************/

    for (i=0; i < N+3; ++i)
      for (j=0; j < N*N+N+1; ++j) mat[j][i] = 0;

    /* First three rows & columns */

    mat[0][0] = 6;
    mat[1][0] = 3;
    mat[2][0] = 5;


    mat[0][1] = full;
    mat[1][2] = full;
    mat[2][3] = full;

    for (i=0; i<N-1; ++i)
    { mat[3+i][0] = 1;
      mat[2+N+i][0] = 2;
      mat[1+2*N+i][0] = 4;
    }

    /* Identity submatrices */

    for (i=0; i<N-1; ++i)
    { mat[3+i][1] = bit[i];
      mat[1+2*N+i][2] = bit[i];
      mat[2+N+i][3] = bit[i];
      for (j=4; j<N+3; ++j)
      { mat[2+N+i][j] = bit[i];
mat[3+(j-1)*(N-1)+i][2] = bit[i];
      }
    }

    /* Fill in the rest of the fixed part */

    for (i=0; i<N-1; ++i)
    { mat[3+i][4+i] = full;
      for (j=0; j<N-1; ++j) mat[3+(3+i)*(N-1)+j][1] = bit[i];
    }

    /* Fill in the part depending on the latin square */
```

```
    for (i=0; i<N-1; ++i)
     for (j=0; j<N-1; ++j)
       mat[1+2*N+ls[i][j]][4+i] = bit[j];


    /******************************/
    /* Initialize bactrack search */
    /******************************/

    for (i=0; i<(N-1)*(N-1); ++i) nodes[i] = 0.0;
    col = 0;
    setcol(col);
    backtrack = 0;


    /************************/
    /* The backtrack search */
    /************************/

    do
    {
      if ((!backtrack) && (test(col)))  /* Proceed to the next level
   of the backtrack tree ? */

/* YES */
      { nodes[col] += 1.0;
++col;
if (col == depth)
{ output(0); printf("\n");
  output(1); printf("\n");
  output(2); printf("\n");
  --col;
  backtrack = 1;
}
else setcol(col);

      }
/* NO */
      else
      {
/* Increase the current column */

backtrack = 0;
if (skip[col]==N+2) i=N+1;
else i=N+2;

        do
{
  ok = 1;
          while (ok && ((mat[3*N+col][i] <<= 1) != endbit))
          { ok = 0;
            for (j=col-1; j >= (col/(N-1))*(N-1); --j)
              ok |= (mat[3*N+j][i] == mat[3*N+col][i]);
          }
```

```
          if ((i>3) && (mat[3*N+col][i] == endbit))
  { mat[3*N+col][i] = 1;
    --i;
    if (i == skip[col]) --i;
    ok = 1;
  }
  else ok = 0;

  ok &= (i>3);

        } while (ok);

if (i<=3)
{
          ok = 1;
          while (ok && ((mat[3*N+col][3] <<= 1) != endbit))
          { ok = 0;
            for (j=col-1; j >= (col/(N-1))*(N-1); --j)
              ok |= (mat[3*N+j][3] == mat[3*N+col][3]);
          }

  if (mat[3*N+col][3] == endbit)
  /* All columns have been tried at this node;
     backtrack to the previous level */
  { backtrack = 1;
    mat[3*N+col][3] = 1;
  }

  for (i=4; i<N+3; ++i)
    mat[3*N+col][i] = (skip[col] != i);
}
if (backtrack) --col;
      }
    } while (col >= 0);

    /* Print summary of the search for this latin square */

    printf("Number of nodes  at levels:\n");
    for (i=0; i<depth; ++i) printf("%.0f ",nodes[i]);
    printf("\n");

  }
}
```

8 Literatura

Projektivne ravnine

[1] D.R.Hughes, F.C.Piper, *Projective Planes*, Springer–Verlag, New York, Heidelberg, Berlin, 1973.

[2] F.W.Stevenson, *Projective Planes*, W.H.Freeman & Co., San Francisco, 1972.

Klasifikacija projektivnih ravnina

[3] W.A.Pierce, *The impossibility of Fano's configuration in a projective plane with eight points per line*, Proceedings of the American Mathematical Society, 4 (1953), 908–912.

[4] M.Hall Jr., *Uniqueness of the projective plane with* 57 *points*, Proceedings of the American Mathematical Society, 4 (1953), 912–916.

[5] M.Hall Jr., *Correction to "Uniqueness of the projective plane with* 57 *points"*, Proceedings of the American Mathematical Society, 5 (1954), 994–997.

[6] M.Hall Jr., J.D.Swift, R.J.Walker, *Uniqueness of the projective plane of order eight*, Mathematical Tables and other Aids to Computation, 10 (1956), 186–194.

[7] M.Hall Jr., J.D.Swift, R.B.Killgrove, *On projective planes of order nine*, Mathematical Tables and other Aids to Computation, 13 (1959), 233–246.

[8] R.B.Killgrove, *A note on the nonexistence of certain projective planes of order nine*, Mathematics of Computation, 14 (1960), 70–71.

[9] E.T.Parker, R.B.Killgrove, *A note on projective planes of order nine*, Mathematics of Computation, 18 (1964), 506–508.

[10] C.W.H.Lam, G.Kolesova, L.H.Thiel, *A computer search for finite projective planes of order* 9, Discrete Mathematics, 92 (1991), 187–195.

Projektivna ravnina reda 10

[11] C.W.H.Lam, L.H.Thiel, S.Swiercz, *The nonexistence of code words of weight* 16 *in a projective plane of order* 10, Journal of Combinatorial Theory, Ser. A, 42 (1986), 207–214.

[12] C.W.H.Lam, L.H.Thiel, S.Swiercz, *A computer search for a projective plane of order* 10, u Deza, Frankl, Rosenberg (urednici), *Algebraic, Extremal and Metric Combinatorics* 1986, Lecture Notes Series 131, London Mathematical Society, 155–165.

[13] C.W.H.Lam, L.H.Thiel, S.Swiercz, *The non-existence of finite projective planes of order* 10, Canadian Journal of Mathematics, Vol. XLI, 6 (1989), 1117–1123.

Latinski kvadrati

[14] J.Dénes, A.D.Keedwell, *Latin Squares and their Applications*, Akadémiai Kiadó, Budimpešta, 1974.

[15] C.J.Colburn, J.H.Dinitz, *Handbook of Combinatorial Designs*, CRC Press, Boca Raton, 1996. Poglavlje II.1, *Latin Squares*, 97–110.

[16] B.D.McKay, E.Rogoyski, *Latin squares of order* 10, The Electronic Journal of Combinatorics, 2 (1995), #N3.

[17] V.Volenec, *Grupoidi, kvazigrupe i petlje*, Školska Knjiga, Zagreb, 1982.

[18] J.W.Brown, *Enumeration of latin squares with application to order* 8, Journal of Combinatorial Theory, 5 (1968), 177–184.

[19] C.W.H.Lam, G.Kolesova, L.H.Thiel, *On the number of* 8×8 *latin squares*, Journal of Combinatorial Theory, Ser. A, 54 (1990), 143–148.

Ostalo

[20] B.W.Keringhan, D.M.Ritchie, *The C Programming Language*, Prentice Hall, 1988.

[21] M.T.Grady, *C! Programming Principles and Practices*, Mitchel Publishing, Santa Cruz, 1988.

[22] B.D.McKay, *Nauty User's Guide (Version* 1.5*)*, Technical Report TR-CS-90-02, Computer Science Department, Australian National University, 1990.

[23] D.E.Knuth, *Estimating the efficiency of backtrack programs*, Mathematics of Computation, 29 (1975), 121–136.

[24] M.Hall Jr., D.E.Knuth, *Combinatorial analysis and computers*, American Matematical Monthly, 72 (1965), no.2, part II, 21–28.