Софийски Университет "Св. Климент Охридски"
Факултет по математика и информатика

Красимир Янков Йорджев

Побитови операции и комбинаторни алгоритми с бинарни матрици, множества и графи

Област на висше образование 4 – Природни науки, математика и информатика

Професионално направление 4.6 – Информатика и компютърни науки

ДИСЕРТАЦИЯ
за присъждане на научната степен
доктор на науките

София – 2014

Съдържание

Увод

Актуалност. Настоящата дисертация е резултат на дългогодишната работа на автора като учен и преподавател по дискретна математика и програмиране. В него са събрани голяма част от научните, научно-приложни и методически изследвания на автора в тази област, намерили отражения в над 80 публикации в реномирани специализирани издания.

Идеята за проблематиката, разисквана в дисертацията възниква при опита ни да подберем подходящи примери при работата ни с **изявените студенти**, тепърва докосващи се до съвременните научни постижения. И така, по естествен начин решавайки винаги актуалния проблем за връзката между университетското образование и съвременната наука се получиха резултатите, които определено имат и самостоятелна научна и научно-практическа стойност.

Разбира се невъзможно е материалът от големия брой публикации да бъде събран в един дисертационен труд и поради тази причина сме се спрели само на част от тях. Разискват се оригинални резултати, получени от автора, които имат отношение към следните раздели от дискретната математика и програмирането:

- езици за програмиране C++, Java и Delphi;
- побитови операции;
- алгоритми и тяхната оценка;
- комбинаторен анализ;
- бинарни матрици и приложения;
- теория на множествата и релационна алгебра;
- теория на графите и приложения;
- формални езици и граматики;

Цели и задачи на дисертационния труд. Като прилагаме методите и средствата на *компютърната алгебра* – тази част от информатиката, която се занимава с разработката, анализа, реализацията и приложението на алгебрични алгоритми[1], получаваме количествени оценки на обекти от различни области на науката и техниката:

занимателна математика – популярната главоблъсканица Судоку;

текстилна техника – структурното многообразие от тъкачни сплитки;

психология – компютърно администриране на личностни въпросници;

комбинаторика – комбинаторни задачи над бинарни матрици;

теория на графите – числови характеристики на биполярни графи;

теоретични основи на информатиката – формални езици и граматики.

[1] Дефиницията за понятието компютърна алгебра буквално е заимствана от [12].

Ще покажем, че използването на побитови операции води до повишаването на алгоритмичната ефективност по отношение на бързодействие и икономия на оперативна памет. Ще опишем и алгоритми, получени допълнително в резултат на работата ни при решаване на основните проблеми, които от своя страна имат и самостоятелно научно и научно-практическо значение.

За постигане на поставените цели бяха решени следните задачи:

- Да се дадат интересни и съдържателни примери, в които използването на побитови операции повишава ефективността (в смисъл на бързодействие и икономия на оперативна памет) на алгоритмите;
- Да се направи математически модел описващ структурното многообразие на различните видове преплитания на нишките при тъкането на платове (тъкачни сплитки). Да се решат някои комбинаторни задачи свързани с проблема. При реализацията на алгоритмите да се използват побитови операции;
- Да се реализират алгоритми за теоретико множествени операции с използване на побитови операции;
- Да се реализира алгоритъм използващ теоретико-множествени операции за решаване на произволно Судоку;
- Да се изследват някои комбинаторни проблеми, свързани с играта Судоку.
- С помощта на теорията на множествата и релационната алгебра да се направи математически модел на произволно психологическо изследване с личностни въпросници;
- Да се изследва множеството от бинарни матрици с точно определен брой единици по редовете и стълбовете;
- Ще казваме, че две бинарни матрици са еквивалентни, ако едната може да се получи от другата чрез преместване на някои от редовете и/или стълбовете. Да се изследва фактормножеството относно така дефинираната релация. Да се опише алгоритъм за получаване на точно по един представител от всеки клас на еквивалентност без да е необходимо да се обхожда цялото базово множество. При реализацията на алгоритмите да се използват побитови операции;
- Да се намери броят на двойките взаимно непресичащи се (mutually disjoint) $n \times n$ S-пермутационни матрици. Като следствие да се намери вероятността две случайно получени S-пермутационни матрици да са непресичащи се;
- Да се опишат някои нови теоретико-графови и теоретико-множествени модели в компютърната лингвистика. На базата на тези модели да се опише полиномиален алгоритъм, проверяващ дали произволен безконтекстен език се съдържа в зададен групов език (проблем, формулиран от А. В. Анисимов [6, 90]).

Научна новост. С изключение на раздел 1.1, който има справочен характер, целият текст на дисертационния труд се основава на публикации на автора, в които се описват нови и оригинални научни и научно-приложни разработки. В началото на всяка глава се дава справка за авторските публикации, от където е взаимстван текстът на изложението в дисертацията.

Методология. Методите на изследванията се основават преди всичко на строги математически доказателства. Алгоритмите многократно са тествани с различни входни данни. Резултатите свързани с психологическите изследвания (глава 3) са проверени и положително оценени от специалисти от областта на психологията.

Глава 1

Побитови операции

Раздел 1.1 има справочен характер.

Раздел 1.2 е публикуван изцяло в [183].

Раздел 1.3 е публикуван в [181, 185, 186]

В тази глава ще опишем основните дефиниции на побитовите операции и ще посочим някои интересни техни приложения при програмиране с езика C++.

В раздел 1.2 ще предложим един алгоритъм за сортиране на произволен целочислен масив, който работи за време $O(n)$, където n е размера на масива. При неговата реализация съществено ще използваме побитови операции.

Съществено се използват побитови операции и в описания в раздел 1.3 алгоритъм за получаване на всички елементи на дадено множество, притежаващи специфични свойства без да се обхожда цялото базово множество.

1.1. Основни дефиниции

Използването на побитови операции е мощно средство за програмиране при езиците C/C++ и Java. Едни от силните страни на тези езици за програмиране са възможността за програмиране на ниско ниво. Някои от средствата за тази възможност са въведените стандартни побитови операции, с помощта на които е възможно директно да се оперира с всеки бит на произволна променлива, разположена в оперативната памет на компютъра. За съжаление в разпространената у нас учебна литература липсва, или е много слабо застъпено описание за работата на съответните побитови оператори [1, 2, 33, 67–70, 124].

За да се възприемат всички възможности на побитовите операции са необходими солидни познания относно понятието "бройна система". Основните свойства на това понятие се изучава още в средното училище и курсовете по теория на числата [20, 39]. Някои разсъждения за приложението на бройните системи при решаването на различни комбинаторни задачи са показани в [25].

Побитовите операции (*bitwise operations*) в езиците за програмиране C/C++ и Java могат да бъдат прилагани единствено за целочислените типове данни, такива като стандартните `short`, `long int`, или върху тип `char`.

Приемаме, че номерацията на *битовете* (съответната цифра в двоичното представяне) в променливите и константите е отдясно наляво, като най-десния бит има номер 0.

Нека `x,y` са целочислени променливи или константи от един и същи тип, за който в оперативната памет са необходими по w бита и нека `z` е променлива от същия тип. Нека `x` и `y` са инициализирани (ако са променливи) и нека е извършено присвояването `z=x`α`y`, където α е една от операциите `&` (*побитова конюнкция*, *побитово И (AND)*), `|` (*побитова дизюнкция*, *побитово ИЛИ (OR)*) или `^` (*побитово събиране по модул 2*, *побитово изключващо ИЛИ (XOR)*). Тогава за всяко $i = 0, 1, 2, \ldots, w-1$ новото съдържание на i-я бит в `z` ще стане както е показано в таблица 1.1.

Побитовото отрицание `~` работи по следния начин: Нека `z = ~x`. Тогава, ако i-ят бит на `x` е 0, то i-ят бит на `z` става 1, а ако i-ят бит на `x` е 1, то i-ят бит на `z` става 0, $i = 0, 1, 2, \ldots, w-1$.

Ако `k` е цяло неотрицателно число, то операторът `z = x<<k` (*побитово изместване на ляво*) ще запише в $(i+k)$-я бит на `z` стойността на k-я бит на `x`, където

i-ти бит на `x`	i-ти бит на `y`	i-ти бит на `z = x & y;`	i-ти бит на `z = x | y;`	i-ти бит на `z = x ^ y;`
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Таблица 1.1. Бинарни побитови операции в езика C++

$i = 0, 1, \ldots, w-k-1$, а най-десните k на брой бита на `x` запълва с нули. Излишните битове на `x` се премахват. Тази операция е равносилна на умножение на `x` с 2^k.

По подобен начин работи и операторът `z=x>>k` (*побитово изместване на дясно*. Но тук трябва да внимаваме, тъй като операцията `>>` е машинно зависима и в различните среди за програмиране този оператор се тълкува по различен начин (виж например [15]) – някъде най-левите k бита на `z` се запълват задължително с 0 (логическо преместване), а другаде най-левите k бита на `z` се запълват със стойността на най-левия (знаковия) бит, т.е. ако числото е отрицателно, то тогава запълването ще бъде с 1 (аритметично преместване). Затова се препоръчва при работа с побитовите операции използуването на `unsigned` типове на променливите. Може също така предварително да запомним знака и преди да извършим операцията да нулираме знаковия бит. След извършването на операцията да възстановим знака. И двете операции могат да бъдат осъществени като умножим по променлива, в която е записано 1 или -1 (знака на `x`). За пример вижте описанието на функцията `Div2` (пример 1.1.5), показано малко по-долу.

В езика за програмиране Java, гореописаният проблем е разрешен като са въведени двата различни оператора `>>` и `>>>` [109, 156].

Асоциативността на побитовите операции е лява.

Приоритетът на операциите в низходящ ред е както следва:

- `~` (побитово НЕ);
- аритметичните операции: `*` (умножение), `/` (деление), `%` (остатък при деление или модул);
- аритметичните операции: `+` (събиране), `-` (изваждане);
- побитовите операции `<<` и `>>`;
- сравняване на два израза: `<`, `>`, `<=`, `>=`, `==`, `!=`;
- побитовите операции `&`,`^` и `|`;
- логическите операции `&&` и `||`.

За повече подробности относно побитовите операции и някои елементарни техни приложения може да видите напр. в [54, 106, 109, 128, 156].

Пример 1.1.1. *За да получим на колко е равен* `i`*-ят бит (0 или 1) на целочислената променлива* `x` *бихме могли да се възползуваме от функцията:*

```
int BitValue(int x, unsigned int i) {
int b = ( (x & 1<<i) == 0 ) ? 0 : 1;
    return b;
}
```

Пример 1.1.2. *Непосредствено от дефиницията на операцията побитово изместване на ляво (<<) следва ефективността на следната функция изчисляваща* 2^n*, където* n *е цяло неотрицателно число:*

```
unsigned int Power2(unsigned int n) {
return 1<<n;
}
```

Когато работим с отрицателни числа трябва да се съобразяваме, че в компютъра представянето на отрицателните числа е чрез така наречения допълнителен код. Следващата функция показва как се кодират целите числа в оперативната памет на компютъра, с който работим. За простота ще работим с тип `short`, но не е проблем функцията да се предефинира и за други целочислени типове.

Цяло число тип `short`	Представяне в ОП
0	0000000000000000
1	0000000000000001
-1	1111111111111111
2	0000000000000010
-2	1111111111111110
$16 = 2^4$	0000000000010000
$-16 = -2^4$	1111111111110000
$26 = 2^4 + 2^3 + 2$	0000000000011010
$-26 = -(2^4 + 2^3 + 2)$	1111111111100110
$41 = 2^5 + 2^3 + 1$	0000000000101001
$-41 = -(2^5 + 2^3 + 1)$	1111111111010111
$32767 = 2^{15} - 1$	0111111111111111
$-32767 = -(2^{15} - 1)$	1000000000000001
$32768 = 2^{15}$	1000000000000000
$-32768 = -2^{15}$	1000000000000000

Таблица 1.2. Представяне на някои числа от тип short в оперативната памет на компютъра

Пример 1.1.3. *Функция показваща представянето на числата от тип* `short` *в оперативната памет на компютъра.*

```
void BinRepl(short n)
{
     int b;
     int d = sizeof(short)*8 - 1;
     while (d>=0)
           {
           b= 1<<d & n ? 1 : 0;
           cout << b;
           d--;
           }
}
```

Някой експерименти с функцията `BinRepl` са дадени в таблица 1.2:

Пример 1.1.4. *Функция разпечатваща дадено цяло число в двоична бройна система. Сравнете тази функция с функцията представена в пример 1.1.3.*

```
void DecToBin(int n)
{
     if (n<0) cout<<'-'; // Печата "-", ако n<0
     n = abs(n);
     int b;
     int d = sizeof(int)*8 - 1;
//Пропуска незначещите нули в началото:
     while ( d>0 && (n & 1<<d ) == 0 ) d--;
     while (d>=0)
           {
           b= 1<<d & n ? 1 : 0;
           cout<<b;
           d--;
           }
}
```

Пример 1.1.5. *За да разделим цялото число* `x` *на* 2^n *целочислено (т.е. като се „изреже“ дробната част на резултата), можем да се възползуваме от функцията:*

```
int Div2(int x, unsigned int n) {
    int s = x<0 ? -1 : 1; // s = знака на x
    x = x*s;  // Нулираме знаковия бит на x
return (x>>n)*s;
}
```

Пример 1.1.6. *Следващата функция пресмята броя на единиците в дадено цяло число* `n` *записано в двоична бройна система. Тук отново игнорираме знака на числото (ако е отрицателно) и работим с абсолютната му стойност.*

```
int NumbOf_1(int n)
{
     n = abs(n);
     int temp=0;
     int d = sizeof(int)*8 - 1;
     for (int i=0; i<d; i++)
         if (n & 1<<i) temp++;
     return temp;
}
```

1.2. Побитова сортировка

В този раздел ще предложим един бърз алгоритъм за сортиране на произволен целочислен масив. И тъй като при неговата реализация съществено ще използваме побитови операции, ще го наречем ”Побитова сортировка”. Ще докажем, че побитовата сортировка работи за време $O(n)$, където n е размера на масива. Това е една отлична оценка по отношение на критерия време. За сравнение в таблица 1.3 даваме някои от най-известните алгоритми за устойчиви сортировка и техните оценки по критерий време [105, 129, 157].

Алгоритъм	англ.	Сложност
Сортировка чрез избор	Selection sort	$O(n^2)$
Метод на мехурчетата	Bubble sort	$O(n^2)$
Размесваща (коктейлна)	Bidirectional bubble sort (Cocktail sort)	$O(n^2)$
Сортировка чрез вмъкване	Insertion sort	$O(n^2)$
Сортировка чрез сливане	Merge sort	$O(n \log n)$
Чрез бинарно дърво	Tree sort	$O(n \log n)$
Сортировка на Тим	Timsort	$O(n \log n)$
Бърза сортировка	Quicksort	$\approx O(n \log n)$
Чрез изброяване	Counting sort	$O(n + m)$
Блокова (кошница)	Bucket sort	$O(n)$

Таблица 1.3. Някои от най-известните алгоритми за сортиране

Забележки:

1. **Timsort** е разработен за ползване с езика Python [122].
2. За **Counting sort** m е втори параметър, даващ броя на уникалните ключове и е необходимо $O(n + m)$ допълнителна памет.
3. За **Bucket sort** е необходимо $O(m)$ допълнителна памет, където m е втори параметър, даващ броя на уникалните ключове и е необходимо и знание за природата на сортираните данни, които излизат извън рамките на функциите "преместване" и "сравняване".

1.2.1. Програмен код на алгоритъма

Създаденият от нас алгоритъм, описан с помощта на езика за програмиране C++ е показан в програмния код 1.2.1. Първо създаваме функция, която сортира масив, чиито елементи са или само неотрицателни или само отрицателни. Втората функция разделя даденият масив на два непресичащи се подмасива съответно само с отрицателни и само с неотрицателни елементи. След сортировката на всеки един от тези подмасиви, ние ги слепваме, така че да се получи един окончателно сортиран масив.

Програмен код 1.2.1. *Побитова сортировка*

```
template <class T>
void BitwiseSort1(T A[],int n){
//сортира целочислени елементи с еднакви знаци
     int t = sizeof(T)*8;  // t - размер на типа T в битове
     T A0[n];  // запомня елементите, за които k-ят бит е 0
     T A1[n];  // запомня елементите, за които k-ят бит е 1
     int n0; // размер на A0
     int n1; // размер на A1
     for (int k=0; k<t-1; k++) {
/* k - номер на бит.
   Номерацията започва от 0. Не проверява знаковия бит  */
```

```
        n0=0;
        n1=0;
/*  проверява на колко е равен k-ят бит
    на i-я елемент на масива:  */
        for (int i=0; i<n; i++){
            if (A[i] & 1<<k) {
                    A1[n1] = A[i];
                    n1++;
                    }
                    else {
                    A0[n0] = A[i];
                    n0++;
                    }
        }

/* Слепване на двата масива.
   В резултат получаваме масив на чийто елементи k-я бит
   е равен на 0 са в началото, а ако е равен на 1 в края */
        for (int i=0; i<n0; i++) A[i] = A0[i];
        for (int i=0; i<n1; i++) A[n0+i] = A1[i];
     }
}
template <class T>
void BitwiseSort(T A[],int n){
    T Aminus[n];  // Масив с отрицателните стойности на A
    T Aplus[n];  // Масив с неотрицателните стойности на A
    int Nm = 0;   // Брой елементи в записани в Aminus
    int Np = 0;   // Брой елементи в записани в Aplus
    for (int i=0; i<n; i++)
        if (A[i] < 0) {
                Aminus[Nm] = A[i];
                Nm++;
                }
                else {
                Aplus[Np] = A[i];
                Np++;
                }
    BitwiseSort1(Aminus,Nm); // Сортира отрицателните елементи
    BitwiseSort1(Aplus,Np); // Сортира неотрицателните елементи

    /* Слепване на двата масива: */
    for (int i=0; i<Nm; i++) A[i] = Aminus[i];
    for (int i=Nm; i<n; i++) A[i] = Aplus[i-Nm];
}
```

1.2.2. Оценка на алгоритъма

Като основен недостатък на описания от нас алгоритъм е фактът, че той е приложим единствено за масиви от цели числа или символи (тип `char`). Причина за това е, че при него съществено се използват побитови операции, които

са приложими единствено над целочислени типове от данни. Но този недостатък се компенсира от неговото бързодействие. Както ще видим по-долу, алгоритъм 1.2.1 работи за време $O(n)$, където n е броят на елементите, които подлежат на сортировка.

Освен чрез многократните експерименти, които направихме, с помощта на следващата теорема ще докажем коректността на създадения от нас алгоритъм.

Теорема 1.2.2. *При всяко изпълнение на алгоритъм 1.2.1 при произволни входни данни (масив от цели числа) в резултат се получава сортиран масив.*

Доказателство. Достатъчно е да докажем, че функция `BitwiseSort1` работи така, че да изпълнява условията на теоремата.

Нека $A = \{a_0, a_1, \ldots, a_{n-1}\}$ е произволен целочислен масив с дължина n и нека $A^{(k)} = \{a_o^{(k)}, a_1^{(k)}, a_{n-1}^{(k)}\}$ е масивът, който се получава след итерация с номер k, където $k = 0, 1, \ldots, t-2$, $t =$ `sizeof(T)*8`, т.е. t е равно на броя на битовете, които заема всеки елемент на A в оперативната памет на компютъра.

Нека x е цяло число. За всяко естествено число $k = 0, 1, 2, \ldots$ дефинираме функциите:

$$\nu_k(x) = x \,\%\, 2^k,$$

където както и в езиците за програмиране C/C++ и Java операторът % означава остатъка при целочислено деление. Очевидно $\nu_{s-1}(x) = x$ ако абсолютната стойност на цялото число x може да бъде записано с не повече от s цифри 0 или 1 в двоична бройна система. Следователно за да докажем, че в резултат на работата на алгоритъма масивът $A^{(t-2)}$ е сортиран, е достатъчно да се докаже, че масивът $\overline{A^{(t-2)}} = \{\nu_{t-2}(a_0^{(t-2)}), \nu_{t-2}(a_1^{(t-2)}), \ldots, \nu_{t-2}(a_{n-1}^{(t-2)})\}$ е сортиран. Прилагайки индуктивни разсъждения, ние ще докажем, че за всяко s, такова че $0 \leq s < t-1$ масивът $\overline{A^{(s)}} = \{\nu_s(a_0^{(s)}), \nu_s(a_1^{(s)}), \ldots \nu_s(a_{n-1}^{(s)})\}$ е сортиран.

При $s = 0$ твърдението следва от факта, че при итерация с номер 0 ($k = 0$), $A^{(0)}$ е подреден така, че първо са всички елементи на масива, които в двоичния си запис завършват на нула, следвани от всички елементи на масива, които в двоичният си запис завършват на 1.

Допускаме, че за някое естествено число s, $0 \leq s < t-2$ масивът $\overline{A^{(s)}} = \{\nu_s(a_0^{(s)}), \nu_s(a_1^{(s)}), \ldots \nu_s(a_{n-1}^{(s)})\}$ е сортиран. Но тогава анализирайки работата на алгоритъма в $(s+1)$-та итерация, лесно се вижда, че масивът `A0`, който се получава от $\overline{A^{(s)}}$ вземайки в същия ред само тези елементи на $A^{(s)}$ имащи 0 в бит с номер $s+1$ е сортиран масив. Аналогично се доказва, че масивът `A1` е сортиран и в бит с номер $s+1$ на всеки от елементите му стои 1. Тогава масивът $\overline{A^{(s+1)}} = \{\nu_{s+1}(a_0^{(s+1)}), \nu_{s+1}(a_1^{(s+1)}), \ldots \nu_{s+1}(a_{n-1}^{(s+1)})\}$, който се получава от сливането на масивите `A0` и `A1` като елементите на `A0` предхождат елементите на `A1` е сортиран. С това доказахме и теоремата.

□

Теорема 1.2.3. *Алгоритъмът, описан с помощта на програмен код 1.2.1 работи за време $O(n)$.*

Доказателство. Твърдението на теоремата следва от факта, че във функция `BitwiseSort1` имаме единствено два вложени един в друг цикъла. Във вътрешния цикъл се извършват точно n итерации, като във всяка итерация се извършват веднъж операцията `&` (побитова конюнкция), веднъж операцията `<<` (побитово изместване в ляво), веднъж условният оператор `if`, веднъж операторът за присвояване

и веднъж операторът за увеличение на стойността на целочислена променлива с 1. Всяка от гореизброените операции се извършва за константно време. Външният цикъл прави $t-2$ итерации, където t е константа и във всяка итерация има освен вътрешните цикли и две нулирания на променлива.

Във функция `BitwiseSort` разделянето на масива на два непресичащи се подмасива се извършва очевидно за време $O(n)$. Двата новополучени два масива се сортират за сумарно време $O(n)$. Последващото сливане на двата сортирани масива с обща дължина n очевидно също се извършва за време $O(n)$.

1.3. Алгоритъм за получаване на всички елементи на дадено множество, притежаващи специфични свойства без да се обхожда цялото базово множество

1.3.1. Полуканонични и канонични бинарни матрици

Бинарна (или *булева*, или (0,1)-матрица) се нарича матрица, чийто елементи принадлежат на множеството $\mathcal{B} = \{0, 1\}$.

Нека n и m са цели положителни числа. С $\mathcal{B}_{n\times m}$ ще означаваме множеството от всички $n \times m$ бинарни матрици, а с $\mathcal{B}_n = \mathcal{B}_{n\times n}$ ще означаваме множеството на всички квадратни $n \times n$ бинарни матрици.

Дефиниция 1.3.1. *Нека $A \in \mathcal{B}_{n\times m}$.*

С $r(A)$ ще означаваме наредената n-торка

$$\langle x_1, x_2, \ldots, x_n\rangle,$$

където $0 \leq x_i \leq 2^m - 1$, $i = 1, 2, \ldots n$ и x_i е естественото число, записано в двоична бройна система, цифрите на които представляват i-я ред на A.

Аналогично със $c(A)$ ще означаваме наредената m-торка

$$\langle y_1, y_2, \ldots, y_m\rangle$$

където $0 \leq y_j \leq 2^n - 1$, $j = 1, 2, \ldots m$ и y_j е естественото число, записано в двоична бройна система, цифрите на които представляват j-я стълб на A.

Разглеждаме множествата:

$$\begin{aligned}\mathcal{R}_{n\times m} &= \{\langle x_1, x_2, \ldots, x_n\rangle \mid 0 \leq x_i \leq 2^m - 1,\ i = 1, 2, \ldots n\} \\ &= \{r(A) \mid A \in \mathcal{B}_{n\times m}\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}_{n\times m} &= \{\langle y_1, y_2, \ldots, y_m\rangle \mid 0 \leq y_j \leq 2^n - 1,\ j = 1, 2, \ldots m\} \\ &= \{c(A) \mid A \in \mathcal{B}_{n\times m}\}\end{aligned}$$

С "$<$" ще означаваме лексикографските наредби в $\mathcal{R}_{n\times m}$ и в $\mathcal{C}_{n\times m}$

Лесно се вижда, че в дефиниция 1.3.1 се описват две изображения

$$r : \mathcal{B}_{n\times m} \to \mathcal{R}_{n\times m}$$

и

$$c : \mathcal{B}_{n\times m} \to \mathcal{C}_{n\times m},$$

които са биективни и следователно

$$\mathcal{R}_{n\times m} \cong \mathcal{B}_{n\times m} \cong \mathcal{C}_{n\times m}.$$

В [131] е доказано, че представянето на елементите на $\mathcal{B}_n$, използвайки наредени n-торки от естествени числа, води до създаването на по-бързи и пестящи памет алгоритми.

Дефиниция 1.3.2. *Нека* $A \in \mathcal{B}_{n\times m}$,

$$r(A) = \langle x_1, x_2, \ldots, x_n\rangle,$$

$$c(A) = \langle y_1, y_2, \ldots, y_m\rangle.$$

Матрицата A *ще наричаме* ***полуканонична****, ако*

$$x_1 \leq x_2 \leq \cdots \leq x_n$$

и

$$y_1 \leq y_2 \leq \cdots \leq y_m.$$

Твърдение 1.3.3. *Нека* $A = [a_{ij}] \in \mathcal{B}_{n\times m}$ *е полуканонична матрица. Тогава съществуват цели числа* i, j*, такива че* $1 \leq i \leq n$, $1 \leq j \leq m$ *и*

$$a_{1\,1} = a_{1\,2} = \cdots = a_{1\,j} = 0, \quad a_{1\,j+1} = a_{1\,j+2} = \cdots = a_{1\,m} = 1, \tag{1.3.1}$$

$$a_{1\,1} = a_{2\,1} = \cdots = a_{i\,1} = 0, \quad a_{i+1\,,1} = a_{i+2\,,1} = \cdots = a_{n\,1} = 1. \tag{1.3.2}$$

Доказателство. Нека $r(A) = \langle x_1, x_2, \ldots x_n\rangle$ и $c(A) = \langle y_1, y_2, \ldots y_m\rangle$. Допускаме, че съществуват цели числа p и q, такива че $1 \leq p < q \leq m$, $a_{1\,p} = 1$ и $a_{1\,q} = 0$. Но в този случай $y_p > y_q$, което противоречи на условието за полуканоничност на матрицата A. Доказахме (1.3.1). Аналогично се доказва и (1.3.2).

□

Нека n е цяло положително число. С $[n]$ ще означаваме множеството

$$[n] = \{1, 2, \ldots, n\}.$$

С $\mathcal{P}_n \subset \mathcal{B}_n = \mathcal{B}_{n\times n}$ ще означаваме множеството от всички *пермутационни* матрици, т.е. множеството от всички $n \times n$ бинарни матрици, имащи точно по една единица на всеки ред и всеки стълб. В сила е изоморфизмът:

$$\mathcal{P}_n \cong \mathcal{S}_n,$$

където с $\mathcal{S}_n$ сме означили *симетричната група* от n-ти ред, т.е. групата от всички взаимно-еднозначни изображения на множеството $[n] = \{1, 2, \ldots n\}$ в себе си.

Както е добре известно [66] умножението на произволна реална или комплексна матрица A отляво с пермутационна матрица (ако умножението е възможно) води до разместване на редовете на матрицата A, а умножението на A отдясно с пермутационна матрица води до разместването на стълбовете на A.

Дефиниция 1.3.4. *Нека* $A, B \in \mathcal{B}_{n\times m}$*. Ще казваме, че матриците* A *и* B *са* ***еквивалентни*** *и ще пишем*

$$A \sim B, \tag{1.3.3}$$

ако съществуват пермутационни матрици $X \in \mathcal{P}_n$ *и* $Y \in \mathcal{P}_m$*, такива че*

$$A = XBY \tag{1.3.4}$$

С други думи $A \sim B$ *ако* A *се получава от* B *след разместване на някои от редовете и/или стълбовете на* B*.*

Очевидно така въведената релация е релация на еквивалентност.

С $\mathcal{T}_n \subset \mathcal{P}_n$ означаваме множеството от всички *транспозиции* в $\mathcal{P}_n$, т.е. множеството от всички $n \times n$ пермутационни матрици, които умножавайки от ляво произволна $n \times m$ матрица разменя местата на точно два реда, а умножавайки от дясно произволна $k \times n$ матрица разменя местата на точно два стълба.

Теорема 1.3.5. *Нека A е произволна матрица от $\mathcal{B}_{n\times m}$, а $<$ е лексикографската наредба в $\mathcal{R}_{n\times m}$ и $\mathcal{C}_{n\times m}$. Тогава:*

а) Ако $X_1, X_2, \cdots, X_s \in \mathcal{T}_n$ са такива, че

$$r(X_1X_2\dots X_sA) < r(X_2X_3\dots X_sA) < \cdots < r(X_sA) < r(A),$$

то

$$c(X_1X_2\dots X_sA) < c(A).$$

b) Ако $Y_1, Y_2, \cdots, Y_t \in \mathcal{T}_m$ са такива, че

$$c(AY_1Y_2\dots Y_t) < c(AY_2Y_3\dots Y_t) < \cdots < c(AY_t) < c(A),$$

то

$$r(AY_1Y_2\dots Y_t) < r(A).$$

Доказателство. а) Индукция по s. Нека $s = 1$ и нека $X_1 \in \mathcal{T}_n$ е транспозиция, която умножавайки произволна матрица $A = [a_{ij}] \in \mathcal{B}_{n\times m}$ отляво разменя местата редовете на A с номера u и v $(1 \le u < v \le n)$, а останалите редове остават на място. С други думи, ако

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1r} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2r} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{u1} & a_{u2} & \cdots & a_{ur} & \cdots & a_{um} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{v1} & a_{v2} & \cdots & a_{vr} & \cdots & a_{vm} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nr} & \cdots & a_{nm} \end{bmatrix}$$

то

$$X_1A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1r} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2r} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{v1} & a_{v2} & \cdots & a_{vr} & \cdots & a_{vm} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{u1} & a_{u2} & \cdots & a_{ur} & \cdots & a_{um} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nr} & \cdots & a_{nm} \end{bmatrix},$$

където $a_{ij} \in \{0, 1\}$, $1 \le i \le n$, $1 \le j \le m$.

Нека

$$r(A) = \langle x_1, x_2, \dots, x_u, \dots, x_v, \dots, x_n \rangle.$$

Тогава

$$r(X_1A) = \langle x_1, x_2, \dots, x_v, \dots, x_u, \dots, x_n \rangle.$$

Тъй като $r(X_1A) < r(A)$, то съгласно свойствата на лексикографската наредба $x_v < x_u$. Съгласно дефиниция 1.3.1 представяниятя на x_u и x_v в двоична бройна система с евентуално допълване при необходимост с незначещи нули в началото е съответно както следва:

$$x_u = a_{u1}a_{u2}\cdots a_{um},$$

$$x_v = a_{v1}a_{v2}\cdots a_{vm}.$$

Тъй като $x_v < x_u$, то съществува цяло число $r \in \{1, 2, \ldots, m\}$, такова че $a_{uj} = a_{vj}$ при $j < r$, $a_{ur} = 1$ и $a_{vr} = 0$.

От тук следва, че ако $c(A) = \langle y_1, y_2, \ldots, y_m\rangle$, $c(X_1A) = \langle z_1, z_2, \ldots, z_m\rangle$, то $y_j = z_j$ когато $j < r$, а представянето на y_r и z_r в двоична бройна система с евентуално допълване при необходимост с незначещи нули в началото е съответно както следва:

$$y_r = a_{1r}a_{2r}\cdots a_{u-1\,r}a_{ur}\cdots a_{vr}\cdots a_{nr},$$

$$z_r = a_{1r}a_{2r}\cdots a_{u-1\,r}a_{vr}\cdots a_{ur}\cdots a_{nr}.$$

Тъй като $a_{ur} = 1$, $a_{vr} = 0$, то $z_r < y_r$, откъдето следва че $c(X_1A) < c(A)$.

Допускаме, че за всяка s-торка от транспозиции $X_1, X_2, \ldots, X_s \in \mathcal{T}_n$ и за всяка матрица $A \in \mathcal{B}_{n\times m}$ от

$$r(X_1X_2\ldots X_sA) < r(X_2\cdots X_sA) < \cdots < r(X_sA) < r(A)$$

следва

$$c(X_1X_2\ldots X_sA) < c(A)$$

и нека $X_{s+1} \in \mathcal{T}_n$ е такава, че

$$r(X_1X_2\ldots X_sX_{s+1}A) < r(X_2\cdots X_sX_{s+1}A) < \cdots < r(X_{s+1}A) < r(A).$$

Съгласно индукционното предположение

$$c(X_{s+1}A) < c(A).$$

Полагаме

$$A_1 = X_{s+1}A.$$

Съгласно индукционното предположение от

$$r(X_1X_2\ldots X_sA_1) < r(X_2\cdots X_sA_1) < \cdots < r(X_sA_1) < r(A_1)$$

следва, че

$$c(X_1X_2\cdots X_sX_{s+1}A) = c(X_1X_2\cdots X_sA_1) < c(A_1) = c(X_{s+1}A) < c(A),$$

с което доказахме а).

b) се доказва аналогично на а).

□

Очевидно е в сила и дуалното на теорема 1.3.5 твърдение, в което навсякъде вместо знака "<" поставим знака ">".

Следствие 1.3.6. *Нека $A \in \mathcal{B}_{n\times m}$. Тогава $r(A)$ е минимален елемент относно лексикографската наредба в множеството $\{r(B) \mid B \sim A\}$ тогава и само тогава когато $c(A)$ е минимален елемент относно лексикографската наредба в множеството $\{c(B) \mid B \sim A\}$.*

□

Следствие 1.3.6 ни дава основание да формулираме следната дефиниция:

Дефиниция 1.3.7. *Матрицата $A \in \mathcal{B}_{n\times m}$ ще наричаме* ***канонична матрица****, ако $r(A)$ е минимален елемент относно лексикографската наредба в множеството $\{r(B) \mid B \sim A\}$, и следователно $c(A)$ е минимален елемент относно лексикографската наредба в множеството $\{c(B) \mid B \sim A\}$.*

Множеството $\mathcal{M} \subseteq \mathcal{B}_{n\times m}$ ще наричаме **затворен клас** относно релацията $\sim$ (дефиниция 1.3.4), ако от $A \in \mathcal{M}$, $B \in \mathcal{B}_{n\times m}$ и $B \sim A$, следва, че $B \in \mathcal{M}$. От дефиниция 1.3.7 непосредствено следва, че във всеки клас на еквивалентност в даден затворен клас относно релацията $\sim$ съществува единствена канонична матрица.

Ако една матрица $A \in B_{n\times m}$ е канонична матрица, то лесно се вижда, че A е полуканонична матрица, но както ще видим в следващия пример обратното твърдение не винаги е вярно.

Пример 1.3.8. *Разглеждаме матриците:*

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad \text{и} \qquad B = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

След непосредствена проверка установяваме, че $A \sim B$. Освен това $r(A) = \langle 7, 9, 10, 14\rangle$, $c(A) = \langle 7, 9, 11, 12\rangle$, $r(B) = \langle 3, 5, 13, 14\rangle$, $c(B) = \langle 3, 7, 9, 14\rangle$. Следователно A и B са две еквивалентни помежду си полуканонични матрици.

От пример 1.3.8 непосредствено следва, че в даден клас на еквивалентност е възможно съществуването на повече от един полуканоничен елемент.

Нека $\mathcal{W}$ е произволно подмножество на $\mathcal{B}_n$, такова че ако $A \in \mathcal{W}$ и $B \sim A$, то $B \in \mathcal{W}$. Тогава, очевидно съществува единствена канонична матрица във всеки клас на еквивалентност във фактор множеството $\mathcal{W}_{/\sim}$. Следователно броят на каноничните матрици $\mathcal{W}$ ще ни даде мощността на фактор множеството $\mathcal{W}_{/\sim}$.

1.3.2. Постановка на задачата

Голям клас от задачи по програмиране имат следната обобщена формулировка:

Задача 1.3.9. *Дадено е множеството $\mathcal{M}$. Да се състави програма, получаваща множеството $\delta(\mathcal{M})$ от всички елементи на $\mathcal{M}$, притежаващи дадени свойства.*

Предполагаме, че е известна процедура, която при зададен елемент на $\mathcal{M}$ отговаря утвърдително или отрицателно дали този елемент принадлежи, или съответно не принадлежи на $\delta(\mathcal{M})$. Решението на задача 1.3.9 в реално време се усложнява, ако множеството $\mathcal{M}$ е голямо, т.е. ако неговата мощност е експоненциална функция на един или повече параметри. В този случай времето необходимо да се обходят всички елементи на множеството и да се провери за всеки елемент дали притежава необходимите свойства е неефективно голямо при растенето на параметрите. Подобни задачи са обект на разглеждане в [13].

Едно решение на гореспоменатия проблем е да се намери множество $\mathcal{K} \subset \mathcal{M}$, такова че $|\mathcal{K}| < |\mathcal{M}|$ и $\delta(\mathcal{M}) \subseteq \mathcal{K}$. Тогава ще се проверява дали притежават зададените свойства не всички елементи на $\mathcal{M}$, а само елементите на $\mathcal{K}$. Идеалният

случай е когато $\mathcal{K} = \delta(\mathcal{M})$, но за съжаление не винаги това е лесна задача от алгоритмична гледна точка. Един критерий за ефективност може да служи частното $\frac{|\mathcal{K}|}{|\mathcal{M}|}$, като ефективността е обратно-пропорционална на стойността на това частно. Различни алгоритми, реализиращи този подход се разискват в [14]. Тук ние ще използуваме побитовите операции за да решим една задача от този клас.

Дефиниция 1.3.10. *Квадратните $n \times n$ бинарни матрици във всеки ред и всеки стълб, на които съществуват точно k, $0 \leq k \leq n$ единици ще наричаме Λ_n^k-матрици, като с Λ_n^k ще означаваме и множеството от тези матрици.*

При въвеждане на означението Λ_n^k сме се съобразили с [76].
Нека n и k да са цели положителни числа. Разглеждаме множествата:

$\mathcal{B}_n$ – множеството от всички квадратни $n \times n$ бинарни матрици;

Λ_n^k – множеството от всички квадратни $n \times n$ бинарни матрици имащи точно k единици във всеки ред и всеки стълб;

$\mathcal{U}_n = \{A \in \mathcal{B}_n \mid r(A) = \langle x_1, x_2, \ldots, x_n \rangle,\ x_1 \leq x_2 \leq \cdots \leq x_n\} \subset \mathcal{B}_n$;

$\mathcal{V}_n = \{A \in \mathcal{U}_n \mid c(A) = \langle y_1, y_2, \ldots, y_n \rangle,\ y_1 \leq y_2 \leq \cdots \leq y_n\} \subset \mathcal{U}_n$;

$\mathcal{Z}_n^k = \Lambda_n^k \cap \mathcal{V}_n$ – множеството от всички полуканонични матрици в Λ_n^k.

Да разгледаме следната задача:

Задача 1.3.11. *Да се опише и реализира алгоритъм, получаващ множеството $\mathcal{Z}_n^k$ без да се разглеждат всички елементи на $\mathcal{B}_n$.*

С други думи търсим алгоритъм за получаване на всички елементи на дадено множество, притежаващи специфични свойства (полуканонични $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб) без да се обхожда цялото базово множество (множеството $\mathcal{B}_n$ от всички $n \times n$ бинарни матрици).

Решавайки задача 1.3.11 ще улесним до минимум решението на задачата за намирането на $\mu(n,k) = \left|\Lambda_{n/\sim}^k\right|$, т.е на броя на класовете на еквивалентност в множеството Λ_n^k, съгласно дефинираната в 1.3.4 релация. Задачата за намирането на броя на класовете на еквивалентност за всяко n и k е открит научен проблем. Задача 1.3.11 ще решим като използваме съществено свойствата на побитовите операции с цел повишаване на ефективността на създадения от нас алгоритъм.

1.3.3. Описание и реализация на алгоритъма с използване на побитови операции

Както е добре известно, съществуват точно 2^n цели неотрицателни числа, които са записани с не повече от n цифри в двоична бройна система. От тях ние трябва да изберем всички, които имат точно k единици в двоичния си запис. Техният брой е $\binom{n}{k} \ll 2^n$. Бихме могли да използуваме функцията `NumbOf_1(int)` от пример 1.1.6 (стр. 11), но тогава ще трябва да я използуваме за всяко цяло число от интервала $[0, 2^n - 1]$, т.е. 2^n пъти. Ние ще опишем алгоритъм, който директно получава необходимите ни елементи, **без да проверява за всяко цяло число** $m \in [0, 2^n - 1]$ дали отговаря на условията. Резултата запомняме в масива `p[]` с размер $c = \binom{n}{k}$. При това полученият масив е сортиран във възходящ ред и няма

дублиращи се елементи. Алгоритъмът се основава на факта, че множеството на всички наредени m-торки $\mathcal{B}^m = \langle b_1, b_2, \ldots, b_m \rangle$, $b_i \in \mathcal{B} = \{0, 1\}$, $i = 1, 2, \ldots, m$, $m = 1, 2, \ldots, n$, се разбива на две непресичащи се подмножества $\mathcal{B}^m = \mathcal{M}_1 \cup \mathcal{M}_2$, $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$, където $\mathcal{M}_1 = \{\langle b_1, b_2, \ldots, b_m \rangle \mid b_1 = 0\}$ и $\mathcal{M}_2 = \{\langle b_1, b_2, \ldots, b_m \rangle \mid b_1 = 1\}$. Описаният рекурсивен алгоритъм съществено ще използува побитови операции.

```
void DataNumb(int p[], unsigned int n, unsigned int k, int& c)
{
if (k==0)
    {
    c = 1;
    p[0] = 0;
    }
    else if (k==n)
        {
        c = 1;
        p[0] = 0;
        for (int i=0; i<k; i++) p[0] |=  1<<i;
        }
        else
            {
            int p1[10000], p2[10000];
            int c1, c2;
            DataNumb(p1, n-1, k, c1);
            DataNumb(p2, n-1, k-1, c2);
            c = c1+c2;
            for (int i=0; i<c1; i++) p[i] = p1[i];
            for (int i=0; i<c2; i++) p[c1+i] = p2[i] | 1<<(n-1);
            }
}
```

Ние ще използуваме побитови операции и при конструирането на следващите две функции.

Функцията

`int n_tuple(int[], int, int, int)`

получава всички $t = \binom{n+k-1}{k}$ (брой комбинации с повторения) на брой наредени n-торки $\langle x_1, x_2, ..., x_n \rangle$, където $0 \leq x_1 \leq x_2 \leq \ldots \leq x_n < c$. x_i, $i = 1, 2, \ldots, n$ са елементи на предварително сортирания масив `p[]` с размер c. При своята работа ще се обръща към функцията

`bool check(int[], int)`,

която се отнася до използването на всяка от получените n-торки, т.е. която проверява дали това е полуканоничен елемент и я извежда. В резултат функцията връща броя на полуканоничните елементи.

```
bool check(int x[], int n, int k)
{
     int yj; // числото представящо (n-j)-я стълб
     int y0=0; // числото предхождащо j-я стълб
     int b;
```

```
     for (int j=n-1; j>=0; j--)
     {
         yj=0;
         for (int i=0; i<n; i++)
         {
             b = 1<<j & x[i] ? 1 : 0;
             yj |= b << (n-1-i);
         }

         if (yj<y0 || (NumbOf_1(yj) != k)) return false;
         y0 = yj;
     }
     // Получили сме полуканоничен елемент. Разпечатваме го.
     for (int i=0; i<n; i++) cout<<x[i]<<"  ";
     cout<<'\n';
     return true;
}

int n_tuple(int p[], int n, int k, int c)
{
     int t=0;
     int a[n], x[n];
     int indx = n-1;
     for (int i=0; i<n; i++) a[i]=0;
     while (indx >= 0)
     {
           for (int i=indx+1; i<n; i++) a[i] = a[indx];

           for (int i=0; i<n; i++) x[i] = p[a[i]];
           if(check(x,n,k)) t++;

           indx = n-1;
           a[indx]++;
           while (indx>=0 && a[indx]==c)
           {
                 indx--;
                 a[indx]++;
           }
     }
     return t;
}
```

Тук ще пропуснем описанието на `main()` функцията.

1.3.4. Някои резултати от работата на алгоритъма

Нека n и k са естествени числа. Да означим с $\nu(n,k)$ броя на всички полуканонични матрици в Λ_n^k, т.е.

$$\nu(n,k) = \left|\mathcal{Z}_n^k\right| = \left|\Lambda_n^k \cap \mathcal{V}_n\right|.$$

Тук за $k = 2, 3, 4$ и 5 ние ще посочим началото на редиците $\{\nu(n,k)\}_{n=k}^{\infty}$ за някои не големи стойности на параметъра n. С помощта на компютърната програма, описана в раздел 1.3.3 са получени следните резултати:

$$\{\nu(n,2)\}_{n=1}^{\infty} = \{0,1,1,2,5,13,42,155,636,2889,14321,76834,443157,\ldots\} \quad (1.3.5)$$

$$\{\nu(n,3)\}_{n=1}^{\infty} = \{0,0,1,1,3,25,272,4070,79221,1906501,\ldots\} \quad (1.3.6)$$

$$\{\nu(n,4)\}_{n=1}^{\infty} = \{0,0,0,1,1,5,161,7776,626649,\ldots\} \quad (1.3.7)$$

$$\{\nu(n,5)\}_{n=1}^{\infty} = \{0,0,0,0,1,1,8,1112,287311,\ldots\} \quad (1.3.8)$$

Числовите редици (1.3.5), (1.3.6), (1.3.7) и (1.3.8) са отбелязани в енциклопедията от числови последователности [164], съответно под номерата A229161 [79], A229162 [80], A229163 [81] и A229164 [82]. Всички те са представени от N. J. A. Sloane, който се позовава на работата [186] на автора на дисертационния труд.

Редицата (1.3.5) се коментира от Brendan McKay и е допълнена от R. H. Hardin с елементите $\nu(12,2) = 76\,834$ и $\nu(13,2) = 443\,157$.

В редицата (1.3.6) е добавен елементът $\nu(10,3) = 1\,906\,501$ от R. H. Hardin.

1.4. Заключение към първа глава

Настоящата глава има въвеждащ характер. Нейното предназначение е да дадем основните дефиниции и кратко описание на операторите за работа с побитовите операции. Дадени са някои интересни примери за приложение на побитовите операции при програмирането с алгоритмичните езици C,C++ и Java.

Определен научен интерес представляват описаният в раздел 1.2 бърз алгоритъм за сортиране на цели числа, който работи за време $O(n)$, където n е броя на числата подлежащи на сортиране и описаният в раздел 1.3 алгоритъм за получаване на всички елементи на дадено множество притежаващи специфични свойства (полуканонични $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб) без да се обхожда цялото базово множество (множеството $\mathcal{B}_n$ от всички $n \times n$ бинарни матрици).

Глава 2

Математическо моделиране в текстилната техника

Резултатите от тази глава намират отражение в публикациите [27, 29, 131, 172, 174, 190, 192].

В настоящата глава ще опишем още един съдържателен пример на задача, при решаването на която е уместно използуването на побитови операции с цел улесняване на решаването и повишаване на ефективността на съответния алгоритъм. По този начин ще подпомогнем работата на преподавателя по програмиране в усилията си на търсене на подходящи примери за демонстрация на побитовите операции в програмирането с езиците C/C++. От друга страна описаните алгоритми имат добро практическо приложение при решаването на една известна комбинаторна задача свързана с класификацията на различните текстилни структури.

2.1. Един пример за използване на побитовите операции в програмирането

2.1.1. Постановка на задачата

Да означим с $\mathfrak{B}_n$ множествата на всички $n \times n$ бинарни матрици, т.е. матрици с n реда и n стълба, всички елементи на които са равни на 0 или 1. Известен факт е, че броят на всички такива матрици е равен на 2^{n^2}. Нека $A, B \in \mathfrak{B}_n$. Ще казваме, че A и B са еквивалентни и ще пишем $A \sim B$, ако B се получава от A след последователно циклично преместване на последния ред или стълб на първо място като при това първият ред (съответно стълб) става втори, втория - трети и т.н., предпоследния ред (съответно стълб) става последен. Не е трудно да се види, че така описаната релация е действително релация на еквивалентност. Така стигаме до формулировката на следната задача по програмиране:

Задача 2.1.1. *Да се състави програма, която при зададено цяло положително число n да дава по един екземпляр от всеки клас на еквивалентност в $\mathfrak{B}_n$ относно гореописаната релация.*

Като следствие от задача 2.1.1 ще решим и комбинаторната задача за намиране на броя на всички класове на еквивалентност в $\mathfrak{B}_n$ по отношение на релацията $\sim$, т.е. за намиране на мощността на фактор множеството $\mathfrak{B}_{n/\sim}$.

Така поставената задача има приложение в тъкачната промишленост. С помощта на елементите на $\mathfrak{B}_n$ могат да бъдат кодирани различните преплитания на нишките на дадена тъкачна структура, като при тази кодировка с помощта на две еквивалентни матрици се кодира на практика изтъкаването на един и същи плат поради цикличността на повторенията на преплитанията [10, 27, 29, 60, 192].

От практическа гледна точка значение имат само тези матрици, които имат на всеки ред и на всеки стълб поне по една 0 и поне по една 1. Да означим с $\mathcal{Q}_n$ множеството от всички такива матрици, $\mathcal{Q}_n \subset \mathfrak{B}_n$. Следващата задача, която всъщност ще решим е усложнен вариант на задача 2.1.1.

Задача 2.1.2. *Да се състави програма, която при зададено цяло положително число n да дава по един екземпляр от всеки клас на еквивалентност в $\mathcal{Q}_n$ относно гореописаната релация.*

2.1.2. Реализация на алгоритъма

Всяка $n \times n$ бинарна матрица A може да се кодира с помощта на n-мерния вектор от цели неотрицателни числа $v = (v_0, v_1, \ldots, v_{n-1})$, където $0 \leq v_i \leq 2^n - 1$, $0 \leq i \leq n-1$. Взаимно еднозначното съответствие се осъществява посредством двоичното представяне на естествените числа, т.е. i-ят ред на матрицата A представлява двоичното представяне на v_i. Редът с номер i на A ще бъде изцяло нулев тогава и само тогава, когато $v_i = 0$, а всички елементи на i-ят ред на A ще бъдат равни на 1 тогава и само тогава когато $v_i = 2^n - 1$.[1] С други думи за да има във всеки ред поне една нула и поне една единица е необходимо и достатъчно за всяко $i = 0, 1, \ldots, n-1$ да е изпълнено $1 \leq v_i \leq 2^n - 2$. За да има поне една нула във всеки стълб на матрицата A е необходимо и достатъчно побитовото AND на всички числа, представляващи редовете на A да бъде равно на 0. За да има поне една единица във всеки стълб на матрицата A е необходимо и достатъчно побитовото OR на всички числа, представляващи редовете на A да бъде равно на $2^n - 1$, т.е. на числото, което се записва в двоична бройна система с точно n на брой единици и нито една значеща нула. Така получаваме следната функция, която проверява дали с помощта на n-торката от цели числа $v = (v_0, v_1, \ldots, v_n)$ се кодира матрица от $\mathcal{Q}_n$, или не.

Програмен код 2.1.3. *Проверява дали целочисленият n-мерен масив $v[n] = \{v[0], v[1], \ldots 1]\}$ представя матрица от $\mathcal{Q}_n$.*

```
int IsQn(unsigned int v[], unsigned int n) {
    // Връща 1, ако с v се кодира матрица от Qn
    // Връща 0, в противен случай
for (int i=0; i <= n-1; i++)
    if  (v[i]<1 || v[i] > (1<<n)-2) return 0;
int x,y;
x = (1<<n) -1;
y=0;
for (int j=0; j <= n-1; j++) {
    x = x & v[j];
    y = y | v[j];
}
if (x != 0) return 0;
if (y != (1<<n)-1) return 0;
return 1;
}
```

Разглеждаме множеството $V = \{(v_0, v_1, \ldots, v_{n-1}) \mid 0 \leq v_i \leq 2^n - 1, i = 0, 1, \ldots, n-1\}$ съставено от всевъзможните разглеждани от нас n-торки от цели неотрицателни числа, представящи съответните редове на дадена бинарна матрица. Всички елементи на V могат да бъдат сравнявани относно лексикографската наредба. Същността на предлагания от нас алгоритъм е последователно да получим всички елементи на V при това сортирани във възходящ ред, т.е. от най-малкия до най-големия относно лексикографската наредба и веднага след получаването им да проверим дали този елемент е минимален в класа на еквивалентност на който принадлежи. В крайна сметка ще отсеем единствено минималните за класа

[1] Виж пример 1.1.2 на стр. 10.

на еквивалентност елементи и това ще бъдат единствените представители на всеки клас на еквивалентност в множествата $\mathfrak{B}_n$ и $\mathcal{Q}_n$ (което се изисква и от задачи 2.1.1 и 2.1.2). За целта ще създадем функция IsMin, която да връща стойност 1, ако подаваният аргумент е минимален в класа на класа на еквивалентност, на който принадлежи и 0 в противен случай. Но преди това се нуждаем от следната помощна функция CicleMove, която по зададени цели неотрицателни числа x и n връща число, което се получава от x преместваики всички битове с един надясно като най-десният преди това бит отиде на мястото на бит с номер $n-1$ (най-десният бит има номер 0). Като страничен ефект на предлаганата от нас функция (което не пречи на нашите цели) е нулирането на всички битове на ляво от този с номер $n-1$. В случая отново ще се възползуваме от услугите на побитовите операции.

Програмен код 2.1.4. *Премества с една позиция на дясно на битовете на цялото число x, което се записва с не повече от n цифри при двоичното му представяне. Най-десният бит се премества n позиции в ляво.*

```
unsigned int CicleMove(unsigned int x, unsigned int n) {
    unsigned int b0 = x & 1;  /* Запомня стойността
                                 на най-десния бит на x */
    x=x & ((1<<n)-1);
    return (x >> 1)  |  (b0 << n-1);
}
```

Следващата помощна функция също ще ни бъде полезна при решаването на главната задача:

Програмен код 2.1.5. *Връща 1, ако $[u_0, u_1, \ldots, u_{n-1}] < [v_0, v_1, \ldots, v_{n-1}]$ спрямо лексикографската наредба и 0, в противен случай.*

```
int IsLess (unsigned int u[],unsigned int v[], int n) {
    int i = 0;
    while ((u[i] == v[i]) && (i<n-1)) i++;
    if (u[i] < v[i]) return 1;
        else return 0;
}
```

Споменатата по-горе функция IsMin би могла да има следния вид:

Програмен код 2.1.6. *Проверка за минималност. Връща 1, ако съгласно лексикографската наредба $[v_0, v_1, \ldots, v_{n-1}]$ е минимален в своя клас на еквивалентност и 0, в противен случай.*

```
int IsMin(unsigned int v[], unsigned int n) {
unsigned int u[32], v1[32];
for (int i = 0; i <= n-1; i++)    v1[i] = v[i];
for (int j = 0; j <= n-1; j++) {
    for (int i = 0; i <= n-1; i++) {
        for (int s = 0; s <= n-1; s++) {
            int s1 = (s+i) % n;
            u[s] = v1[s1];
        }
```

```
    if (IsLess(u,v,n) ) return 0;
    }
    for (int i=0; i <= n-1; i++)   v1[i]=CicleMove(v1[i],n);
    }
return 1;
}
```

Възползвайки се от описаните по-горе функции предлагаме следното решение на задачи 2.1.1 и 2.1.2 (например за n=4).

Програмен код 2.1.7. *Главна програма.*

```
int main() {
const int n=4;
int i;
unsigned long int NBn = 0;  // Брой елементи в Bn
unsigned long int NQn = 0;  // Брой елементи в Qn
unsigned long int NBnEq = 0; // Брой класове на еквивалентност в Bn
unsigned long int NQnEq = 0; // Брой класове на еквивалентност в Qn
unsigned int v[n];
int r=(1<<n)-1;

for (i = 0; i<n; i++)  v[i]=0;

do {
    i=n-1;

    for (int k=0; k<=r; k++) {
        v[i]=k;
        NBn++;
        if (IsQn(v,n) ) NQn++;
        if (IsMin(v,n) ) {
            NBnEq++;
            if (IsQn(v,n))   NQnEq++;
        }
    }
    while (v[i]==r) i--;
    if (i>=0) {
        v[i]++;

        for (int k=n-1; k>i; k--) {
            v[k]=0;
        }

    }

} while ( i>=0 );

cout<<"Брой елементи в Bn "<<NBn<<'\n';
cout<<"Брой елементи в Qn "<<NQn<<'\n';
cout<<"Брой класове на еквивалентност в Bn "<<NBnEq<<'\n';
```

```
cout<<"Брой класове на еквивалентност в Qn "<<NQnEq<<'\n';

return 0;
}
```

В програмен код 2.1.7 за краткост не е описано извеждането на всички получени елементи, а е получен само техният брой. За самото извеждане на отделните редове на всяка от получените матрици можем да се възползуваме от описаната по-долу функция `BinPrn` (програмен код 2.1.8) и след организиране на цикъл по номера на реда да изведем и цялата матрица.

Програмен код 2.1.8. *Нека x е цяло число, за което сме сигурни, че принадлежи на интервала $0 \leq x \leq 2^n - 1$, т.е. за двоичния му запис не са необходими повече от n цифри 0 или 1. Тогава за представянето на x в двоична бройна система, записано с помощта на точно n цифри 0 или 1 и евентуално с известен брой незначещи нули в началото можем да използуваме следната функция:*[2]

```
void BinPrn(int x, unsigned int n) {
   int z;
   for (int i = n-1; i >= 0; i--) {
z = x & (1<<i);
if (z == 0) cout<<'0';
else cout<<'1';
}
}
```

2.2. Математическо моделиране на някои класове от тъкачни структури

2.2.1. Постановка на задачата

Както подчертахме в раздел 2.1.1 преплитането на нишките в дадена тъкачна структура може да се кодира с помощта на квадратна *бинарна* матрица, т. е. матрица, всички елементи на която са равни на 0 или 1. Задължително условие е във всеки ред и всеки стълб на матрицата да има поне една нула и поне една единица за да може да съществува тъкан, представен с помощта на тази матрица. При това две различни матрици ще съответстват на една и съща тъкачна структура, тогава и само тогава, когато едната матрица се получава от другата след няколко последователни циклични премествания на първия ред или стълб на последно място.

Както и в раздел 2.1 с $\mathfrak{B}_n$ ще означаваме множеството на всички квадратни $n \times n$ бинарни матрици, а с $\mathcal{Q}_n$ множеството от всички $n \times n$ бинарни матрици, имащи поне една единица и поне една нула на всеки ред и всеки стълб. Очевидно $\mathcal{Q}_n \subset \mathfrak{B}_n$. По отношение на необходимите дефиниции и означения от теория на матриците ще се съобразяваме с [34, 55, 66, 139]. Не е трудно да се види, че

$$|\mathfrak{B}_n| = 2^{n^2} \tag{2.2.1}$$

[2] Сравнете описаната в този програмен код функция `BinPrn` с функцията `DecToBin` (пример 1.1.4) описана в глава 1 на страница 11.

Ако $A = [a_{ij}]_{n\times n} \in \mathfrak{B}_n$, то с $A^T = [a_{ji}]_{n\times n}$, $1 \leq i,j \leq n$ ще означаваме *транспонираната* на A матрица.

Интерес представлява подмножеството $\mathcal{P}_n \subset \mathcal{Q}_n$ съставено от всички *пермутационни* матрици, т.е. бинарни матрици в които на всеки ред и всеки стълб има точно по една единица. Както е добре известно [34, 66], множеството $\mathcal{P}_n$ заедно с операцията умножение на матрици е група, изоморфна на симетричната група $\mathcal{S}_n$, където множеството

$$\mathcal{S}_n = \left\{ \begin{pmatrix} 1 & 2 & \cdots & n \\ i_1 & i_2 & \cdots & i_n \end{pmatrix} \middle| \, i_k \neq i_l \text{ for } k \neq l,\ k,l,i_k,i_l \in [n], \right\} \tag{2.2.2}$$

се състои от всички взаимно-еднозначни изображения на елементите на множеството $[n] = \{1,2,\ldots,n\}$ в себе си. При това, ако $M \in \mathcal{P}_n$ и образът на M при този изоморфизъм е $\begin{pmatrix} 1 & 2 & \cdots & n \\ i_1 & i_2 & \cdots & i_n \end{pmatrix} \in \mathcal{S}_n$, то това е равносилно на условието единствената единица на първия ред на M да е на i_1-во място, единицата на втория ред на M да е на i_2-ро място, и така нататък, единицата на n-я ред на M да е на i_n-то място.

Нека $t \in \{1,2,\ldots,n\}$ и нека $\rho = \begin{pmatrix} 1 & 2 & \cdots & t & \cdots & n \\ i_1 & i_2 & \cdots & i_t & \cdots & i_n \end{pmatrix} \in \mathcal{S}_n$. Тогава образа i_t на числото t при изображението ρ ще означаваме с $i_t = \rho(t)$.

Съобразявайки се с дефинициите и означенията, използвани в [34] и [66] имаме:

$$\begin{pmatrix} 1 & 2 & \cdots & t & \cdots & n \\ i_1 & i_2 & \cdots & i_t & \cdots & i_n \end{pmatrix} \begin{pmatrix} 1 & 2 & \cdots & t & \cdots & n \\ j_1 & j_2 & \cdots & j_t & \cdots & _n \end{pmatrix} =$$

$$= \begin{pmatrix} i_1 & i_2 & \cdots & i_t & \cdots & i_n \\ j_1 & j_2 & \cdots & j_t & \cdots & _n \end{pmatrix}$$

С други думи, ако $\rho, \rho_1, \rho_2 \in \mathcal{S}_n$ са такива, че $\rho = \rho_1\rho_2$, то $\rho(i) = \rho_2(\rho_1(t))$ за всяко $t \in [n]$, т.е приемаме, че образът на даден елемент от $[n]$ при произведението на две изображения се получава "от ляво на дясно".

Както е известно [66], ако умножим произволна $n \times n$ матрица A отдясно с произволна пермутационна матрица $M \in \mathcal{P}_n$, то това е равносилно на разместване на стълбовете на A. При това, ако съответният елемент на $M \in \mathcal{P}_n$ при описания по-горе изоморфизъм е $\begin{pmatrix} 1 & 2 & \cdots & n \\ i_1 & i_2 & \cdots & i_n \end{pmatrix} \in \mathcal{S}_n$, то след умножението ще получим матрица с k-ти стълб равен на i_k-я стълб на A, $k = 1,2,\ldots,n$. Аналогично разместване на редовете ще получим, ако A умножим отляво с M^T.

Единичен елемент на групата $\mathcal{P}_n$ е единичната матрица E_n с единици по главния диагонал и нули навсякъде другаде. Единичен елемент на групата $\mathcal{S}_n$ е елемента $\varepsilon_n = \begin{pmatrix} 1 & 2 & \cdots & n \\ 1 & 2 & \cdots & n \end{pmatrix}$.

Нека да припомним, че бинарните $n \times n$ матрици A и B са еквивалентни и ще пишем $A \sim B$, ако едната матрица се получава от другата след няколко последователни циклични премествания на първия ред или стълб на последно място. С други думи, ако $A, B \in \mathcal{Q}_n$ и $A \sim B$, то с помощта на двете матрици се кодира една и съща тъкачна структура (плат). Класът на еквивалентност по отношение на релацията $\sim$ с представител матрицата A ще означаваме с $\overline{A}$, а множествата от класовете на еквивалентност в $\mathfrak{B}_n$ и $\mathcal{Q}_n$ (фактор множествата) по

отношение на $\sim$, съответно с $\overline{\mathfrak{B}_n}$ и $\overline{\mathcal{Q}_n}$. Считаме, че $\overline{\mathfrak{B}_n}$ и $\overline{\mathcal{Q}_n}$ са описани, ако е зададен по един произволен представител на всеки клас на еквивалентност.

Класовете на еквивалентност в $\mathfrak{B}_n$ относно релацията на еквивалентност $\sim$ са частен случай на *двойни съседни класове* (виж [37, т.1,гл.2,§1.1] [97],[118, §1.7], [165]). Те имат голямо приложение в теория на групите от субституции (виж [126, §1.12,§2.6]) и при линейното представяне на крайните групи (виж [101, §§44÷45], [159])).

Дефиниция 2.2.1. *Елементите на* $\overline{\mathcal{Q}_n}$ *ще наричаме* ***сплитки****. В случая числото* n *се нарича* ***повтор*** *на сплитките от* $\overline{\mathcal{Q}_n}$.

Термините в дефиниция 2.2.1 са взаимствани от науката *сплиткознание*, която изучава строежа, дизайна и физикомеханичните свойства вследствие на различните преплитания на нишките след изтъкаването на дадена текстилна структура (плат).

По естествен начин възникват редица комбинаторни задачи с практическа насоченост в тъкачната промишленост, свързани с различни подмножества на $\overline{\mathcal{Q}_n}$, т. е. с различни класове от сплитки. Някои от тези класове са предмет на разглеждане в следващия раздел.

2.2.2. За някои класове от сплитки.

Лесно се вижда, че ако $A \in \mathcal{P}_n$ и $B \sim A$, то и $B \in \mathcal{P}_n$. Сплитки, чийто представители са елементи на множеството $\mathcal{P}_n$ от всички пермутационни матрици се наричат *първични* сплитки. Формула и алгоритъм за изчисляване броя на всички първични сплитки при произволен повтор n са дадени в [29, 172]. В тези разработки при описание на алгоритъма е използуван апарат от теория на графите.

Разглеждаме матрицата

$$P = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} = [p_{i\,j}] \in \mathcal{P}_n, \tag{2.2.3}$$

където $p_{1\,2} = p_{2\,3} = \cdots = p_{i\,i+1} = \cdots = p_{n-1\,n} = p_{n\,1} = 1$ и това са единствените единици в P, а всички останали елементи са равни на нула. При описания по-горе изоморфизъм на групата от пермутационни матрици със симетричната група, матрицата P ще съответства на елемента

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \cdots & n-1 & n \\ 2 & 3 & 4 & \cdots & n & 1 \end{pmatrix} \in \mathcal{S}_n. \tag{2.2.4}$$

Не е трудно да се пресметне, че ако E_n е единичната $n \times n$ матрица, то

$$P^t \neq E_n \quad \text{при } 1 \leq t < n, \tag{2.2.5}$$

$$P^n = E_n \tag{2.2.6}$$

и

$$P^{t+n} = P^t \tag{2.2.7}$$

за всяко естествено число t.

Нека $A \in \mathfrak{B}_n$ и нека

$$B = PA \tag{2.2.8}$$

и

$$C = AP. \tag{2.2.9}$$

Непосредствено се доказва [66], че първият ред на B е равен на втория ред на A, вторият ред на B е равен съответно на третия на A и т. н., последният ред на B е равен на първия ред на A, т. е. матрицата B се получава от матрицата A чрез преместване на първия ред на последно място, а останалите редове се преместват с един ред нагоре.

Аналогично се убеждаваме, че C се получава от A чрез преместване на последния стълб на първо място, а останалите стълбове се преместват с един надясно.

Вземайки в предвид всичко казано по-горе, лесно се доказва следната

Лема 2.2.2. *Нека $A, B \in \mathfrak{B}_n$. Тогава $A \sim B$, тогава и само тогава, когато съществуват естествени числа k, l, такива че*

$$A = P^k B P^l,$$

където P е матрицата, зададена с помощта на формула (2.2.3). От формули (2.2.7), (2.2.5) и (2.2.6) следва, че числата k и l е достатъчно да се търсят в интервала $[0, n-1]$.

□

Следствие 2.2.3. *Всички елементи на даден клас на еквивалентност на $\mathfrak{B}_n$ относно релацията $\sim$ могат да бъдат разположени в правоъгълна таблица с размери $s \times t$, където s и t са делители на n.*

□

Следствие 2.2.4. *Всеки клас на еквивалентност на $\mathfrak{B}_n$ относно релацията $\sim$ съдържа не повече от n^2 елемента.*

□

Освен добре известната от линейната алгебра операция транспониране на матрица, тук ще въведем още няколко подобни матрични операции.

Нека

$$A = \begin{bmatrix} a_{1\,1} & a_{1\,2} & \cdots & a_{1\,n} \\ a_{2\,1} & a_{2\,2} & \cdots & a_{2\,n} \\ \vdots & \vdots & & \vdots \\ a_{n\,1} & a_{n\,2} & \cdots & a_{n\,n} \end{bmatrix} \in \mathcal{B}_n \tag{2.2.10}$$

Ако A е представената с помощта на формула (2.2.10) квадратна бинарна матрица, то по дефиниция

$$A^S = \begin{bmatrix} a_{1\,n} & a_{1\,n-1} & \cdots & a_{1\,1} \\ a_{2\,n} & a_{2\,n-1} & \cdots & a_{2\,1} \\ \vdots & \vdots & & \vdots \\ a_{n\,n} & a_{n\,n-1} & \cdots & a_{n\,1} \end{bmatrix}, \tag{2.2.11}$$

т. е. A^S се получава от A като последният стълб става първи, предпоследният - втори и т.н., първият стълб става последен. С други думи, ако $A = [a_{i\,j}]_{n \times n}$, то $A^S = [a_{i\,n-j+1}]_{n \times n}$, $1 \leq i, j \leq n$.

Очевидно е, че

$$\left(A^S\right)^S = A.$$

Дефиниция 2.2.5. *Ще казваме, че матрицата $A \in \mathfrak{B}_n$ е* ***огледална*** *на матрицата $B \in \mathfrak{B}_n$, ако $A^S = B$, където A^S е операцията, дефинирана с помощта на формули (2.2.10) и (2.2.11).*

Лесно се вижда, че ако матрицата A е огледална на матрицата B, то B е огледална на A, т. е. релацията "огледалност" е симетрична.

В общия случай $A \neq A^S$. Освен това, ако $A = B^S$ и $B = C^S$, то в общия случай ще имаме $A = B^S = \left(C^S\right)^S = C \neq C^S$. Следователно релацията "огледалност" не е рефлексивна и не е транзитивна.

Разглеждаме матрицата

$$S = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} = [s_{ij}] \in \mathcal{P}_n, \tag{2.2.12}$$

където за всяко $i = 1, 2, \ldots, n$ $s_{i\,n-i+1} = 1$ и $s_{ij} = 0$ при $j \neq n-i+1$. При описания по-горе изоморфизъм на групата от пермутационни матрици със симетричната група, матрицата S ще съответства на елемента

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ n & n-1 & n-2 & \cdots & 1 \end{pmatrix} \in \mathcal{S}_n. \tag{2.2.13}$$

Очевидно S е симетрична матрица, т.е. $S^T = S$. Непосредствено се проверява, че $S^2 = E_n$, където E_n е единичната $n \times n$ матрица.

Не е трудно да се забележи, че за всяко $A \in \mathfrak{B}_n$ е изпълнено

$$A^S = AS. \tag{2.2.14}$$

Лема 2.2.6. *Ако P и S са матриците зададени с формули съответно (2.2.3) и (2.2.12), то за всяко $l = 0, 1, 2, \ldots, n-1$ е в сила:*

$$P^l S = S P^{n-l} \tag{2.2.15}$$

Доказателство. Да означим с $\oplus$ и с $\ominus$ операциите съответно събиране и изваждане в пръстена $\mathbb{Z}_n = \{1, 2, \ldots, n \equiv 0\}$ на остатъците по модул n. За да има съответствие с приетите от нас означения на елементите от $\mathcal{S}_n$ и тъй като $0 \equiv n \pmod{n}$, то нулевият елемент в $\mathbb{Z}_n$ ще означаваме с n (вместо с 0). Ако $\pi \in \mathcal{S}_n$ и $\sigma \in \mathcal{S}_n$ са елементи, съответстващи на матриците $P \in \mathcal{P}_n$ и $S \in \mathcal{P}_n$ при изоморфизма на групите $\mathcal{P}_n$ и $\mathcal{S}_n$, описани съгласно формули (2.2.4), (2.2.13), (2.2.3) и (2.2.12), то лесно се проверява, че за всяко $t = 1, 2, \ldots, n$ е изпълнено:

$$\pi(t) = t \oplus 1 \tag{2.2.16}$$

$$\sigma(t) = n \oplus 1 \ominus t \tag{2.2.17}$$

По индукция ще докажем, че всяко цяло положително число l е изпълнено:

$$\pi^l(t) = t \oplus l \tag{2.2.18}$$

И наистина при $l = 1$, твърдението следва от (2.2.16). Допускаме, че е в сила равенството $\pi^l(t) = t \oplus l$. Тогава получаваме $\pi^{l+1}(t) = \pi(\pi^l(t)) = \pi(t \oplus l) = t \oplus l \oplus 1$,

откъдето следва, че равенството (2.2.18) е вярно за всяко цяло положително число l.

Използувайки равенства (2.2.16), (2.2.17) и (2.2.18) последователно получаваме:

$$\pi^l\sigma(t) = \sigma(\pi^l(t)) = \sigma(t \oplus l) = n \oplus 1 \ominus (t \oplus l) = n \oplus 1 \ominus t \ominus l$$

$$\sigma\pi^{n-l}(t) = \pi^{n-l}(\sigma(t)) = \pi^{n-l}(n \oplus 1 \ominus t) = (n \oplus 1 \ominus t) \oplus (n - l) =$$

$$= 2n \oplus 1 \ominus t \ominus l = n \oplus 1 \ominus t \ominus l$$

Последното равенство е в сила, тъй като $2n \equiv n \equiv 0 \pmod{n}$. Виждаме, че $\pi^l\sigma(t) = \sigma\pi^{n-l}(t)$ за всяко $t = 1, 2, \ldots, n$ и следователно $\pi^l\sigma = \sigma\pi^{n-l}$. Вземайки в предвид изоморфизма на групите $\mathcal{S}_n$ и $\mathcal{P}_n$ следва и твърдението на лемата.

□

Теорема 2.2.7. *Ако $A \sim A^S$ и $B \sim A$ то $B \sim B^S$.*

Доказателство. Тъй като $B \sim A$, то съгласно лема 2.2.2 съществуват $k, l \in \{1, 2, \ldots, n\}$, такива че $B = P^kAP^l$. Прилагайки лема 2.2.6 получаваме $BS = P^kAP^lS = P^kASP^{n-l}$, откъдето следва, че $BS \sim AS$, т.е. съгласно (2.2.14) $B^S \sim A^S$. Но $A^S \sim A \sim B$ и поради транзитивността на релацията $\sim$ получаваме $B^S \sim B \Rightarrow B \sim B^S$.

□

Теорема 2.2.7 ни дава основание да дадем следната

Дефиниция 2.2.8. *Нека $A \in \mathcal{Q}_n$. Ще казваме, че A е представител на* **самоогледална** *(или* **огледална сама на себе си***) сплитка, ако $A \sim A^S$.*

Да означим с

$$\overline{\mathcal{M}_n} \subset \overline{\mathcal{Q}_n}$$

множеството от всички самоогледални сплитки с повтор равен на n.

Пример 2.2.9.

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \; A_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \in \overline{\mathcal{M}_5} \tag{2.2.19}$$

са представители на две различни самоогледални сплитки с повтор равен на 5.

Ако $A \in \mathfrak{B}_n$ е представената с помощта на формула (2.2.10) квадратна бинарна матрица, то дефинираме операцията

$$A^R = \begin{bmatrix} a_{1\,n} & a_{2\,n} & \cdots & a_{n\,n} \\ a_{1\,n-1} & a_{2\,n-1} & \cdots & a_{n\,n-1} \\ \vdots & \vdots & & \vdots \\ a_{1\,1} & a_{2\,1} & \cdots & a_{n\,1} \end{bmatrix} \tag{2.2.20}$$

С други думи матрицата A^R се получава от матрицата A след завъртане на $90°$ по посока обратна на часовниковата стрелка.

Лесно се вижда, че

$$A^R = \left(A^S\right)^T = (AS)^T = S^T A^T = SA^T \tag{2.2.21}$$

Очевидно

$$\left(\left(\left(A^R\right)^R\right)^R\right)^R = A.$$

В общия случай $A^R \neq A$.

Лема 2.2.10. *Ако P е дефинираната с помощта на формула (2.2.3) бинарна матрица, то*

$$P^T = P^{n-1}$$

Доказателство. Ако $P = [p_{ij}]_{n\times n}$ и $P^T = [p'_{ij}]_{n\times n}$, то по дефиниция $p'_{ij} = p_{ji}$ за всеки $i, j \in \{1, 2, \dots n\}$. Нека $PP^T = Q = [q_{ij}]_{n\times n} \in \mathcal{P}_n$. Тогава за всяко $i = 1, 2, \dots, n$ имаме $q_{ii} = \sum_{k=1}^{n} p_{ik}p'_{ki} = \sum_{k=1}^{n} p_{ik}^2 = (n-1)0 + 1 = 1$ и това ще бъде единствената единица на i-ти ред на матрицата $Q = PP^T$. Следователно $PP^T = E_n$, където E_n е единичната $n \times n$ матрица. Умножаваме отляво двете страни на последното равенство с P^{n-1} и имайки в предвид, че $P^n = E_n$, то получаваме $P^{n-1}PP^T = P^{n-1}E_n$, откъдето окончателно получаваме, че $P^T = P^{n-1}$.

□

Теорема 2.2.11. *Ако $A \sim A^R$ и $B \sim A$ то $B \sim B^R$.*

Доказателство. $B \sim A$, откъдето съгласно лема 2.2.2 съществуват естествени числа k и l, такива че $B = P^k AP^l$. Тогава, прилагайки формула (2.2.21), лема 2.2.6 и лема 2.2.10 последователно получаваме $B^R = SB^T = S(P^k AP^l)^T = S(P^T)^l A^T (P^T)^k = S(P^{n-1})^l A^T (P^{n-1})^k = SP^{nl-l}A^T P^{kn-k} = SP^{n-l}A^T P^{n-k} = P^l SA^T P^{n-k} = P^l A^R P^{n-k} \sim A^R$. Следователно $B^R \sim A^R \sim A \sim B$.

□

Теорема 2.2.11 ни дава основание да дадем следната

Дефиниция 2.2.12. *Нека $A \in \mathcal{Q}_n$. Ако $A \sim A^R$, то ще казваме, че A е представител на* **ротационно устойчива** *сплитка.*

Да означим с

$$\overline{\mathcal{R}}_n \subset \overline{\mathcal{Q}}_n$$

множеството от всички ротационно устойчиви сплитки с повтор равен на n.

Пример 2.2.13.

$$A_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}, \; A_4 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \in \overline{\mathcal{Q}_5} \tag{2.2.22}$$

са представители на две различни ротационно устойчиви сплитки с повтор равен на 5.

Ротационно устойчивите сплитки играят важна роля в практиката. Това означава, че ако бъде изтъкан плат, чиято тъкачна структура се кодира с матрица, представител на ротационно устойчива сплитка, то този плат ще има същите експлоатационни характеристики (изключвайки разбира се цветовите) след завъртане на 90°.

2.3. Алгоритми за изследване на структурното многообразие на тъкачните сплитки със средствата на обектно-ориентираното програмиране

2.3.1. Постановка на задачата

Разглеждаме множеството $\mathfrak{B} = \{0, 1\}$. $\mathfrak{B}$ заедно с операциите конюнкция &&, дизюнкция $\|$ и отрицание ! образуват известната *булева алгебра* $\mathfrak{B}(\&\&, \|, !)$, играеща важна роля в компютърната техника и програмирането. Ще използваме посочените означения за операциите в $\mathfrak{B}(\&\&, \|, !)$, съобразявайки се със синтаксиса на аналогичните операции в широко разпространените в последно време езици за програмиране C, C++ и Java.

Нека $A = [a_{ij}]_{n\times n}$ и $B = [b_{ij}]_{n\times n}$ са матрици от $\mathfrak{B}_n$ (множеството на всички квадратни $n \times n$ бинарни матрици). Тук и по-надолу до края на тази глава отново ще се съобразим със синтаксиса на езиците C++ и Java и индексите ще започват от 0, т.е. $i, j \in \{0, 1, 2, \ldots, n-1\}$. Разглеждаме следните операции в $\mathfrak{B}_n$, дефинирани съобразно нашите цели както следва:

покомпонентна конюнкция

$$A \,\&\&\, B = C = [c_{ij}]_{n\times n} \tag{2.3.1}$$

където по дефиниция за всеки $i, j \in \{0, 1, 2 \ldots, n-1\}$

$$c_{ij} = a_{ij} \,\&\&\, b_{ij}.$$

покомпонентна дизюнкция

$$A \,\|\, B = C = [c_{ij}]_{n\times n} \tag{2.3.2}$$

където по дефиниция за всеки $i, j \in \{0, 1, 2 \ldots, n-1\}$

$$c_{ij} = a_{ij} \,\|\, b_{ij}.$$

покомпонентно отрицание

$$!A = C = [c_{ij}]_{n\times n} \tag{2.3.3}$$

където по дефиниция за всеки $i, j \in \{0, 1, 2 \ldots, n-1\}$

$$c_{ij} = !a_{ij}.$$

транспониране

$$t(A) = C = [c_{ij}]_{n\times n} \tag{2.3.4}$$

където по дефиниция за всеки $i, j \in \{0, 1, 2 \ldots, n-1\}$

$$c_{ij} = a_{ji}.$$

логическо произведение

$$A * B = C = [c_{ij}]_{n\times n} \tag{2.3.5}$$

където по дефиниция за всеки $i, j \in \{0, 1, 2 \ldots, n-1\}$

$$c_{ij} = \bigvee_{k=0}^{n-1} (a_{ik} \,\&\&\, b_{kj}) =$$

$$= (a_{i\,0}\ \&\&\ b_{0\,j})\ \|\ (a_{i\,1}\ \&\&\ b_{1\,j})\ \|\ \cdots\ \|\ (a_{i\,n-1}\ \&\&\ b_{n-1\,j}).$$

Не е трудно да се забележи, че така въведените по-горе бинарни операции са асоциативни.

По такъв начин $\mathfrak{B}_n$ заедно с описаните по-горе операции $\&\&$, $\|$, $!$, $t()$ и $*$ образуват алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$. Тук и навсякъде в дисертационния труд под *алгебра* ще разбираме понятието *абстрактна алгебра* съгласно дефиницията дадена в [104], а именно множество заедно с някои операции над елементите от множеството, изпълняващи система от аксиоматични закони.

В $\mathfrak{B}_n(\&\&, \|, !, t(), *)$ по естествен начин може да бъде въведена наредба, а именно лексикографската наредба.

Разглеждаме следното множество от стандартни операции с цели числа в езиците за програмиране C++ и Java:

$$\mathcal{O} = \{\texttt{+,-,*,/,\% ,<<,>>,\&,|,\^{},\~{},\&\&,||,!,if ,<,<=,>,>=,==,!=}\} \tag{2.3.6}$$

изразяващи съответно събиране, изваждане, умножение, деление, целочислено деление, побитово преместване наляво и надясно, побитово "и", побитово "или", побитово "изключващо или", побитово "не", дизюнкция, конюнкция, отрицание, проверка на условие и операциите за сравнение. Ще считаме, че необходимите времена за изпълнението на всяка една от операциите от множеството $\mathcal{O}$ са съпоставими, т.е. ако t_1 и t_2 са съответно времената за изпълнение на произволни две операции от $\mathcal{O}$, то $t_1 = Ct_2$, където C е константа. Алгоритмите в настоящата работа по отношение на фактора време ще оценяваме съгласно броя на необходимите операции от множеството $\mathcal{O}$.

Един от основните принципи на обектно-ориентираното програмиране е капсулирането, което се изразява във факта, че клиентът знае какви член данни, свойства и методи притежават обектите и на какви събития реагират, при това не е задължително да има информация относно тяхната реализация и по-конкретно не е длъжен да притежава информация за алгоритмите, по които работят член функциите на съответния клас. Тук естествено може да възникне въпросът - ако имаме два различни класа, с които може да се опише един и същи обект от математиката или от реалния свят и които притежават едни и същи свойства и методи, кой от двата класа да предпочетем. Най-често отговорът е тривиален - този клас, за чийто обекти е необходимо по-малко оперативна памет и чийто методи работят по-бързо, т.е. чийто методи в общия случай използуват по-малко на брой стандартни компютърни операции.

Целта на раздел 2.3 е да покажем, че представянето на бинарните матрици с помощта на наредена n-торка цели числа и използуването на побитовите операции водят до реализацията на по-добър клас (по смисъла, посочен по-горе) в сравнение със стандартното представяне на бинарните матрици с помощта на двумерен $n \times n$ целочислен масив.

В следващият подраздел 2.3.2 ще опишем декларацията два класа описващи алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$ имащи еднаква декларация, но различна реализация.

Ще докажем, че използването на побитови операции дава възможност да се реализират по-бързи методи, т.е. води до по-ниска степен на броя на използваните операциите от множеството $\mathcal{O}$, съставено от стандартните операции за езиците за програмиране C, C++ и Java.

2.3.2. Два класа описващи алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$.

За първия клас ще използуваме *стандартното* (общоприето) представяне на бинарните матрици с помощта на двумерен $n \times n$ целочислен масив и последващите

от този факт стандартни алгоритми за изпълнение на операциите (2.3.1) ÷ (2.3.5). Да означим този клас с `Bn_array`.

Една квадратна бинарна $n \times n$ матрица може да бъде представена и с помощта на наредена n-торка от цели неотрицателни числа, принадлежащи на затворения интервал $[0,\ 2^n - 1]$. Еднозначността на представянето следва от еднозначността на представянето на естествените числа в двоична бройна система. Да означим с `Bn_tuple` класът, описващ алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$ по този начин.

Ще направим така, че двата класа да имат еднакви спецификации (при терминологията се съобразяваме с [2, 67–70]), които ще опишем еднократно с името `Bn_X`, т.е. под `X` ще разбираме или `array`, или `tuple` в зависимост от случая.

Нека двата класа, описани общо под името `Bn_X`, да имат следната декларация:

Програмен код 2.3.1. *Спецификация на класа* `Bn_X`*, където* `X` *е или* `array`*, или* `tuple`

```
class Bn_X {
    int n;
    int *Matr;
  public:

/* конструктор без параметър: */
    Bn_X();

/* деструктор: */
    ~Bn_X();

/* конструктор с параметър n указващ реда
                    на квадратната матрица: */
    Bn_X (unsigned int);

/* връща размера (реда) на матрицата: */
    int get_n() { return n; };

/* присвоява стойност 1 на [i,j]-я елемент на матрицата: */
    void set_1 (int,int);

/* нулира [i,j]-я елемент на матрицата: */
    void set_0 (int,int);

/* запълване на i-я ред на матрицата чрез
двоичното представяне на естественото число r
(само за класа Bn_tuple, при "X"="array" да се пропусне!): */
    void set_row(int,int);

/* получаване на [i,j]-я елемент на матрицата: */
    int get_element (int,int);

/* получаване на числото представящо i-я ред на матрицата
(само за класа Bn_tuple, при "X"="array" да се пропусне!): */
    int get_row (int);
```

```
/* транспониране на матрица: */
    Bn_X t (Bn_X);

/* предефиниране на операторите съгласно
(2.3.1), (2.3.2), (2.3.3) и (2.3.5): */
    Bn_X operator && (Bn_X);
    Bn_X operator || (Bn_X);
    Bn_X operator ! ();
    Bn_X operator * (Bn_X);

/* дефиниране на наредба (лексикографска) */
    int operator < (Bn_X);
};
```

Забележка 1. При предефинирането на операторите &&, || и *, ако размерите на двата операнда не са равни, за улеснение ще направим така, че връщаният резултат да е нулева матрица с ред съответстващ на първия операнд, но това няма да бъде истинската стойност на резултата. Нещо повече, в този случай операцията не е дефинирана, т.е. връщаният резултат от операторната функция е недействителен и трябва да внимаваме за подобни ситуации.

Забележка 2. Еднакви и за двата класа `Bn_array` и `Bn_tuple` ще бъдат конструкторът без параметър и деструкторът. Всъщност при работата с обекти от класа `Bn_X`, от математическа и алгоритмична гледна точка е задължително указването размера на матрицата и този размер остава непроменяем. В този аспект използуването на конструктор без параметър е безсмислено и не влиза в нашите интереси. Все пак описваме такъв конструктор за пълнота на изложението:

Програмен код 2.3.2. *Конструктор без параметър.*

```
Bn_X :: Bn_X() {
    n=1;
    Matr = new int[1];
}
```

Програмен код 2.3.3. *Деструктор.*

```
Bn_X :: ~Bn_X() {
    delete [] Matr;
}
```

Забележка 3. За определеност при тестването на програмата сме използували универсалния целочислен тип int, който може да бъде заменен с произволен друг целочислен тип на данните.

Забележка 4. Тъй като обектите от двата класа ще заделят динамично оперативна памет с помощта на указател и оператора `new`, то за да работят приложенията използващи такива обекти е необходимо предефинирането на операциите от "голямата тройка" [2] - конструктора за копиране, деструктора и оператора за присвояване.

2.3.3. Стандартна реализация на класа Bn_array.

При `X` съвпадащ с `array` предлагаме следната (*стандартна*) реализация на методите на класа `Bn_array` (като разбира се включим и описаните по-горе конструктор без параметри и деструктор):

Програмен код 2.3.4. *Конструктор с параметър, указващ реда на квадратната* $n \times n$ *матрица.*

```
Bn_array :: Bn_array (unsigned int k) {
  n=k;
  int s=k*k;
  Matr = new int[s];
  for (int p=0; p<s; p++)
      *(Matr+p) = 0; // началната инициализация
                     // е нулевата матрица
}
```

Програмен код 2.3.5. *Присвоява стойност 1 на* $[i, j]$*-я елемент на матрицата.*

```
void Bn_array :: set_1 (int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
      cout<<"недопустима стойност на параметър \n";
  else *(Matr +i*n+j) = 1;
}
```

Програмен код 2.3.6. *Нулира* $[i, j]$*-я елемент на матрицата*

```
void Bn_array :: set_0 (int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
      cout<<"недопустима стойност на параметър \n";
  else *(Matr +i*n+j) = 0;
}
```

Програмен код 2.3.7. *Получаване на* $[i, j]$*-я елемент на матрицата.*

```
int Bn_array :: get_element (int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
     cout<<"недопустима стойност на параметър \n";
        return *(Matr +i*n+j);
}
```

Програмен код 2.3.8. *Транспониране на матрица.*

```
Bn_array Bn_array :: t (const Bn_array &A) {
  int m=A.get_n();
  Bn_array temp(m);
  for (int i=0; i<m; i++)
     for (int j=0; j<m; j++)
        *(temp.Matr + i*m+j) = *(A.Matr + j*m+i);
  return temp;
}
```

Програмен код 2.3.9. *Предефиниране на оператора* `&&`.

```
Bn_array Bn_array :: operator && (const Bn_array &B) {
  Bn_array temp(n);
  int n2 = n*n;
  if (B.get_n() != n)
    cout<<"недопустима стойност на параметър \n";
  else
    for (int p=0; p<n2; p++)
      *(temp.Matr + p) = *(this->Matr + p) && *(B.Matr + p);
  return temp;
}
```

Програмен код 2.3.10. *Предефиниране на оператора* `||`.

```
Bn_array Bn_array :: operator || (const Bn_array &B) {
  Bn_array temp(n);
  int n2 = n*n;
  if (B.get_n() != n)
    cout<<"недопустима стойност на параметър \n";
  else
    for (int p=0; p<n2; p++)
      *(temp.Matr + p) = *(this->Matr + p) || *(B.Matr + p);
return temp;
}
```

Програмен код 2.3.11. *Предефиниране на оператора* `!`.

```
Bn_array Bn_array :: operator ! () {
  int n2 = n*n;
  for (int p=0; p<n2; p++)
     *(this->Matr + p) = *(this->Matr + p) ? 0 : 1;
  return *this;
}
```

Програмен код 2.3.12. *Предефиниране на оператора* `*`.

```
Bn_array Bn_array :: operator * (const Bn_array &B) {
  Bn_array temp(n);
  int c;
  if (B.get_n() != n)
    cout<<"недопустима стойност на параметър \n";
    else
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++) {
        c=0;
        for (int k=0; k<n; k++)
           c = c||(*(this->Matr + i*n+k) && *(B.Matr +k*n+j));
        *(temp.Matr +i*n+j)=c;
      }
  return temp;
}
```

Програмен код 2.3.13. *Предефиниране на оператора* <.

```
int Bn_array :: operator < (const Bn_array &B) {
  int temp = 1;
  int n2 = n*n;
  if (B.get_n() != n) {
     cout<<"недопустима стойност на параметър \n";
     temp = -1;
  }
  else
    for (int p=0; p<n2; p++)
      if ( *(this->Matr + p) >= *(B.Matr + p) )  {
         temp = 0; break;
      }
  return temp;
}
```

Лесно се убеждаваме във верността на следната теорема:

Теорема 2.3.14. *За всяко цяло положително число n при компютърното представянето на езика за програмиране C++ на алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$ с помощта на класа* `Bn_array` *са в сила следните твърдения:*

(i) При стандартната реализация (стандартното предефиниране) на операторните функции $\&\&$, $\|$, $!$, $<$ и операцията транспониране, всяка една от тях ще извършва по $O(n^2)$ операции от множеството $\mathcal{O}$;

(ii) При стандартната реализация (стандартното предефиниране) на операторната функция $*$, същата ще извършва по $O(n^3)$ операции от множеството $\mathcal{O}$;

(iii) За всеки обект от клас Bn_ array са необходими $O(n^2) * \mathrm{sizeof}\,(\mathrm{int})$ байта оперативна памет;

(iv) При инициализация на обект от клас Bn_ array ще се извършват $O(n^2)$ операции от множеството $\mathcal{O}$.

Доказателството на твърдения (i) и (ii) следва непосредствено от броя на вложените цикли във всеки един от методите и максималния брой на итерациите във всеки един от циклите. Твърдения (iii) и (iv) са очевидни.

□

2.3.4. Реализация на класа Bn_tuple.

Както вече подчертахме по-горе, съществува взаимно-еднозначно съответствие между множеството от всички $n \times n$ бинарни матрици и множеството на всички наредени n-торки от цели числа, принадлежащи на затворения интервал $[0,\ 2^n - 1]$, основаващо се на двоичното представяне на целите числа. Тази идея е заложена и при реализацията на класа `Bn_tuple`.

За класа `Bn_tuple` предлагаме следните методи, включвайки и конструктора без параметри и деструктора, описани в раздел 2.3.2. При тяхната реализация съществена роля ще играят побитовите операции побитова конюнкция `&`, побитова дизюнкция `|`, побитово изключващо "или" `^` и побитово отрицание `~`, използването на които ще повиши ефективността и най-вече бързодействието на заложените в тях алгоритми.

Програмен код 2.3.15. *Конструктор с параметър, указващ реда на квадратната $n \times n$ матрица.*

```
Bn_tuple :: Bn_tuple (unsigned int k) {
  n=k;
  Matr = new int[n];
  for (int p=0; p<n; p++)
     *(Matr+p) = 0; //началната инициализация
                    // е нулевата матрица
}
```

Програмен код 2.3.16. *Присвоява стойност 1 на $[i,j]$-я елемент на матрицата.*

```
void Bn_tuple :: set_1 (int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
     cout<<"недопустима стойност на параметър \n";
  else *(Matr +i) |= 1<<(n-1-j);
}
```

Програмен код 2.3.17. *Нулира $[i,j]$-я елемент на матрицата.*

```
void Bn_tuple :: set_0(int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
     cout<<"недопустима стойност на параметър \n";
  else *(Matr +i) &= ~(1<<(n-1-j));
}
```

Програмен код 2.3.18. *Запълва i-я ред на матрицата чрез двоичното представяне на естественото число* `r`.

```
void Bn_tuple :: set_row (int i,int r) {
  if (i<0 || i>=n || r<0 || r>= (1<<n)
     cout<<"недопустима стойност на параметър \n";
  else *(Matr +i) = r;
}
```

Програмен код 2.3.19. *Получаване на $[i,j]$-я елемент на матрицата.*

```
int Bn_tuple :: get_element(int i,int j) {
  if (i<0 || i>=n || j<0 || j>=n)
     cout<<"недопустима стойност на параметър \n";
  return (*(Matr +i)) & (1<<(n-1-j)) ? 1 : 0;
}
```

Програмен код 2.3.20. *Получава естественото число, което в двоично бройна система дава i-я ред на матрицата.*

```
int Bn_tuple :: get_row (int i) {
  if (i<0 || i>=n )
     cout<<"недопустима стойност на параметър \n";
  return *(Matr +i);
}
```

Програмен код 2.3.21. *Транспониране на матрица.*

```
Bn_tuple Bn_tuple :: t (const Bn_tuple &A) {
  int m=A.get_n();
  Bn_tuple temp(m);
  int k;
  for (int i=0; i<m; i++)
     for (int j=0; j<m; j++) {
        k=A.get_element(i,j);
        if (k) temp.set_1(j,i);
           else temp.set_0(j,i);
    }
  return temp;
}
```

Програмен код 2.3.22. *Предефиниране на оператор* `&&`.

```
Bn_tuple Bn_tuple :: operator && (const Bn_tuple &B) {
  Bn_tuple temp(n);
  if (B.get_n() != n)
     cout<<"недопустима стойност на параметър \n";
  else
    for (int p=0; p<n; p++)
      *(temp.Matr + p) = *(this->Matr + p) && *(B.Matr + p);
  return temp;
}
```

Програмен код 2.3.23. *Предефиниране на оператор* `||`.

```
Bn_tuple Bn_tuple :: operator || (const Bn_tuple &B) {
  Bn_tuple temp(n);
  if (B.get_n() != n)
    cout<<"недопустима стойност на параметър \n";
  else
    for (int p=0; p<n; p++)
      *(temp.Matr + p) = *(this->Matr + p) || *(B.Matr + p);
  return temp;
}
```

Програмен код 2.3.24. *Предефиниране на оператор* `!`.

```
Bn_tuple Bn_tuple :: operator ! () {
  for (int p=0; p<n; p++)
     *(this->Matr + p) = *(this->Matr + p) ? 0 : 1;
  return *this;
}
```

Програмен код 2.3.25. *Предефиниране на оператор* `*`.

```
Bn_tuple Bn_tuple :: operator * (const Bn_tuple &B) {
  Bn_tuple temp(n), TB(n);
  TB = t(B);
```

```
  int c, r_i, r_j;
  if (B.get_n() != n)
     cout<<"недопустима стойност на параметър \n";
     else
     for (int i=0; i<n; i++)
        for (int j=0; j<n; j++) {
          r_i = this->get_row(i);
          r_j = TB.get_row(j);
          c = r_i & r_j;
          if (c==0) temp.set_0(i,j);
               else temp.set_1(i,j);
        }
    return temp;
}
```

Програмен код 2.3.26. *Предефиниране на оператора* <.

```
int Bn_tuple :: operator < (const Bn_tuple &B) {
  int temp = 1;
  if (B.get_n() != n)
     { cout<<"недопустима стойност на параметър \n";
       temp = -1;
     }
     else
        for (int p=0; p<n; p++)
           if ( *(Matr + p) >= *(B.Matr + p) )
              { temp = 0;
                break;
              }
  return temp;
}
```

Аналогично на теорема 2.3.14 се убеждаваме и във верността на

Теорема 2.3.27. *За всяко цяло положително число n при компютърното представяне на езика за програмиране C++ на алгебрата $\mathfrak{B}_n(\&\&, \|, !, t(), *)$ с помощта на класа Bn_ tuple са в сила следните твърдения:*

(i) При описаната по-горе реализация на операторните функции &&, ||, !, >, всяка една от тях ще извършва по $O(n)$ операции от множеството $\mathcal{O}$;

(ii) Операцията транспониране ще извършва $O(n^2)$ операции от множеството $\mathcal{O}$.

(iii) При описаната по-горе реализация на операторната функция $*$, същата ще извършва по $O(n^2)$ операции от множеството $\mathcal{O}$;

(iv) За всеки обект от клас Bn_ tuple са достатъчни $O(n) * \text{sizeof (int)}$ байта оперативна памет;

(v) Инициализация на всеки обект от клас Bn_ tuple е възможно да се осъществи с помощта на $O(n)$ операции от множеството $\mathcal{O}$.

□

Виждаме, че за да създадем класа Bn_array е значително по-леко и не би представлявало проблем дори за начинаещия програмист, като при описание на алгоритмите се съобразяваме основно с дефинициите на съответните операции.

От друга страна сравнявайки теорема 2.3.14 с теорема 2.3.27 се убеждаваме във верността на следното твърдение:

Теорема 2.3.28. *Алгоритмите, които използват обекти от класа Bn_tuple ще бъдат значително по-бързи от алгоритмите, използващи обекти от класа Bn_array със съществена икономия на оперативна памет.*

□

2.4. Количествена оценка на множествата от всички самоогледални и всички ротационно устойчиви сплитки при зададен повтор n.

2.4.1. Постановка на задачата

Ще продължим представянето на елементите на $\mathfrak{B}_n$ с помощта на наредени n-торки естествени числа $< k_1, k_2, \ldots, k_n >$, където $0 \leq k_i \leq 2^n - 1$, $i = 1, 2, \ldots, n$. Взаимно-еднозначното съответствие се основава на еднозначното представяне на естествените числа в двоична бройна система, т.е. числото k_i записано в двоична бройна система (с евентуално известен брой незначещи нули в началото) ще представлява i-я ред на съответната бинарна матрица. Както доказахме в раздел 2.3.4, теорема 2.3.28, с помощта на това представяне съществуват по-бързи и по-икономични по отношение на използвана оперативна памет алгоритми. Ще се възползуваме от този факт за да създадем алгоритъм, намиращ точно по един представител от всеки клас на еквивалентност съответно от дефинираните в раздел 2.2 фактормножества $\overline{\mathcal{Q}_n}$, $\overline{\mathcal{M}_n}$ и $\overline{\mathcal{R}_n}$. При това полученият представител ще бъде минималният в съответния клас на еквивалентност по отношение на лексикографската наредба, която по естествен начин се въвежда в множеството $\mathbb{N}^n$ от всички наредени n-торки от цели неотрицателни числа. Като следствие получаваме и алгоритъм за решаване на комбинаторната задача за намиране броя на класовете на еквивалентност в множествата $\mathcal{Q}_n$, $\mathcal{M}_n$ и $\mathcal{R}_n$ относно релацията $\sim$ при зададено естествено число n.

2.4.2. Описание на алгоритъма

Матриците P и S, зададени съответно с формули (2.2.3) и (2.2.12) се кодират с помощта на наредени n-торки както следва:

$$P \ :< 2^{n-2}, 2^{n-3}, \ldots, 2^1, 2^0, 2^{n-1} > \tag{2.4.1}$$

$$S \ :< 2^0, 2^1, 2^2, \ldots, 2^{n-2}, 2^{n-1} > \tag{2.4.2}$$

В езиците за програмиране например C, C++, Java функцията $y = 2^k$, където k е цяло неотрицателно число може да бъде пресметната като се използва еднократно операторът

`x=1<<k;`,

където присъства единствено операцията побитово изместване на ляво `<<` принадлежаща на множеството $\mathcal{O}$ от стандартни операции за програмните езици C, C++ Java (виж пример 1.1.2).

На страница 35 с помощта на формула 2.3.5 е дефинирана операцията *логическо произведение* на две бинарни матрици, която означихме с "$*$".

Аналогично на класическото доказателство за асоциативността на операцията произведение на матрици (виж напр. [34]) се доказва, че операцията логическо

произведение на бинарни матрици е асоциативна. Следователно $\mathfrak{B}_n$ с въведената операция логическо произведение е моноид с единица - единичната матрица E_n. При това $\mathcal{P}_n$ е собствена подгрупа на този моноид.

На страница 43 с помощта на програмен код 2.3.25 описахме алгоритъм работещ за време $O(n^2)$ и осъществяващ операцията логическо произведение на две бинарни матрици, всяка от които се представя еднозначно чрез наредена n-торка от естествени числа. В същото време за да осъществим произведение на две матрици съгласно класическата дефиниция са необходими $O(n^3)$ операции.

Ако бинарната матрица A е представена с помощта на наредена n-торка числа, то за да проверим принадлежи ли A на множеството $\mathcal{Q}_n \subset \mathfrak{B}_n$ може да използуваме следното очевидно твърдение:

Лема 2.4.1. *Нека $A \in \mathfrak{B}_n$ и нека A се представя с помощта на наредената n-торка $< k_1, k_2, \ldots, k_n >$, където $0 \leq k_i \leq 2^n - 1$, $i = 1, 2, \ldots, n$ и нека $c \mid$ и $\&$ означим съответно операциите побитова дизюнкция и побитова конюнкция. Тогава:*

(i) Числото k_i, $i = 1, 2, \ldots, n$ представя изцяло нулев ред тогава и само тогава, когато $k_i = 0$;

(ii) Числото k_i, $i = 1, 2, \ldots, n$ представя изцяло единичен ред тогава и само тогава, когато $k_i = 2^n - 1$;

(iii) j-я стълб на A е изцяло нулев тогава и само тогава, когато

$$(k_1 \mid k_2 \mid \cdots \mid k_n) \,\&\, 2^j = 0;$$

(iv) j-я стълб на A е изцяло съставен от единици тогава и само тогава, когато

$$(k_1 \,\&\, k_2 \,\&\, \cdots \,\&\, k_n) \,\&\, 2^j \neq 0.$$

□

Алгоритъма, който ще опишем по-долу се основава на следните твърдения:

Лема 2.4.2. *Ако $A \in \mathfrak{B}_n$, $B \in \mathcal{P}_n$, то*

$$A * B = AB$$

и

$$B * A = BA$$

Доказателство. Нека $A = [a_{ij}]_{n\times n}$, $B = [b_{ij}]_{n\times n}$, $U = A * B = [u_{ij}]_{n\times n}$ и $V = AB = [v_{ij}]_{n\times n}$, $i, j = 1, 2, \ldots, n$. Нека единствената единица в j-я стълб на $B \in \mathcal{P}_n$ е на s-то място, т.е. $b_{sj} = 1$ и $b_{kj} = 0$ при $k \neq s$. Тогава по дефиниция

$$u_{ij} = \bigvee_{k=1}^{n} (a_{ik} \,\&\, b_{kj}) = \begin{cases} 1 & \text{for} \quad a_{is} = 1 \\ 0 & \text{for} \quad a_{is} = 0 \end{cases}$$

и

$$v_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = \begin{cases} 1 & \text{for} \quad a_{is} = 1 \\ 0 & \text{for} \quad a_{is} = 0 \end{cases}$$

Следователно $u_{ij} = v_{ij}$ за всеки $i, j \in \{1, 2, \ldots, n\}$.

Аналогично се доказва, че $B * A = BA$.

□

Лема 2.4.3. *Нека $A \in \mathfrak{B}_n$ се задава с помощта на наредената n-торка $< k_1, k_2, \ldots, k_n >$ и нека A е минималният елемент в класа на еквивалентност относно лексикографската наредба в $\mathbb{N}^n$. Тогава $k_1 \leq k_t$ за всяко $t = 2, 3, \ldots, n$.*

Доказателство. Допускаме, че съществува $t \in \{2, 3, \ldots, n\}$, такова че $k_t < k_1$. Тогава ако преместим първия ред на последно място и това го направим последователно $t-1$ пъти, ще получим матрица $A' \in \mathfrak{B}_n$, такава че $A' \sim A$ и A' ще се представя с помощта на n-торката $< k_t, k_{t+1}, \ldots, k_n, k_1, \ldots, k_{t-1} >$. Тогава очевидно $A' < A$ спрямо лексикографската наредба в $\mathbb{N}^n$, което противоречи на минималността на A в класът на еквивалентност $\overline{A}$.

□

Така достигаме до следния обобщен алгоритъм за получаване на точно по един представител от всеки клас на еквивалентност във фактормножествата $\overline{\mathcal{Q}_n}$, $\overline{\mathcal{M}_n}$ и $\overline{\mathcal{R}_n}$

Алгоритъм 2.4.4. .

1. *Генериране на всевъзможните наредени n-торки от естествени числа $< k_1, k_2, \ldots, k_n >$, такива че $1 \leq k_i \leq 2^n - 2$ и $k_1 \leq k_i$ при $i = 2, 3, \ldots, n$;*

2. *Проверка за принадлежност на получените в т. 1 елементи на множеството $\mathcal{Q}_n$ съгласно лема 2.4.1 (iii) (iv) (случаите (i) и (ii) сме отхвърлили още при генерирането на елементите в т. 1 съгласно лема 2.4.3);*

3. *Проверка за минималност в съответният клас на еквивалентност на получените в т. 2 елементи. Съгласно леми 2.2.2 и 2.4.2 A е минимален в $\overline{A}$ тогава и само тогава, когато $A \leq P^k * A * P^l$ за всеки $k, l \in \{0, 1, \ldots, n-1\}$, където P е матрицата представена с n-торката (2.4.1);*

4. *Проверка за принадлежност на множеството $\mathcal{M}_n$ на елементите, получени в т. 3 съгласно дефиниция 2.2.8 и прилагайки леми 2.2.2 и 2.4.2;*

5. *Проверка за принадлежност на множеството $\mathcal{R}_n$ на елементите, получени в т. 3 съгласно дефиниция 2.2.12 и прилагайки леми 2.2.2 и 2.4.2.*

□

2.5. Заключение към втора глава

В тази глава направихме математическо описание на структурното многообразие на различните видове преплитания на нишките при тъкането на платове (тъкачни сплитки). Това изисква решаването на различни комбинаторни задачи, свързани с бинарни матрици. При реализацията на алгоритмите използвахме побитови операции като доказахме, че това води до повишаване на ефективността.

В множеството $\mathfrak{B}_n$ от всички $n \times n$ бинарни матрици въведохме операциите покомпонентна конюнкция `&&`, покомпонентна дизюнкция `||`, покомпонентно отрицание `!` и логическо произведение $*$. В $\mathfrak{B}_n$ въведохме линейна наредба $<$. Доказахме, че съществуват алгоритми за всеки от които са достатъчни $O(n)$ стандартни операции и реализиращи така дефинираните операции `&&`, `||`, `!` и $<$. Доказахме, че съществува алгоритъм за който са достатъчни $O(n^2)$ операции за да реализира операцията логическо произведение $*$ на две бинарни матрици. При реализацията на тези алгоритми представихме бинарните матрици с помощта на наредени

n-торки от цели неотрицателни числа и съществено използвахме побитови операции. В същото време, както е известно за стандартната реализация на операциите `&&`, `||`, `!` и `<` съгласно класическите дефиниции са необходими $O(n^2)$ стандартни операции, а за реализацията на операцията логическо произведение `*` на две бинарни матрици съгласно класическата дефиниция са необходими $O(n^3)$ стандартни операции. За всеки обект от описаният в тази глава C++ клас `Bn_tuple` са необходими $O(n)$`*sizeof (int)` байта оперативна памет. Стандартно за всяка целочислена матрица са необходими $O(n^2)$`*sizeof (int)` байта оперативна памет. И не на последно място доказахме, че за инициализацията на обект от клас `Bn_tuple` са достатъчни $O(n)$ стандартни операции, докато за инициализацията на произволна $n \times n$ целочислена матрица са необходими $O(n^2)$ стандартни операции. Това доказва и ползата от побитовите операции в програмирането.

За демонстрацията на идеите, заложени в настоящата глава решихме и една практическа задача с принос в текстилната техника за получаване на някои количествени оценки за структурното многообразие на различните текстилни материи. За тази цел създадохме програмата "Сплитки", написан на език за програмиране C++ и реализиращ идеите, разисквани в тази глава. Някои обобщени количествени характеристики, получени в резултат от работата на описаната компютърна програма са обобщени в таблица 2.1.

n	2	3	4	5	6
$\lvert\mathfrak{B}_n\rvert$	16	512	65 536	33 554 432	$2^{36} > 2^{32} - 1$
$\lvert\mathcal{Q}_n\rvert$	2	102	22 874	17 633 670	$> 2^{32} - 1$
$\lvert\overline{\mathfrak{B}_n}\rvert$	7	64	4 156	1 342 208	1 908 897 152
$\lvert\overline{\mathcal{Q}_n}\rvert$	1	14	1 446	705 366	1 304 451 482
$\lvert\overline{\mathcal{M}_n}\rvert$	1	2	142	1 302	586 060
$\lvert\overline{\mathcal{R}_n}\rvert$	1	2	18	74	902

Таблица 2.1. Обобщени количествени характеристики, получени в резултат от работата на алгоритъм 2.4.4

Глава 3

Множествено - релационен подход в занимателната математика и в психологическите изследвания

Материалът от раздел 3.2 е публикуван в [191, 195].

Материалът от раздел 3.3 е публикуван в [132].

Материалът от раздели 3.4, 3.5 и 3.6 е публикуван в [28, 47, 147, 148, 153, 193, 194].

3.1. Предварителни сведения

Материалът в настоящата глава е замислен да подпомогне преподавателя по програмиране в стремежа си да даде съдържателни, интересни и забавни примери за ползата на понятието множество в програмирането. В главата се разглеждат някои аспекти на обучението по програмиране. Набляга се на занимателния момент при подбора на подходящи примери за демонстрация на отделните езикови конструкции и структури от данни. За целта обучаемите трябва добре да познават основните дефиниции от теория на множествата и да владеят основните операции с множества. От тук следва и добре известният факт, че да си добър програмист е необходимо (но не и достатъчно) да си добър математик.

В раздел 3.2 ще разгледаме понятието множество във връзка с популярната в последно време математическа главоблъсканица Судоку. Ще покажем, как теоретико-множествените операции водят до ефективни алгоритми за решаване на Судоку. Ще разгледаме и някои комбинаторни задачи свързани със Судоку–матриците.

В раздел 3.3 се обсъжда въпроса за приложение на побитовите операции във връзка с описанието на крайни множества и алгоритми за операции с тях, реализирани с помощта на езиците за програмиране C++ и Java, където липсват стандартни езикови средства специално за тези цели.

Операциите с множества намират своето достойно място и при описание на модела на данните. Най-разпространеният в момента модел е релационният модел, основаващ се на операциите в релационната алгебра – естествено продължение на операциите в класическата теория на множествата. На този аспект на приложенията на теория на множествата ще се спрем в раздели 3.4, 3.5 и 3.6, където с помощта на теоретико-множествения апарат ще направим математико-информатичен модел на едно личностно психологическо изследване.

3.2. Един занимателен пример за използването на понятието множество в програмирането. Компютърът решава Судоку и комбинаторни задачи над Судоку–матрици

В този раздел ще наблегнем на занимателния момент при подбора на подходящи примери за демонстрация на отделните езикови конструкции и структури от данни. Такъв пример е разгледаният алгоритъм за решаване на широко разпространената в последно време главоблъсканица „Судоку”. Това е направено във връзка с понятието множество и неговото приложение в програмирането. С помощта на създадената програма са решени и някои комбинаторни задачи, свързани

със Судоку-матриците [191].

Класически пример за използването на множества в програмирането е станала програмната реализация на задачата за намиране на прости числа по метода наречен „Решето на Ератостен” [62, 102, 127, 154]. Тук сме длъжни да отбележим, че в [62] при реализацията на този алгоритъм е допусната грешка, причислявайки числото 1 към множеството на простите числа. Подобна грешка е недопустима за едно учебно помагало, тъй като би довела до объркване от страна на ползващите го.

В [195] е показано е, че с помощта на езика на множествата и операциите над тях може да бъде конструиран значително по-бърз алгоритъм, решаващ задачата за получаване на всички квадратни бинарни матрици, имащи еднакъв брой единици на всеки ред и всеки стълб.

Тук ще се спрем на занимателна и актуална задача, която се приема с голям интерес от страна на учащите – алгоритми за решаване на Судоку. Това е широко разпространена в днешно време главоблъсканица, неизменно присъстваща в развлекателните страници на болшинството вестници и списания и в развлекателни интернет сайтове. Sudoku, или Su Doku, е японска дума (или фраза) означаваща ”място на числата” [167]. От друга страна, Судоку-матриците имат и интересни комбинаторни приложения, например в теорията на кодирането и теорията на дизайните [94, 145]. В [103] е показана връзката между множеството на всички $n^2 \times n^2$ пермутационни матрици (т.е. бинарни матрици с точно една единица на всеки ред и всеки стълб) и множеството на всички Судоку-матрици.

Някои психологически аспекти за ползата от заниманието със Судоку във връзка с развитието на мисленето са показани в [141].

Математически главоблъсканици от рода на Судоку и подобни са събрани в книгата [35].

3.2.1. Постановка на задачата

Нека n е произволно цяло положително число и нека $m = n^2$. Нека $S = [s_{ij}]_{m \times m}$ е квадратна $m \times m$ матрица, всички елементи на които са цели числа принадлежащи на затворения интервал $[1, m]$. С помощта на $n-1$ хоризонтални и $n-1$ вертикални линии, прекарани съответно между някои от редовете и някои от стълбовете, матрицата S е разделена на n^2 не застъпващи се $n \times n$ квадратни подматрици, които ще наричаме блокове (на фиг. 3.1 е показана матрицата S при $n = 3$).

s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}
s_{21}	s_{22}	s_{23}	s_{24}	s_{25}	s_{26}	s_{27}	s_{28}	s_{29}
s_{31}	s_{32}	s_{33}	s_{34}	s_{35}	s_{36}	s_{37}	s_{38}	s_{39}
s_{41}	s_{42}	s_{43}	s_{44}	s_{45}	s_{46}	s_{47}	s_{48}	s_{49}
s_{51}	s_{52}	s_{53}	s_{54}	s_{55}	s_{56}	s_{57}	s_{58}	s_{59}
s_{61}	s_{62}	s_{63}	s_{64}	s_{65}	s_{66}	s_{67}	s_{68}	s_{69}
s_{71}	s_{72}	s_{73}	s_{74}	s_{75}	s_{76}	s_{77}	s_{78}	s_{79}
s_{81}	s_{82}	s_{83}	s_{84}	s_{85}	s_{86}	s_{87}	s_{88}	s_{89}
s_{91}	s_{92}	s_{93}	s_{94}	s_{95}	s_{96}	s_{97}	s_{98}	s_{99}

Фигура 3.1. 9×9 Судоку-матрица

Нека да означим блоковете в дефинираната по-горе матрица $S = [s_{i\,j}]_{n^2 \times n^2}$ с

A_{kl}, $1 \leq k, l \leq n$. Тогава по дефиниция, ако $s_{ij} \in A_{kl}$, то

$$(k-1)n < i \leq kn$$

и

$$(l-1)n < j \leq ln.$$

Нека s_{ij} принадлежи на блока A_{kl} и нека да са известни i и j. Тогава лесно можем да съобразим, че k и l могат да бъдат изчислени с помощта на формулите

$$k = \left[\frac{i-1}{n}\right] + 1$$

и

$$l = \left[\frac{j-1}{n}\right] + 1,$$

където с $[x]$ сме означили както обикновено функцията цяла част на реалното число x.

Ще казваме, че $S = [s_{ij}]_{n^2 \times n^2}$, $1 \leq i, j \leq n^2$, е *Судоку-матрица*, ако във всеки ред, всеки стълб и всеки блок съществува точно по едно число от множеството $Z_{n^2} = \{1, 2, \ldots, n^2\}$.

Дадена е Судоку-матрица, на която са изтрити някои от елементите. Липсващите елементи на S при нужда ще отъждествяваме с 0. Задачата на главоблъсканицата се състои в това да се възстановят липсващите елементи на Судоку-матрицата. Предполага се, че авторите на конкретната главоблъсканица така са подбрали липсващите елементи, че задачата да има единствено решение. Това условие, ние ще пропуснем и няма да се съобразяваме с него. Нещо повече, в нашата разработка така ще направим програмният продукт, че същия да показва и броя на възможните решения. Ако задачата няма решение, то този брой ще бъде нула. Най-разпространените главоблъсканици Судоку са при $n = 3$, т.е. при $m = 9$.

3.2.2. Описание на алгоритъма

Тук ще разгледаме някои методически аспекти, свързани с преподаването на дисциплината програмиране като изтъкнем занимателния елемент в подбора на подходящи примери. Ще опишем и анализираме алгоритъм за създаване на компютърна програма, намираща всички решения (ако съществуват) на произволно Судоку. За целта съществено ще използуваме познанията ни от теория на множествата.

Разглеждаме множествата R_i, C_j и B_{kl}, където $1 \leq i, j \leq m = n^2$, $1 \leq k, l \leq n$. За всяко $i = 1, 2, \ldots, m$, множеството R_j се състои от всички липсващи числа в i-я ред на матрицата. Аналогично дефинираме и множествата C_j, $j = 1, 2, \ldots, m$ съответно за липсващите числа в j-я стълб и B_{kl}, $k, l = 1, 2, \ldots, n$ съответно за липсващите числа в блоковете A_{kl} на S.

Началото на алгоритъма се състои в многократно обхождане на всички елементи $s_{ij} \in S$, такива че $s_{ij} = 0$, т.е. това са елементите, чийто истински стойности трябва да открием.

Нека $s_{ij} = 0$ и нека $s_{ij} \in A_{kl}$. Полагаме

$$P = R_i \bigcap C_j \bigcap B_{kl}.$$

Тогава са възможни следните три случая:

i) $P = \phi$ (празното множество). В този случай задачата няма решение;

ii) $P = \{d\}$, $d \in Z_m = \{1, 2, \ldots, m\}$, т.е. броят $|P|$ на елементите на P е равен на 1 (P е едномоментно множество). Тогава единствената възможност за s_{ij} е $s_{ij} = d$, т.е. в този случай сме открили неизвестната стойност на s_{ij}. След това премахваме общия елемент d от множествата R_i, C_j и B_{kl}, след което преминаваме към следващия нулев елемент на матрицата S (ако съществува такъв);

iii) $|P| \geq 2$. Тогава нищо не можем да кажем за неизвестната стойност на s_{ij} и преминаваме към следващия липсващ (нулев) елемент на матрицата S.

Обхождането на нулевите елементи на матрицата S продължава докато не настъпи едно от следните събития:

e1) Задачата доказано няма решение, т.е. намерени са $i, j \in \{1, 2, \ldots, m\}$ за които е изпълнено $s_{ij} = 0$, но $P = R_i \bigcap C_j \bigcap B_{kl} = \phi$;

e2) Всички елементи на S станат строго положителни;

e3) Обходени са всички нулеви елементи на S, но не е настъпило нито събитие e1, нито събитие e2. С други думи при всички оставащи нулеви елементи на S, винаги е в сила описаният по-горе случай iii).

В случай, че настъпи едно от събитията e1 или e2, процедурата спира своята работа и извежда получения резултат.

В случай, че настъпи събитие e3, алгоритъмът трябва да продължи използувайки други методи, например да приложи метода на "пробите и грешките". В конкретния случай този метод се състои в следното:

Избираме произволно $s_{ij} \in S$, такова че $s_{ij} = 0$ и нека $k = \left[\frac{i-1}{n}\right] + 1$, $l = \left[\frac{j-1}{n}\right] + 1$. Нека $P = R_i \bigcap C_j \bigcap B_{kl} = \{d_1, d_2, \ldots, d_t\}$. Тогава за всяко $d_r \in P$, $r = 1, 2, \ldots, t$ полагаме $s_{ij} = d_r$. Такова полагане ще наричаме *случайна проба*. В предложената по-долу програмна реализация на алгоритъма, ще броим и всички случайни грешки до достигане на дадено решение. След това решаваме задачата за намиране на неизвестните елементи на Судоку-матрицата с един неизвестен елемент по-малко. Тук е удобно използуването на рекурсия. База на рекурсията, т.е. ще излезем от процедурата, ако настъпи събитие e1 или e2. Това неминуемо би трябвало да се получи (т.е. няма да попаднем във „вечен цикъл"), тъй като при случайните проби намаляваме броя на нулевите елементи с 1.

За реализацията на описания алгоритъм за решаване на Судоку, най-удобен е език за програмиране Паскал по две основни причини:

1) Паскал се избира от повечето учебни заведения в системата на средното образование в качеството му на първи език за програмиране [3, 4]. В този смисъл се изпълнява една от основните цели на разработката – да подпомогне учителя в подготовката му и в процеса на търсене на интересни приложения на преподавания материал. По този начин и интересът на учениците към преподавания материал се повишава.

2) В езика Паскал има вградени средства за работа с множества.

Изложените по-горе идеи могат да бъдат реализирани и на произволен друг език за програмиране, например C++. Но в този случай трябва да търсим допълнителни средства за работа с множества (нещо което ще направим в следващите раздели).

С учебна цел би могло да се създаде и собствен клас „множество“ като се опишат специфичните методи. Това би било прекрасно упражнение, като се има в предвид и факта, че базовото („универсално“) множество е с относително малка мощност. Например „стандартната“ главоблъсканица Судоку е с базово множество целите числа от 1 до 9 плюс празното множество. Това ще направим в раздел 3.3, където основната ни цел е да търсим нови примери за комбинаторни приложения на побитовите операции.

3.2.3. Комбинаторни задачи със Судоку–матрици

Очевидно, ако за решаване на дадено Судоку няма нужда от прилагане на метода на пробите и грешките, т.е. задачата може да бъде решена само с прилагане на теоретико-множествени операции (0 на брой случайни проби), то това Судоку има единствено решение. Такова Судоку е показано на фигура 3.2.

								3
5		3		8	2		4	1
6	2	4	5				7	
7	8	5		9		1	6	
4		6	2			3		
	3		1	7	6			
1								4
		7	8			2		
3	4	8	6			7	1	5

Фигура 3.2

Обратното твърдение не е вярно, както показва и следващият пример, илюстриран на фигура 3.3, където имаме единствено решение и програмата е направила 218 случайни проби само до достигането му, като общо направените случайни проби (след намирането на последното решение програмата продължава, ако е необходимо с пробите) са 332.

7			3					4
	9				1			
	1	2			4			
			9		7	4		
	5			3				
	8			2		1		7
		6						3
			7			5		
8	4						2	9

Фигура 3.3

Понякога авторите на Судоку не изследват за еднозначност на решението. Например показаното на фигура 3.4 Судоку (и поместена във всекидневник, чието заглавие се въздържаме да споменем под рубриката „лесно Судоку“) има 4 различни решения, като програмата прави общ брой 16 случайни проби до намирането на последното решение, като общия брой случайни проби е 18.

Показаното на фигура 3.5 Судоку няма решение, въпреки че на пръв поглед в условието всичко е на ред и няма противоречие. За да достигне до този извод на програмата и е достатъчно да направи общо 21 случайни проби.

		4	9		3	8		
	2		8			9		3
3		8		4			1	
7				9	4	1		
2	4		7					8
	3		2				7	5
8		7		2			3	
	5				7	2		9
		2		3	8		5	

Фигура 3.4

	5		9		2		4	
2		7				9		8
	4		8	7		3		
	1			2		8		9
5		8		9				3
7			5		3		1	
3		4		5				2
			2		7	4	8	
	7	2			8		3	

Фигура 3.5

Ако стартираме програмата и дадем нулеви стойности на всички елементи на матрицата S, то с помощта на създадената от нас програма ще получим всевъзможните $m \times m$ Судоку–матрици.

При $n = 2$ получаваме, че броят на всички 4×4 Судоку–матрици е равен на 288. При това за да се получи този резултат компютърът е направил 568 случайни проби.

При $n = 3$ за улеснение премахнахме преброяването на случайните проби и след повече от 10 денонощна непрекъсната работа на нашия (не много бърз и не много мощен компютър) получихме, че съществуват точно

$$96\,670\,903\,752\,021\,072\,936\,960$$

на брой 9×9 Судоку матрици. Този резултат напълно съвпада и с резултата получен с други методи в [108].

А ето и още някои комбинаторни резултата, свързани с различни видове Судоку матрици и получени с помощта на експерименти с описания по-горе програмен продукт.

- Съществуват 283 576 Судоку-матрици от вида, показан на фигура 3.6;
- Съществуват 6 280 Судоку-матрици от вида, показан на фигура 3.7;
- Съществува единствено решение на Судоку от вида, показан на фигура 3.8, при това не се налага при решаването му да се използва методът на пробите и грешките (0 случайни проби);
- Съществуват 680 Судоку–матрици от вида, показан на фигура 3.9;
- Съществуват 744 Судоку–матрици от вида, показан на фигура 3.10;
- Съществуват 430 848 Судоку–матрици от вида, показан на фигура 3.11;

1	2	3	*	*	*	*	*	*
4	5	6	*	*	*	*	*	*
7	8	9	*	*	*	*	*	*
*	*	*	1	2	3	*	*	*
*	*	*	4	5	6	*	*	*
*	*	*	7	8	9	*	*	*
*	*	*	*	*	*	1	2	3
*	*	*	*	*	*	4	5	6
*	*	*	*	*	*	7	8	9

Фигура 3.6

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	1	*	*	*	*	*	*
5	6	4	*	*	*	*	*	*
8	9	7	*	*	*	*	*	*
3	1	2	*	*	*	*	*	*
6	4	5	*	*	*	*	*	*
9	7	8	*	*	*	*	*	*

Фигура 3.7

1	2	3	4	*	6	7	8	9
4	5	6	7	*	9	1	2	3
7	8	9	*	*	*	4	5	6
2	3	*	*	*	*	*	9	1
*	*	*	*	*	*	*	*	*
8	9	*	*	*	*	*	6	7
3	4	5	*	*	*	9	1	2
6	7	8	9	*	2	3	4	5
9	1	2	3	*	5	6	7	8

Фигура 3.8

1	2	3	4	*	6	7	8	9
4	5	6	*	*	*	1	2	3
7	8	9	*	*	*	4	5	6
2	*	*	*	*	*	*	*	1
*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	7
3	4	5	*	*	*	9	1	2
6	7	8	*	*	*	3	4	5
9	1	2	3	*	5	6	7	8

Фигура 3.9

- Съществуват 6 280 Судоку–матрици от вида, показан на фигура 3.12;
- Съществуват 1 728 Судоку-матрици от вида, показан на фигура 3.13;
- Съществуват 22 Судоку-матрици от вида, показан на фигура 3.14;

1	2	3	4	*	6	7	8	9
4	5	6	*	*	*	1	2	3
7	8	*	*	*	*	*	5	6
2	*	*	*	*	*	*	*	1
*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	7
3	4	*	*	*	*	*	1	2
6	7	8	*	*	*	3	4	5
9	1	2	3	*	5	6	7	8

Фигура 3.10

1	2	3	*	*	*	7	8	9
4	5	6	*	*	*	1	2	3
7	8	9	*	*	*	4	5	6
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
3	4	5	*	*	*	9	1	2
6	7	8	*	*	*	3	4	5
9	1	2	*	*	*	6	7	8

Фигура 3.11

*	*	*	1	2	3	*	*	*
*	*	*	4	5	6	*	*	*
*	*	*	7	8	9	*	*	*
1	2	3	9	7	8	4	5	6
4	5	6	3	1	2	7	8	9
7	8	9	6	4	5	1	2	3
*	*	*	2	3	1	*	*	*
*	*	*	5	6	4	*	*	*
*	*	*	8	9	7	*	*	*

Фигура 3.12

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*

Фигура 3.13

- Съществуват 8 Судоку-матрици от вида, показан на фигура 3.15;

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	*	*	*
5	6	7	8	9	1	*	*	*
8	9	1	2	3	4	*	*	*
3	4	5	*	*	*	*	*	*
6	7	8	*	*	*	*	*	*
9	1	2	*	*	*	*	*	*

Фигура 3.14

1	*	*	7	*	*	4	*	*
*	2	*	*	8	*	*	5	*
*	*	3	*	*	9	*	*	6
7	*	*	4	*	*	1	*	*
*	8	*	*	5	*	*	2	*
*	*	9	*	*	6	*	*	3
4	*	*	1	*	*	7	*	*
*	5	*	*	2	*	*	8	*
*	*	6	*	*	3	*	*	9

Фигура 3.15

3.3. Побитовите операции във връзка с понятието множество

За съжаление, както вече подчертахме на страница 52 в езиците за програмиране C/C++ и Java няма стандартен тип „множество“, както това е направено например в езика за програмиране Паскал. По тази причина, ако искаме да използуваме езика на множествата при реализацията на някои наши алгоритми и в последствие съставим програма на някой от тези езици, е необходимо да търсим допълнителни средства за работа с множества – например реализираните в Standard Template Library (STL) асоциативни контейнери `set` и `multiset` [2, 100, 140]. Също така може да се използува и шаблонния клас `set` от системата за компютърна алгебра „Symbolic C++“ и чийто програмен код е даден в подробности в [163].

Целта на настоящия раздел е да покажем ползата от побитовите операции при реализацията на теоретико–множествени операции със средствата на обектно–ориентираното програмиране на базата на езика за програмиране C++, което разбира се лесно може да бъде пренесено и на език за програмиране Java, който има същият синтаксис както и езиците C/C++ [31, 41, 91, 109, 156]. Тук ние ще създадем собствен клас „множество“ като се опишат специфичните методи, където съществено ще използваме побитовите операции за значително подобряване на ефективността и бързодействието на алгоритмите.

3.3.1. Описание на модела

Нека $M = \{\mu_0, \mu_1, \ldots, \mu_{m-1}\}$, $|M| = m$ е крайно и дискретно множество. Както е широко разпространено в практиката [40] всяко подмножество $A \subseteq M$ може да се представи с помощта на булев вектор $b(A) = [b_0, b_1, \ldots, b_{m-1}]$, където $b_i = 1 \Leftrightarrow \mu_i \in A$ и $b_i = 0 \Leftrightarrow \mu_i \notin A$, $i = 0, 1, 2, \ldots, m-1$. Значителна икономия на оперативна памет бихме могли да постигнем, ако вместо булев вектор използуваме

представянето на естествените числа в двоична бройна система [131] (виж също и теорема 4.7.5 на страница 91). Числото 0 ще съответства на празното множество, а числото 2^m-1, което в двоична бройна система се записва с помощта на m единици ще съответства на базовото множество M. Така по естествен начин получаваме взаимно-еднозначно съответствие между целите числа от затворения интервал $[0, 2^m - 1]$ и множеството от всички подмножества на M. При това цялото число $a \in [0, 2^m - 1]$ ще съответства на множеството $A \subseteq M$, ако за всяко $k = 0, 1, \ldots, m-1$, k-ят бит на двоичното представяне на a е 1 тогава и само тогава когато $\mu_i \in A$. В този случай при компютърната реализация на теоретико множествените операции естествено възниква необходимостта от използването на побитови операции.

Подобен подход е удобен и значително ефективен при сравнително малка мощност $m = |M|$ на базовото множество M. Съществено значение има и операционната система и средата за програмиране, които използуваме. Причината за това е, че за да кодираме по описания по-горе начин едно множество, което е подмножество на M, където $|M| = m$ са необходими m бита. И ако в средата за програмиране за целочислените типове данни се отделят k бита, то за да реализираме идеите, изложени по-горе ще са необходими $\left[\frac{m}{k}\right] + 1$ променливи от дадения тип, където с $[x]$ сме означили "цяла част на x". За нашата задача – написването на програма, решаваща главоблъсканица Судоку с размер $n^2 \times n^2$ използувайки апарата от теория на множествата при $n \leq 5$, са напълно достатъчни 4 байта (32 бита). Това е размерът на стандартният тип `int`, който съвпада с типът `long` при повечето съвременни операционни системи и съответните среди за програмиране. В този случай всяко множество от вида $A = \{\alpha_1, \alpha_2, \ldots, \alpha_s\}$, където $1 \leq \alpha_i \leq n^2$, $i = 1, 2, \ldots, s$, $0 \leq s \leq n^2$, или празното множество, може да бъде еднозначно кодирано с единствено цяло число.

Още по-добри резултати по отношение на икономия на оперативна памет бихме получили и ако използваме тип `short`, за който са необходими 2 байта, но тогава базовото множество не може да бъде с мощност по-голяма от 16 (при мощност на базовото множество равно на 16 ще трябва да използваме тип `unsigned short`). За съвсем малки базови множества с мощност не превишаваща 8 можем да използваме и тип `char`, или въведения допълнително в някои среди за програмиране (не навсякъде съществува) тип `byte`.

3.3.2. Описание на клас „множество“ с използуване на побитови операции и предефиниране на оператори

Разглеждаме множеството

$$\mathbb{Z}_{31} = \{1, 2, \ldots, 31\},$$

което ще наричаме *базово множество*.

Тук ще опишем клас, обектите на който могат да бъдат всички подмножества, включително и празното на $\mathbb{Z}_{31}$. Методи на този клас ще бъдат различните операции с множества. Класът ще има едно единствено поле – число n от тип `int`, двоичният запис на което ще ни дава и описание на самото множество. При това k-ят бит на този запис е единица тогава и само тогава, когато числото k принадлежи на множеството, представено от n. За улеснение сме се абстрахирали от знаковия бит на число, записано в компютъра с фиксирана запетая. Поради тази причина не сме включили и числото 32 в базовото множество.

Създаденият от нас клас има два конструктора. Първият е без параметри и инициализира празното множество. Вторият е с един параметър, цяло число, двоичният запис на което определя множеството. По този начин празното множество

може да бъде инициализирано по два начина – без параметър или с параметър равен на 0. Универсалното множество, съдържащо всички числа от 1 до 31 се инициализира със стандартната за редици среди за програмиране константа `Maxint`, която е равна на $2^{31} - 1$, т.е. най-голямото числа, което се записва в 31 бита и което е максималното число за целочисления тип `int`. В случай, че вместо с `int` навсякъде работим с `unsigned int`, то това число ще бъде равно на $2^{32} - 1$. Използвайки побитовата операция `<<` изместване в ляво без проблем тази константа може да бъде пресметната, както е показано в следващия пример.

Пример 3.3.1.
```
Set A, B(0);
unsigned int mx = (1<<31) - 1;
Set U(mx);
```

В пример 3.3.1 множествата `A` и `B` са инициализирани като празни, т.е. не съдържат нито един елемент, а `U` е универсалното множество, т.е. множеството съдържащо всички числа от 1 до 31.

Нека са дадени множествата $A, B \subseteq \mathbb{Z}_{31}$, които ще бъдат обекти на създавания от нас клас и числото $k \in \mathbb{Z}_{31}$. Разглеждаме следните операции с множества, които ще реализираме като методи на класа и които с помощта на предефиниране на някои оператори ще имат собствени подходящи означения. Тези означения са взаимствани от познатите ни от средното училище [3, 4] означения за операции с множества в езика за програмиране Паскал:

- Сечение $A \cap B$ на две множества. Ще го означаваме с `A*B`.
- Обединение $A \cup B$ на две множества. Ще го означаваме с `A+B`.
- Обединение $A \cup \{k\}$ на множеството A с едноелементното множество $\{k\}$. Ще го означаваме с `A+k`.
- Добавяне на елемента k към множеството A. Ще го означаваме с `k+A`.

Тук трябва да отбележим, че от алгоритмична гледна точка `A+k` и `k+A` се реализират по различен начин, съобразявайки се със стандарта на езика C++, независимо от факта, че от математическа гледна точка това е една и съща операция съгласно комутативния закон при обединението на две множества.

- Премахване на елемента k от множеството A. Ако $k \notin A$, то A не се променя. Тази операция ще означаваме с `A-k`.
- Разлика $A \setminus B = \{k \mid k \in A \ \& \ k \notin B\}$ на две множества. Ще я означаваме с `A-B`.
- Проверка дали $A \supseteq B$, т.е дали множеството A съдържа в себе си множеството B. Ще я означаваме с `A>=B`. Резултатът е истина или лъжа.
- Проверка дали $A \subseteq B$, т.е дали множеството A е подмножество на множеството B. Ще я означаваме с `A<=B`. Резултатът е истина или лъжа.
- Проверката дали множествата A и B са равни по между си ще означаваме с `A==B`. Тук, както и при останалите две операции сме се съобразили единствено със синтаксиса на езиците C/C++ и Java. Резултатът е истина или лъжа.

- Проверката дали множествата A и B са различни ще означаваме с `A!=B`. Резултатът е истина или лъжа.

- За да проверим дали даден елемент $k \in \mathbb{Z}_{31}$ е елемент на множеството $A \subseteq \mathbb{Z}_{31}$ ще използваме метода (член функцията) `in` във вида `A.in(k)`. Резултатът е истина или лъжа.

Една примерна декларацията и реализация на класът „множество“ със съществуването използване на побитови операции са дадени в [132].

3.4. Въведение в компютърното администриране на психологически тестове

3.4.1. Основни дефиниции

Под *психологическо тестиране* ще разбираме всяко психологическо изследване, извършвано с помощта на предварително подготвен *тест* – списък с въпроси или твърдения, на които изследваното лице или група от хора трябва да отговорят или да дадат своето мнение. Отделните единици на теста се наричат *айтеми*.

Под *компютърно психологическо тестиране* ще разбираме всяко психологическо тестиране, при което един или няколко етапа от процедурата са проведени с помощта на компютър.

Под *компютърно администриране на психологически тестове* ще разбираме компютърното пресъздаване на цялостната процедура на психологическите изследвания – от конструирането на теста, неговото провеждане, изчисляване на резултатите, съхранение и поддръжка на получената база от данни, статистическата им обработка и техния анализ и интерпретация.

Под *личностни въпросници* ще разбираме психологически тестове, предназначени за описване и оценка особеностите на поведенческа, емоционалната и мотивационна сфера, на междуличностните отношения и атитюди на индивида [59]. Характерно при личностните въпросници (за разлика от тестовете за постижения и интелигентност) е че айтемите са под формата на въпроси или твърдения, при чиито отговори изследваното лице трябва самостоятелно да съобщи някои сведения за себе си, за своите преживявания и отношения. Формàта на отговорите на айтемите също е специфичен – най-често те се описват с помощта на крайно множество, от предварително известни отговори или твърдения, което ще означаваме с Ans. В нашите разглеждания ще акцентираме предимно на крайни множества от възможни отговори на всеки айтем от теста. Например Ans={”да”, ”не”}, Ans={”вярно”, ”грешно”}, Ans={”харесва ми”, ”не ми харесва”}, Ans={”често”, ”понякога”, ”никога”}, Ans={”вярно”, ”не знам”, ”невярно”}, Ans={”съгласен”, ”не уверен”, ”не съгласен”}. В практиката се използуват и непрекъснати рейтингови скали, но ние ще разглеждаме единствено рейтингови скали, които представляват крайно и следователно дискретно множество от реални числа.

Адаптацията на тест (от лат. adaptatio - приспособяване) представлява комплекс от процедури, обезпечаващи адекватността на теста в новите условия на неговото приложение [11]. В най-голяма степен това се отнася за преводните психологически тестове. За да бъдат преводните психологически тестове надеждни средства за осигуряване на точни изследователски данни, като предпоставка за достоверността на научните изводи, е необходима тяхната адаптация към конкретните националните, религиозни и социални условия на приложение [148]. В

този смисъл компютърното администриране значително би улеснило работата на изследователя в тази област.

За повече подробности по отношение на основните дефиниции от областта на психологическите измервания и методите за тяхното провеждане препоръчваме книгите [59, 63, 64, 88, 144].

3.4.2. Предимства и недостатъци на компютърното тестиране

Освен безспорно очевидните преимущества на компютърното тестиране - бързото провеждане, високата скорост при обработката и лесното архивиране на резултатите, към достойнствата на съвременните компютъризирани методики, в сравнение с традиционните, могат да се отнесат и следните:

- Неизменчивостта на компютърните програми обезпечава постоянни условия на тестиране, което е трудно постижимо при традиционните тестове.
- Намалява се до минимум субективното влияние на психолога върху резултатите от изследването.
- Има възможност за изключване на психолога като допълнителен стресов фактор при изследването, което увеличава искреността и оттам надеждността на теста.
- Обезпечава се точна и еднозначна регистрация на множеството реакции на изследваните лица.
- Има възможност да се възстановят и проследят в последователност действията на изследваните лица.
- Сравнително лесно се създават единни банки с данните от различни изследвани лица (база данни).
- Съществува възможността от автоматизирано конструиране на тестове.
- Психолозите се освобождават от рутинната и трудоемка работа, както при проверка на данните, така и при конструиране или адаптация на тестове.
- Съществува възможност за разширяване практиката на груповото тестиране.
- Разширява се възможността за използване на мощни математико - статистически апарати за анализ на данни, което улеснява разработването на нови процедури за анализ.
- Облекчава се обезпечаването на конфиденциалност на личните резултати от тестирането.
- Съществува възможност за провеждане на масови изследвания (например по Internet).
- Улеснява се запазването на диагностичните данни (на магнитни носители вместо на хартиени), с което се намалява себестойността на изследванията.

- Създават се благоприятни условия за прилагане на експрес-методики, провеждането на които позволява бързо получаване на резултати, което в много случаи (например в професионалната психодиагностика) може да има решаващо значение.

- Създава се възможност за обединение на тестовете в батерии.

- Компютърната процедура има минимално негативно въздействие, каквото нерядко съществува при директното взаимодействие между експериментатора и изследваното лице (снижаване действието на защитния механизъм при изследвания).

- Създава се възможност за актуализация на игровия мотив у изследваното лица (чрез оформление на теста като игра), което прави процесът на тестиране по-привлекателен и повишава достоверността на резултатите.

- Създава се възможност за провеждане на анализ на поведението на изследвания непосредствено в хода на изследването, отчитане на многобройните параметри в различните ситуации, за организация на диалог в режим на работа на програмата (т.н. адаптивни тестове).

- Чрез средствата на компютърната графика има възможност да се представят на тестирания динамични обекти (динамично стимулирана среда), става възможно използването на полимодални стимули.

- Появява се възможността за индивидуализация на психодиагностичните изследвания.

- Обезпечава се тясната връзка с решаването на практически задачи.

Наред с преимуществата, които носят компютрите, някои специалисти по психология изтъкват и следните недостатъци:

- Разработването на нови компютърни тестове е сложна, трудоемка и скъпа процедура, с която могат да се справят само добри специалисти - програмисти.

- Необходимо е специално обучение на операторите на подобни програми за работа с компютърните тестове.

- Отсъства индивидуален подход при тестирането.

- Недостатък е и латентността на етапите на обработка и интерпретация на резултатите (те са предварително зафиксирани от програмиста в компютърната програма).

- Някои утвърдени и станали класически тестове в електронен вид довеждат до съществени изменения на диагностичните качества, преди всичко на валидността и надеждността. Тоест за да бъдат използвани в психодиагностичната практика автоматизираните версии на вече стандартизирани тестове се нуждаят от специална проверка – т.н. ревалидизация. Също така е нужно повторно обосноваване на надеждността на компютърните методики и тяхната рестандартизация.

- При обработката на получените резултати най-често се използва мощен софтуер с общо предназначение като например системите SPSS, MatLab, Statistica, S-Plus и други подобни, за усвояването на които се изискват не малки усилия, докато при компютърното тестиране се използува неголяма част от техните възможности, което води до известна неефективност в работата на психолога - изследовател.

- Отсъствието на непосредствено наблюдение и жив контакт с изследвания носи същите трудности и ограничения, които произтичат от груповото тестиране.

- При около 30% от изследваните се наблюдава психологическа бариера [11], т.н. "феномен на компютърната тревожност", дори в 5%, от случаите са регистрирани състояния, сходни с фобиите.

- Някои от тестовете на практика не могат да се транслират в компютърен вид (каквито са повечето проективни методи).

- На практика е затруднено прилагането на компютърни тестове в полеви условия. Задължително условие за прилагането им в подобни условия в случая е използването на преносими компютри.

Описаните недостатъци предизвикват отрицателно отношение на някои психолози към компютърните тестове. Твърде ограничено, например е използването на такива тестове в клиничната психодиагностика, тъй като цената на грешката там е може да бъде твърде висока. Видно е, обаче че компютърните тестове имат голямо бъдеще, при което навярно много от недостатъците на процедурата ще бъдат разрешени благодарение на по-нататъшното развитие на компютърната техника. Този оптимизъм проличава в нарастващия интерес към компютърната психодиагностика, в инструментариума на която се включват над 1000 компютърни теста [11].

Болшинството от описаните недостатъци за компютърно тестиране могат да бъдат преодоляни с помощта на създаване на ново поколение тясно специализиран софтуер за психологически изследвания (експертна система в психологията). Създаването на необходимия софтуер за компютърно администриране на психологически тестове изисква определени знания и умения в областта на програмирането, което твърде често не е в компетенцията на психолога - експериментатор. От друга страна самото създаване на теста, процедурата по провеждането, обработката и анализа на резултатите изисква задълбочени знания от областта на психологията. В този смисъл разработването на специализирания софтуер изисква активното взаимодействие на специалисти и от двете области на науката. Резултатът – създаване на софтуер – генератор и анализатор на психологически тестове, автоматизиращ в максимална степен дейностите по компютърното администриране на психологически тестове, ще е икономически изгоден и оправдан, както от страна на време, така и на човешки ресурси.

3.5. Релационен модел на личностни психологически тестове

Данните в един софтуерен продукт могат да се структурират, преработват и съхраняват по различни начини. Данните и техните връзки представляват абстракция от факти и връзки от действителния свят [121]. Много често тази абстракция е достатъчно сложна и изисква специален математически апарат за нейното

описание, което се изразява в *модела на данните*. Използването на един или друг модел означава, че в дадена *информационна система* се избират едни или други принципи за структуриране и опериране с данните. Днес най-разпространеният модел на данните – това е релационният модел [48, 49, 74], предложен за пръв път от E. F. Codd в началото на седемдесетте години на миналия век и описан в серия от статии, първата от която е [99]. За тази своя разработка E. F. Codd през 1980 година получава престижната награда на американската асоциация по компютърна техника на името на А. Тюринг.

Нека е дадена фамилията от множества

$$D = \{D_1, D_2, \ldots, D_m\},$$

които ще наричаме *домейни*. Да разгледаме декартовото произведение

$$W = D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n} = \\ = \{\langle x_1, x_2, \ldots, x_n\rangle \mid x_{i_k} \in D_{i_k},\ D_{i_k} \in D,\ k = 1, 2, \ldots, n\}$$

Възможно е за някои s и t да е изпълнено $D_{i_s} = D_{i_t}$. Всяко подмножество

$$\rho \subseteq W$$

се нарича *n-арна релация* над D. От практическа гледна точка интерес представляват крайните релации, т.е. всевъзможните крайни подмножества на W и по тази причина в настоящата работа под „релация" ще разбираме „крайна релация" (освен, ако изрично не е подчертано противното), независимо, че се допуска множествата D_i, $i = 1, 2, \ldots m$ евентуално да бъдат безкрайни (например безкрайни множества от реални числа).

Да означим с R множеството от всички n-арни релации образувани от n, $n = 1, 2, \ldots$ (не непременно различни) множества (домейни) от фамилията D.

Нека $\rho \in R$ и нека $r^i = \langle r_1^i, r_2^i, \ldots, r_n^i\rangle$ е i-я елемент на ρ. Тогава r^i се нарича i-ти *запис* на релацията ρ. В случая i е индекс на елемент, а не степенен показател. Компонентата r_j^i на r^i се нарича *стойност на j-я атрибут в i-я запис на ρ*. Стойностите на j-тите атрибути на всички записи на ρ образуват j-то *поле* на ρ. От дефиницията на понятието поле следва, че елементите на j-то поле могат да приемат стойности от един единствен домейн $D_{i_j} \in D$.

В множеството R от всички релации над D при определени условия е възможно дефинирането на различни операции – обединение, сечение, разлика, допълнение, проекция, композиция, индексация, сортировка и т.н. Между отделните атрибути са възможни и съществуването на връзки, отговарящи на известни условия. По такъв начин R заедно с въведените операции и връзки между атрибутите се превръща в алгебра, която се нарича релационна алгебра. Релационната алгебра е в основата на релационния модел на данните. Системите за управление на бази от данни имащи в основата си релационния модел се наричат още бази от данни от релационен тип. Основни сведения от областта на теорията на релационната алгебра могат да се получат в [18, 142].

Всяка релация $\rho \in R$ визуално може да се представи във вид на двумерна правоъгълна таблица, в която i-ят ред представлява i-я запис, а j-ят стълб представлява j-тото поле на ρ. Това представяне е взаимно еднозначно. Поради тази причина и за по-лесното възприемане от всеки потребител на основните понятия от релационната алгебра в повечето програмни системи за управление на бази от данни от релационен тип се говори и оперира основно с понятието *таблица* като синоним на понятието релация над фамилия от домейни [52, 74, 75, 121], независимо, че от практическа гледна точка видовете таблици могат да бъдат много

повече и не всяка таблица може да представлява конкретна релация, т.е. не всеки вид таблица може да бъде по какъвто и да е начин използувана в една система за бази данни от релационен тип. Например, календарът е вид правоъгълна таблица, която в общия случай не може да се отъждестви с дефинираното по-горе понятие релация.

3.6. Множества и релации над тях, необходими при създаването на личностни психологически тестове

За създаването на един личностен психологически тест, проведен с помощта на компютър са необходими да се уточнят и дефинират следните множества:

3.6.1. Множество Qst

съставено от въпроси или твърдения, които психологът – изследовател предоставя за отговор или мнение на изследваното лице, или група от хора. Елементите на Qst са айтемите на теста.

3.6.2. Множество $Z_m = \{1, 2, \ldots, m\}$,

където $m = |Qst|$ е равно на броя на елементите на множеството Qst. Всеки елемент на Z_m представлява уникален номер на айтемите от множеството Qst.

За психолога е от особено значение редът на представяне на айтемите, тъй като последователността на обсъжданите айтеми влияе на отговорите и следователно този ред има значение при оформянето на окончателния извод и интерпретация на резултатите от теста. В този смисъл възниква и необходимостта от множеството Z_m.

3.6.3. Релация $\varphi \subset Qst \times Z_m$

Между множествата Qst и Z_m съществува взаимно еднозначно (биективно) изображение $\varphi \subset Qst \times Z_m$, като авторът на психологическия тест за всяко $q \in Qst$ много внимателно трябва да определи образа $k = \varphi(q)$ на елемента q при изображението φ. В случая k представлява номер на айтема q и определя реда на представянето на айтемите пред изследваното лице. Както бе подчертано по-горе определянето на номера на всеки айтем е от значение за окончателния извод и е от компетенцията на психолога – автор на теста. Освен това номерът не може да излезе извън интервала от естествени числа $[1, m]$. В този смисъл при една система за управление на база от данни, необходима при създаването на компютърен личностен психологически тест е необходимо при изтриване на айтеми, или при вмъкването между два айтема на нов айтем да се предвиди автоматично преномериране. В редица системи за управление на база от данни е предвидено създаване на *автоматично номеруемо поле* (auto increment field) играещо роля на *първичен ключ* (primary key). За полета от такъв тип след посочените по-горе операции деление и вмъкване на айтем не се извършва преномерация, нещо повече промяната на стойност на първичен ключ е недопустима. В този смисъл полето Z_m съществено се различава от автоматично номеруемото поле.

3.6.4. Множество Ans

от възможни отговори на айтемите (виж страница 60).

3.6.5. Множество Ctg

от *психологически категории* (*личностни характеристики*), които са предмет на анализ и оценка, касаещо изследваното лице или група от хора с помощта на айтемите от множеството Qst и избрания конкретен отговор. Не е задължително всеки айтем да има отношение към дадена психологическа категория.

3.6.6. Крайната фамилия $T = \{T_c \subseteq Qst \mid c \in Ctg\}$ от подмножества на Qst

При това елементът $q \in Qst$ принадлежи на подмножеството T_c тогава и само тогава, когато айтемът q има отношение към психологическата категория категория $c \in Ctg$ при съответното психологическо изследване. Очевидно $\bigcup\limits_{c \in Ctg} T_c \subseteq Qst$ и ако $q \in Qst$ и $q \notin \bigcup\limits_{c \in Ctg} T_c$, то айтемът q не влияе на цялостното изследване и може да отпадне.

3.6.7. Множество от реални вектори Scl

Нека $Ans = \{a_1, a_2, \ldots, a_k\}$, т.е. $|Ans| = k$. Разглеждаме множеството Scl състоящо се от реални k-мерни вектори. Ако $S = [s_1, s_2, \ldots, s_k]$ е вектор от множеството Scl, то смисълът на всеки елемент s_i на S от практическа гледна точка представлява оценъчна числова стойност за психологическите категории от множеството Ctg при условие, че изследваното лице отговоря на айтема $q \in Qst$ с отговор или твърдение $a_i \in Ans$, $i = 1, 2, \ldots, k$. Много често тези оценъчни стойности са 1 или 0. Например, ако $|Ans| = 2$ и приемем, че при всеки положителен отговор оценъчната стойност е равна на 1 и при отрицателен отговор – на 0 за всеки айтем $q \in Qst$, то при сумиране лесно можем да пресметнем общия брой на положителните отговори.

3.6.8. Множество от функции $f_c : Qst \to Scl$, където $c \in Ctg$

За всяка психологическа категория $c \in Ctg$ се определя по една функция

$$f_c : Qst \to Scl$$

с помощта на които психологът оценява изследваното лице по отношение на категорията $c \in Ctg$. Такава функция се нарича *скала за психологическата категория* $c \in Ctg$. Много честа практика е името на скалата да съвпада с името на психологическата категория. Множеството от функции

$$\{f_c \mid c \in Ctg\},$$

състоящо се от скалите на разглежданите психологически категории към които дадения личностен психологически тест има отношение се нарича *скала на теста*.

Как да се оценяват различните айтеми при съответните отговори, т.е. за всяко $c \in Ctg$, как се дефинира всяка от функциите $f_c : Qst \to Scl$ и как това влияе на общата оценка от провежданите тестове на практика се определя след достатъчно мащабни психологически и статистически изследвания [42, 57, 59, 63, 88, 144]. Ако бъдат използувани готови тестове, които ще се използват на ново място или върху нови обекти за изследване, различаващи се по пол, възраст, религия и т.н. от изследвани да момента лица, то е необходимо тестовете да бъдат адаптирани към специфичните условия на приложение [148]. С други думи при адаптацията на даден тест става и предефиниране на функциите f_c.

3.6.9. Релация $Bnd = \{\langle c, q, f_c(q)\rangle \mid c \in Ctg, q \in Qst\} \subset Ctg \times Qst \times Scl$,

където Ctg, Qst и Scl са дефинираните съответно в раздели 3.6.5, 3.6.1 и 3.6.7 множества, а f_c, $c \in Ctg$ е дефинираната в 3.6.8 функция (скала за психологическата категория c). Смисълът на всеки запис в Bnd е за всяка категория $c \in Ctg$, какви оценки (брой точки) се дават, ако изследваното лице даде отговор $a \in Ans$ на айтема $q \in Qst$ съгласно скалата $f_c(q)$.

3.6.10. Множество от числови интервали

Нека $c \in Ctg$ и нека

$$m_c = \min \sum_{q \in Qst} \mu\left(f_c\left(q\right)\right),$$

където

$$\mu \ : \ Scl \to \mathbb{R}, \quad \mu([x_1, x_2, \ldots, x_k]) = \min(x_1, x_2, \ldots, x_k).$$

С други думи m_c е минималната оценъчна стойност (минимален брой точки), която е възможно да се получи при произволно тестиране, касаещо психологическата категория $c \in Ctg$ и в зависимост от съответната скала $f_c \ : \ Qst \to Scl$.

Аналогично определяме максималната оценъчна стойност

$$M_c = \max \sum_{q \in Qst} \nu\left(f_c\left(q\right)\right),$$

където

$$\nu \ : \ Scl \to \mathbb{R}, \quad \nu([x_1, x_2, \ldots, x_k]) = \max(x_1, x_2, \ldots, x_k).$$

Очевидно $m_c < M_c$. Разглеждаме крайната редица от числа

$$m_c = bc_0 < bc_1 < \cdots < bc_{l_c - 1} < bc_{l_c} = M_c.$$

С означението l_c подчертаваме факта, че броят на членовете на редицата зависи от категорията $c \in Ctg$. Интервалът $[m_c, M_c]$ разделяме на подинтервалите $[bc_0, bc_1]$, $(bc_1, bc_2]$, $(bc_2, bc_3]$,..., $(bc_{l_c-1}, bc_{l_c}]$. Да означим с D_c множеството от всички такива интервали $\Delta c_1 = [bc_0, bc_1]$ и $\Delta c_i = (bc_{i-1}, bc_i]$, $i = 2, 3, \ldots, l_c$, съответстващи на психологическата категория $c \in Ctg$.

Числото l_c и всеки един от интервалите Δc_i, $i = 1, 2, \ldots l_c$ също е необходимо да се получат в резултат на задълбочени психологически изследвания [148].

Означаваме с

$$D = \bigcup_{c \in Ctg} D_c$$

множеството от всички числови интервали, които получаваме по описания по-горе начин и които биха могли да се използуват при даден личностен психологически тест.

3.6.11. Множество от текстове

За всеки от дефинираните в 3.6.10 интервал Δc_i, $i = 1, 2, \ldots, l_c$, $c \in Ctg$ създаваме текст tc_i, $i = 1, 2, \ldots, l_c$, който би съответствал на текста, даден от професионален психолог относно психологическото състояние на изследваното лице по отношение на психологическата категория $c \in Ctg$, в случай че *сумарната оценка* (сумата от всички конкретни оценки за съответния айтем при проведения тест съгласно скалата f_c) попадне в интервала Δc_i. Така за всяко $c \in Ctg$ получаваме множеството от текстове

$$T_c = \{tc_i \mid i = 1, 2, \ldots, l_c\}.$$

3.6.12. Множество от интерпретации

Нека $c \in Ctg$. От казаното по-горе следва, че съществува функционална зависимост

$$g_c \ : \ D_c \to T_c.$$

Нека

$$G_c = \{\langle \Delta c_i, tc_i \rangle \mid g_c(\Delta c_i) = tc_i, \ i = 1, 2, \ldots, l_c\}$$

е съответната бинарна релация, съответстваща на тази зависимост (график на функцията g_c). Функционалната зависимост $g_c : D_c \to T_c$ се нарича *интерпретация* на теста, относно психологическата категория $c \in Ctg$.

Означаваме с

$$T = \bigcup_{c \in Ctg} T_c$$

множеството от всевъзможните интерпретации, които могат да се дадат от психолога изследовател след провеждането на даден личностен психологически тест.

3.6.13. Релация

$$Ntp = \{\langle c, \Delta c_i, g_c(\Delta c_i)\rangle \mid c \in Ctg, \ \Delta c_i \in D_c\} \subset Ctg \times D \times T,$$

където D_c и D са разгледаните в 3.6.10 множества от числови интервали, а g_c, $c \in Ctg$ и T са съответно функциите и множеството от интерпретации, дефинирани в 3.6.12.

Смисълът на всеки запис в Ntp е следният: Ако на дадена психологическа категория $c \in Ctg$ при провеждане на теста изследваното лице събере r на брой точки (получена оценъчна стойност за категорията), то проверяваме на кой числов интервал $\Delta c_i \in D_c$, $c \in Ctg$ принадлежи r. Тогава правим съответната експертна интерпретация на резултатите от теста в съответствие с категорията $c \in Ctg$ и дефинираната в 3.6.12 функция $g_c : D_c \to T_c$.

3.7. Заключение към трета глава

Трябва да отбележим повишения интерес на студентите, както от специалностите ”информатика” и ”математика и информатика”, така и от специалност ”психология” към описаната в тази глава тематика. Задаването на теми от тази област за курсови задачи и дипломни работи, свързани с компютърното администриране в психологията, интернет тестирането [47] и занимателната математика се приемат с определен интерес [17, 21, 32, 45, 46, 50, 53, 61]. Тук ние посочваме само най-добрите от тях. Четирима студенти (Ивелина Пенева, Богдана Кирилиева-Шиварова, Гергана Праскова и Иван Петров) докладваха своите разработки от тази област на международни научни форуми [47, 147, 153, 194]. Бяха защитени две докторски дисертации, свързани с тази тематика, където авторът на настоящия дисертационен труд К. Йорджев бе научен ръководител или консултант [32, 46].

За реализацията на идеите за компютърно администриране на психологическите изследвания, съвместно със студенти–дипломанти и докторанти към ЮЗУ ”Неофит Рилски”, Благоевград, създадохме различни реализации на софтуер–генератор на психологически тестове [21, 32, 45, 46, 50, 53, 61]. Това на практика е помощник на психолога-експериментатор при създаването компютъризирани личностни тестове. Тези програми работят като тестов редактор с улеснен интерфейс

за въвеждане на необходимите компоненти на множествата и релациите, описани в раздел 3.6, както и вградена система за управление на базата данни свързани с конкретен новосъздаден тест.

В тази глава посочихме само някой от приложенията на теория на множествата в програмирането. Показахме, че освен за чисто комбинаторни приложения [195] множественият подход може с успех да се прилага и при математическото моделиране и компютърното администриране в психологията и занимателната математика [191]. Тъй като не всеки език (като напр. Паскал) поддържа вградени средства за работа с множества, то на помощ ни идват средствата на обектно-ориентираното програмиране за създаване на специални класове за работа с множества. И тук побитовите операции улесняват значително решаването на тази задача.

Глава 4

Комбинаторни задачи над бинарни матрици

Резултатите от раздел 4.2 са публикувани в [24, 173].
Резултатите от раздел 4.3 са публикувани в [184].
Резултатите от раздел 4.4 са публикувани в [186, 188].
Резултатите от раздел 4.5 са публикувани в [26, 136, 137].
Резултатите от раздел 4.6 са публикувани в [176, 177].
Резултатите от раздел 4.7 са публикувани в [30, 177].
Резултатите от раздел 4.8 са публикувани в [189].

4.1. Предварителни сведения

Бинарна (или *булева*, или (0,1)-матрица) се нарича матрица, чийто елементи принадлежат на множеството $\mathfrak{B} = \{0, 1\}$. С $\mathfrak{B}_n$ означаваме множеството на всички квадратни $n \times n$ бинарни матрици.

Някои комбинаторни задачи, свързани с бинарни матрици и с приложение в текстилната техника, решихме в глава 2.

Използвайки означенията от [76], Λ_n^k-матрици ще наричаме всички квадратни бинарни матрици от n-ти ред, във всеки ред и всеки стълб на които съществуват точно k единици.

Да означим с $\lambda(n, k)$ броя на всички Λ_n^k-матрици.

Лесно се доказва следната добре известна формула:

$$\lambda(n, 1) = n! \tag{4.1.1}$$

В [65] е показана формулата:

$$\lambda(n, 2) = \sum_{2x_2+3x_3+\cdots+nx_n=n} \frac{(n!)^2}{\prod_{r=2}^{n} x_r!(2r)^{x_r}} \tag{4.1.2}$$

Една от първите рекурсивни формули за изчисляване на $\lambda(n, 2)$ е дадена в [87] (виж също [117, стр. 763]):

$$\begin{aligned} &\lambda(n, 2) = \frac{1}{2}n(n-1)^2\left[(2n-3)\lambda(n-2, 2) + (n-2)^2\lambda(n-3, 2)\right],\ n \geq 4 \\ &\lambda(1, 2) = 0,\ \lambda(2, 2) = 1,\ \lambda(3, 2) = 6 \end{aligned} \tag{4.1.3}$$

В [115] е посочена следната рекурсивна формула за намиране на $\lambda(n, 2)$:

$$\begin{aligned} &\lambda(n, 2) = (n-1)n\lambda(n-1, 2) + \frac{(n-1)^2 n}{2}\lambda(n-2, 2), \quad n \geq 3 \\ &\lambda(1, 2) = 0,\ \lambda(2, 2) = 1 \end{aligned} \tag{4.1.4}$$

В [24] е изведена следната рекурсивна система за намиране на $\lambda(n, 2)$:

$$\left|\begin{array}{l} \lambda(n+1, 2) = n(2n-1)\lambda(n, 2) + n^2\lambda(n-1, 2) - \pi(n+1)\ ; \quad n \geq 2 \\ \pi(n+1) = \frac{n^2(n-1)^2}{4}[8(n-2)(n-3)\lambda(n-2, 2) + \\ \qquad +(n-2)^2\lambda(n-3, 2) - 4\pi(n-1)]\ ; \quad n \geq 4 \\ \lambda(1, 2) = 0, \quad \lambda(2, 2) = 1, \quad \pi(1) = \pi(2) = \pi(3) = 0, \quad \pi(4) = 9 \end{array}\right. \tag{4.1.5}$$

като $\pi(n)$ посочва броя на специален клас Λ_n^2-матрици.

В [161] е посочена следната явна формула за изчисляване на $\lambda(n,3)$

$$\lambda(n,3) = \frac{n!^2}{6^n} \sum \frac{(-1)^\beta(\beta+3\gamma)!2^\alpha 3^\beta}{\alpha!\beta!\gamma!^2 6^\gamma} \tag{4.1.6}$$

където сумата е по всички $\dfrac{(n+2)(n+1)}{2}$ на брой решения в неотрицателни цели числа на уравнението $\alpha+\beta+\gamma = n$. Както е отбелязано и в [160] формулата (4.1.6) не ни дава голяма възможност да изследваме поведението на $\lambda(n,3)$.

Нека n е цяло положително число. С $[n]$ означаваме множеството

$$[n] = \{1, 2, \ldots, n\}.$$

Нека P_{ij}, $1 \leq i,j \leq n$ са n^2 на брой квадратни $n \times n$ матрици, елементите на които принадлежат на множеството $[n^2] = \{1, 2, \ldots, n^2\}$. Тогава $n^2 \times n^2$ матрицата

$$P = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix}$$

се нарича *Судоку матрица*, ако всеки нейн ред, всеки нейн стълб, както и всяка подматрица P_{ij}, $1 \leq i,j \leq n$ образуват пермутации на елементите на множеството $[n^2]$, т.е. всяко число $s \in \{1, 2, \ldots, n^2\}$ се среща точно по веднъж във всеки ред, всеки стълб и всяка подматрица P_{ij}. Подматриците P_{ij} ще наричаме блокове на P.

Популярната главоблъсканица Судоку и алгоритъм за нейното решаване с прилагане на теоретико-множествен подход е разгледана в глава 3. Там са обсъдени и някои методически аспекти в курсовете по програмиране. В тази глава ще продължим с изследвания в тази област.

Една квадратна бинарна матрица се нарича *пермутационна матрица*, ако във всеки ред и във всеки стълб има точно по една 1.

Нека n е положително цяло число и нека $A \in \mathfrak{B}_{n^2}$ е $n^2 \times n^2$ бинарна матрица. С помощта на $n-1$ хоризонтални и $n-1$ вертикални линии A е разбита на n^2 непресичащи се квадратни $n \times n$ подматрици A_{kl}, $1 \leq k,l \leq n$, т.е.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}, \tag{4.1.7}$$

Подматриците A_{kl}, $1 \leq k,l \leq n$ ще наричаме блокове.

Матрицата $A \in \mathfrak{B}_{n^2}$ се нарича S-пермутационна, ако във всеки ред, всеки стълб и всеки блок на A има точно една 1. Означаваме със Σ_{n^2} множеството от всички $n^2 \times n^2$ S-пермутационни матрици.

Както е доказано в [103]

$$|\Sigma_{n^2}| = (n!)^{2n} \tag{4.1.8}$$

В следващите раздели, съответно на страници 97 и 104 ще дадем нови доказателства на този факт.

Дефиниция 4.1.1. Две бинарни матрици $A = (a_{ij}) \in \mathfrak{B}_m$ и $B = (b_{ij}) \in \mathfrak{B}_m$ ще наричаме *непресичащи се* (*disjoint*), ако не съществуват елементи с еднакви индекси a_{ij} и b_{ij} такива, че $a_{ij} = b_{ij} = 1$, т.е. ако $a_{ij} = 1$, то $b_{ij} = 0$ и ако $b_{ij} = 1$, то $a_{ij} = 0$, $1 \leq i,j \leq m$.

Понятието S-пермутационна матрица е въведено от Geir Dahl в [103] във връзка с популярната главоблъсканица Судоку.

Очевидно една квадратна $n^2 \times n^2$ матрица M с елементи от $[n^2] = \{1, 2, \ldots, n^2\}$ е Судоку матрица тогава и само тогава, когато съществуват Σ_{n^2} матриците

$$A_1, A_2, \ldots, A_{n^2},$$

всеки две от които са непресичащи се по между си и такива че M може да се представи във вида

$$M = 1 \cdot A_1 + 2 \cdot A_2 + \cdots + n^2 \cdot A_{n^2} \tag{4.1.9}$$

За всички недефинирани понятия и означения, свързани с матричната алгебра и комбинаторика над бинарни матрици, както и за недоказани тук известни твърдения, препоръчваме книгите [9, 55, 66, 77, 84].

4.2. Върху един открит проблем над бинарни матрици

Да разгледаме следната не решена все още в общия случай задача:

Проблем 4.2.1. *Да се намери броят на всички $n \times n$ бинарни матрици с точно k на брой единици на всеки ред и всеки стълб, т.е. броят $\lambda(n, k)$ на всички Λ_n^k-матрици.*

Проблем 4.2.1 не е решен в общия случай, т.е. не е известна до този момент формула за намиране на броя на Λ_n^k-матриците при всяко n и k. Известни са формули изчисляващи функцията $\lambda(n, k)$ за всяко n, но при сравнително малки стойности на k и по-точно за $k = 1$, $k = 2$ и $k = 3$ описани в раздел 4.1. За $k > 3$ на нас не ни е известна формула за изчисляване на функцията $\lambda(n, k)$.

4.2.1. Едно разбиване на множеството Λ_n^k

Формулата (4.1.4), показана на страница 70, се появява за първи път в работата [115] и е изведена по начин, който е приложим единствено за намиране на броя на Λ_n^2-матриците. Предложеният от нас метод, който ще изложим в раздел 4.2.2 и който на практика се явява ново доказателство на формула (4.1.4) ще ни доближи до намирането на аналогични формули за по-големи стойности на k. В случая k показва броя на единиците във всеки ред и всеки стълб на съответните квадратни матрици.

Да въведем означенията:

$$\Lambda_n^{k+} = |\{A = [a_{ij}] \in \Lambda_n^k |\ a_{n\,n} = 1\}| \tag{4.2.1}$$

$$\Lambda_n^{k-} = |\{A = [a_{ij}] \in \Lambda_n^k |\ a_{n\,n} = 0\}| \tag{4.2.2}$$

Очевидно

$$\Lambda_n^{k+} \cap \Lambda_n^{k-} = \emptyset \quad \text{и} \quad \Lambda_n^{k+} \cup \Lambda_n^{k-} = \Lambda_n^k, \tag{4.2.3}$$

т.е $\{\Lambda_n^{k+}, \Lambda_n^{k-}\}$ представлява разбиване на множеството Λ_n^k.

Полагаме:

$$\lambda^+(n, k) = |\Lambda_n^{k+}| \tag{4.2.4}$$

$$\lambda^-(n, k) = |\Lambda_n^{k-}| \tag{4.2.5}$$

Теорема 4.2.2. *В сила е равенството*

$$\lambda^-(n,k) = \frac{n-k}{k}\lambda^+(n,k), \tag{4.2.6}$$

където $\lambda^+(n,k)$ *и* $\lambda^-(n,k)$ *се задават съответно с формули (4.2.4) и (4.2.5).*

Доказателство. Нека A и B са Λ_n^k-матрици. Ще казваме, че A и B са ρ-еквивалентни ($A\rho B$), ако след премахването на стълбовете с единици на последно място (т.е. в n-ти ред) получаваме еднакви матрици от ред $n\times(n-k)$. Очевидно ρ е релация на еквивалентност. Означаваме с ρ_A класът на еквивалентност по модул ρ съдържащ A.

Нека $A=(a_{ij})$ е Λ_n^k-матрица. Да означим с p^+ броят на всички ρ-еквивалентни на A матрици, за които елементът в най-долния десен ъгъл е равен на 1 и с p^- броят на всички ρ-еквивалентни на A матрици, за които този елемент е равен на 0. Нека $j_1, j_2, ..., j_k$ са номерата на вектор-стълбовете на матрицата A с единица на последно място. Множеството $J=\{j_1, j_2, ..., j_k\}$ разбиваме на подмножества J_r, $r=1,2,\dots s$, такива че j_r и j_t попадат в едно и също подмножество тогава и само тогава, когато са номера на два равни стълба. Лесно се вижда, че $J=\bigcup_{r=1}^{s} J_r$ и $J_u\cap J_v=\emptyset$ за $u\neq v$. Полагаме $k_r=|J_r|$, $r=1,2,\dots,s$. Очевидно

$$k_1+k_2+\dots+k_s=k. \tag{4.2.7}$$

Нека C е матрица от ред $n\times(n-k)$, която се получава от A, зачертвайки стълбовете с номера $j_1, j_2, \dots, j_k$. Тогава с помощта на различните начини на добавяне на нови стълбове към C ще получим всички елементи на множеството ρ_A. Най-напред на произволно място да добавим k_1 равни по между си стълбове, които от своя страна са равни на стълбовете на матрицата A с номера, принадлежащи на множеството J_1. Това може да стане по $\binom{n-k+k_1}{k_1}$ начина. След това добавяме по $\binom{n-k+k_1+k_2}{k_2}$ възможни начина k_2 на брой еднакви стълбове, равни на стълбовете на A с номера от J_2 и т.н. Следователно

$$|\rho_A| =$$
$$= \binom{n-k+k_1}{k_1}\binom{n-k+k_1+k_2}{k_2}\cdots\binom{n-k+k_1+k_2+\dots+k_s}{k_s} =$$
$$= \binom{n}{k_s}\binom{n-k_s}{k_{s-1}}\binom{n-k_s-k_{s-1}}{k_{s-2}}\cdots\binom{n-k_s-k_{s-1}-k_{s-2}+\dots+k_2}{k_1} =$$
$$= \frac{n!(n-k_s)!(n-k_s-k_{s-1})!\cdots(n-k_s-k_{s-1}-\dots-k_2)!}{k_s!(n-k_s)!k_{s-1}!(n-k_s-k_{s-1})!\cdots k_1!(n-k_s-k_{s-1}-\dots-k_2-k_1)!} =$$
$$= \frac{n!}{k_1!k_2!\cdots k_s!(n-k)!}$$

Аналогично за p^-, като вземем в предвид, че в този случай след последния стълб на матрицата C не можем да поставяме нови стълбове (добавяните вектор стълбове с единици на последно място), получаваме:

$$p^- = \frac{(n-1)!}{k_1!k_2!\cdots k_s!(n-1-k)!}$$

За p^+ получаваме:

$$p^+ = |\rho_A| - p^- = \frac{n!}{k_1!k_2!\cdots k_s!(n-k)!} - \frac{(n-1)!}{k_1!k_2!\cdots k_s!(n-1-k)!} =$$

$$= \frac{k(n-1)!}{k_1!k_2!\cdots k_s!(n-k)!}$$

Тогава $\frac{p^-}{p^+} = \frac{n-k}{k}$, т.е. $p^- = \frac{n-k}{k}p^+$. Сумирайки по класовете на еквивалентност, получаваме и твърдението, което трябваше да докажем.

□

Вземайки предвид равенствата (4.2.1) ÷ (4.2.5) от теорема 4.2.2 непосредствено следва

Следствие 4.2.3.

$$\lambda(n,k) = \lambda^+(n,k) + \lambda^-(n,k) = \lambda^+(n,k) + \frac{n-k}{k}\lambda^+(n,k) = \frac{n}{k}\lambda^+(n,k) \qquad (4.2.8)$$

□

Теорема 4.2.2 и следствие 4.2.3 са полезни в такъв смисъл, че намирането на $\lambda^+(n,k)$ е по-лека задача от намирането на общия брой на всички Λ_n^k-матрици.

4.2.2. Друго доказателство на теоремата на I. Good и J. Grook

За да получим формула за броя на квадратните $n \times n$ бинарни матрици, съгласно следствие 4.2.3 на теорема 4.2.2 е достатъчно да намерим формула за $\lambda^+(n,2)$. Това ще направим с помощта на следната

Теорема 4.2.4. *При $n \geq 3$ е в сила равенството*

$$\lambda^+(n,2) = 2(n-1)\lambda(n-1,2) + (n-1)^2\lambda(n-2,2)$$

Доказателство. Нека $A = (a_{ij})$ е Λ_{n-1}^2-матрица. От A можем да получим Λ_n^{2+}-матрицата $B = (b_{ij})$ по следния начин: Избираме p, q, такива, че $a_{pq} = 1$. Това може да стане по $2(n-1)$ начина. Полагаме $b_{pq} = 0, b_{pn} = b_{nq} = b_{nn} = 1, b_{in} = b_{in} = b_{nj} = 0$ за $i \neq p, j \neq q$ и $b_{ij} = a_{ij}$ за $(i,j) \neq (p,q)$. Лесно се вижда, че B е Λ_n^2-матрица с единица в най-долния десен ъгъл. Освен това, от B еднозначно могат да се определят p и q и да се възстанови матрицата A. Следователно $\lambda^+(n,2) = 2(n-1)\lambda(n-1,2) + t$, където t е броят на всички Λ_n^2-матрици с единица в долния десен ъгъл, които не могат да се получат по описания току що начин. Това са Λ_n^{2+}-матриците, $B = (b_{ij})$ за които съществуват p и q, такива, че $b_{pq} = b_{nq} = b_{pn} = b_{nn} = 1$ и това са единствените единици (по две на ред и стълб) на p-я и n-я редове и q-я и n-я стълбове. Но тогава, премахвайки p-и и n-и редове и q-и и n-и стълбове, очевидно получаваме Λ_{n-2}^2-матрица. Обратно, от всяка Λ_{n-2}^2-матрица можем да получим Λ_n^{2+}-матрица, вмъквайки на p-то и n-то място нови редове и на q-то и на n-то място нови стълбове с нули навсякъде, с изключение на пресечните им точки. Тъй като p и q се менят от 1 до $n-1$ то тогава $t = (n-1)^2\lambda(n-2,2)$. Теоремата е доказана.

□

Прилагайки следствие 4.2.3 и теорема 4.2.4 непосредствено получаваме:

Теорема 4.2.5. *[115] Броят на всички $n \times n$ квадратни бинарни матрици с точно две единици на всеки ред и всеки стълб се дава с помощта на рекурентната зависимост*

$$\lambda(n,2) = (n-1)n\lambda(n-1,2) + \frac{(n-1)^2 n}{2}\lambda(n-2,2) \quad \textit{при} \quad n \geq 3$$
$$\lambda(1,2) = 0,\ \lambda(2,2) = 1$$

4.2.3. Върху броя на Λ_n^3-матриците

Целта на настоящия раздел е да се доближим до намирането на рекурсивна формула за изчисляване на $\lambda(n,3)$, с помощта на която се избягват някои неудобства при използването на формула (4.1.6) показана на страница 71.

Нека $X = [x_{ij}] \in \Lambda_n^{3+}$ и нека всички единици в последния ред и последния стълб са елементите x_{np}, x_{nq}, x_{sn}, x_{tn}, x_{nn}, където $s,t,p,q \in \{1,2,\ldots,n-1\}$, $s \neq t$, $p \neq q$. С $\widetilde{X}$ ще означаваме 2×2 подматрицата

$$\widetilde{X} = \begin{bmatrix} x_{sp} & x_{sq} \\ x_{tp} & x_{tq} \end{bmatrix}$$

Множеството Λ_n^{3+} разбиваме на следните непресичащи се подмножества:

$$\mathrm{A}_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right\}$$

$$\mathrm{B}_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} \in \left\{ \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right\} \right\}$$

$$\Gamma_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} \in \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \right\} \right\}$$

$$\Delta_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} \in \left\{ \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \right\} \right\}$$

$$\mathrm{E}_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} \in \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right\} \right\}$$

$$\mathrm{Z}_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} \in \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right\} \right\}$$

$$\mathrm{H}_n = \left\{ X \in \Lambda_n^{3+} \,\middle|\, \widetilde{X} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right\}$$

Полагаме

$$\alpha_n = |\mathrm{A}_n|$$
$$\beta_n = |\mathrm{B}_n|$$
$$\gamma_n = |\Gamma_n|$$
$$\delta_n = |\Delta_n|$$
$$\epsilon_n = |\mathrm{E}_n|$$
$$\zeta_n = |\mathrm{Z}_n|$$
$$\eta_n = |\mathrm{H}_n|$$

Очевидно

$$\gamma_n = \delta_n \tag{4.2.9}$$

и

$$\lambda^+(n,3) = \alpha_n + \beta_n + \gamma_n + \delta_n + \epsilon_n + \zeta_n + \eta_n. \tag{4.2.10}$$

Теорема 4.2.6.

$$\lambda^+(n,3) = \frac{3(n-1)(3n-8)}{2}\lambda(n-1,3) + \alpha_n + \beta_n + 2\gamma_n - \eta_n$$

Доказателство. Нека $Y = (y_{ij}) \in \Lambda^3_{n-1}$. В Y избираме две единици, нележащи на един ред или стълб и нека това са елементите y_{sp} и $y_{tq} \in Y$, $s,t,p,q \in \{1,2,\ldots,n-1\}$, $s \neq t, p \neq q$. Това може да стане по $\frac{3(n-1)[3(n-1)-5]}{2} = \frac{3(n-1)(3n-8)}{2}$ начина. Избраните единици превръщаме в нули и в Y на последно място добавяме още един ред (n-ти) и още един стълб (n-ти), така че $y_{sn} = y_{tn} = y_{np} = y_{nq} = y_{nn} = 1$ и $y_{in} = y_{nj} = 0$ за $i \notin \{s,t,n\}$, $j \notin \{p,q,n\}$. Очевидно така получената матрица принадлежи на едно от множествата E_n, Z_n или H_n.

Обратно, нека $X = (x_{ij})$ е матрица от E_n или Z_n. Тогава $\widetilde{X}$ има единствен нулев диагонал, който превръщаме в единичен и премахваме n-ти ред и n-ти стълб. По такъв начин еднозначно получаваме единствена Λ^3_{n-1}-матрица.

Нека $\widetilde{X} = [x_{ij}] \in \mathrm{H}_n$ и нека $\widetilde{X} = \begin{bmatrix} x_{sp} & x_{sq} \\ x_{tp} & x_{tq} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Избираме единия диагонал и го правим единичен. Премахваме n-ти ред и n-ти стълб. Получаваме Λ^3_n-матрица. Следователно на всяка H_n-матрица съответстват две Λ^3_n-матрици. Тогава

$$\lambda^+(n,3) = \frac{3(n-1)(3n-8)}{2}\lambda(n-1,3) - \eta_n + t$$

където $t = \alpha_n + \beta_n + \gamma_n + \delta_n$. Вземайки в предвид (4.2.9) и (4.2.10) получаваме доказателството на теоремата.

□

4.3. Конструктивно доказателство на формулата на В. Е. Тараканов

Ако M е произволно крайно множество, както обикновено $|M|$ ще означава броя на елементите на M, а с $\mathcal{P}(M)$ ще означаваме множеството от всички пермутации на елементите от M. Както е добре известно $\mathcal{P}(M) \cong \mathcal{S}_n$, където $\mathcal{S}_n$ е симетричната група. Ако $\rho \in \mathcal{S}_n$, то с $\rho(i)$ означаваме образа на $i \in [n]$ при изображението ρ.

В настоящия раздел ще разгледаме следната алгоритмична задача, при чието решение, ще използваме теория на множествата, т.е. при програмната ѝ реализация би било удобно да се използват побитови операции:

Задача 4.3.1. *Да се състави алгоритъм с помощта на който при зададено естествено число $n \geq 2$ да се получават всички Λ^2_n-матрици.*

Да отбележим, че формула 4.1.2, получена от В. Е. Тараканов и публикувана в [65] не дава решение на задача 4.3.1, а само посочва броя на обектите, които се търсят. Решение на задача 4.3.1 ще дадем с помощта на алгоритъм 4.3.2, описан подолу. Освен това формула 4.1.2 е получена от В. Е. Тараканов, разглеждайки една релация на еквивалентност в множеството от всички наредени двойки противоречиви изображения $(\rho,\sigma) \in \mathcal{S}_n \times \mathcal{S}_n$ (да припомним, че $\rho,\sigma \in \mathcal{S}_n$ се наричат *противоречиви* или *без неподвижни точки* [77], ако $\rho(i) \neq \sigma(i)$ за всяко $i = 1,2,\ldots,n$). В настоящата работа ще предложим ново конструктивно доказателство на формула 4.1.2, като предложим алгоритъм за получаване на всички Λ^2_n-матрици. При описанието на този алгоритъм ще използваме различни действия с множества. Както подчертахме в глава 3 за компютърното осъществяване на тези действия е удобно да използваме побитовите операции. Вградените в широко използваните езици за програмиране C, C++, Java побитови операции [106, 128, 156] водят до по-бързи и ефективни алгоритми за работа с множества [132, 163].

Разбиване на множеството A ще наричаме всяко множество $\{A_i\}_{i\in I}$ от подмножества на A, такива че $\bigcup\limits_{i\in I} A_i = A$, $A_i \cap A_j = \emptyset$ при $i \neq j$. Подмножествата A_i, $i \in I$ се наричат *блокове* на разбиването. Казваме, че дадено разбиване е от тип $1^{x_1}2^{x_2}\dots n^{x_n}$, ако съществуват точно x_k блока с мощност k $(k = 1, 2, \dots n)$. Както е добре известно [9, 84] съществува взаимно-еднозначно съответствие между множеството от възможните типове на разбивания на A, $|A| = n$ и множеството от всички решения на уравнението $1 \cdot x_1 + 2 \cdot x_2 + \dots + n \cdot x_n = n$.

Алгоритъм 4.3.2. *Получаване на всички Λ_n^2-матрици.*

Begin

1. За всяко решение на уравнението $2x_2 + 3x_3 + \dots + nx_n = n$ **Do begin**

2. За всяко разбиване $\mathbf{C}$ *на* $[n]$ *от тип* $1^0 2^{x_2} 3^{x_3} \dots n^{x_n}$ **Do begin**

3. За всяко разбиване $\mathbf{A}$ *на* $[n]$ *от тип* $1^0 2^{x_2} 3^{x_3} \dots n^{x_n}$ **Do begin**

4. За всяко $k = 2, 3, \dots n$, *за които* $x_k \neq 0$ **Do begin**

5. Образуваме множествата

$$\mathcal{C} := \left\{C_1^{(k)}, C_2^{(k)}, \dots, C_{x_k}^{(k)}\right\}$$

и

$$\mathcal{A} := \left\{A_1^{(k)}, A_2^{(k)}, \dots, A_{x_k}^{(k)}\right\}$$

където $C_i^{(k)} \in \mathbf{C}$, $\left|C_i^{(k)}\right| = k$, $A_i^{(k)} \in \mathbf{A}$, $\left|A_i^{(k)}\right| = k$, $i = 1, 2, \dots, x_k$;

6. За всяко $\rho \in \mathcal{S}_{x_k}$ **Do begin**

7. За всяко $i = 1, 2, \dots, x_k$ **Do begin**

8. $l := 0$; $L_i^{(k)} := \emptyset$;

% l - цяло число; $L_i^{(k)}$ - множество от цели числа %

9. $c_1 := \min\{c \mid c \in C_i^{(k)}\}$;

10. За всяка пермутация $c_2 c_3 \dots c_k \in \mathcal{P}\left(C_i^{(k)} \setminus \{c_1\}\right)$ **Do begin**

11. За всяка пермутация $a_1 a_2 \dots a_k \in \mathcal{P}\left(A_{\rho(i)}^{(k)}\right)$ **Do begin**

12. Ако $a_1 > a_2$ *То върни се в 11;*

13. $l := l + 1$; $L_i^{(k)} := L_i^{(k)} \cup \{l\}$;

14. Получаваме множеството от тримерни вектори:

$$N_i^{(k)}[l] := \left\{\begin{bmatrix} c_1 \\ a_1 \\ a_2 \end{bmatrix}, \begin{bmatrix} c_2 \\ a_2 \\ a_3 \end{bmatrix}, \dots, \begin{bmatrix} c_{k-1} \\ a_{k-1} \\ a_k \end{bmatrix}, \begin{bmatrix} c_k \\ a_k \\ a_1 \end{bmatrix}\right\};$$

15. **End** *11*

16. **End** *10*

17. **End** *7*

18. Образуваме множеството $Q_{x_k} := L_1^{(k)} \times L_2^{(k)} \times \dots \times L_{x_k}^{(k)}$;

19. За всяка x_k*-торка* $\overline{q} = \langle l_1, l_2, \dots, l_{x_k}\rangle \in Q_{x_k}$ **Do begin**

20. Образуваме множеството от тримерни вектори:

$$M[\rho; \overline{q}] := \bigcup_{i=1}^{x_k} N_i^{(k)}[l_i]$$

21. **End** *19*

22. **End** *6*

23. **End** *4*

24. За всяка $(n-1)$*-торка* $\langle \rho_2, \rho_3, \dots \rho_n\rangle \in \mathcal{S}_{x_2} \times \mathcal{S}_{x_3} \times \dots \times \mathcal{S}_{x_n}$ **Do begin**

% Ако $x_k = 0$, то по дефиниция $\mathcal{S}_{x_k} = \emptyset$ и $|\mathcal{S}_{x_k}| = 0! = 1$ %

25. За всяка $(n-1)$-торка $\langle \overline{q_2}, \overline{q_3}, \ldots, \overline{q_n} \rangle \in Q_{x_2} \times Q_{x_3} \times \cdots \times Q_{x_n}$ **Do begin**

% Ако $x_k = 0$, то по дефиниция $Q_{x_k} = \emptyset$ и $|Q_{x_k}| = 0! = 1$ %

26. Образуваме множеството от тримерни вектори

$$R = \bigcup_{k=2}^{n} M[\rho_k; \overline{q_k}] \ ;$$

% Ако $x_k = 0$, то по дефиниция $M[\rho_k; q_k] = \emptyset$ и $|M[\rho_k; q_k]| = 0! = 1$ %

27. От R еднозначно получаваме Λ_n^2-матрицата α, такава,че ако $\begin{bmatrix} c \\ u \\ v \end{bmatrix} \in R$, то в α на c-ти стълб единствените две единици се намират на редове с номера u и v;

28. **End** *25*

29. **End** *24*

30. **End** *3*

31. **End** *2*

32. **End** *1*

End.

Теорема 4.3.3. *С помощта на алгоритъм 4.3.2 се получават всички Λ_n^2-матрици, при това при всяко изпълнение на оператор 27 се получава различна Λ_n^2-матрица.*

Доказателство.Нека $w' = \begin{bmatrix} c' \\ u' \\ v' \end{bmatrix}$ и $w'' = \begin{bmatrix} c'' \\ u'' \\ v'' \end{bmatrix}$ са два вектора от множество R, получавано при изпълнение на оператор 26 от алгоритъм 4.3.2. Ако съществуват i, k и l, такива че $w', w'' \in N_i^{(k)}[l]$ (т.е. w' и w'' се получават при еднократно изпълнение на оператор 14), то съгласно оператори 10 и 14 $c' \neq c''$. Съгласно оператори 20 и 26 не е възможно да съществуват i, k_1, k_2, l, t $l \neq t$, такива че $w' \in N_i^{(k_1)}[l]$, $w'' \in N_i^{(k_2)}[t]$ и при това w' и w'' да са от едно и също множество R, получавано при изпълнение на оператор 26. Ако $w' \in N_i^{(k_1)}[l]$, $w'' \in N_j^{(k_2)}[t]$ $i \neq j$, то c' и c'' са от различни блокове на разбиването $\mathbf{C}$, т.е. отново $c' \neq c''$. Освен това очевидно за всяко $c \in [n]$ съществува блок $C_i^{(k)} \in \mathbf{C}$, такъв че $c \in C_i^{(k)}$ и съгласно оператори 10 и 14 съществуват естествено число l и $u, v \in [n]$, такива че $\begin{bmatrix} c \\ u \\ v \end{bmatrix} \in N_i^{(k)}[l]$. Следователно първите елементи на всеки вектор в множеството R при еднократно изпълнение на оператор 26 коректно посочва номер на стълб в матрица.

Нека $a \in [n]$. Тъй като оператори 11,12,14 и 26 са вътрешни за циклите с номера 2 и 3, то съгласно тези оператори съществуват единствени тримерни вектори w' и w'' $w' \neq w''$ от множеството R при еднократно изпълнение на оператор 26, такива че a е на втора позиция в w' и на трета позиция в w''. Следователно в матриците, получаващи се в оператор 27 има точно две единици на всеки ред и съгласно оператор 27 точно две единици на всеки стълб.

Допускаме, че съществуват две множества R_1 и R_2, получавани при различни изпълнения на оператор 26, които са равни помежду си. Тогава $R_1 = R_2 = N_1^{(2)}[l_1] \cup \cdots \cup N_{x_2}^{(2)}[l_{x_2}] \cup \cdots \cup N_{x_n}^{(n)}[l_{x_n}]$, което е невъзможно, имайки предвид начина на работа на оператори 14, 20 и 26 и с какви обекти работят при всяко тяхно изпълнение. Следователно при всяко изпълнение на оператор 27 се получава различна Λ_n^2-матрица.

Да означим с T множеството от матриците, получавани при работата на оператор 27. Тъй като в T няма повтарящи се елементи, то $T \subseteq \Lambda_n^2$.

Нека α е произволна Λ_n^2-матрица. Ще покажем, че $\alpha \in T$. Нека в началото α няма маркирани редове и стълбове. В α построяваме *маршрут*, започвайки от горната единична клетка на най-левия немаркиран стълб на матрицата, продължаваме до следващата по-долу 1 в стълба, след което достигаме до втората единица от същия ред, след това до втората единица от същия стълб и т.н. докато се върнем до първоначалната клетка. Движейки се по този маршрут маркираме редовете и стълбовете по които минаваме, като минавайки по c-ти стълб получаваме вектора $\begin{bmatrix} c \\ u \\ v \end{bmatrix}$ (първо сме преминали през единичната клетка на u-ти ред, след което през единичната клетка на v-ти ред). Продължаваме същия процес с немаркираните редове и стълбове, докато маркираме цялата матрица. Нека c_1 е минималното число, измежду числата, стоящи на първа позиция на векторите съответстващи на даден маршрут. Тъй като началото на цикъла започва от горната единична клетка на най-левия стълб от маршрута, то за вектора $\begin{bmatrix} c_1 \\ u \\ v \end{bmatrix}$ е изпълнено $u < v$. Следователно за всеки маршрут получаваме множество от вектори, които се получават и в оператор 14. Обединявайки множествата от всички вектори, които се получават от маршрути в α с еднаква дължина получаваме множество, което се получава и в оператор 20. Обединявайки всички такива множества, съответстващи на дадена матрица, получаваме множество, което се получава и в оператор 26. Следователно $|\Lambda_n^2| \leq |T|$, откъдето следва и твърдението на теоремата.

□

С помощта на алгоритъм 4.3.2 ще докажем и следващата теорема, доказана в [65] по различен начин.

Теорема 4.3.4. [65] *За броя на всички Λ_n^2-матрици е в сила формулата:*

$$\lambda(n,2) = \sum_{2x_2+3x_3+\cdots+nx_n=n} \frac{(n!)^2}{\prod_{k=2}^{n} x_k!(2k)^{x_k}}$$

Доказателство. Да означим с ζ_j броя на повторенията на цикъл с номер j в алгоритъм 4.3.2. Тъй като алгоритъм 4.3.2 е построен така, че всеки път при изпълнение на оператори 14, 20, 26 и следователно и оператор 27 (следва от теорема 4.3.3) се получават различни обекти, то

$$\lambda(n,2) = \zeta_1\zeta_2\zeta_3\zeta_{24}\zeta_{25}. \tag{4.3.1}$$

Лесно се вижда, че

$$\zeta_1 = \sum_{2x_2+3x_3+\cdots+nx_n=n} 1$$

Съгласно една добре известна формула (виж напр. 3.15(ii) от [84]) имаме

$$\zeta_2 = \zeta_3 = \frac{n!}{\prod_{k=2}^{n} x_k!(k!)^{x_k}}$$

За ζ_{24} и ζ_{25} получаваме съответно:

$$\zeta_{24} = \prod_{k=2}^{n} |\mathcal{S}_{x_k}| = \prod_{k=2}^{n} x_k!$$

$$\zeta_{25} = \prod_{k=2}^{n} |Q_{x_k}| = \prod_{k=2}^{n} \left(\prod_{j=1}^{x_k} \left| L_j^{(k)} \right| \right) = \prod_{k=2}^{n} \left(\prod_{j=1}^{x_k} \zeta_{10}\zeta_{11} \right) =$$
$$= \prod_{k=2}^{n} \left(\prod_{j=1}^{x_k} (k-1)! \cdot \frac{k!}{2} \right) = \prod_{k=2}^{n} \frac{(k!)^{2x_k}}{(2k)^{x_k}}$$

Замествайки в 4.3.1 получаваме и твърдението на теоремата.

□

Сравнявайки доказателството на теорема 4.3.4 с формула 4.1.2 се убеждаваме във високата ефективност на алгоритъм 4.3.2. Това на практика е доказателство на факта, че алгоритъм 4.3.2 работи точно толкова, колкото е необходимо.

4.4. Функцията $\mu(n,k) = \left|\Lambda^k_{n/\sim}\right|$ и числата на Фибоначи

Нека n и k са цели положителни числа. Означаваме с $\mu(n,k)$ броя на класовете на еквивалентност в множеството Λ^k_n относно описаната в дефиниция 1.3.4 (стр. 16) релация на еквивалентност, т.е.

$$\mu(n,k) = \left|\Lambda^k_{n/\sim}\right|, \tag{4.4.1}$$

където с Λ^k_n означаваме множеството от всички квадратни $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб.

На нас не ни е известна формула за изчисляване на тази функция и смятаме е че това е открит за науката проблем, който е интересен за разрешаване от математическа гледна точка.

Както отбелязахме в раздел 1.3.1 броя на класовете на еквивалентност в дадено множество от бинарни матрици, което е затворен клас относно релацията $\sim$ (какъвто очевидно е Λ^k_n) е равно на броя на **каноничните матрици** (дефиниция 1.3.7 на стр. 19) в това множество.

Очевидно, при $k = 0$, нулевата $n \times n$ матрица $[0]_{n\times n}$ е единствената матрица от множеството Λ^0_n. При $k = n$, съществува единствена $n \times n$ бинарна матрица от Λ^n_n и това е матрицата $[1]_{n\times n}$, всички елементи на които са равни на 1. Следователно

$$\mu(n,0) = \mu(n,n) = 1. \tag{4.4.2}$$

При $k = 1$ ако $A \in \Lambda^1_n$ е канонична матрица, то очевидно $r(A) = \langle 1, 2, 4, \ldots, 2^{n-1} \rangle$, т.е. A е бинарна матрица с 1 на втория (не главния) диагонал и 0 на всякъде другаде. При $k = n-1$, ако $A \in \Lambda^{n-1}_n$ е канонична матрица, то лесно се вижда, че A е матрица с 0 на главния диагонал и 1 на всякъде другаде. Следователно

$$\mu(n,1) = \mu(n,n-1) = 1. \tag{4.4.3}$$

Както е добре известно [93, 130], редицата на Фибоначи $\{f_n\}_{n=0}^{\infty}$ се дефинира с помощта на следната рекурентна зависимост:

$$f_0 = f_1 = 1, \qquad f_n = f_{n-1} + f_{n-2} \quad \text{за} \quad n = 2, 3, \ldots \tag{4.4.4}$$

Лема 4.4.1. *Нека $A = [\alpha_{i\,j}] \in \Lambda^k_n$ е канонична матрица. Тогава:*

$$\alpha_{1\,1} = \alpha_{1\,2} = \cdots = \alpha_{1\,n-k} = 0, \quad \alpha_{1\,n-k+1} = \alpha_{1\,n-k+2} = \cdots = \alpha_{1\,n} = 1,$$

$$\alpha_{1\,1} = \alpha_{2\,1} = \cdots = \alpha_{n-k\,1} = 0, \quad \alpha_{n-k+1\,1} = \alpha_{n-k+2\,1} = \cdots = \alpha_{n\,1} = 1,$$

Доказателство. Следва непосредствено от твърдение 1.3.3 (стр. 16) и факта, че всяка канонична матрица е и полуканонична.

□

Следствие 4.4.2. *Ако A е канонична матрица принадлежаща на множеството Λ_n^k, $r(A) = \langle x_1, x_2, \ldots, x_n\rangle$ и $c(A) = \langle y_1, y_2, \ldots, y_n\rangle$, то $x_1 = y_1 = 2^k - 1$.*

□

С помощта на следващата теорема, ние ще докажем, че редицата $\{\mu(k+2,k)\}_{k=0}^{\infty}$ съвпада с редицата на Фибоначи (4.4.4).

Теорема 4.4.3. *Нека $\{\mu(k+2,k)\}_{k=0}^{\infty}$ е редицата, дефинирана с (4.4.1) при $n = k+2$ и нека $\{f_k\}_{k=0}^{\infty}$ е редицата на Фибоначи. Тогава за всяко естествено число $k = 0, 1, 2, 3, \ldots$ е в сила:*

$$\mu(k+2,k) = f_k.$$

Доказателство. При $k = 0$ твърдението следва от (4.4.4) и (4.4.2), а при $k = 1$ от (4.4.4) и (4.4.3).

При $k = 2$ съществуват точно две канонични матрици $A_1, A_2 \in \Lambda_4^2$, такива че $r(A_1) = \langle 3, 3, 12, 12\rangle$ and $r(A_2) = \langle 3, 5, 10, 12\rangle$. Този факт се проверява непосредствено. Следователно

$$\mu(2,0) = f_0, \quad \mu(3,1) = f_1 \quad \text{и} \quad \mu(4,2) = f_2$$

Нека k е произволно цяло число, такова че $k \geq 3$ и нека $A = [\alpha_{i\,j}] \in \Lambda_{k+2}^k$, $1 \leq i, j \leq k+2$ е канонична матрица. Тогава съгласно лема 4.4.1 $\alpha_{1\,1} = \alpha_{1\,2} = \alpha_{2\,1} = 0$ и $\alpha_{1\,3} = \alpha_{1\,4} = \ldots = \alpha_{1\,n} = \alpha_{3\,1} = \alpha_{4\,1} = \ldots = \alpha_{n\,1} = 1$. Следователно са възможни следните два случая:

i) $\alpha_{2\,2} = 0$, т.е. A има вида

$$A = \begin{bmatrix} 0 & 0 & 1 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ 1 & 1 & & & \\ \vdots & \vdots & & B & \\ 1 & 1 & & & \end{bmatrix}$$

Означаваме с $\mathcal{M}_1$ множеството на всички канонични матрици от този вид и нека A е произволна матрица от $\mathcal{M}_1$. В A премахваме първия и втория редове и първия и втория стълбове. Получаваме матрицата $B \in \Lambda_k^{k-2}$. Лесно се вижда, че B е канонична матрица.

Обратно, нека $B = [\beta_{i\,j}] \in \Lambda_k^{k-2}$ $(k \geq 3)$ и нека B е канонична матрица. От B получаваме матрицата $A = [\alpha_{i\,j}] \in \Lambda_{k+2}^k$ по следния начин: $\alpha_{1\,1} = \alpha_{1\,2} = \alpha_{2\,1} = \alpha_{2\,2} = 0$, $\alpha_{1\,j} = \alpha_{2\,j} = 1$, $3 \leq j \leq k+2$ и $\alpha_{i\,1} = \alpha_{i\,2} = 1$, $3 \leq i \leq k+2$. За всеки $i, j \in \{3, 4, \ldots, k+2\}$ полагаме $\alpha_{i\,j} = \beta_{i-2\,j-2}$. Лесно се вижда, че така получената матрица A е канонична матрица.

Следователно $|\mathcal{M}_1| = \mu(k, k-2)$ за всяко естествено число $k \geq 3$.

ii) $\alpha_{2\,2} = 1$, т.е. A има вида

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 0 & 1 & \cdots & 1 \\ 1 & 0 & & & & \\ 1 & 1 & & & & \\ \vdots & \vdots & & & & \\ 1 & 1 & & & & \end{bmatrix}$$

Нека $\mathcal{M}_2$ е множеството от всички матрици от от този вид и нека $A = [\alpha_{ij}]$ е произволна матрица от $\mathcal{M}_2$, където $\alpha_{22} = 1$. Променяме α_{22} да приеме стойност 0 и премахваме първия ред и първия стълб на A. По този начин получаваме матрица от Λ_{k+1}^{k-1}, която лесно се вижда че е канонична.

Обратно, нека $B = [\beta_{ij}] \in \Lambda_{k+1}^{k-1}$ и нека B да е канонична матрица. Съгласно лема 4.4.1 $\beta_{11} = \beta_{12} = \beta_{21} = 0$. Променяме β_{11} от 0 на 1. Добавяме в B един първи ред и един първи стълб и получаваме матрицата $A = [\alpha_{ij}] \in \Lambda_{k+2}^{k}$, така че $\alpha_{11} = \alpha_{12} = \alpha_{21} = 0$, $\alpha_{1j} = 1$ при $j = 3, 4, \ldots, k+2$, $\alpha_{i1} = 1$ при $3, 4, \ldots, k+2$ и $\alpha_{s+1\,t+1} = \beta_{st}$ при $s, t \in \{1, 2, \ldots, k+1\}$. Лесно се вижда, че така получената матрица A е канонична и $A \in \mathcal{M}_2$. Следователно $|\mathcal{M}_2| = \mu(k+1, k-1)$ за всяко естествено число $k \geq 3$.

Ако $\mathcal{M}$ е множеството от всички канонични матрици от Λ_{k+2}^{k}, то очевидно $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$ и $\mathcal{M}_1 \cup \mathcal{M}_2 = \mathcal{M}$. Следователно при $k \in \mathbb{N}$, $k \geq 3$ е изпълнено $\mu(k+2, k) = |\mathcal{M}| = |\mathcal{M}_1| + |\mathcal{M}_2| = \mu(k, k-2) + \mu(k+1, k-1)$, което доказва и теоремата.

□

4.5. Бинарни матрици във връзка с теорията на k–значните функции

Нека да разгледаме следните множества от бинарни матрици:

$\mathcal{L}_{s\times t}(p, q)$ - множеството на всички $s \times t$ бинарни матрици, имащи поне една единица във всеки ред и ако броят на единиците в j-я стълб е равен на t_j, то $p \leq t_j \leq q$, $j = 1, 2, \ldots, t$.

$\mathcal{R}_{s\times t}(p) = \mathcal{L}_{s\times t}(p, p)$ - множеството на всички $s \times t$ бинарни матрици, имащи във всеки стълб точно по p на брои единици, като във всеки ред матриците притежават поне една единица;

$\mathcal{H}_s(p) \subseteq \mathcal{R}_{s\times s}(p)$ - множеството на всички бинарни $s \times s$ матрици с точно p единици на всеки ред и всеки стълб.

Изброените по-горе множествата намират приложение в теорията на k–значните функции [133–135].

Теорема 4.5.1. *Ако $1 \leq p \leq q \leq s$, то броят на бинарните матрици от множеството $\mathcal{L}_{s\times t}(p, q)$ се дава с равенството:*

$$|\mathcal{L}_{s\times t}(p, q)| = \sum_{i=0}^{s-p} (-1)^i \binom{s}{i} \left[\sum_{j=p}^{q} \binom{s-i}{j} \right]^t \tag{4.5.1}$$

Доказателство. Числото $R(p, q, s, t) = \left[\sum_{j=p}^{q} \binom{s}{j} \right]^t$ показва броя на всички бинарни матрици с размерност $s \times t$, във всеки стълб на които броят на единиците е в интервала $[p, q]$.

Нека A е $s \times t$ бинарна матрица. Ще казваме, че A притежава свойството r_j, ако j-ят ред на A съдържа само нули, $1 \leq j \leq s$. Тогава, очевидно броят на матриците от множеството $\mathcal{L}_{s\times t}(p, q)$, притежаващи свойствата $r_{j_1}, r_{j_2}, \ldots, r_{j_i}$, $i = 1, 2, \ldots s$ е равен на $R(p, q, s-i, t)$. Тогава прилагайки принципа на включване и изключване получаваме:

$$|\mathcal{L}_{s\times t}(p, q)| = \sum_{i=0}^{s} (-1)^i \binom{s}{i} R(p, q, s-i, t) \tag{4.5.2}$$

Като вземем предвид, че за всяка матрица от множеството $\mathcal{L}_{s\times t}(p,q)$ във всеки стълб има поне p на брой единици, т.е. не може да има повече от $s-p$ изцяло нулеви реда, то $R(p,q,r,t)=0$ при $r>s-p$ и достигаме до формула (4.5.1).

□

Като непосредствено следствие от теорема 4.5.1 полагайки $p=q$ се получава следното твърдение:

Следствие 4.5.2. ([135] Lemma 1) *Броят $|\mathcal{R}_{s\times t}(p)|$ на всички $s\times t$ бинарни матрици, имащи във всеки стълб точно по p на брой единици, като във всеки ред матриците притежават поне една единица се дава с формулата:*

$$|\mathcal{R}_{s\times t}(p)|=\sum_{i=0}^{s-p}(-1)^i\binom{s}{i}\binom{s-i}{p}^t \tag{4.5.3}$$

□

Ще разгледаме и някои специални подмножества на множеството $\mathcal{L}_{s\times t}(p,q)$:

$\mathcal{L}_{s\times t}(p,q]$ - множеството на всички матрици от $\mathcal{L}_{s\times t}(p,q)$, имащи в поне един стълб точно q на брой единици;

$\mathcal{L}_{s\times t}[p,q)$ - множеството на матрици от $\mathcal{L}_{s\times t}(p,q)$, имащи в поне един стълб точно p на брой единици;

$\mathcal{L}_{s\times t}[p,q]$ - множеството матрици от $\mathcal{L}_{s\times t}(p,q)$, имащи в поне един стълб точно q на брой единици и в поне един стълб точно p на брой единици.

Теорема 4.5.1 ще използуваме и при намирането на количествена оценка за множествата $\mathcal{L}_{s\times t}(p,q]$, $\mathcal{L}_{s\times t}[p,q)$, и $\mathcal{L}_{s\times t}[p,q]$.

Следствие 4.5.3. *Ако $1\le p<q\le s$, то броят на бинарните матрици от множеството $\mathcal{L}_{s\times t}(p,q]$ се дава съответно с равенството:*

$$|\mathcal{L}_{s\times t}(p,q]|=\sum_{i=0}^{s-p}(-1)^i\binom{s}{i}\left\{\left[\sum_{j=p}^{q}\binom{s-i}{j}\right]^t-\left[\sum_{j=p}^{q-1}\binom{s-i}{j}\right]^t\right\} \tag{4.5.4}$$

Доказателство. До равенството (4.5.4) ще достигнем като вземем предвид, че $\mathcal{L}_{s\times t}(p,q]=\mathcal{L}_{s\times t}(p,q)\setminus\mathcal{L}_{s\times t}(p,q-1)$.

□

Следствие 4.5.4. *Ако $1\le p<q\le s$, то в сила е равенството:*

$$|\mathcal{L}_{s\times t}[p,q)|=\sum_{i=0}^{s-p}(-1)^i\binom{s}{i}\left\{\left[\sum_{j=p}^{q}\binom{s-i}{j}\right]^t-\left[\sum_{j=p+1}^{q}\binom{s-i}{j}\right]^t\right\} \tag{4.5.5}$$

Доказателство. До равенството (4.5.5) ще достигнем като вземем предвид, че $\mathcal{L}_{s\times t}[p,q)=\mathcal{L}_{s\times t}(p,q)\setminus\mathcal{L}_{s\times t}(p+1,q)$.

□

Следствие 4.5.5. *Ако $1\le p\le q\le s$, то в сила е неравенството:*

$$\begin{aligned}&|\mathcal{L}_{s\times t}[p,q]|=\\&\sum_{i=0}^{s-p}(-1)^i\binom{s}{i}\left\{\left[\sum_{j=p}^{q}\binom{s-i}{j}\right]^t-\left[\sum_{j=p+1}^{q}\binom{s-i}{j}\right]^t-\left[\sum_{j=p}^{q-1}\binom{s-i}{j}\right]^t\right\}\end{aligned} \tag{4.5.6}$$

Доказателство. До равенството (4.5.6) ще достигнем вземайки предвид едно от следните равенства:

$$\begin{aligned}
&\mathcal{L}_{s\times t}[p,q] = \\
&= \mathcal{L}_{s\times t}[p,q) \bigcap \mathcal{L}_{s\times t}(p,q] = \\
&= \Big(\mathcal{L}_{s\times t}[p,q) \bigcup \mathcal{L}_{s\times t}(p,q]\Big) \setminus \mathcal{L}_{s\times t}(p,q) = \\
&= (\mathcal{L}_{s\times t}(p,q) \setminus \mathcal{L}_{s\times t}(p+1,q)) \setminus \mathcal{L}_{s\times t}(p,q-1).
\end{aligned}$$

□

Някои комбинаторни тъждества и неравенства и връзката им с функциите в k-значната логика като непосредствени следствия от теорема 4.5.1 са дадени в [136].

Теорема 4.5.1 намира и непосредствено приложение при получаването и доказателството на различни комбинаторни тъждества [26, 137].

4.6. Комбинаторни задачи над бинарни матрици в курсовете по програмиране

За да повишим интереса на студентите към дадена дисциплина или дял от науката съществуват различни методи. Тук ще се спрем на два от тях, които както сочи нашият скромен опит с успех могат да се прилагат в курсовете по програмиране:

1. Първо да наблегнем на факта, че все още не е известно решението в общия случай на дадена математическа задача, каквито са например проблем 4.2.1 формулиран на страница 72 и усложненият му вариант – проблем 4.6.1, които ще формулираме по-долу на страница 85. След това да дадем на студентите за самостоятелна работа да напишат програма, решаваща задачата при някои частни случаи на параметрите.

2. Като посочим, че за решаването на дадена задача по програмиране съществува по-ефективен алгоритъм в сравнение със стандартните алгоритми, които по принцип не затрудняват болшинството от изявените студенти. Много често това е свързано с доказателството на някои чисто математически твърдения (понякога и непубликувани все още).

Подобни подходи ще опишем в настоящият раздел.

Нека n е положително цяло число и нека $A \in \mathfrak{B}_{n^2}$ е $n^2 \times n^2$ бинарна матрица. С помощта на $n-1$ хоризонтални и $n-1$ вертикални линии A е разбита на n^2 квадратни $n \times n$ подматрици A_{kl}, $1 \le k,l \le n$, т.е.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}, \tag{4.6.1}$$

Подматриците A_{kl}, $1 \le k,l \le n$ ще наричаме блокове.

Нека да усложним проблем 4.2.1 като добавим още едно изискване, а именно:

Проблем 4.6.1. *Да се намери броят $\varsigma(n,k)$ на всички $n^2 \times n^2$ бинарни матрици имащи точно k на брой единици на всеки ред, всеки стълб и във всеки $n \times n$ блок.*

Както е показано в [103] проблем 4.6.1 има връзка с решаването на различни комбинаторни задачи, свързани с популярната през последните години главоблъсканица Судоку.

При $k = 1$ проблем 4.6.1 е решен в [103], където е показано, че

$$\varsigma(n,1) = (n!)^{2n} \tag{4.6.2}$$

На нас не ни е известна формула за изчисляване на функцията $\varsigma(n,k)$ при $k > 1$.

В [103] е посочено следното очевидно твърдение:

Твърдение 4.6.2. [103] *Квадратната $n^2 \times n^2$ матрица P с елементи от $[n^2] = \{1, 2, \ldots, n^2\}$ е Судоку матрица тогава и само тогава, когато съществуват Σ_{n^2} матриците $A_1, A_2, \ldots, A_{n^2}$, всеки две от които са непресичащи се по между си и такива че P може да се представи във вида*

$$P = 1 \cdot A_1 + 2 \cdot A_2 + \cdots + n^2 \cdot A_{n^2}$$

□

Да разгледаме следната задача по програмиране:

Задача 4.6.3. *Да се напише програма, която по зададено положително цяло число n да получава всички $n^2 \times n^2$ S-пермутационни матрици.*

Опитът показва, че болшинството от добрите студенти без проблем се справят с така поставената задача, но за съжаление представените решения обикновено не са от най-ефективните. Да опишем едно от стандартните решения най-често предлагани от студентите:

Стандартно решение на задача 4.6.3

Лесно може да се забележи, че ако премахнем условието във всеки блок на получаваните $n^2 \times n^2$ бинарни матрици да има единствена 1, то разглежданата задача може да се сведе до задачата за получаването на всички пермутации на целите числа от 1 до n^2. Това е една често обсъждана в курсовете по програмиране комбинаторна задача и чието решение може да бъде намерено в редица учебни помагала, например в [40]. И наистина нека $\pi = \langle p_1, p_2, \ldots, p_m \rangle$ е пермутация на числата от 1 до m. Тогава получаваме $m \times m$ бинарната матрица $B = (b_{ij}) \in \mathfrak{B}_m$, такава че $b_{ij} = 1$ тогава и само тогава, когато $p_i = j$, $1 \leq i, j \leq m$. Лесно се вижда, че така получената матрица B има точно една 1 на всеки ред и всеки стълб. Оттук идва и името на такива матрици - *пермутационни*. Така получаваме следния алгоритъм за решаване на задача 4.6.3:

Алгоритъм 4.6.4. *Получаване на всички S-пермутационни матрици (**стандартен алгоритъм**).*

1. *Получаваме всички пермутации на целите числа от 1 до n^2;*
2. *За всяка пермутация $\pi = \langle p_1, p_2, \ldots, p_{n^2} \rangle$ получена в стъпка 1 получаваме бинарната матрица $A = (\alpha_{ij}) \in \mathfrak{B}_{n^2}$, такава че $\alpha_{ij} = 1$ тогава и само тогава когато $p_i = j$. Във всички останали случаи $\alpha_{ij} = 0$, $1 \leq i, j \leq n^2$;*

3. *За всяка от получените в стъпка 2 матрица проверяваме дали във всеки блок има единствена 1. Ако да, то получената матрица е S-пермутационна, ако не, то премахваме тази матрица от списъка.*

□

За съжаление в алгоритъм 4.6.4 се получават редица ненужни матрици и се губи не малко време при проверката дали получената матрица отговаря на зададените условия (стъпка 3). Общо в алгоритъм 4.6.4 се получават $(n^2)!$ пермутационни матрици, докато съгласно формула (4.1.8) броят на S-пермутационните матрици е равен на $(n!)^{2n} < n^2!$ при $n \geq 2$. Не е за пренебрегване и факта, че самата програмна реализация на стъпка 3 изисква известни усилия и известни математически умения.

При $n = 2$ имаме $2^2! = 24$, $(2!)^4 = 16$; при $n = 3$ имаме $3^2! = 362\ 880$, $(3!)^6 = 46\ 656$; при $n = 4$ имаме $4^2! = 20\ 922\ 789\ 888\ 000$, $(4!)^8 = 110\ 075\ 314\ 176$ и т.н. Може да се докаже, че при растенето на n отношението $\dfrac{n^2!}{(n!)^{2n}}$ расте. Това доказва и малката ефективност на алгоритъм 4.6.4.

В настоящата работа ще докажем, че съществува алгоритъм получаващ всички S-пермутационни матрици и само тях, т.е. без да получава други матрици, които не са S-пермутационни. Подобни алгоритми са известни под името *алгоритми за класификация на комбинаторни обекти с отхвърляне на изоморфните (ненужните)* [13, 14] (вижте също и глава 1, раздел 1.3).

По този начин се намаляват до минимум итерациите в алгоритъма и при всяка итерация се избягва проверката дали получената матрица е S-пермутационна. Такъв алгоритъм очевидно ще бъде по-бърз и по-ефективен от алгоритъм 4.6.4. Търсеният алгоритъм се основава на доказаната по-долу теорема 4.6.6.

Да означим с Ψ_n множеството на всички $(2n) \times n$ матрици, които за краткост ще наричаме Ψ_n матрици, всеки ред на която представлява пермутация на елементите на $[n]$. Очевидно

$$|\Psi_n| = (n!)^{2n} \tag{4.6.3}$$

Малко по-сложна дефиниция за понятието "непресичащи се" ще дадем за Ψ_n матриците:

Дефиниция 4.6.5. *Нека $C = [c_{ij}]$ и $D = [d_{ij}]$ са две Ψ_n матрици. Ще казваме, че C и D са* непресичащи се, *ако не съществуват естествени числа $s, t \in \{1, 2, \ldots n\}$, такива че наредената двойка $\langle c_{st}, c_{n+t\,s}\rangle$ да бъде равна на наредената двойка $\langle d_{st}, d_{n+t\,s}\rangle$.*

Теорема 4.6.6. *Съществува биективно изображение от Ψ_n в Σ_{n^2} при което двойка непресичащи се матрици от Ψ_n да съответстват на двойка непресичащи се матрици от Σ_{n^2}*

Доказателство. Нека $P = [p_{ij}]_{2n\times n} \in \Psi_n$. От P получаваме единствена матрица от Σ_{n^2} с помощта на следния алгоритъм:

Алгоритъм 4.6.7. *По дадена $P = [p_{ij}]_{2n\times n} \in \Psi_n$ получава точно една матрица ат Σ_{n^2}*

1. ***for*** $s = 1, 2, \ldots, n$ ***do***
2. ***for*** $t = 1, 2, \ldots, n$ ***do***
   begin

3. $k := p_{st}$;

4. $l := p_{n+t,\,s}$;

5. *Получаваме* $n \times n$ *матрицата* $A_{st} = [a_{ij}]_{n\times n}$, *такава че* $a_{kl} = 1$ *и* $a_{ij} = 0$ *при всички останали случаи;*

   end;

6. *Получаваме матрицата* A *съгласно формула (4.6.1);*

Нека $s \in [n] = \{1, 2, \ldots, n\}$. Тъй като наредената n-торка

$$\langle p_{s\,1}, p_{s\,2}, \ldots, p_{s\,n}\rangle$$

представляваща s-я ред на матрицата P е пермутация, то на всеки ред на $n \times n^2$ матрицата

$$R_s = [\ A_{s1} \quad A_{s2} \quad \cdots \quad A_{sn}\ ]$$

има единствена единица. В случая, за всяко $j = 1, 2, \ldots, n$, A_{sj} е бинарна $n\times n$ матрица. Аналогично за всяко $t \in [n]$, понеже наредената n-торка $\langle p_{n+t\,1}, p_{n+t\,2}, \ldots, p_{n+t\,n}\rangle$, представляваща $(n + t)$-я ред на P е пермутация, то във всеки стълб на $n^2 \times n$ матрицата

$$C_t = \begin{bmatrix} A_{1t} \\ A_{2t} \\ \vdots \\ A_{nt} \end{bmatrix}$$

има единствена единица, където A_{it}, $i = 1, 2, \ldots, n$ е бинарна $n \times n$ матрица. Следователно получената с помощта на алгоритъм 4.6.7 матрица A е Σ_{n^2} матрица.

Тъй като за всяко $P \in \Psi_n$ с помощта на алгоритъм 4.6.7 се получава единствен елемент от Σ_{n^2}, то този алгоритъм е описание на изображение $\varphi : \Psi_n \to \Sigma_{n^2}$. Лесно се вижда, че по зададени различни елементи от Ψ_n с помощта на алгоритъм 4.6.7 получаваме различни елементи на Σ_{n^2}. Следователно φ е инекция. Но съгласно формули (4.6.3) и (4.1.8)

$$|\Sigma_{n^2}| = |\Psi_n|\,,$$

откъдето следва, че φ е биекция.

Анализирайки алгоритъм 4.6.7 стигаме до заключението, че P и Q са непресичащи се матрици от Ψ_n тогава и само тогава, когато $\varphi(P)$ и $\varphi(Q)$ са непресичащи се матрици от Σ_{n^2} съгласно дефиниции 4.1.1 и 4.6.5. Теоремата е доказана.

□

Като следствие от теорема 4.6.6 получаваме следния обобщен алгоритъм за получаване на всички S-пермутационни матрици, който поради изказаните по-горе доводи очевидно ще бъде по-ефективен и по-бърз от алгоритъм 4.6.4 решаващ същата задача:

Алгоритъм 4.6.8. *Получаване на всички S-пермутационни матрици (****без да генерираме ненужни обекти****).*

1. *Получаваме всички пермутации на целите числа от 1 до* n^2;

2. *С помощта на получените в стъпка 1 пермутации получаваме всички* Ψ_n *матрици;*

3. *От всяка* Ψ_n *матрица, получена в стъпка 2 получаваме поредната S-пермутационна матрица с помощта на алгоритъм 4.6.7.*

4.7. Случайни пермутации, случайни Судоку матрици и алгоритми със случайни обекти

Нека $\mathfrak{M}$ е крайно множество. Под *генератор на случайни обекти* от $\mathfrak{M}$ ще разбираме всеки алгоритъм $\mathcal{A}_{\mathfrak{M}}$ получаващ по случаен начин елемент на $\mathfrak{M}$. Елементите, получени с помощта на генератор за случайни обекти ще наричаме *случайни елементи* на $\mathfrak{M}$ (случайни числа, случайни матрици, случайни пермутации и т.н). Ще считаме, че вероятностите за получаване на различните случайни елементи на $\mathfrak{M}$ с помощта на $\mathcal{A}_{\mathfrak{M}}$ са равни по между си и равни съответно на $\frac{1}{|\mathfrak{M}|}$. С $T(\mathcal{A}_{\mathfrak{M}})$ ще означаваме времето за работа на генератора на случайни обекти за получаване на един случаен елемент от $\mathfrak{M}$.

В този раздел ще разгледаме алгоритми, които използват в своята логика случайността в различните и проявления (*randomized algorithms*). Такива алгоритми са особено полезни математически методи за решаване на голям клас от задачи като се използва генератор на случайни обекти. Обикновено това са генератори на случайни числа с помощта на които се получават и по-сложни случайни обекти [112, 146]. Най-известните класове от алгоритми от този вид са методите *Монте Карло* и *Лас Вегас*.

Методите *Монте Карло* се използват основно при задачи, където се пресмята с приближение. При тях времето за работа е детерминирано, т.е работи се за фиксиран брой стъпки (итерации) но резултатът е случайна величина, която дава приблизителна стойност на търсения резултат. Тук важното е грешката да е възможно най-малка [56, 58, 78, 113]. Тъй като измерването в различните технически системи и технологии в зависимост от прецизността на измервателните уреди обикновено дава приблизителна стойност, то методите Монте Карло заемат своето достойно място в измервателните системи и процеси. В този смисъл са и разглежданията в някои глави на книгата [30].

При другия клас от алгоритми - методите *Лас Вегас* винаги се получава точен резултат, но времето за работа е случайна величина [125, 143, 158].

Разгледаните в настоящият раздел проблеми намират теоретично и практическо приложение както в областта на математиката и информатиката, така и при математическото моделиране в обществените и социални науки [51, 150–152].

Тук ще използуваме метода Лас Вегас при решаването на следния клас от задачи: Нека n и $m = m(n)$ и са естествени числа. Разглеждаме множеството $\mathcal{U}$ състоящо се от обекти зависещи от m параметъра като всеки параметър принадлежи на крайното множество $\mathfrak{M}$. Нека $\mathcal{V} \subset \mathcal{U}$. Задачата се състои в получаването на (поне един) обект, принадлежащ на множеството $\mathcal{V}$. Броят на елементите на множествата $\mathcal{U}$ и $\mathcal{V}$ зависи единствено от параметъра m, който от своя страна е целочислена функция на аргумента n. Стандартният алгоритъм използуван много често и реализиращ метода Лас Вегас се описва най-общо по следния начин:

Алгоритъм 4.7.1.

1. *С помощта на генератор на случайни обекти $\mathcal{A}_{\mathfrak{M}}$ получаваме последователно $m = m(n)$ на брой случайни елементи на $\mathfrak{M}$ с помощта на които инициализираме параметрите на обекта $u \in \mathcal{U}$;*

2. *Проверяваме дали $u \in \mathcal{V}$. Ако не, то повтаряме всичко от начало.*

С други думи, ако разполагаме с генератор на случайни обекти, методът Лас Вегас може да ни послужи като генератор на по-сложни случайни обекти. Методът

Лас Вегас може да бъде в основата на различни алгоритми, решаващи една и съща задача и имащи различна ефективност в различните конкретни ситуации, както ще видим и по-долу.

Ефективността на алгоритъм 4.7.1 зависи от конкретния случай и може да бъде оценявана по отношение на следните два критерия:

Вероятностна оценка: Ако с $p(n)$ означим вероятността след генериране на $m = m(n)$ случайни елемента на $\mathfrak{M}$ да получим обект от $\mathcal{V}$, то съгласно формулата за класическа вероятност ще имаме

$$p(n) = \frac{|\mathcal{V}|}{|\mathcal{U}|} \tag{4.7.1}$$

Оценка за време: Да означим с $\tau(n)$ времето за изпълнение на една итерация на алгоритъм 4.7.1. Тогава

$$\tau(n) = m(n)T(\mathcal{A}_{\mathfrak{M}}) + \theta(n), \tag{4.7.2}$$

където $\theta(n)$ е времето за проверка дали полученият обект принадлежи на множеството $\mathcal{V}$.

Очевидно оценката на ефективността на алгоритъм 4.7.1 ще бъде право пропорционална на $p(n)$ и обратно пропорционална на $\tau(n)$.

Особен интерес представляват случаите с вероятностна оценка равна на 1, т.е. случаите при които сме конструирали алгоритъма така, че същият да получава директно елемент на множеството $\mathcal{V}$ и да не се налага проверка за принадлежност. В този случай се изпълнява единствена итерация т.е. няма повторения. В раздел 4.7.1 ще опишем два алгоритъма за получаване на случайни пермутации, единият от които е с вероятностна оценка 1 и който като цяло ще бъде очевидно по-ефективен от другия с вероятностна оценка $p(n) = \dfrac{n!}{n^n}$.

В болшинството среди за програмиране са добавени стандартни процедури за получаване на случайни числа от множеството $[k]$. Този факт ще считаме за даденост и ще го използуваме в по-нататъшните ни разсъждения. Нека подобна процедура означим с $\mathcal{A}_k$. В настоящата работа приемаме, че $T(\mathcal{A}_k) \approx T(\mathcal{A}_l) \approx t_0 = const$ при $k \neq l$.

Нека k е цяло положително число. Както и на други места в дисертационния труд с $[k]$ означаваме множеството от цели числа

$$[k] = \{1, 2, \ldots, k\}.$$

С $\mathcal{P}_k$ означаваме множеството от всички пермутации на елементите на $[k]$. Както е добре известно $|\mathcal{P}_k| = k!$.

В следващите подраздели ще демонстрираме казаното по-горе като анализираме получаването по метода Лас Вегас на произволна пермутация от m елемента, произволна $n^2 \times n^2$ Судоку матрица и произволна Ψ_n матрица, т.е. $(2n) \times n$ матрица с $2n$ реда и n стълба, всеки ред на която представлява пермутация на елементите на $[n]$ (вижте раздел 4.6).

Методът Лас Вегас се използува широко при решаването на задачи, които са доказано NP-пълни. Подробно за NP-пълните задачи и тяхното приложение вижте в [111] или [123]. В [168] и [169] се дава доказателство, че решаването на популярната главоблъсканица Судоку е NP-пълна задача. Как да напишем компютърна програма за решаване на Судоку с използуването на понятието множество и в съчетание с метода на "пробите и грешките" е описано в раздел 3.2.

4.7.1. Случайни пермутации

При пермутациите на елементите от множеството $[n] = \{1, 2, \ldots, n\}$ имаме $m(n) = n$.

Да означим $p_1(n)$ вероятността да получим случайна пермутация от $\mathcal{P}_n$ с помощта на алгоритъм 4.7.1. Тогава съгласно формула (4.7.1) получаваме

$$p_1(n) = \frac{n!}{n^n} \tag{4.7.3}$$

Твърдение 4.7.2. *Съществува алгоритъм, работещ за време $O(n)$ и проверяващ дали наредената n-торка $\rho = \langle a_1, a_2, \ldots a_n \rangle$ е пермутация, $a_i \in [n]$, $i = 1, 2, \ldots, n$.*

За доказателство предлагаме следния алгоритъм, който лесно се вижда, че работи за време $O(n)$:

Алгоритъм 4.7.3. *Проверява дали наредената n-торка*

$$\rho = \langle a_1, a_2, \ldots a_n \rangle, \quad a_i \in [n]$$

е пермутация.

1. *Декларираме масива с n елемента $v[1], v[2], \ldots, v[n]$ и инициализираме всички негови елементи с 0;*
2. ***for*** $i = 1, 2, \ldots, n$ ***do***
   begin
3. $v[a_i] := v[a_i] + 1$;
4. ***if*** $v[a_i] > 1$ ***then*** $\langle a_1, a_2, \ldots, a_i, \ldots, a_n \rangle \notin \mathcal{P}_n$ *тъй като числото a_i се среща повече от веднъж в ρ и изход от алгоритъма;*
   end.

□

Нека да означим с $\tau_1(n)$ времето за изпълнение на една итерация на алгоритъм 4.7.1 при получаването на случайна пермутация на елементите на $[n]$, а с $\theta_1(n)$ времето за проверка произволна n-торка числа от $[n]$ да принадлежи на $\mathcal{P}_n$. Тогава вземайки предвид формула (4.7.2) и твърдение 4.7.2 получаваме следната оценка за време:

$$\tau_1(n) = nT(\mathcal{A}_n) + \theta_1(n) = nt_0 + O(n) = O(n). \tag{4.7.4}$$

Следващият алгоритъм използва генериране се случайни числа, но има вероятностна оценка равна на 1, т.е. в алгоритъм 4.7.1 стъпка 2 не се изпълнява, тъй като още при първото генериране на n случайни числа получената наредена n-торка е пермутация.

Алгоритъм 4.7.4. *Получаване на случайна пермутация*

$$\rho = \langle a_1, a_2, \ldots, a_i, \ldots, a_n \rangle \in \mathcal{P}_n,$$

където $a_i \in [n]$, $i = 1, 2, \ldots, n$, $a_i \neq a_j$ при $i \neq j$.

1. *Декларираме масива с n елемента $v[1], v[2], \ldots, v[n]$;*
2. ***for*** $k = 1, 2, \ldots, n$ ***do*** $v[k] := k$;
3. ***for*** $k = 1, 2, \ldots, n$ ***do***
   begin
4. *Генерираме случайното число $x \in [n-k+1] = \{1, 2, \ldots, n-k+1\}$;*
5. $a_k := v[x]$;
   // Премахваме елемента $v[x]$ и намаляваме броя на елементите на масива с 1:
6. ***for*** $j = x, x+1, \ldots, n-k$ ***do*** $v[j] := v[j+1]$;
   end.

Лесно се вижда верността на следното твърдение:

Твърдение 4.7.5. *Алгоритъм 4.7.4 получаващ случайна пермутация има вероятностна оценка*

$$p_2(n) = 1 \tag{4.7.5}$$

и оценка за време

$$\tau_2(n) = t_0 \left[O(n) + O(n-1) + \cdots + O(1)\right] = O(n^2). \tag{4.7.6}$$

□

Виждаме, че алгоритъм 4.7.1 за получаване на случайни пермутации е по-ефективен от алгоритъм 4.7.4 по отношение на фактора време. От друга страна, фактът, че вероятностната оценка на алгоритъм 4.7.4 е равна на единица за разлика от алгоритъм 4.7.1 с вероятностна оценка равна на $p_1(n) = \dfrac{n!}{n^n}$, прави алгоритъм 4.7.4 като цяло значително по-ефективен от алгоритъм 4.7.1 при прилагането му за получаване на случайни пермутации.

4.7.2. Случайни $(2n) \times n$ матрици, всеки ред на която представлява пермутация на елементите на $[n]$ (Ψ_n-матрици)

Както и в раздел 4.6 нека да означим с Ψ_n множеството на всички $(2n) \times n$ матрици, които за краткост наричаме Ψ_n матрици, всеки ред на която представлява пермутация на елементите на $[n]$. В случая $\mathfrak{M} = [n]$ и $m(n) = 2n^2$. Очевидно

$$|\Psi_n| = (n!)^{2n} \tag{4.7.7}$$

Лесно се убеждаваме във верността на следното твърдение:

Твърдение 4.7.6. *При получаването на случайна Ψ_n матрица с помощта на алгоритъм 4.7.1 имаме следните оценки:*

Вероятностна оценка:

$$p_3(n) = \frac{|\mathcal{V}|}{|\mathcal{U}|} = \frac{(n!)^{2n}}{n^{2n^2}} \tag{4.7.8}$$

Оценка за време:

$$\tau_3(n) = m(n)T(\mathcal{A}_n) + 2n\tau_1(n) = 2n^2 t_0 + 2nO(n) = O(n^2) \tag{4.7.9}$$

където $\tau_1(n)$ *се получава съгласно формула (4.7.4).*

□

Както ще видим по-надолу (твърдение 4.7.8) следващият алгоритъм ще бъде по-добър от алгоритъм 4.7.1 за получаване на случайна Ψ_n матрица по отношение на вероятностната оценка.

Алгоритъм 4.7.7. *Получаване на случайна* Ψ_n *матрица с вероятностна оценка равна на 1.*

1. ***for*** $k = 1, 2, \ldots, n, n+1, \ldots 2n$ ***do***

   Получаваме случайна пермутация с помощта на алгоритъм 4.7.4, която ще бъде k*-я ред на матрицата;*

На практика алгоритъм 4.7.7 представлява $2n$ пъти повтаряне на алгоритъм 4.7.4, който съгласно твърдение 4.7.5 има вероятностна оценка равна на 1 и следователно и алгоритъм 4.7.7 ще има вероятностна оценка равна на 1. Така получаваме следното

Твърдение 4.7.8. *Алгоритъм 4.7.7, получаващ случайна* Ψ_n *матрица, има вероятностна оценка*

$$p_4(n) = 1 \tag{4.7.10}$$

и оценка за време

$$\tau_4(n) = 2nO(n^2) = O(n^3). \tag{4.7.11}$$

□

Както видяхме в раздел 4.6 Ψ_n-матриците могат с успех да бъдат използвани при конструирането на ефективни алгоритми за получаване на Судоку матрици.

Една квадратна бинарна матрица се нарича *пермутационна*, ако на всеки ред и на всеки стълб има точно по една 1. Да припомним, че със Σ_{n^2} означаваме множеството от всички $n^2 \times n^2$ S-пермутационни матрици, т.е. $n^2 \times n^2$ пермутационни матрици от вида

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}, \tag{4.7.12}$$

където за всеки $s, t \in \{1, 2, \ldots, n\}$ A_{st} е квадратна $n \times n$ бинарна подматрица (блок) имаща единствен елемент равен на 1, а останалите елементи са равни на 0.

Следователно, съгласно 4.1.8, ако получим случайна Σ_{n^2} матрица с помощта на алгоритъм 4.7.1 ще имаме следната вероятностна оценка:

$$p_5(n) = \frac{(n!)^{2n}}{2^{(n^2)^2}} = \frac{(n!)^{2n}}{2^{n^4}} \tag{4.7.13}$$

За да получим случайна Σ_{n^2} матрица използувайки метода Лас Вегас трябва да генерираме $m(n) = (n^2)^2 = n^4$ случайни числа принадлежащи на множеството

$\{0,1\}$. Следователно, независимо какъв алгоритъм, основаващ се на метода Лас Вегас използуваме, ще получим следната оценка за време:

$$\tau_5(n) = m(n)T(\mathcal{A}_2) + \theta_5(n) = n^4 t_0 + \theta_5(n) = O(n^z), \quad z \geq 4, \tag{4.7.14}$$

където $\theta_5(n)$ е времето за проверка дали дадена бинарна матрица принадлежи на множеството Σ_{n^2}.

За да проверим дали дадена бинарна матрица принадлежи на множеството Σ_{n^2} може да се използува например следния алгоритъм, работещ за време $O(n^4)$, т.е. $\theta_5 = O(n^4)$ и следователно $z = 4$.

Алгоритъм 4.7.9. *Проверява дали бинарната $n^2 \times n^2$ матрица $B = [b_{ij}] \in \Sigma_{n^2}$.*

1. ***for*** $i = 1, 2, \ldots, n^2$ ***do***
   begin
2. *r:=0;*
3. *c:=0;*
4. ***for*** $j = 1, 2, \ldots, n^2$ ***do***
   begin
5. $r := r + b_{ij}$;
6. ***if*** $r > 1$ ***then*** *B не е пермутационна матрица и изход от алгоритъма;*
7. $c := c + b_{ji}$;
8. ***if*** $c > 1$ ***then*** *B не е пермутационна матрица и изход от алгоритъма;*
   end;
9. ***if*** $r = 0$ ***or*** $c = 0$ ***then*** *B не е пермутационна матрица и изход от алгоритъма;*
   end;
10. ***for*** $s = 0, 1, \ldots, n-1$ ***do***
11. ***for*** $t = 0, 1, \ldots, n-1$ ***do***
    begin
12. $x := 0$;
13. ***for*** $i = 1, 2, \ldots, n$ ***do***
14. ***for*** $j = 1, 2, \ldots, n$ ***do*** $x := x + b_{sn+i\ tn+j}$;
15. ***if*** $x \neq 1$ ***then*** $B \notin \Sigma_{n^2}$ *и изход от алгоритъма;*
    end.

Сравнявайки (4.7.13) с (4.7.8) и (4.7.10), както и (4.7.14) с (4.7.9) и (4.7.11) стигаме до хипотезата, че алгоритмите, използващи случайни Ψ_n матрици се очаква да бъдат по-ефективни по отношение както на вероятностната оценка, така и по отношение на оценката за време в сравнение с алгоритмите, използващи случайни Σ_{n^2} матрици и решаващи аналогични задачи. Това стимулира изучаването на свойствата на Ψ_n матриците.

В раздел 4.6 (теорема 4.6.6) доказахме, че съществува биективно изображение от Ψ_n в Σ_{n^2}, при което двойка непресичащи се матрици от Ψ_n да съответстват на двойка непресичащи се матрици от Σ_{n^2}.

4.7.3. Получаване на случайни Судоку матрици

Да означим с $\mathcal{M}_{n^2}$ множеството на всички квадратни $n^2 \times n^2$ матрици с елементи, принадлежащи на множеството $[n^2] = \{1, 2, \ldots, n^2\}$, а със σ_n – броя на всички $n^2 \times n^2$ Судоку матрици. Очевидно $|\mathcal{M}_{n^2}| = (n^2)^{n^2} = n^{2n^2}$. Тогава, ако използуваме алгоритъм 4.7.1 за получаване на случайна Судоку матрица, съгласно формула (4.7.1) имаме следната вероятностна оценка:

$$p_6(n) = \frac{\sigma_n}{n^{2n^2}} \tag{4.7.15}$$

При $n = 2$, $\sigma_2 = 288$ [132, 191]. При $n = 3$ в [108] е показано, че съществуват точно

$$\sigma_3 = 6\ 670\ 903\ 752\ 021\ 072\ 936\ 960 =$$
$$= 9! \times 72^2 \times 2^7 \times 27\ 704\ 267\ 971 =$$
$$2^{20} \times 3^8 \times 5^1 \times 7^1 \times 27\ 704\ 267\ 971^1 \sim 6.671 \times 10^{21}$$

на брой Судоку матрици. Вижте също и страница 54. На нас не ни е известна обща формула за броя σ_n на Судоку матриците при произволно n. Считаме, че това е открит за науката математически проблем.

Ако получим по случаен начин матрицата $P \in \mathcal{M}_{n^2}$ с елементи от $[n^2]$, то съгласно алгоритъм 4.7.1 е необходимо да се провери дали всеки ред, всеки стълб и всеки блок на P е пермутация на елементите на $[n^2]$. Съгласно твърдение 4.7.2 всяка проверка може да стане за време $O(n)$. Следователно при прилагането на алгоритъм 4.7.1 за получаване на случайна Судоку матрица ще получим следната оценка за време:

$$\tau_6(n) = \left(n^2\right)^2 + 2nO(n) + n^2O(n) = O(n^4). \tag{4.7.16}$$

Тук ще посочим по-ефективен алгоритъм за получаване на случайна Судоку матрица, който се основава на твърденията и алгоритмите, разгледани в предходните раздели. Същността му е последователно да получим n^2 на брой случайни Ψ_n матрици (алгоритъм 4.7.7). При всяка новополучена Ψ_n матрица се проверява дали тя е непресичаща се със всяка от преди това получените. При проверката съществено се използуват критериите, описани в теорема 4.6.6 и твърдение 4.6.2. Ако получената матрица се пресича с поне една от преди това получените матрици се генерира на нейно място нова случайна Ψ_n матрица.

Алгоритъм 4.7.10. *Получава случайна Судоку матрица*

1. *Декларираме $n^2 \times n^2$ матрицата S и инициализираме всички нейни елементи с 0;*

2. ***for*** $k = 1, 2, \ldots, n^2$ ***do***
   begin
3. *Декларираме* $n^2 \times n^2$ *матрицата* B *и инициализираме всички нейни елементи с 0;*
4. $z :=$ **true**;
5. ***while*** z ***do***
   begin
6. *Получаваме случайна матрица* $P_k \in \Psi_n$ *(алгоритъм 4.7.7)*
7. *Получаваме* Σ_{n^2} *матрицата* $A = \varphi(P_k)$ *(алгоритъм 4.6.7), където* φ *е дефинираното в теорема 4.6.6 биективно изображение;*
8. $C := B + A$;
9. ***if*** *Всички елементи на матрицата* C *са равни на 0 или 1* ***then***
   /* $P_k = \varphi^{-1}(A)$ *е непресичаща се с всяка от матриците* $P_1, P_2, \ldots, P_{k-1}$ *(теорема 4.6.6)* */
10. ***begin***
11. $B := C$;
12. $S := S + k \cdot A$ *(proposition 4.6.2);*
13. $z :=$ **false**;
    end;
    end;
end.

4.8. Върху вероятността две случайно получени S-пермутационни матрици да са непресичащи се

Да означим с $\mathcal{P}_n$ групата от всички $n \times n$ пермутационни матрици, и със $\mathcal{S}_n$ симетричната група от n-ти ред, т.е. групата от всички взаимно-еднозначни изображения от множеството $[n]$ в себе си. Ако $x \in [n]$, $\rho \in \mathcal{S}_n$, то образа на елемента x при изображението ρ ще означаваме с $\rho(x)$. Както е добре известно [34, 66], съществува изоморфизъм

$$\theta : \mathcal{P}_n \to \mathcal{S}_n,$$

така че ако $A = [a_{ij}] \in \mathcal{P}_n$ и $\theta(A) = \rho \in \mathcal{S}_n$, то

$$a_{ij} = 1 \Leftrightarrow \rho(i) = j. \tag{4.8.1}$$

Нека $i \in [n]$ и нека $\rho \in \mathcal{S}_n$. Ние ще наричаме i *неподвижна точка* или *среща* (*rencontre*) ако $\rho(i) = i$. Както е добре известно (виж напр. [84, стр. 159], или [77,

зад. 1.6.22 б)]) броят $e_p(n)$ на всички пермутации от $\mathcal{S}_n$ с точно p неподвижни точки е равен на

$$e_p(n) = \frac{n!}{p!} \sum_{k=0}^{n-p} \frac{(-1)^k}{k!} \tag{4.8.2}$$

Лесно се вижда, че

$$e_{n-1}(n) = 0 \tag{4.8.3}$$

При $p = 0$, т.е. ако $\rho(i) \neq i$ за всяко $i = 1, 2, \ldots, n$, $\rho \in \mathcal{S}_n$ се нарича *безпорядък* (*без неподвижни точки*) (*derangement*). Броят на всички безпорядъци в $\mathcal{S}_n$ (виж напр. [84, стр. 159], или [77, зад. 1.6.22 а)]) е равен на

$$c_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} \tag{4.8.4}$$

В [110] Roberto Fontana предлага алгоритъм който по случаен начин получава множество от n^2 на брой взаимно непресичащи се $n^2 \times n^2$ S-пермутационни матрици, където $n = 2, 3$. При $n = 3$ той изпълнява алгоритъма 1000 пъти и намира 105 различни множества от по девет взаимно непресичащи се S-пермутационни матрици. Тогава съгласно (4.1.9) той заключава, че съществуват най-малко $9! \cdot 105 = 38\,102\,400$ Судоку матрици. Това число е прекалено малко в сравнение с действителният брой на 9×9 Судоку матриците. В [108] е показано, че съществуват точно

$$9! \cdot 72^2 \cdot 2^7 \cdot 27\,704\,267\,971 = 6\,670\,903\,752\,021\,072\,936\,960$$

на брой 9×9 Судоку матрици.

За да изследваме ефективността на алгоритъма на Fontana е необходимо да изчислим вероятността две случайно генерирани матрици да бъдат непресичащи се и вземем в предвид и формула (4.1.8).

Теорема 4.8.1. *Нека $A \in \Sigma_{n^2}$. Тогава броят на всички матрици $B \in \Sigma_{n^2}$, които не се пресичат с A е равен на*

$$\xi_n = (n!)^{2n} \left(\sum_{k=0}^{n} \frac{(-1)^k}{k!} \right)^n \left(2 - \sum_{k=0}^{n} \frac{(-1)^k}{k!} \right)^n + R_n, \tag{4.8.5}$$

където

$$R_n \geq 0.$$

Доказателство. Лесно се вижда, че ако $A, B \in \Sigma_{n^2}$, то съществуват единствени $n^2 \times n^2$ пермутационни матрици $C, D \in \mathcal{P}_{n^2}$ от вида

$$C = \begin{bmatrix} C_1 & O & O & \cdots & O \\ O & C_2 & O & \cdots & O \\ O & O & C_3 & \cdots & O \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ O & O & O & \cdots & C_n \end{bmatrix}, \; D = \begin{bmatrix} D_1 & O & O & \cdots & O \\ O & D_2 & O & \cdots & O \\ O & O & D_3 & \cdots & O \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ O & O & O & \cdots & D_n \end{bmatrix} \tag{4.8.6}$$

където $C_i, D_i \in \mathcal{P}_n$ са пермутационни $n \times n$ матрици, $i = 1, 2, \ldots, n$, а O е нулевата $n \times n$ матрица и такива че

$$B = CAD. \tag{4.8.7}$$

Нека за всяко $i = 1, 2, \ldots, n$ елементите $\theta(C_i) \in \mathcal{S}_n$ да са без неподвижни точки или за всяко $i = 1, 2, \ldots, n$ елементите $\theta(D_i) \in \mathcal{S}_n$ да са без неподвижни

точки, където $\theta : \mathcal{P}_n \to \mathcal{S}_n$ е изоморфизма дефиниран с помощта на формулата (4.8.1). Лесно се вижда, че в този случай, който ние ще наричаме *основен случай*, S-пермутационните матрици A и $B = CAD$ са непресичащи се. Това съгласно (4.8.4) може да стане по

$$\zeta_n = \left(c_n \cdot n! + n! \cdot c_n - c_n^2\right)^n$$

$$= \left[2\,(n!)^2 \sum_{k=0}^{n} \frac{(-1)^k}{k!} - (n!)^2 \left(\sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^2\right]^n$$

$$= (n!)^{2n} \left(\sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n \left(2 - \sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n$$

начина. Очевидно

$$\xi_n = \zeta_n + R_n,$$

където $R_n \geq 0$ и R_n преброява всички случаи когато съществуват $i, j \in [n]$, такива че $\theta(C_i)$ и $\theta(D_j)$ имат неподвижни точки, но A и $B = CAD$ са непресичащи се.

□

Следствие 4.8.2. *Формула (4.1.8) непосредствено следва от формула (4.8.7).*

□

Следствие 4.8.2 е друго доказателство на твърдение 1 от [103].

Следствие 4.8.3. *Броят на всички непресичащи се ненаредени двойки от $n^2 \times n^2$ S-пермутационни матрици е равен на*

$$\eta_n = \frac{(n!)^{2n}}{2} \xi_n =$$

$$= \frac{(n!)^{2n}}{2} \left[(n!)^{2n} \left(\sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n \left(2 - \sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n + R_n\right],$$

където

$$R_n \geq 0.$$

Доказателството следва непосредствено от теорема 4.8.1, формула (4.1.8) и вземайки предвид факта, че релацията "непресичащи се" в множеството Σ_{n^2} на всички S-пермутационни матрици е симетрична и антирефлексивна.

□

Следствие 4.8.4. *Вероятността $p(n)$ две случайно получени $n^2 \times n^2$ S-пермутационни матрици да са непресичащи се е равна на*

$$p(n) = \frac{\xi_n}{(n!)^{2n} - 1} = \frac{(n!)^{2n} \left(\sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n \left(2 - \sum_{k=0}^{n} \frac{(-1)^k}{k!}\right)^n + R_n}{(n!)^{2n} - 1}.$$

Доказателство. Прилагайки следствие 4.8.3 и формула (4.1.8) получаваме:

$$p(n) = \frac{\eta_n}{\binom{|\Sigma_{n^2}|}{2}} =$$

$$= \frac{\frac{(n!)^{2n}}{2}\left[(n!)^{2n}\left(\sum_{k=0}^{n}\frac{(-1)^k}{k!}\right)^n\left(2-\sum_{k=0}^{n}\frac{(-1)^k}{k!}\right)^n + R_n\right]}{\frac{(n!)^{2n}\left((n!)^{2n}-1\right)}{2}} =$$

$$= \frac{(n!)^{2n}\left(\sum_{k=0}^{n}\frac{(-1)^k}{k!}\right)^n\left(2-\sum_{k=0}^{n}\frac{(-1)^k}{k!}\right)^n + R_n}{(n!)^{2n}-1} =$$

$$= \frac{\xi_n}{(n!)^{2n}-1}.$$

□

4.8.1. Приложение на теорема 4.8.1 и нейните следствия за $n = 2$ и $n = 3$

Ние ще намерим стойността на ξ_n, η_n и $p(n)$ при $n = 2$ и $n = 3$. За по-големи стойности на n се изискват допълнителни усилия.

При $n = 2$

Имайки предвид (4.8.3) при $n = 2$ всяка от матриците C_1, C_2, D_1 и D_2, дефинирани с (4.8.6) и (4.8.7) е или безпорядък, или единичната матрица E_2 , където $\theta : \mathcal{P}_n \to \mathcal{S}_n$ е изоморфизма, дефиниран с помощта на формулата (4.8.1).

Нека $C_1 = E_2$. Тогава за да бъдат матриците A и $B = CAD$ непресичащи се е необходимо $\theta(D_1)$ и $\theta(D_2)$ да бъдат без неподвижни точки, но това е основния случай. (За дефиницията на този термин виж в доказателството на теорема 4.8.1). Аналогично се разглеждат случаите когато $C_2 = E_2$, $D_1 = E_2$ или $D_2 = E_2$. Следователно при $n = 2$ е изпълнено $R_2 = 0$. Така ние получаваме:

$$\xi_2 = 2^4\left(\sum_{k=0}^{2}\frac{(-1)^k}{k!}\right)^2\left(2-\sum_{k=0}^{2}\frac{(-1)^k}{k!}\right)^2 = 9$$

$$\eta_2 = \frac{2^4}{2}\xi_2 = 72$$

$$p(2) = \frac{\xi_2}{2^4-1} = \frac{3}{5}$$

Достоверността на получените резултати се потвърждава от получената с други методи в [178] стойност за $\eta_2 = 72$, където е използвана техника от теория на графите [182] и които ще опишем в глава 5.

При $n = 3$

За да изчислим R_3 е необходимо да разгледаме всички възможности за пермутационните матрици C_i и D_j, $1 \leq i, j \leq 3$ дефинирани с помощта на (4.8.6) и (4.8.7). Нека да означим с $\mathcal{E}_0$ множеството от всички безпорядъци в $\mathcal{S}_3$, т.е. $\mathcal{E}_0 = \{\rho \in \mathcal{S}_3 \mid \rho(i) \neq i,\ i = 1, 2, 3\}$. Необходимо е да се разгледат следните случаи:

i) Съществуват единствени $i, j \in \{1, 2, 3\}$, такива че $\theta(C_i), \theta(D_j) \notin \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се.

ii) Съществуват единствени $i, j \in \{1, 2, 3\}$, такива че $\theta(C_i), \theta(D_j) \in \mathcal{E}_0$, при това A и $B = CAD$ да са непресичащи се.

iii) Съществуват $i_1, i_2, i_3, j_1, j_2, j_3 \in \{1, 2, 3\}$, $i_s \neq i_t$, $j_s \neq j_t$ при $s \neq t$, такива че $\theta(C_{i_1}), \theta(C_{i_2}), \theta(D_{j_1}) \notin \mathcal{E}_0$, $\theta(C_{i_3}), \theta(D_{j_2}), \theta(D_{j_3}) \in \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се.

iv) Съществуват $i_1, i_2, i_3, j_1, j_2, j_3 \in \{1, 2, 3\}$, $i_s \neq i_t$, $j_s \neq j_t$ при $s \neq t$, такива че $\theta(C_{i_1}), \theta(D_{j_1}), \theta(D_{j_2}) \notin \mathcal{E}_0$, $\theta(C_{i_2}), \theta(C_{i_3}), \theta(D_{j_3}) \in \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се. Разглежда се аналогично на случай iii.

v) $\theta(C_1), \theta(C_2), \theta(C_3) \notin \mathcal{E}_0$ и съществува единствено $j \in \{1, 2, 3\}$, такова че $\theta(D_j) \notin \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се.

vi) $\theta(D_1), \theta(D_2), \theta(D_3) \notin \mathcal{E}_0$, съществува единствено $i \in \{1, 2, 3\}$, такова че $\theta(C_i) \notin \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се. Разглежда се аналогично на случай v.

vii) Съществува единствено $i \in \{1, 2, 3\}$, такова че $\theta(C_i) \in \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се.

viii) Съществува единствено $j \in \{1, 2, 3\}$, такова че $\theta(D_j) \in \mathcal{E}_0$, при това A и $B = CAD$ са непресичащи се. Разглежда се аналогично на случай vii.

След направените рутинни изчисления, ние получихме, че $R_3 = 19\,008$. За ζ_3 получаваме:

$$\zeta_3 = (3!)^6 \left(\sum_{k=0}^{3} \frac{(-1)^k}{k!} \right)^3 \left(2 - \sum_{k=0}^{3} \frac{(-1)^k}{k!} \right)^3 = 8\,000,$$

откъдето

$$\xi_3 = \zeta_3 + R_3 = 27\,008.$$

$$\eta_3 = \frac{(3!)^6}{2} \xi_3 = 630\,042\,624$$

$$p(3) = \frac{\xi_3}{(3!)^6 - 1} = \frac{27\,008}{46655} \approx 0,579$$

Достоверността на получените резултати се потвърждава от получената с други методи в [178] стойност за $\eta_3 = 630\,042\,624$, където е използвана техника от теория на графите [182] и които ще опишем в глава 5.

4.9. Заключение към четвърта глава

В настоящата глава насочихме вниманието си основно на бинарните матрици. Тук ние разгледахме някои чисто комбинаторни задачи, които възникнаха при решаването на задачи, свързани с математическото моделиране на обекти от реалния свят, каквито са например задачите, свързани с приложението на бинарните матрици в текстилната техника и които решихме в глава 2. Решавайки задачите, свързани с практиката получихме и някои интересни резултати, които имат чисто

теоретично значение от областта на алгебрата и комбинаториката и така възникна идеята за тяхното обстойно разглеждане. В процеса на работата възникнаха задачи, които обединихме и разгледахме в настоящата глава.

С помощта на едно разбиване на множеството Λ_n^k от всички $n \times n$ бинарни матрици с точно k на брой единици на всеки ред и всеки стълб получихме съотношението 4.2.6 (стр. 73). С негова помощ предложихме ново оригинално доказателство на известно твърдение, формулирано и доказано от I. Good и J. Grook в [115] и се доближихме в значителна степен до решаването на открития научен проблем за намирането на подобни съотношения при по-голяма стойност на целочисления параметър n.

В тази глава, както и на други места в дисертацията приложихме, не дотам широко разпространения подход за доказателството на математически твърдения като анализираме и точно оценим работата на алгоритъм с проверена и строго доказана ефективност. В случая ние демонстрирахме този метод, предлагайки ново конструктивно доказателство на известно твърдение, формулирано и доказано с чисто алгебрични методи (прилагайки известни твърдения от теория на групите) от В. Е. Тараканов в [65].

В раздел 4.4 разглеждаме функцията $\mu(n,k) = \left|\Lambda^k_{n/\sim}\right|$, където с Λ_n^k означаваме множеството от всички квадратни $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб, а $\sim$ е релация на еквивалентност, такава че две матрици са еквивалентни по между си, ако едната се получава от другата след разместване на редовете и/или стълбовете. Доказахме, че при $n = k+2$ редицата $\{\mu(k+2,k)\}_{k=0}^{\infty}$ съвпада с редицата на Фибоначи (теорема 4.4.3).

В сътрудничество с доц. д-р Д. Ковачев разгледахме някои комбинаторни задачи над бинарни матрици имащи приложение в k-значната логика. Резултатите от тези изследвания за описани в раздел 4.5.

В раздел 4.6 споделихме част от нашия опит за работата ни с изявени студенти по програмиране с цел повишаване на техния интерес като им предлагаме да опишат и реализират "по-добри" комбинаторни алгоритми за някои математически проблеми, които в някои случаи все още не са решени в аналитичен вид. Тук отново избираме примери, свързани с бинарни матрици и тяхното приложение в занимателната математика.

В раздел 4.7 са разгледани алгоритми за получаване на случайни обекти. Обърнато е внимание на метода Лас Вегас за получаване на случайни пермутации във връзка с един клас бинарни матрици, а именно S-пермутационните матрици. Така, получавайки по случаен начин n^2 на брой взаимно непресичащи се (mutually disjoint) S-пермутационни матрици, ние получаваме алгоритъм за получаване на случайна Судоку матрица. Обърнато е специално внимание на оценката на тези алгоритми по отношение на критерия време и е разгледано подробно понятието вероятностна оценка.

Някои аспекти по отношение на вероятността две случайно получени S-пермутационни матрици да са непресичащи се са разгледани в раздел 4.8. Пълното решение на тази задача с използване биполярни графи е дадено в следващата глава 5.

Глава 5

S-пермутационни матрици и биполярни графи

Резултатите от тази глава са публикувани в [178] и [182]. Първоначалният замисъл бе това да е една публикация, но по препоръка на рецензентите от издателство "Elsevier" материалът бе разделен на две статии. Така раздел 5.2 изцяло е публикуван в [182], а раздели 5.3 и 5.4 изцяло са публикувани в [178]. Уводната част (раздел 5.1) и заключителната част (раздел 5.5) присъстват в различна степен и в двете публикации.

Част от резултатите в обобщен вид са докладвани на конференцията [187].

5.1. Биполярни графи – основни дефиниции и означения

Нека m е цяло положително число и нека $\mathcal{S}_m$ е симетричната група от m-ти ред. Ако $x \in [m] = \{1, 2, \ldots, m\}$, $\rho \in \mathcal{S}_m$, то образът на елементите x при изображението ρ означаваме с $\rho(x)$.

Биполярен граф се нарича графът

$$g = \langle R_g \cup C_g, E_g \rangle,$$

където R_g и C_g са непразни множества, такива че $R_g \cap C_g = \emptyset$, елементите на които ще наричаме *върхове*. $E_g \subseteq R_g \times C_g = \{\langle r, c\rangle \mid r \in R_g, c \in C_g\}$ - множество от *ребра*. В нашите разглеждания не се допускат повтарящи се ребра.

Нека $g' = \langle R_{g'} \cup C_{g'}, E_{g'} \rangle$ и $g'' = \langle R_{g''} \cup C_{g''}, E_{g''} \rangle$. Ще казваме, че графите g' и g'' са *изоморфни* и ще пишем $g' \cong g''$, ако $R_{g'} = R_{g''}$, $C_{g'} = C_{g''}$ и съществуват $\rho \in \mathcal{S}_m$ и $\sigma \in \mathcal{S}_n$, където $m = |R_{g'}| = |R_{g''}|$, $n = |C_{g'}| = |C_{g''}|$, такива че $\langle r, c\rangle \in E_{g'} \iff \langle \rho(r), \sigma(c)\rangle \in E_{g''}$. Предмет на настоящата работа са биполярни графи, разглеждани с точност до изоморфизъм.

За повече подробности от теорията на графите препоръчваме книгите [38, 73, 107, 119].

Нека n и k са цели положителни числа и нека $0 \leq k \leq n^2$. Да означим с $\mathfrak{G}_{n,k}$ множеството от всевъзможните биполярни графи без кратни ребра от вида $g = \langle R_g \cup C_g, E_g \rangle$, разгледани с точност до изоморфизъм, такива че $|R_g| = |C_g| = n$ и $|E_g| = k$.

Нека $g = \langle R_g \cup C_g, E_g \rangle \in \mathfrak{G}_{n,k}$ за някои естествени числа n и k и нека $v \in V_g = R_g \cup C_g$. С $\gamma(v)$ ще означаваме множеството от всички върхове от V_g, инцидентни с v, т.е. $u \in \gamma(v)$ тогава и само тогава, когато съществува ребро в E_g съединяващо u и v. Ако v е изолиран връх (т.е. не съществува ребро, инцидентно с v), то по дефиниция $\gamma(v) = \emptyset$ и $|\gamma(v)| = 0$. Очевидно, ако $v \in R_g$, то $\gamma(v) \subseteq C_g$, а ако $v \in C_g$, то $\gamma(v) \subseteq R_g$. Очевидно

$$\sum_{v \in V_g} |\gamma(v)| = 2k.$$

Нека $g = \langle R_g \cup C_g, E_g \rangle \in \mathfrak{G}_{n,k}$ и нека $u, v \in V_g = R_g \cup C_g$. Ще казваме, че u и v са еквивалентни и ще пишем $u \sim v$, ако $\gamma(u) = \gamma(v)$. Ако u и v са изолирани, то по дефиниция $u \sim v$ тогава и само тогава, когато u и v принадлежат едновременно или на R_g, или на C_g. Очевидно така въведената релация е релация на еквивалентност.

С $V_{g/\sim}$ ще означаваме полученото фактормножество (множеството от класовете на еквивалентност) относно релацията $\sim$ и нека

$$V_{g/\sim} = \{\Delta_1, \Delta_2, \ldots, \Delta_s\},$$

където $\Delta_i \subseteq R_g$, или $\Delta_i \subseteq C_g$, $i = 1, 2, \ldots s$, $2 \leq s \leq 2n$. Полагаме

$$\delta_i = |\Delta_i|, \quad 1 \leq \delta_i \leq n, \quad i = 1, 2, \ldots, s$$

и нека за всяко $g \in \mathfrak{G}_{n,k}$ да дефинираме мултимножеството (множество с повторение)

$$[g] = \{\delta_1, \delta_2, \ldots \delta_s\},$$

където $\delta_1, \delta_2, \ldots, \delta_s$ са естествените числа, получени по описания по-горе начин. Очевидно

$$\sum_{i=1}^{s} \delta_i = 2n.$$

Ако $z_1\ z_2\ \ldots\ z_n$ е пермутация на елементите на множеството $[n] = \{1, 2, \ldots, n\}$ и тази пермутация означим на кратко с π, то $\pi(i)$ в случая ще означава i-я елемент на тази пермутация, т.е. $\pi(i) = z_i$, $i = 1, 2, \ldots, n$.

5.2. Върху броят на двойките непресичащи се S-пермутационни матрици – теоретико графов подход

Лема 5.2.1. *Броят $b(n,k)$ на всички $n \times n$ бинарни матрици с точно k, $k = 0, 1, \ldots, n^2$, единици е равен на*

$$b(n,k) = (n!)^2 \sum_{g \in \mathfrak{G}_{n,k}} \frac{1}{\prod\limits_{\delta \in [g]} \delta!} \tag{5.2.1}$$

Доказателство. Нека $A = [a_{ij}]_{n\times n}$ е $n \times n$ бинарна матрица с точно k единици. Тогава построяваме граф $g = \langle R_g \cup C_g, E_g \rangle$, така, че множеството $R_g = \{r_1, r_2, \ldots, r_n\}$ да съответствува на редовете на A, а $C_g = \{c_1, c_2, \ldots, c_n\}$ да съответствува на стълбовете на A, при това има ребро съединяващо върховете r_i и c_j тогава и само тогава, когато $a_{ij} = 1$. Очевидно, така построеният граф принадлежи на $\mathfrak{G}_{n,k}$.

Обратно, нека $g = \langle R_g \cup C_g, E_g \rangle \in \mathfrak{G}_{n,k}$. Номерираме по произволен начин върховете от R_g с помощта на естествените числа от 1 до n без да повтаряме някой от номерата. По аналогичен начин номерираме и върховете от C_g. Това може да стане по $(n!)^2$ начина. Тогава образуваме бинарната $n\times n$ матрица $A = [a_{ij}]_{n\times n}$, така че $a_{ij} = 1$ тогава и само тогава когато съществува ребро в E_g, свързващо върха с номер i от R_g и върха с номер j от C_g. Тъй като $g \in \mathfrak{G}_{n,k}$, то така построената матрица има точно k единици. Лесно се вижда, че при $q, r \in [n]$ q-ти и r-ти редове на A са равни помежду си (т.е. матрицата A не се променя, ако разменим местата на тези два реда) тогава и само тогава, когато върховете от R_g съответно с номера q и r са еквивалентни относно релацията $\sim$. Аналогично твърдение е в сила и за стълбовете на матрицата A и върховете от множеството C_g, което доказва и верността на формула (5.2.1).

□

Следствие 5.2.2. *За всяко цяло число $n \geq 2$ и всяко цяло число k такова, че $0 \leq k \leq n^2$ е изпълнено:*

$$\sum_{g \in \mathfrak{G}_{n,k}} \frac{1}{\prod\limits_{\delta \in [g]} \delta!} = \frac{\binom{n^2}{k}}{(n!)^2} = \frac{(n^2)!}{(n!)^2 k!(n^2-k)!} \tag{5.2.2}$$

□

Дефиниция 5.2.3. *Да означим с Π_n множеството от всички $n \times n$ матрици образувани така, че $\pi \in \Pi_n$ тогава и само тогава, когато са изпълнени следните три условия:*

i) *елементите на π са наредени двойки числа $\langle i, j \rangle$, където $1 \leq i, j \leq n$;*

ii) *ако*

$$[\langle a_1, b_1 \rangle \quad \langle a_2, b_2 \rangle \quad \cdots \quad \langle a_n, b_n \rangle]$$

представлява i-я ред на π за някое $i \in [n]$, то числата $a_1\ a_2\ \ldots\ a_n$ в този ред образуват пермутация на елементите на множеството $[n]$;

iii) *ако*

$$\begin{bmatrix} \langle a_1, b_1 \rangle \\ \langle a_2, b_2 \rangle \\ \vdots \\ \langle a_n, b_n \rangle \end{bmatrix}$$

представлява j-я стълб на π за някое $j \in [n]$, то числата $b_1, b_2, \ldots, b_n$ в този ред образуват пермутация на елементите на множеството $[n]$.

От дефиниция 5.2.3 следва, че всеки ред и всеки стълб на матрица от множеството Π_n може да се отъждествi с пермутация на елементите на множеството $[n]$. Обратно за всяка $(2n)$-торка

$$\langle \langle \rho_1, \rho_2, \ldots, \rho_n \rangle, \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle \rangle,$$

където $\rho_i = \rho_i(1)\ \rho_i(2)\ \ldots\ \rho_i(n)$, $\sigma_j = \sigma_j(1)\ \sigma_j(2)\ \ldots\ \sigma_j(n)$, $1 \leq i, j \leq n$ са пермутации на елементите на $[n]$, то матрицата

$$\pi = \begin{bmatrix} \langle \rho_1(1), \sigma_1(1) \rangle & \langle \rho_1(2), \sigma_2(1) \rangle & \cdots & \langle \rho_1(n), \sigma_n(1) \rangle \\ \langle \rho_2(1), \sigma_1(2) \rangle & \langle \rho_2(2), \sigma_2(2) \rangle & \cdots & \langle \rho_2(n), \sigma_n(2) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \rho_n(1), \sigma_1(n) \rangle & \langle \rho_n(2), \sigma_2(n) \rangle & \cdots & \langle \rho_n(n), \sigma_n(n) \rangle \end{bmatrix}$$

е матрица от Π_n. Следователно

$$|\Pi_n| = (n!)^{2n} \tag{5.2.3}$$

Ще казваме, че матриците $\pi', \pi'' \in \Pi_n$, където $\pi' = \left[p'_{ij}\right]_{n \times n}$, $\pi'' = \left[p''_{ij}\right]_{n \times n}$ са *непресичащи се*, ако $p'_{ij} \neq p''_{ij}$ за всяка двойка индекси $i, j \in [n]$.

Лема 5.2.4. *Нека $n \geq 2$ е цяло число. Тогава между множествата Σ_{n^2} и Π_n съществува взаимно еднозначно съответствие.*

Доказателство. Нека $A \in \Sigma_{n^2}$. Тогава A има вида, показан с помощта на формула (4.7.12) и за всеки $i, j \in [n]$ в блока A_{ij} се съдържа единствена единица. Нека тази единица има координати (a_i, b_j). За всеки $i, j \in [n]$ получаваме наредените двойки числа $\langle a_i, b_j \rangle$, съответстващи на тези координати. Тъй като във всеки ред и във всеки стълб на A има единствена единица, то матрицата $[\alpha_{ij}]_{n\times n}$, където $\alpha_{ij} = \langle a_i, b_j \rangle$, $1 \leq i, j \leq n$, която се получава от така получените наредени двойки числа е матрица от Π_n, т.е. матрица удовлетворяваща условия i), ii) и iii).

Обратно, нека $[\alpha_{ij}]_{n\times n} \in \Pi_n$, където $\alpha_{ij} = \langle a_i, b_j \rangle$, $i, j \in [n]$, $a_i, b_j \in [n]$. Тогава за всеки $i, j \in [n]$ образуваме бинарните $n \times n$ матрици A_{ij} с единствена единица имаща координати (a_i, b_j). Така получаваме матрица от вида (4.7.12). Съгласно свойства i), ii) и iii) от дефиниция 5.2.3, то очевидно така получената матрица е S-пермутационна.

□

От лема 5.2.4 и формула (5.2.3) непосредствено следва и доказателството на твърдение 1 от [103].

Следствие 5.2.5. [103, Твърдение 1] *Броят на всички $n^2 \times n^2$ S-пермутационни матрици е равен на*

$$|\Sigma_{n^2}| = (n!)^{2n} \tag{5.2.4}$$

□

Така получихме още едно доказателство на твърдение 1 от [103] (сравнете със следствие 4.8.2).

Следствие 5.2.6. *Броят на всевъзможните двойки непресичащи се матрици от Σ_{n^2} е равен на броя на всевъзможните двойки непресичащи се матрици от Π_n.*

Доказателство. Лесно се вижда, че относно описаното в лема 5.2.4 взаимно еднозначно съответствие всяка двойка непресичащи се матрици от Σ_{n^2} ще съответства на двойка непресичащи се матрици от Π_n и обратно на всяка двойка непресичащи се матрици от Π_n ще съответства на двойка непресичащи се матрици от Σ_{n^2}.

□

Нека $\pi', \pi'' \in \Pi_n$, $\pi' = \left[p'_{ij}\right]_{n\times n}$, $\pi'' = \left[p''_{ij}\right]_{n\times n}$ и нека $i, j \in [n] = \{1, 2, \ldots, n\}$ са такива че $p'_{ij} = p''_{ij}$. В този случай ще казваме, че p'_{ij} and p''_{ij} са *покомпонентно равни* елементи.

Очевидно две Π_n-матрици са непресичащи се тогава и само тогава, когато нямат покомпонентно равни елементи.

Пример 5.2.7. Разглеждаме следните Π_3-матрици:

$$\pi' = [p'_{ij}] = \begin{bmatrix} \langle 1,2\rangle & \langle 3,1\rangle & \langle 2,3\rangle \\ \langle 2,1\rangle & \langle 3,3\rangle & \langle 1,2\rangle \\ \langle 3,3\rangle & \langle 1,2\rangle & \langle 2,1\rangle \end{bmatrix}$$

$$\pi'' = [p''_{ij}] = \begin{bmatrix} \langle 1,3\rangle & \langle 3,2\rangle & \langle 2,1\rangle \\ \langle 3,1\rangle & \langle 1,1\rangle & \langle 2,2\rangle \\ \langle 3,2\rangle & \langle 1,3\rangle & \langle 2,3\rangle \end{bmatrix}$$

$$\pi''' = [p'''_{ij}] = \begin{bmatrix} \langle 1,2\rangle & \langle 3,3\rangle & \langle 2,1\rangle \\ \langle 2,1\rangle & \langle 3,2\rangle & \langle 1,2\rangle \\ \langle 3,3\rangle & \langle 1,1\rangle & \langle 2,3\rangle \end{bmatrix}$$

Матриците π' и π'' са непресичащи се, защото нямат покомпонентно равни елементи.

Матриците π'' и π''' са пресичащи се, защото имат два покомпонентно равни елементи: $p''_{13} = p'''_{13} = \langle 2,1\rangle$ и $p''_{33} = p'''_{33} = \langle 2,3\rangle$.

Матриците π' и π''' са пресичащи се, защото имат четири покомпонентно равни елементи: $p'_{11} = p'''_{11} = \langle 1,2\rangle$, $p'_{21} = p'''_{21} = \langle 2,1\rangle$, $p'_{23} = p'''_{23} = \langle 1,2\rangle$ и $p'_{31} = p'''_{31} = \langle 3,3\rangle$,

Лема 5.2.8. *Броят $q(n,k)$ на всевъзможните наредени двойки матрици $\langle \pi', \pi''\rangle$, където $\pi', \pi'' \in \Pi_n$, имащи поне k, $k = 0,1,\ldots,n^2$, покомпонентно равни елемента е равен на*

$$q(n,k) = (n!)^{2(n+1)} \sum_{g\in\mathfrak{G}_{n,k}} \frac{\prod\limits_{v\in R_g\cup C_g} (n-|\gamma(v)|)!}{\prod\limits_{\delta\in[g]} \delta!} \tag{5.2.5}$$

Доказателство. Нека $\pi' = \left[p'_{ij}\right]_{n\times n}, \pi'' = \left[p''_{ij}\right]_{n\times n} \in \Pi_n$ и нека π' и π'' имат точно k на брой покомпонентно равни елемента. Тогава еднозначно получаваме бинарната $n\times n$ матрица $A = [a_{ij}]_{n\times n}$, такава че $a_{ij} = 1$ тогава и само тогава, когато $p'_{ij} = p''_{ij}$, $i,j \in [n]$. Съгласно лема 5.2.1 съществува единствен граф $g \in \mathfrak{G}_{n,k}$, съответстващ на матрицата A и респективно на наредената двойка матрици $\langle \pi', \pi''\rangle \in \Pi\times\Pi$.

Обратно, нека $g = \langle R_g \cup C_g, E_g\rangle \in \mathfrak{G}_{n,k}$ и нека $V_g = R_g \cup C_g$. Тогава съществуват $\dfrac{(n!)^2}{\prod\limits_{\delta\in[g]}\delta!}$ на брой $n\times n$ бинарни матрици, съответстващи на g по описаното в лема 5.2.1 правило и нека $A = [a_{ij}]_{n\times n}$ е една от тях. Нека $\pi = [p_{ij}]_{n\times n}$ е произволна матрица от Π_n. Търсим броя $h(\pi, A)$ на всички матрици $\pi' = [p'_{ij}]_{n\times n} \in \Pi_n$, такива че $p'_{ij} = p_{ij}$, ако $a_{ij} = 1$. Допуска се да съществуват $s,t \in [n]$ такива, че $a_{st} = 0$ и $p'_{st} = p_{st}$. Нека i-ят ред ($i = 1,2,\ldots,n$) на π да съответствува на пермутацията ρ_i на елементите на $[n]$ и нека i-ят ред на матрицата A да съответства на върха $r_i \in R_g$ на графа g (виж лема 5.2.1). Тогава съществуват $(n-|\gamma(r_i)|)!$ на брой пермутации ρ'_i на елементите на $[n]$, такива че ако $a_{it} = 1$, то $\rho_i(t) = \rho'_i(t)$, $t \in [n]$. По аналогичен начин доказваме и съответното твърдение за стълбовете на π. Следователно $h(\pi, A) = \prod\limits_{v\in V_g} (n-|\gamma(v)|)!$. От всичко казано до тук следва, че за всяко $\pi \in \Pi_n$ съществуват

$$\sum_{g\in\mathfrak{G}_{n,k}} \frac{(n!)^2}{\prod\limits_{\delta\in[g]}\delta!} \prod_{v\in V_g} (n-|\gamma(v)|)!$$

матрици от Π_n, които имат поне k на брой елементи, които са покомпонентно равни на съответните елементи на π. Но съгласно формула (5.2.3) $|\Pi_n| = (n!)^{2n}$, откъдето следва и формула (5.2.5).

□

Теорема 5.2.9. *Нека $n \geq$ е цяло число. Тогава броят D_{n^2} на всевъзможните*

наредени двойки непресичащи се матрици от Σ_{n^2} *е равен на*

$$D_{n^2} = (n!)^{4n} + (n!)^{2(n+1)} \sum_{k=1}^{n^2} (-1)^k \sum_{g \in \mathfrak{G}_{n,k}} \frac{\prod\limits_{v \in R_g \cup C_g} (n - |\gamma(v)|)!}{\prod\limits_{\delta \in [g]} \delta!}. \tag{5.2.6}$$

Броят d_{n^2} *на всевъзможните ненаредени двойки непресичащи се матрици от* Σ_{n^2} *е равен на*

$$d_{n^2} = \frac{1}{2} D_{n^2} \tag{5.2.7}$$

Доказателство. Нека $n \geq 2$ е цяло число. Тогава прилагайки лема 5.2.4 и принципа на включване-изключване получаваме, че броят D_{n^2} на всевъзможните наредени двойки непресичащи се матрици от Σ_{n^2} е равен на

$$D_{n^2} = |\Pi_n|^2 + \sum_{k=1}^{n^2} (-1)^k q(n,k),$$

където функцията $q(n,k)$ е дефинирана в лема 5.2.8 и може да бъде изчислена с помощта на формула (5.2.5), а $|\Pi_n|$ с помощта на формула (5.2.3). Така получаваме и доказателството на формула (5.2.6).

Формула (5.2.7) получаваме вземайки предвид, че релацията "непресичащи се" е симетрична и антирефлексивна.

Теорема 5.2.9 е доказана.

□

За да приложим теорема 5.2.9 е необходимо да опишем всички с точност до изоморфизъм биполярни графи $g = \langle R_g \cup C_g, E_g \rangle$, където $|R_g| = |C_g| = n$.

Нека n и k са цели положителни числа и нека $g \in \mathfrak{G}_{n,k}$. Разглеждаме наредената $(n+1)$-торка

$$\langle \psi \rangle(g) = \langle \psi_0(g), \psi_1(g), \ldots, \psi_n(g) \rangle,$$

където $\psi_i(g)$, $i = 0, 1, \ldots, n$ е равен на броя на върховете на g, инцидентни с точно i на брой ребра. Очевидно за всяко $g \in \mathfrak{G}_{n,k}$ е изпълнено $\sum\limits_{i=1}^{n} i\psi_i(g) = 2k$. Тогава формула (5.2.6) може да се представи във вида:

$$D_{n^2} = (n!)^{4n} + (n!)^{2(n+1)} \sum_{k=1}^{n^2} (-1)^k \sum_{g \in \mathfrak{G}_{n,k}} \frac{\prod\limits_{i=0}^{n} [(n-i)!]^{\psi_i(g)}}{\prod\limits_{\delta \in [g]} \delta!}.$$

И тъй като $(n-n)! = 0! = 1$ и $[n-(n-1)]! = 1! = 1$, то получаваме:

$$D_{n^2} = (n!)^{4n} + (n!)^{2(n+1)} \sum_{k=1}^{n^2} (-1)^k \sum_{g \in \mathfrak{G}_{n,k}} \frac{\prod\limits_{i=0}^{n-2} [(n-i)!]^{\psi_i(g)}}{\prod\limits_{\delta \in [g]} \delta!}. \tag{5.2.8}$$

Следователно, за да приложим формула (5.2.8) е необходимо за всеки биполярен граф $g \in \mathfrak{G}_{n,k}$ и за множеството $\mathfrak{G}_{n,k}$ от биполярни графи да получим следните числови характеристики:

$$\omega(g) = \frac{\prod_{i=0}^{n-2} [(n-i)!]^{\psi_i(g)}}{\prod_{\delta \in [g]} \delta!} \tag{5.2.9}$$

и

$$\theta(n,k) = \sum_{g \in \mathfrak{G}_{n,k}} \omega(g) \tag{5.2.10}$$

Използувайки числовите характеристики, описани с формули (5.2.9) и (5.2.10), получаваме следната формула за изчисляване на D_{n^2}:

Теорема 5.2.10. *Нека $n \geq 2$ е цяло число. Тогава*

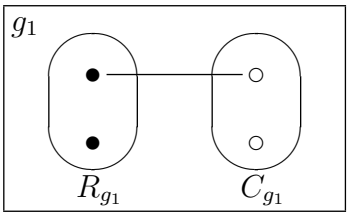
$$D_{n^2} = (n!)^{4n} + (n!)^{2(n+1)} \sum_{k=1}^{n^2} (-1)^k \theta(n,k), \tag{5.2.11}$$

където $\theta(n,k)$ е описана с помощта на формули (5.2.10) и (5.2.9) функция.

5.3. Пресмятане на броя D_4 на всички наредени двойки непресичащи се S-пермутационни матрици при $n = 2$

5.3.1. $k = 1$

При $n = 2$ и $k = 1$, $\mathfrak{G}_{2,1}$ се състои от единствен с точност до изоморфизъм граф g_1 изобразен на фигура 5.1.



Фигура 5.1. $n = 2$, $k = 1$

За графа $g_1 \in \mathfrak{G}_{2,1}$ имаме:

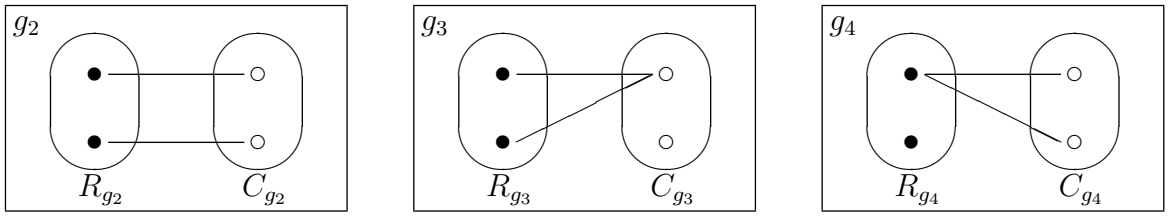
$$[g_1] = \{1,1,1,1\}$$
$$\langle\psi\rangle(g_1) = \langle\psi_0(g_1), \psi_1(g_1), \psi_2(g_1)\rangle = \langle 2,2,0\rangle$$

Тогава получаваме:

$$\omega(g_1) = \frac{[(2-0)!]^2}{1!\,1!\,1!\,1!} = 4$$

и следователно

$$\theta(2,1) = \sum_{g \in \mathfrak{G}_{2,1}} \omega(g) = 4. \tag{5.3.1}$$



Фигура 5.2. $n = 2$, $k = 2$

5.3.2. $k = 2$

Множеството $\mathfrak{G}_{2,2}$ се състои от трите графа g_2, g_3 и g_4, изобразени на фигура 5.2 с точност до изоморфизъм.

За графа $g_2 \in \mathfrak{G}_{2,2}$ имаме:

$$[g_2] = \{1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_2) = \langle\psi_0(g_2), \psi_1(g_2), \psi_2(g_2)\rangle = \langle 0, 4, 0\rangle$$

$$\omega(g_1) = \frac{[(2-0)!]^0}{1!\ 1!\ 1!\ 1!} = 1$$

За графите $g_3 \in \mathfrak{G}_{2,2}$ и $g_4 \in \mathfrak{G}_{2,2}$ имаме:

$$[g_3] = [g_4] = \{2, 1, 1\}$$

$$\langle\psi\rangle(g_3) = \langle\psi\rangle(g_4) = \langle 1, 2, 1\rangle$$

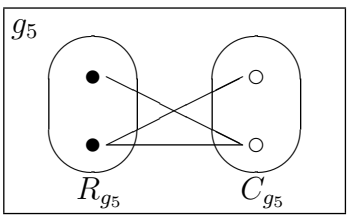
$$\omega(g_3) = \omega(g_4) = \frac{[(2-0)!]^1}{2!\ 1!\ 1!} = 1$$

Тогава за множеството $\mathfrak{G}_{2,2}$ получаваме:

$$\theta(2, 2) = \sum_{g \in \mathfrak{G}_{2,2}} \omega(g) = 1 + 1 + 1 = 3. \tag{5.3.2}$$

5.3.3. $k = 3$

При $n = 2$ и $k = 3$ имаме единствен с точност до изоморфизъм граф g_5, показан на фигура 5.3



Фигура 5.3. $n = 2$, $k = 3$

За графа $g_5 \in \mathfrak{G}_{2,3}$ имаме:

$$[g_5] = \{1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_5) = \langle\psi_0(g_5), \psi_1(g_5), \psi_2(g_5)\rangle = \langle 0, 2, 2\rangle$$

Тогава получаваме:

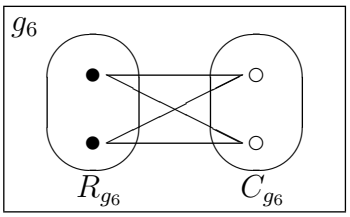
$$\omega(g_5) = \frac{[(2-0)!]^0}{1!\,1!\,1!\,1!} = 1$$

и следователно

$$\theta(2,3) = \sum_{g \in \mathfrak{G}_{2,3}} \omega(g) = 1. \tag{5.3.3}$$

5.3.4. $k = 4$

При $n = 2$ и $k = 4$ съществува единствен граф и това е пълният 2×2 биполярен граф g_6, който е показан на фигура 5.4.



Фигура 5.4. $n = 2$, $k = 4$

За графа $g_6 \in \mathfrak{G}_{2,4}$ имаме:

$$[g_6] = \{2, 2\}$$
$$\langle\psi\rangle(g_6) = \langle\psi_0(g_6), \psi_1(g_6), \psi_2(g_6)\rangle = \langle 0, 0, 4\rangle$$

Тогава получаваме:

$$\omega(g_6) = \frac{[(2-0)!]^0}{2!\,2!} = \frac{1}{4}$$

и следователно

$$\theta(2,4) = \sum_{g \in \mathfrak{G}_{2,1}} \omega(g) = \frac{1}{4}. \tag{5.3.4}$$

Вземайки предвид формули (5.2.11), (5.3.1), (5.3.2), (5.3.3) и (5.3.4) за броя D_4 на всички наредени двойки непресичащи се S-пермутационни матрици при $n = 2$, окончателно получаваме:

$$\begin{aligned} D_4 &= (2!)^8 + (2!)^6\,[-\theta(2,1) + \theta(2,2) - \theta(2,3) + \theta(2,4)] = \\ &= 256 + 64\left(-4 + 3 - 1 + \tfrac{1}{4}\right) = 144. \end{aligned} \tag{5.3.5}$$

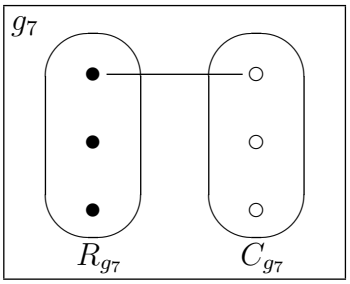
Броят d_4 на всевъзможните ненаредени двойки непресичащи се матрици от Σ_4 е равен на

$$d_4 = \frac{1}{2} D_4 = 72 \tag{5.3.6}$$

5.4. Пресмятане на броя D_9 на всички наредени двойки непресичащи се S-пермутационни матрици при $n = 3$

5.4.1. $k = 1$

Единственият биполярен граф принадлежащ на множеството $\mathfrak{G}_{3,1}$ е графа g_7 изобразен на фигура 5.5.



Фигура 5.5. $n = 3$, $k = 1$

За графа $g_7 \in \mathfrak{G}_{3,1}$ имаме:

$$[g_7] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_7) = \langle\psi_0(g_7), \psi_1(g_7), \psi_2(g_7), \psi_3(g_7), \psi_4(g_8)\rangle = \langle 4, 2, 0, 0\rangle$$

Тогава получаваме:

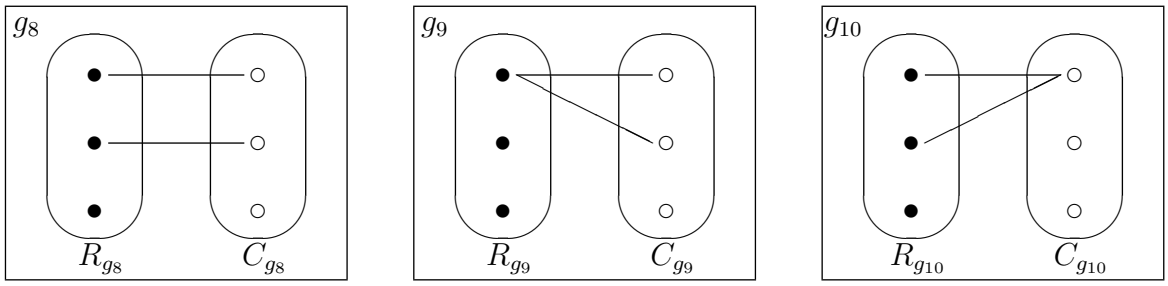
$$\omega(g_7) = \frac{[(3-0)!]^4\,[(3-1)!]^2}{1!\;1!\;2!\;2!} = \frac{6^4 \cdot 2^2}{1 \cdot 1 \cdot 2 \cdot 2} = 1296$$

и следователно

$$\theta(3,1) = \sum_{g \in \mathfrak{G}_{3,1}} \omega(g) = 1296. \tag{5.4.1}$$

5.4.2. $k = 2$

В този случай $\mathfrak{G}_{3,2} = \{g_8, g_9, g_{10}\}$. Биполярните графи g_8, g_9 и g_{10} са показани на фигура 5.6.



Фигура 5.6. $n = 3$, $k = 2$

За графа $g_8 \in \mathfrak{G}_{3,2}$ имаме:

$$[g_8] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_8) = \langle\psi_0(g_8), \psi_1(g_8), \psi_2(g_8), \psi_3(g_8), \psi_4(g_8)\rangle = \langle 2, 4, 0, 0\rangle$$

$$\omega(g_8) = \frac{[(3-0)!]^2\,[(3-1)!]^4}{1!\;1!\;1!\;1!\;1!\;1!} = 6^2 \cdot 2^4 = 576$$

За двата биполярни графа g_9 и g_{10} от $\mathfrak{G}_{3,2}$ имаме:

$$[g_9] = [g_{10}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_9) = \langle\psi\rangle(g_{10}) = \langle 3,2,1,0\rangle$$

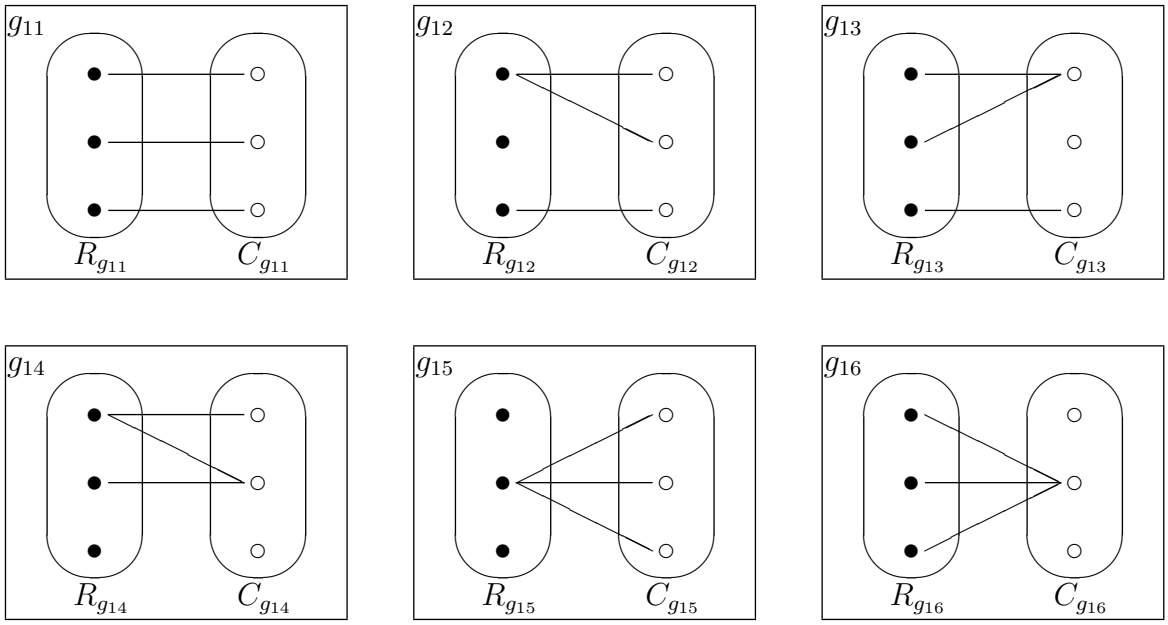
$$\omega(g_9) = \omega(g_{10}) = \frac{[(3-0)!]^3\,[(3-1)!]^2}{1!\;1!\;2!\;2!} = \frac{6^3\cdot 2^2}{1\cdot 1\cdot 2\cdot 2} = 216$$

Тогава

$$\theta(3,2) = \sum_{g\in\mathfrak{G}_{3,2}} \omega(g) = 576 + 216 + 216 = 1008. \tag{5.4.2}$$

5.4.3. $k = 3$

При $n = 3$ и $k = 3$ множеството $\mathfrak{G}_{3,3}$ се състои от шест биполярни графа, показани на фигура 5.7.



Фигура 5.7. $n = 3$, $k = 3$

За графа $g_{11} \in \mathfrak{G}_{3,3}$:

$$[g_{11}] = \{1,1,1,1,1,1\}$$

$$\langle\psi\rangle(g_{11}) = \langle 0,6,0,0\rangle$$

$$\omega(g_{11}) = \frac{[(3-0)!]^0\,[(3-1)!]^6}{1!\;1!\;1!\;1!\;1!\;1!} = 6^0\cdot 2^6 = 64$$

За графите $g_{12}, g_{13} \in \mathfrak{G}_{3,3}$:

$$[g_{12}] = [g_{13}] = \{1,1,1,1,2\}$$

$$\langle\psi\rangle(g_{12}) = \langle\psi\rangle(g_{13}) = \langle 1,4,1,0\rangle$$

$$\omega(g_{12}) = \omega(g_{13}) = \frac{[(3-0)!]^1\,[(3-1)!]^4}{1!\;1!\;1!\;1!\;2!} = \frac{6^1\cdot 2^4}{2} = 48$$

За графа $g_{14} \in \mathfrak{G}_{3,3}$:

$$[g_{14}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{14}) = \langle 2, 2, 2, 0\rangle$$

$$\omega(g_{14}) = \frac{[(3-0)!]^2\,[(3-1)!]^2}{1!\;1!\;1!\;1!\;1!\;1!} = 6^2 \cdot 2^2 = 144$$

За графите $g_{15}, g_{16} \in \mathfrak{G}_{3,3}$:

$$[g_{15}] = [g_{16}] = \{1, 2, 3\}$$

$$\langle\psi\rangle(g_{15}) = \langle\psi\rangle(g_{16}) = \langle 2, 3, 0, 1\rangle$$

$$\omega(g_{15}) = \omega(g_{16}) = \frac{[(3-0)!]^2\,[(3-1)!]^3}{1!\;2!\;3!} = \frac{6^2 \cdot 2^3}{2 \cdot 6} = 24$$

Тогава получаваме:

$$\theta(3,3) = \sum_{g \in \mathfrak{G}_{3,3}} \omega(g) = 64 + 48 + 48 + 144 + 24 + 24 = 352. \tag{5.4.3}$$

5.4.4. $k = 4$

Множеството $\mathfrak{G}_{3,4}$ се състои от седемте графа, изобразени на фигура 5.8.
За графа $g_{17} \in \mathfrak{G}_{3,4}$:

$$[g_{17}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_{17}) = \langle 2, 0, 4, 0\rangle$$

$$\omega(g_{17}) = \frac{[(3-0)!]^2\,[(3-1)!]^0}{1!\;1!\;2!\;2!} = \frac{6^2 \cdot 2^0}{2^2} = 9$$

За графа $g_{18} \in \mathfrak{G}_{3,4}$:

$$[g_{18}] = \{1, 1, 2, 2\}$$

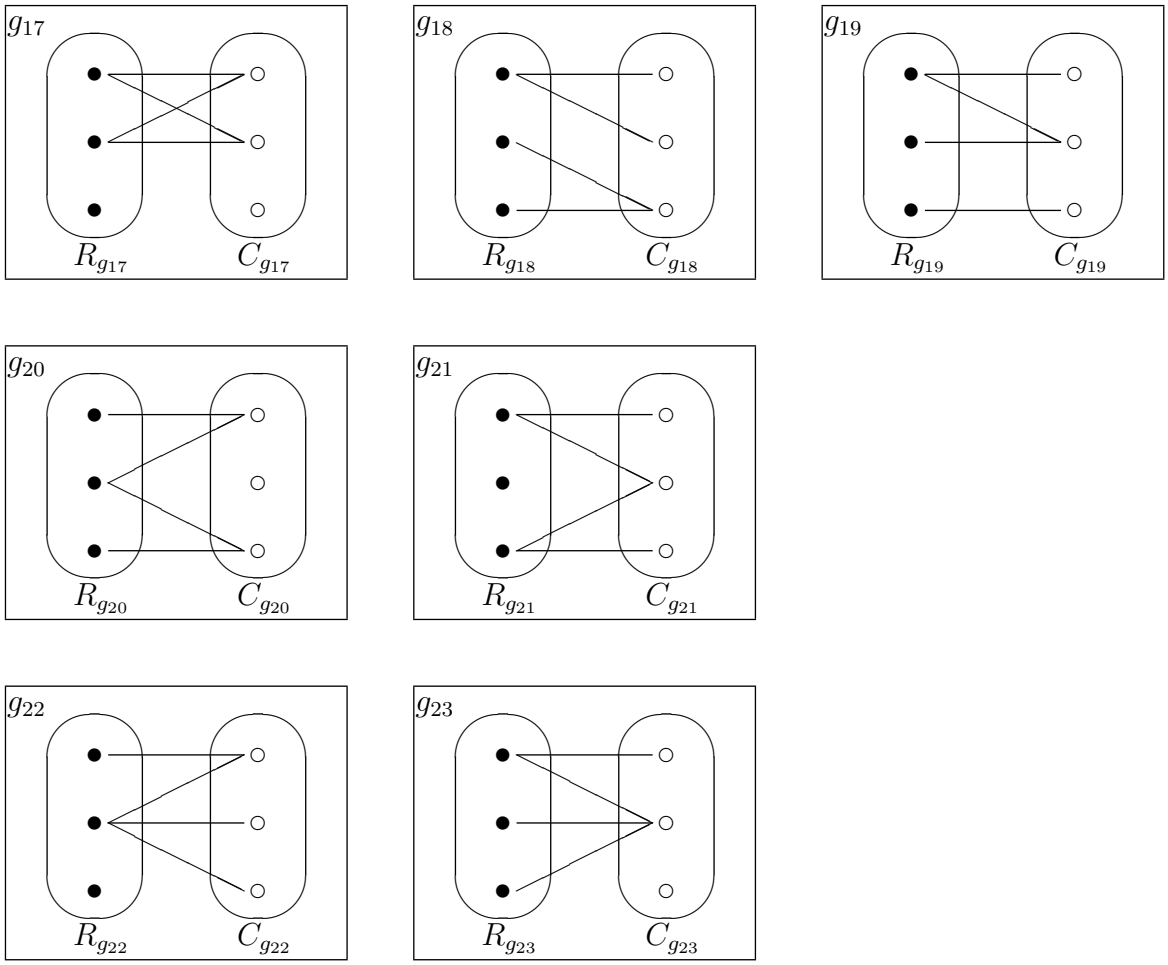
$$\langle\psi\rangle(g_{18}) = \langle 0, 4, 2, 0\rangle$$

$$\omega(g_{18}) = \frac{[(3-0)!]^0\,[(3-1)!]^4}{1!\;1!\;2!\;2!} = \frac{6^0 \cdot 2^4}{2^2} = 4$$

За графа $g_{19} \in \mathfrak{G}_{3,4}$:

$$[g_{19}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{19}) = \langle 0, 4, 2, 0\rangle$$



Фигура 5.8. $n = 3$, $k = 4$

$$\omega(g_{19}) = \frac{[(3-0)!]^0\,[(3-1)!]^4}{1!\;1!\;1!\;1!\;1!\;1!} = 6^0 \cdot 2^4 = 16$$

За графите $g_{20}, g_{21} \in \mathfrak{G}_{3,4}$:

$$[g_{20}] = [g_{21}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{20}) = \langle\psi\rangle(g_{21}) = \langle 1, 2, 3, 0\rangle$$

$$\omega(g_{20}) = \omega(g_{21}) = \frac{[(3-0)!]^1\,[(3-1)!]^2}{1!\;1!\;1!\;1!\;1!\;1!} = 6^1 \cdot 2^2 = 24$$

За графите $g_{22}, g_{23} \in \mathfrak{G}_{3,4}$:

$$[g_{22}] = [g_{23}] = \{1, 1, 1, 1, 2\}$$

$$\langle\psi\rangle(g_{22}) = \langle\psi\rangle(g_{23}) = \langle 1, 3, 1, 1\rangle$$

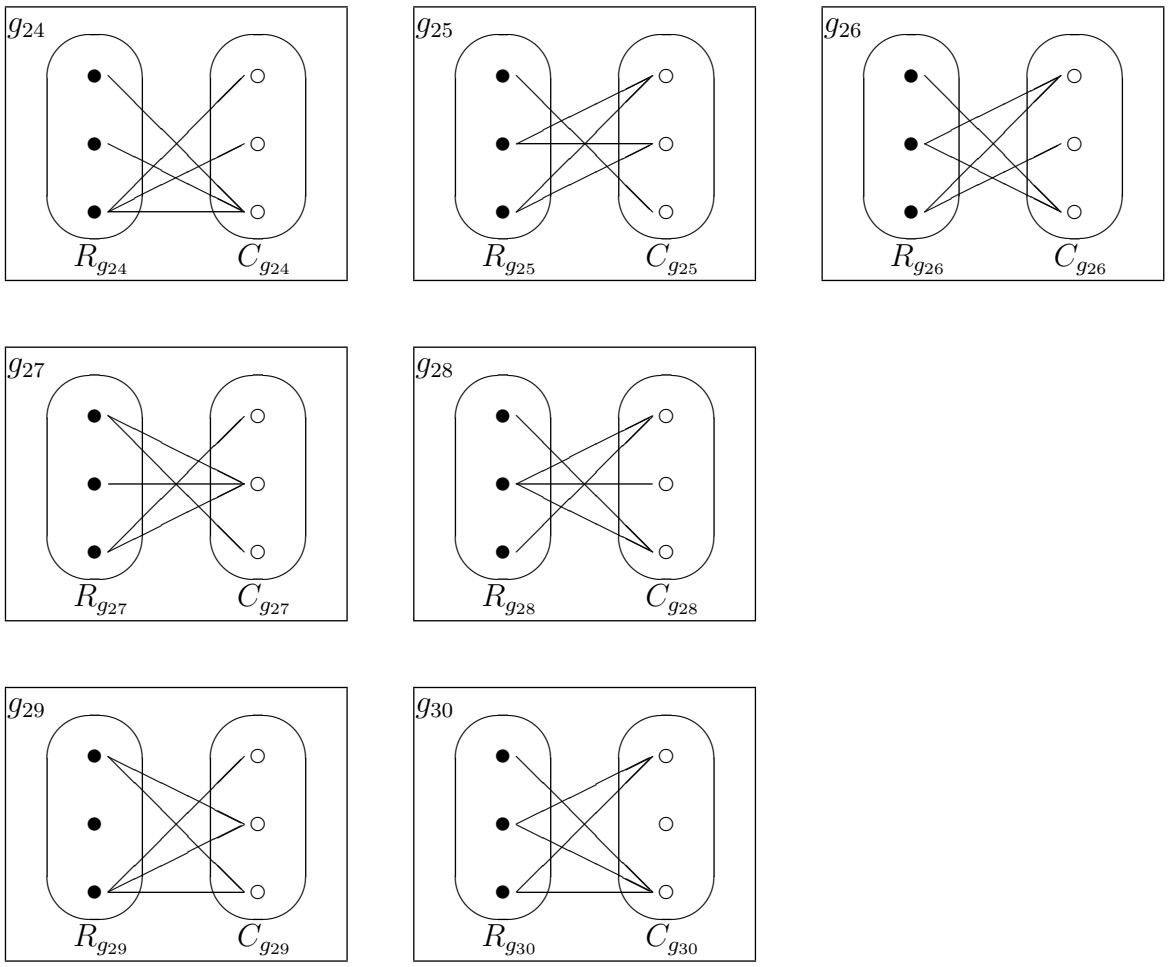
$$\omega(g_{22}) = \omega(g_{23}) = \frac{[(3-0)!]^1\,[(3-1)!]^3}{1!\;1!\;1!\;1!\;2!} = \frac{6^1 \cdot 2^3}{2} = 24$$

Тогава получаваме:

$$\theta(3,4) = \sum_{g \in \mathfrak{G}_{3,4}} \omega(g) = 9 + 4 + 16 + 24 + 24 + 24 + 24 = 125. \tag{5.4.4}$$

5.4.5. $k = 5$

Множеството $\mathfrak{G}_{3,5}$ се състои от графите $g_{24} \div g_{30}$, изобразени на фигура 5.9.



Фигура 5.9. $n = 3$, $k = 5$

За графа $g_{24} \in \mathfrak{G}_{3,5}$:

$$[g_{24}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_{24}) = \langle 0, 4, 0, 2\rangle$$

$$\omega(g_{18}) = \frac{[(3-0)!]^0\,[(3-1)!]^4}{1!\;1!\;2!\;2!} = \frac{6^0 \cdot 2^4}{2^2} = 4$$

За графа $g_{25} \in \mathfrak{G}_{3,5}$:

$$[g_{25}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_{25}) = \langle 0, 2, 4, 0\rangle$$

$$\omega(g_{18}) = \frac{[(3-0)!]^0\,[(3-1)!]^2}{1!\;1!\;2!\;2!} = \frac{6^0 \cdot 2^2}{2^2} = 1$$

За графа $g_{26} \in \mathfrak{G}_{3,5}$:

$$[g_{26}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{26}) = \langle 0, 2, 4, 0\rangle$$

$$\omega(g_{26}) = \frac{[(3-0)!]^0\,[(3-1)!]^2}{1!\;1!\;1!\;1!\;1!\;1!} = 6^0 \cdot 2^2 = 4$$

За графите $g_{27}, g_{28} \in \mathfrak{G}_{3,5}$:

$$[g_{27}] = [g_{28}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{27}) = \langle\psi\rangle(g_{28}) = \langle 0, 3, 2, 1\rangle$$

$$\omega(g_{27}) = \omega(g_{28}) = \frac{[(3-0)!]^0\,[(3-1)!]^3}{1!\;1!\;1!\;1!\;1!\;1!} = 6^0 \cdot 2^3 = 8$$

За графите $g_{29}, g_{30} \in \mathfrak{G}_{3,5}$:

$$[g_{29}] = [g_{30}] = \{1, 1, 1, 1, 2\}$$

$$\langle\psi\rangle(g_{29}) = \langle\psi\rangle(g_{30}) = \langle 1, 1, 3, 1\rangle$$

$$\omega(g_{29}) = \omega(g_{30}) = \frac{[(3-0)!]^1\,[(3-1)!]^1}{1!\;1!\;1!\;1!\;2!} = \frac{6^1 \cdot 2^1}{2} = 6$$

Тогава получаваме:

$$\theta(3,5) = \sum_{g \in \mathfrak{G}_{3,5}} \omega(g) = 4 + 1 + 4 + 8 + 8 + 6 + 6 = 37. \tag{5.4.5}$$

5.4.6. $k = 6$

При $n = 3$ и $k = 6$ имаме шест биполярни графа, принадлежащи на множеството $\mathfrak{G}_{3,6}$, които са показани на фигура 5.10

За графа $g_{31} \in \mathfrak{G}_{3,6}$ получаваме:

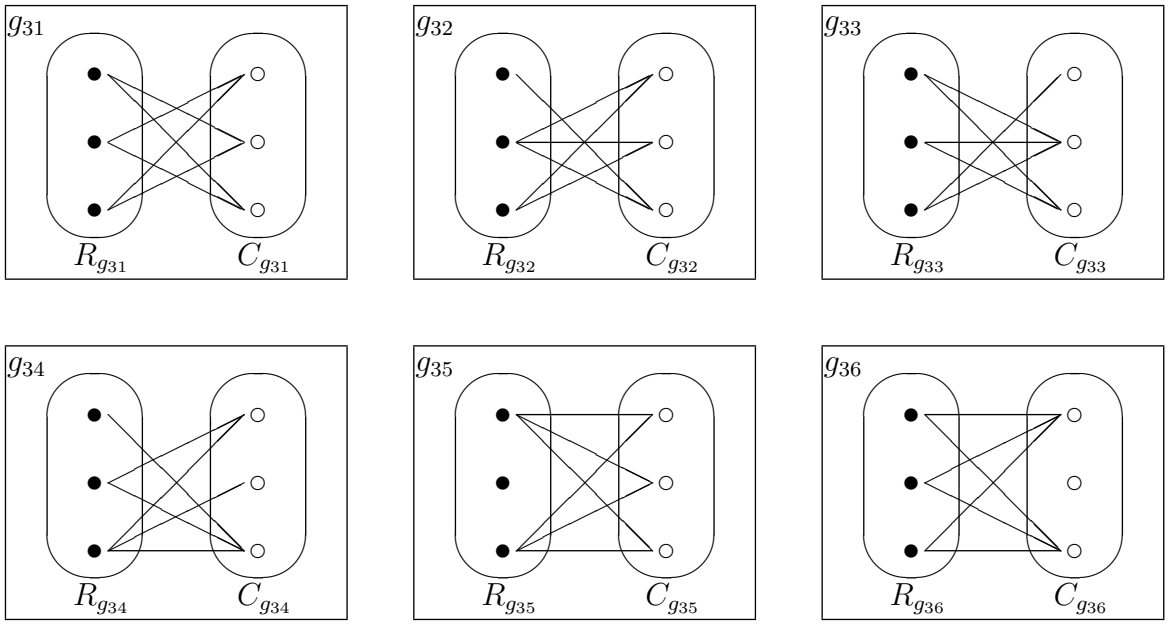
$$[g_{31}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{31}) = \langle 0, 0, 6, 0\rangle$$

$$\omega(g_{31}) = \frac{[(3-0)!]^0\,[(3-1)!]^0}{1!\;1!\;1!\;1!\;1!\;1!} = 1$$

За графите $g_{32}, g_{33} \in \mathfrak{G}_{3,6}$ получаваме:

$$[g_{32}] = [g_{33}] = \{1, 1, 1, 1, 2\}$$



Фигура 5.10. $n = 3$, $k = 6$

$$\langle\psi\rangle(g_{32}) = \langle\psi\rangle(g_{33}) = \langle 0, 1, 4, 1\rangle$$

$$\omega(g_{32}) = \omega(g_{33}) = \frac{[(3-0)!]^0\,[(3-1)!]^1}{1!\;1!\;1!\;1!\;2!} = \frac{6^0 \cdot 2^1}{2} = 1$$

За графа $g_{34} \in \mathfrak{G}_{3,6}$ получаваме:

$$[g_{34}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{34}) = \langle 0, 2, 2, 2\rangle$$

$$\omega(g_{34}) = \frac{[(3-0)!]^0\,[(3-1)!]^2}{1!\;1!\;1!\;1!\;1!\;1!} = \frac{6^0 \cdot 2^2}{1} = 4$$

За графите $g_{35}, g_{36} \in \mathfrak{G}_{3,6}$ получаваме:

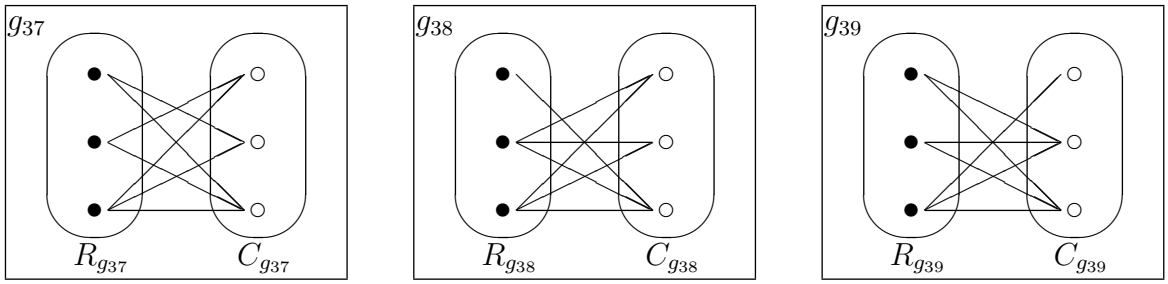
$$[g_{35}] = [g_{36}] = \{1, 2, 3\}$$

$$\langle\psi\rangle(g_{35}) = \langle\psi\rangle(g_{36}) = \langle 1, 0, 3, 2\rangle$$

$$\omega(g_{35}) = \omega(g_{36}) = \frac{[(3-0)!]^1\,[(3-1)!]^0}{1!\;2!\;3!} = \frac{6^1 \cdot 2^0}{2 \cdot 6} = \frac{1}{2}$$

Тогава:

$$\theta(3,6) = \sum_{g \in \mathfrak{G}_{3,6}} \omega(g) = 1 + 1 + 1 + 4 + \frac{1}{2} + \frac{1}{2} = 8 \tag{5.4.6}$$



Фигура 5.11. $n = 3$, $k = 7$

5.4.7. $k = 7$

При $n = 3$ и $k = 7$ имаме три графа, принадлежащи на множествата $\mathfrak{G}_{3,7} = \{g_{37}, g_{38}, g_{39}\}$, които са показани на фигура 5.11.

За графа $g_{37} \in \mathfrak{G}_{3,7}$ е изпълнено:

$$[g_{37}] = \{1, 1, 1, 1, 1, 1\}$$

$$\langle\psi\rangle(g_{37}) = \langle 0, 0, 4, 2\rangle$$

$$\omega(g_{37}) = \frac{[(3-0)!]^0\,[(3-1)!]^0}{1!\;1!\;1!\;1!\;1!\;1!} = \frac{6^0 \cdot 2^0}{1} = 1$$

За графите $g_{38}, g_{39} \in \mathfrak{G}_{3,7}$ получаваме:

$$[g_{38}] = [g_{39}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_{38}) = \langle\psi\rangle(g_{39}) = \langle 0, 1, 2, 3\rangle$$

$$\omega(g_{38}) = \omega(g_{39}) = \frac{[(3-0)!]^0\,[(3-1)!]^1}{1!\;1!\;2!\;2!} = \frac{6^0 \cdot 2^1}{2^2} = \frac{1}{2}$$

Тогава:

$$\theta(3,7) = \sum_{g \in \mathfrak{G}_{3,7}} \omega(g) = 1 + \frac{1}{2} + \frac{1}{2} = 2 \tag{5.4.7}$$

5.4.8. $k = 8$

При $n = 3$ и $k = 8$ имаме единствен граф $g_{40} \in \mathfrak{G}_{3,8}$, който е изобразен на фигура 5.12.

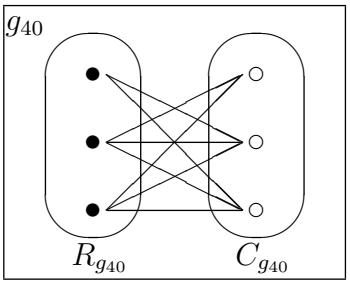
За графа $g_{40} \in \mathfrak{G}_{3,8}$ е изпълнено:

$$[g_{40}] = \{1, 1, 2, 2\}$$

$$\langle\psi\rangle(g_{40}) = \langle 0, 0, 2, 4\rangle$$

$$\omega(g_{40}) = \frac{[(3-0)!]^0\,[(3-1)!]^0}{1!\;1!\;2!\;2!} = \frac{6^0 \cdot 2^0}{2^2} = \frac{1}{4}$$

Следователно

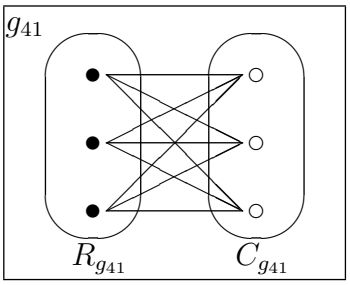


Фигура 5.12. $n = 3$, $k = 8$

$$\theta(3,8) = \sum_{g \in \mathfrak{G}_{3,8}} \omega(g) = \frac{1}{4} \tag{5.4.8}$$

5.4.9. $k = 9$

При $n = 3$ и $k = 9$ е възможен единствено пълният биполярен граф $g_{41} \in \mathfrak{G}_{3,9}$, изобразен на фигура 5.13.



Фигура 5.13. $n = 3$, $k = 9$

За графа $g_{41} \in \mathfrak{G}_{3,9}$ е изпълнено:

$$[g_{41}] = \{3,3\}$$

$$\langle\psi\rangle(g_{41}) = \langle 0,0,0,6\rangle$$

$$\omega(g_{41}) = \frac{[(3-0)!]^0\,[(3-1)!]^0}{3!\;3!} = \frac{6^0 \cdot 2^0}{6^2} = \frac{1}{36}$$

Следователно

$$\theta(3,9) = \sum_{g \in \mathfrak{G}_{3,9}} \omega(g) = \frac{1}{36} \tag{5.4.9}$$

Тогава вземайки предвид формула (5.2.11) и формули с номера от (5.4.1) до (5.4.9), за броя на всички наредени двойки непресичащи се S-пермутационни матрици при $n = 3$, т.е. при $N = 9$, получаваме:

$$\begin{gathered} D_9 = \\ = (3!)^{12} + (3!)^8 \left[\sum_{k=1}^{9} (-1)^k \theta(n,k)\right] = \\ = 2\ 176\ 782\ 336 + \\ +1\ 679\ 616 \left(-1296 + 1008 - 352 + 125 - 37 + 8 - 2 + \tfrac{1}{4} - \tfrac{1}{36}\right) = \\ = 1\ 260\ 085\ 248. \end{gathered} \tag{5.4.10}$$

За броя на всички ненаредени двойки непресичащи се S-пермутационни матрици при $n = 3$, т.е. при $N = 9$ получаваме:

$$d_9 = \frac{1}{2} D_9 = 630\ 042\ 624 \tag{5.4.11}$$

5.5. Върху един комбинаторен проблем от теория на графите, свързан с броя на Судоку матриците

Проблем 5.5.1. *Нека $n \geq 2$ е цяло число и нека G е неориентиран граф с $(n!)^{2n}$ върха. Нека всеки връх на G да е отъждествен с елемент от множеството Σ_{n^2} от всички $n^2 \times n^2$ S-пермутационни матрици. Два върха са свързани с ребро тогава и само тогава, когато съответните матрици са непресичащи се. Да се намери броя на всички пълни подграфи на G с n^2 върха.*

Да отбележим, че броят на ребрата в графа G, дефиниран в проблем 5.5.1, е равен на d_{n^2} и може да бъде изчислен с помощта на формули (5.2.6) и (5.2.7) (съответно с формули (5.2.9), (5.2.10), (5.2.11) и (5.2.7)).

Да означим с z_n решението на проблем 5.5.1 и нека σ_n е броят на всички $n^2 \times n^2$ Судоку матрици. Тогава съгласно твърдение 4.6.2 и начина на построяване на графа G, следва че е в сила зависимостта

$$z_n = \frac{\sigma_n}{(n^2)!} \tag{5.5.1}$$

На нас не ни е известна универсална формула за намиране на броя на всички $n^2 \times n^2$ Судоку матрици за всяко естествено число $n \geq 2$ и считаме, че това е открит комбинаторен проблем. Известни са само някои частни случай. Така например, при $n = 2$ е известно, че $\sigma_2 = 288$ (виж стр.54). Тогава съгласно формула (5.5.1) получаваме:

$$z_2 = \frac{\sigma_2}{4!} = \frac{288}{24} = 12$$

При $n = 3$ в [108] е показано, че съществуват точно

$$\sigma_3 = 6\ 670\ 903\ 752\ 021\ 072\ 936\ 960 =$$

$$= 9! \times 72^2 \times 2^7 \times 27\ 704\ 267\ 971 =$$

$$2^{20} \times 3^8 \times 5^1 \times 7^1 \times 27\ 704\ 267\ 971^1 \sim 6.671 \times 10^{21}$$

на брой Судоку матрици. Тогава съгласно формула (5.5.1) получаваме:

$$z_3 = \frac{\sigma_3}{9!} = \frac{6\ 670\ 903\ 752\ 021\ 072\ 936\ 960}{362\ 880} = 18\ 383\ 222\ 420\ 692\ 992$$

5.6. Заключение към пета глава

В тази глава използвахме един интересен подход с използване на теория на графите за решаване на чисто-комбинаторни задачи. С помощта на графови модели намерихме броя на всевъзможните двойки взаимно непресичащи се $n^2 \times n^2$ S-пермутационни матрици. От своя страна S-пермутационните матрици са интересен клас от бинарни матрици, свързани с популярната в последно време математическа главоблъсканица Судоку [35, 72, 167].

Глава 6

Теоретико-графови модели в компютърната лингвистика

Резултатите от раздел 6.1 са публикувани в [77, 175].
Резултатите от раздел 6.2 са публикувани в [179].
Резултатите от раздел 6.3 са публикувани в [180] като преди това са анонсирани на руски език на конференцията [23].

6.1. Формално-лингвистичен подход при решаване на една занимателна задача

6.1.1. Постановка на задачата

В този раздел продължаваме да експлоатираме принципа "Чрез развлечение към знание". Идеята е да повишим студентския интерес чрез подбора на интересни примери за демонстрация на преподавания материал. Тук ще разгледаме следната занимателна задача:

Задача 6.1.1. Задача за Ханойските кули. *Има три отвесни колони. На първата от тях са поставени N диска. Всички дискове са различни и подредени по големина - в основата най-големия, а на върха най-малкия. Задачата е да се преместят дисковете от първата на третата колона като се използува втората колона като помощна и се спазват следните правила:*

- *на всеки ход може да се пренася един единствен диск, а през това време всички останали дискове да бъдат на някои от трите колони;*
- *забранено е по-голям диск да лежи върху по-малък.*

Да се опише алгоритъм решаващ така поставената задача при произволно зададен брой n на дисковете.

Задачата за Ханойските кули е класически пример за използуване на рекурсия в програмирането [7, 8, 92, 166]. Тук ние ще разгледаме тази задача от гледна точка на математическата (компютърна) лингвистика - дял от дисциплините "дискретна математика" и "дискретни структури", изучавани от студентите съответно от специалности "информатика" и "компютърни системи и технологии" във висшите учебни заведения [7, 19, 36, 44, 77, 98].

Има три интересни гледни точки на задачата за Ханойските кули, свързани с терминологията на компютърната лингвистика:

1. Да се опише алгоритъм, решаващ задача 6.1.1, с помощта на краен автомат;
2. Да се опише алгоритъм, решаващ задача 6.1.1, с помощта на безконтекстна граматика;
3. Да се опише алгоритъм, решаващ задача 6.1.1, с помощта на магазинен автомат.

Решение на първата задача е описано в [85] (на френски език) и в по-нова версия на английски език в [86]. Искам да изкажа благодарността си на един от авторите на горепосочените разработки професор Jean-Paul Allouche, който в резултат на лична кореспонденция ми предостави необходимите материали и ме подтикна към по-нататъшните наши разработки, свързани с тази тематика.

Тук ние ще разгледаме втората и третата гледни точки на поставената задача за Ханойските кули от гледна точка на дискретната математика. Тези задачи са формулирани на български език с указания за тяхното решение в учебното пособие [77], а формулировката и пълното тяхно решение на английски език е представено в публикацията [175].

В този раздел ние ще припомним само най-необходимите дефиниции и означения. За повече подробности от областта математическата лингвистика и по-специално на безконтекстните граматики и езици и магазинните автомати виж книгите [18, 19, 36, 43, 44, 77, 123]. Алгебричните свойства на безконтекстните граматики и езици в подробности са разгледани в [98, 114, 138]. Редица практически приложения на формалните граматики и езици и на магазинните автомати са разгледани в [83, 123].

6.1.2. Безконтекстни (контекстно-свободни) граматики и езици

Нека V е крайно и непразно множество. Елементите на това множество ще наричаме *букви*, а самото множество V - *азбука*.

Дума над азбуката V ще наричаме всяка крайна редица от букви на V. Дума, в която не влиза нито една буквaq се нарича *празна дума*, която ще бележим с ε. С V^* ще означаваме множеството от всички думи над V, включително и празната. Под *дължина* на една дума се разбира броя на буквите в нея. Дължината на думата α ще означаваме с $|\alpha|$.

Нека α и β са две думи над азбуката V. Под *конкатенация* (*произведение, слепване*) $\alpha\beta$ на двете думи ще разбираме думата, получена от последователното дописване на буквите на β след последната буква на α.

Нека V е азбука. Всяко подмножество L на V^* се нарича *формален език* (или само *език*) над азбуката V.

Под *пораждаща граматика* (или само *граматика*) Γ ще разбираме наредената четворка $\Gamma = \langle V, W, S, P\rangle$, където V е крайно множество (азбука) от *терминални* символи, W-множество от *нетерминални* символи, S-*начален символ* на граматиката, който е и елемент на W, а P е множество от наредени двойки (α, β), за които $\alpha, \beta \in (V \cup W)^*$, като в α има поне един нетерминален символ. В редица източници е поставено допълнително условие множествата W и P да бъдат крайни. За нашите нужди това условие не е необходимо, като е достатъчно тези множества да са изброими. Елементите на P се наричат *правила*. Ако $(\alpha, \beta) \in P$, то ще означаваме $\alpha \to \beta$, като символът " $\to$ " не принадлежи на $V \cup W$.

Нека μ и ν са две думи от $(V \cup W)^*$. Ще казваме, че μ се *извежда непосредствено* от ν в граматиката $\Gamma = \langle V, W, S, P\rangle$ и ще пишем $\nu \overset{\Gamma}{\vdash} \mu$ (или само $\nu \vdash \mu$, ако Γ се подразбира), ако съществуват думи $\alpha_1, \alpha_2 \in (V \cup W)^*$ и правило $\alpha \to \beta$ в P така, че $\nu = \alpha_1\alpha\alpha_2$ и $\mu = \alpha_1\beta\alpha_2$.

Ако $\omega_0, \omega_1, \ldots, \omega_n$ са думи над $V \cup W$, за които $\omega_0 \overset{\Gamma}{\vdash} \omega_1 \overset{\Gamma}{\vdash} \ldots \overset{\Gamma}{\vdash} \omega_n$, ще казваме, че тази редица от думи е *извод* на ω_n от ω_0 в Γ, което ще означаваме с $\omega_0 \overset{\Gamma}{\models} \omega_n$ или само $\omega_0 \models \omega_n$, ако Γ се подразбира. Броят n на непосредствените изводи $\omega_i \overset{\Gamma}{\vdash} \omega_{i+1}$ ще наричаме *дължина на извода*.

Множеството $L(\Gamma) = \{\omega \in V^* \mid S \overset{\Gamma}{\models} \omega\}$ се нарича *формален език над* V, *породен от граматиката* Γ. Граматиките Γ_1 и Γ_2 са *еквивалентни*, ако $L(\Gamma_1) = L(\Gamma_2)$.

Една граматика $\Gamma = \langle V, W, S, P\rangle$ е *безконтекстна*, ако всичките ѝ правила са от вида

$$A \to \omega, \quad A \in V, \quad \omega \in (V \cup W)^*.$$

Да се върнем към задачата за Ханойските кули.

Задача 6.1.2. *За дадено цяло положително число* N *да се построи безконтекстна граматика* Γ_N *с входна азбука, кодираща възможните премествания на дисковете и ако* $\omega \in L(\Gamma_N)$, *то* ω *да описва алгоритъм, решаващ задача 6.1.1. Да се докаже, че за всяко цяло положително число* N *езикът* $L(\Gamma_N)$ *не е празен, т.е. за всяко цяло положително число съществува алгоритъм, решаващ задачата за Ханойските кули*[1].

Решение. Разглеждаме безконтекстната граматика $\Gamma_N = \langle V, W, S, P\rangle$, където $V = \{p_{ij} \mid i, j \in \{1, 2, 3\},\ i \neq j\}$. Смисълът на p_{ij} е "Премести най-горния диск от i-та колона на j-та колона". По този начин, ако $\omega = \pi_1\pi_2 \ldots \pi_k$, където $\pi_i \in V$, $i = 1, 2, \ldots, k$, то ω описва алгоритъм за последователно преместване на k диска по трите колони. $W = \{h_{ij}(n) \mid i, j \in \{1, 2, 3\},\ i \neq j,\ n = 1, 2, \ldots, N\}$, началният символ $S = h_{13}(N)$, а P се състои от правилата $h_{ij}(1) \to p_{ij}$ и $h_{ij}(n) \to h_{ik}(n-1)p_{ij}h_{kj}(n-1)$ за $n = 2, 3, \ldots, N$, където $i, j, k \in \{1, 2, 3\}$, $i \neq j$, $i \neq k$, $k \neq j$. Очевидно така построената граматика Γ_N е безконтекстна.

Нека $\omega \in L(\Gamma_N)$, т.е. допускаме, че съществува извод $h_{13}(N) \overset{\Gamma_N}{\models} \omega$ и нека дължината на извода е равен на s. Ако $s = 1$, то очевидно това е възможно тогава и само тогава, когато броят на дисковете $N = 1$ и имаме непосредствения извод $h_{13}(1) \overset{\Gamma_N}{\vdash} p_{13}$ и при наличие на единствен диск, p_{13} описва алгоритъм за решаване на задача 6.1.1. Аналогично $h_{ij}(1) \overset{\Gamma_N}{\vdash} p_{ij}$ е извод с дължина 1 с начален символ $h_{ij}(1)$ и p_{ij} описва алгоритъм преместващ единствен диск от колона i на колона j, където $i, j \in \{1, 2, 3\}$ и $i \neq j$. Нека $s > 1$. Допускаме, че ако съществува извод с дължина по-малка от s от вида $h_{ij}(n) \overset{\Gamma_N}{\models} \alpha$, където $\alpha \in V^*$, $n \leq N$, то α описва преместването на n диска от колона i върху колона j, използувайки колона k за помощна, съгласно посочените в задача 6.1.1 ограничения, където $i, j, k \in \{1, 2, 3\}$, $i \neq j$, $j \neq k$, $k \neq i$. При $s > 1$ очевидно изводът $h_{13}(N) \overset{\Gamma_N}{\models} \omega$ с дължина s (ако съществува) ще има вида $h_{13}(N) \overset{\Gamma_N}{\vdash} h_{12}(N-1)p_{13}h_{23}(N-1) \overset{\Gamma_N}{\models} \omega$ и съществуват изводи $h_{12}(N-1) \overset{\Gamma_N}{\models} \omega_1$ и $h_{23}(N-1) \overset{\Gamma_N}{\models} \omega_2$ с дължини по-малки от s, където $\omega_1, \omega_2 \in V^*$ и $\omega = \omega_1 p_{13} \omega_2$. Съгласно индукционното предположение ω_1 описва алгоритъм за преместването на $N-1$ диска от първата на втората колона, използувайки третата за помощна, а ω_2 описва алгоритъм за преместване на $N-1$ диска от втора колона на трета, използувайки първата за помощна и съблюдавайки ограниченията от задача 6.1.1. Тогава $\omega = \omega_1 p_{13} \omega_2$ описва следния алгоритъм: Първо преместваме съобразявайки се с ограниченията от задача 6.1.1 най-горните t на брой диска от първа колона на втора, след което преместваме най-долния и

[1] Не е необходимо $L(\Gamma_N)$ да описва всички решения на задача 6.1.1. Някои от тях могат да се окажат неефективни, така например, ако включват безсмислено преместване на диск като веднага след това местим същия диск на друга колона.

най-голям диск от първа колона на празната трета и накрая преместваме t на брой диска от втора колона на трета. Следователно $\omega = \omega_1 p_{13} \omega_2$ (ако съществува) описва решение на задачата за Ханойските кули.

Да докажем сега, че за всяко цяло положително число N езикът $L(\Gamma_N)$ не е празен. При $N = 1$ единственото правило от P, което може да бъде приложено е $h_{13} \to p_{13}$ и следователно $L(\Gamma_1) = \{p_{13}\}$, т.е. $L(\Gamma_1)$ не е празен език. Допускаме, че за всяко цяло положително число $t \leq N$ езиците $L(\Gamma_t)$ не са празни и полагаме $N = t + 1$. Разглеждаме безконтекстните граматики $\Gamma'_t = \langle V, W, h_{12}(t), P\rangle$ и $\Gamma''_t = \langle V, W, h_{23}(t), P\rangle$. Очевидно Γ'_t и Γ''_t работят аналогично на Γ_t и съгласно доказаното по-горе, ако $\omega' \in L(\Gamma'_t)$, то ω' описва алгоритъм за преместване на t диска от първа колона на втора с помощна трета колона, съгласно ограниченията описани в задача 6.1.1, а ако $\omega'' \in L(\Gamma''_t)$, то ω'' описва съответния алгоритъм за преместване на t диска от втора колона на трета с помощна първа колона. Съгласно индукционното предположение ω' и ω'' съществуват. Тогава в Γ_{t+1} съществува извода $h_{13}(t+1) \overset{\Gamma_{t+1}}{\vdash} h_{12}(t)p_{13}h_{23}(t) \overset{\Gamma_{t+1}}{\models} \omega' p_{13} \omega''$, където $\omega' \in L(\Gamma'_t)$ и $\omega'' \in L(\Gamma''_t)$. Следователно $\omega \in L(\Gamma_{t+1})$, т.е. $L(\Gamma_{t+1})$ не е празен език.

□

При $N = 5$ се получава думата $p_{13}\ p_{12}\ p_{32}\ p_{13}\ p_{21}\ p_{23}\ p_{13}\ p_{12}\ p_{32}\ p_{31}\ p_{21}\ p_{32}\ p_{13}\ p_{12}\ p_{32}\ p_{13}\ p_{21}\ p_{23}\ p_{13}\ p_{21}\ p_{32}\ p_{31}\ p_{21}\ p_{23}\ p_{13}\ p_{12}\ p_{32}\ p_{13}\ p_{21}\ p_{23}\ p_{13}$ Можем да проверим верността на алгоритъма, използувайки например пет поредни карти.

Лесно се доказва (например с помощта на индукция по N) следното

Твърдение 6.1.3. *Нека N е цяло положително число, а Γ_N е дефинираната при решението на задача 6.1.2 безконтекстна граматика, то*

$$|L(\Gamma_N)| = 1$$

и ако $\omega \in L(\Gamma_N)$, то

$$|\omega| = 2^N - 1$$

□

С други думи за всяко цяло положително число N граматиката Γ_N поражда точно една дума, която описва алгоритъм за решаване на задачата за Ханойските кули с помощта на точно $2^N - 1$ преместванията на дисковете от една колона на друга.

6.1.3. Магазинни (стекови) автомати

Под *недетерминиран магазинен* (*стеков*) *автомат* (НМА) ще разбираме всяка наредена седморка

$$M = \langle K, V, W, \delta, q_0, z_0, F\rangle,$$

където:

- K е крайно множество от състояния на автомата (*азбука на състоянията*);
- V е крайно множество от входни букви (*входна азбука*);
- W е крайно, непразно множество от *магазинни* (*стекови*) символи (*магазинна азбука*);
- $\delta : K \times (V \cup \{\varepsilon\}) \times W \to \mathcal{P}(K \times W^*)$ [2] е *функция на преходите*;
- $q_0 \in K$ е *начално състояние* на автомата;

[2] Както обикновено с $\mathcal{P}(A)$ се означава множеството от всички подмножества на множеството A, включително и празното.

- $z_0 \in W$ е *начален* магазинен символ;

- $F \subseteq K$ е множество от *крайни* (*заключителни*) състояния.

Конфигурация на недетерминирания магазинен автомат M наричаме наредената тройка $\langle q, \alpha, \gamma \rangle \in K \times V^* \times W^*$.

Нека $\alpha = a_1 a_2 \dots a_s \in V^*$, $\gamma = z_1 z_2 \dots z_t \in W^*$. Тогава функцията на преходите δ определя преходите от конфигурацията $\langle q, \alpha, \gamma \rangle$ към следващите конфигурации по следния начин:

(i) за всяка двойка $\langle p, \gamma' \rangle \in \delta(q, a_1, z_1)$ конфигурацията $\langle q, \alpha, \gamma \rangle$ преминава в конфигурацията $\langle p, \alpha_1, \gamma_1 \rangle$, където $\alpha_1 = a_2 a_3 \dots a_s$, $\gamma_1 = \gamma' z_2 z_3 \dots z_t$, което ще означаваме с $\langle q, \alpha, \gamma \rangle \vdash \langle p, \alpha_1, \gamma_1 \rangle$.

(ii) за всяка двойка $\langle p, \gamma' \rangle \in \delta(q, \varepsilon, z_1)$ конфигурацията $\langle q, \alpha, \gamma \rangle$ преминава в конфигурацията $\langle p, \alpha, \gamma' z_2 z_3 \dots z_t \rangle$, което ще означаваме с $\langle q, \alpha, \gamma \rangle \vdash \langle p, \alpha, \gamma' z_2 z_3 \dots z_t \rangle$.

Ако на недетерминирания магазинен автомат в началото е подадена думата $\alpha = a_1 a_2 \dots a_s$, то по началната конфигурация $\langle q_0, \alpha, z_0 \rangle$ се получават следващите възможни конфигурации с помощта на функцията на преходите δ. За всяка от новите конфигурации, с помощта на δ, се получават всички възможни следващи конфигурации и т. н.

Недетерминираният магазинен автомат M *разпознава думата* $\alpha = a_1 a_2 \dots a_s$ *чрез заключително състояние,* ако при работата си при зададена в началото думата α, той достигне до конфигурация от вида $\langle q, \varepsilon, \gamma \rangle$, за някое $\gamma \in W^*$, при това $q \in F$.

Недетерминираният магазинен автомат M *разпознава думата* $\alpha = a_1 a_2 \dots a_s$ *чрез празен магазин,* ако при работата си при зададена в началото думата α, той достигне до конфигурация от вида $\langle q, \varepsilon, \varepsilon \rangle$.

Магазинният автомат $M = \langle K, V, W, \delta, q_0, z_0, F \rangle$ се нарича *детерминиран*, ако за всяко $q \in K$ и $z \in W$ или

(1) $\delta(q, a, z)$ съдържа не повече от един елемент за всяко $a \in V$ и $\delta(q, \varepsilon, z) = \emptyset$ или

(2) $\delta(q, a, z) = \emptyset$ за всяко $a \in V$ и $\delta(q, \varepsilon, z)$ съдържа не повече от един елемент.

Език, който се разпознава от някой ДМА се нарича *детерминиран.*

Както е известно връзката между безконтекстните езици и магазинните автомати се дава с помощта на следните твърдения:

За всеки безконтекстен език L съществува недетерминиран магазинен автомат M, такъв че L се разпознава от M чрез заключително състояние.

Езикът L се разпознава от недетерминиран магазинен автомат чрез празен магазин тогава и само тогава, когато L се разпознава от недетерминиран магазинен автомат чрез заключително състояние.

Ако L е език, който се разпознава от недетерминиран магазинен автомат, то L е безконтекстен език.

Задача 6.1.4. *За всяко цяло положително число $N \geq 2$ да се построи детерминиран магазинен автомат M_N, който при своята работа да имитира работата на монасите при решаването на задачата за Ханойските кули. (виж зад. 6.1.1).*

Решение. Търсеният автомат е следният: $M = \langle K, \emptyset, W, \delta, q_0, z_0, \emptyset \rangle$, където $K = \{q_0\}$, $W = \{p_{ij} \mid i \neq j,\ i, j = 1, 2, 3\} \cup \{h_{ij}(n) \mid i \neq j,\ i, j = 1, 2, 3,\ n = 1, 2, \dots, N-1\} \cup \{z_0\}$, а с $\emptyset$ както обикновено сме означили празното множество. Нека $i, j, k \in \{1, 2, 3\}$, $i \neq j$, $j \neq k$, $k \neq i$. Тогава функцията на преходите δ дефинираме по следния начин:

$$\delta(q_0, \varepsilon, z_0) = \langle q_0, h_{12}(N-1) p_{13} h_{23}(N-1) \rangle \tag{6.1.1}$$

$$\delta(q_0, \varepsilon, h_{ij}(1)) = \langle q_0, p_{ij} \rangle \tag{6.1.2}$$

$$\delta(q_0, \varepsilon, h_{ij}(n)) = \langle q_0, h_{ik}(n-1)p_{ij}h_{kj}(n-1) \rangle \quad n = 2, 3, \ldots, N \tag{6.1.3}$$

$$\delta(q_0, \varepsilon, p_{ij}) = \langle q_0, \varepsilon \rangle. \tag{6.1.4}$$

Веднага след включване на M_N, преди да е подаден какъвто и да е входен сигнал, автоматът заменя началния магазинен символ z_0 с думата $h_{12}(N-1)p_{13}h_{23}(N-1)$ съгласно (6.1.1) и след извършване на определен брой действия в зависимост от текущия магазинен символ съгласно (6.1.2), (6.1.3), или (6.1.4). Освен това допълнително приемаме, че автоматът M_N е конструиран така, че след прочитане на магазинен символ от вида p_{ij} $i, j \in \{1, 2, 3\}$, $i \neq j$, едновременно с действието съгласно (6.1.4) се извършва и друго действие, а именно преместване на най-горния диск от i-та колона на j-та. Функцията на преходите е дефинирана така, че след краен брой тактове автоматът изпразва своя магазин и спира. Това е така, тъй като ако текущият магазинен символ е от вида p_{ij}, то M_N го изтрива, а ако текущият магазинен символ е от вида $h_{ij}(n)$, то при следващия такт параметърът n намалява с единица, ако $n > 1$, или $h_{ij}(n)$ преминава в символа p_{ij} при $n = 1$, след което този символ бива изтрит.

Ще докажем, че при $1 \leq s < N$ автоматът M_N от конфигурация $\langle q_0, \varepsilon, h_{ij}(s)\gamma \rangle$, където $i, j \in \{1, 2, 3\}$, $i \neq j$, $\gamma \in W^*$ в следствие на своята работа достига до конфигурация $\langle q_0, \varepsilon, \gamma \rangle$ и при това премества съгласно ограниченията от задача 6.1.1 s диска от колона i на колона j. При $s = 1$ имаме $\langle q_0, \varepsilon, h_{ij}(1)\gamma \rangle \vdash \langle q_0, \varepsilon, p_{ij}\gamma \rangle \vdash \langle q_0, \varepsilon, \gamma \rangle$, т.е. твърдението е изпълнено. Допускаме, че твърдението е изпълнено за всяко t, такова че $1 \leq t < s < N$ и нека $t = s$. Тогава при $i, j, k \in \{1, 2, 3\}$, $i \neq j$, $j \neq k$, $k \neq i$ имаме $\langle q_0, \varepsilon, h_{ij}(t)\gamma \rangle \vdash \langle q_0, \varepsilon, h_{ik}(t-1)p_{ij}h_{kj}(t-1)\gamma \rangle$. Съгласно индукционното предположение M_N достига до конфигурация $\langle q_0, \varepsilon, p_{ij}h_{kj}\gamma \rangle$ премествайки $t-1$ диска от i-та колона на k-та колона, след което преминава в конфигурация $\langle q_0, \varepsilon, h_{kj}(t-1)\gamma \rangle$, премествайки следващия диск от колона i на колона j и отново съгласно индукционното предположение премества $t-1$ диска (които очевидно всичките са с по-малки размери) върху този диск на колона j, взети от колона k. Следователно твърдението е вярно за всяко $s = 1, 2, \ldots, N-1$.

Съгласно току що доказаното твърдение имаме:
$\langle q_0, \varepsilon, z_0 \rangle \vdash \langle q_0, \varepsilon, h_{12}(N-1)p_{13}h_{23}(N-1) \rangle \vdash \ldots \vdash \langle q_0, \varepsilon, p_{13}h_{23}(N-1) \rangle \vdash \langle q_0, \varepsilon, h_{23}(N-1) \rangle \vdash \ldots \vdash \langle q_0, \varepsilon, \varepsilon \rangle$, като при това автоматът премества най-горните $N-1$ диска от първа на втора колона, след което премества най-долния и най-голям от първа на трета колона и накрая премества дисковете, които са $N-1$ на брой от втора на трета колона, като при това се съблюдават ограниченията, описани в задача 6.1.1. Следователно магазинният автомат M_N решава задачата за Ханойските кули.

□

6.2. Още едно представяне на безконтекстните граматики с помощта на крайни ориентирани графи

6.2.1. Въведение в проблема и постановка на задачата

Безконтекстните граматики са широко използвани при описание на синтаксиса на езиците за програмиране и естествените езици. От друга страна, теорията на графите е добър апарат при моделирането ни изчислителни процеси и устройства. По тази причина възникват множество алгоритми, свързани с теория на графите [38, 162]. Теория на графите съществено се използва и в компютърната лингвистика. Както е добре известно, крайните автомати и магазинните автомати,

които разпознават съответно автоматните и безконтекстните езици се моделират с помощта на крайни ориентирани графи. Целта на настоящия раздел е за всяка безконтекстна граматика да построим краен ориентиран граф, моделиращ нейната работа, без да построяваме за целта магазинен автомат. В този смисъл крайният граф, която ще построим тук ще бъде различен от известните други аналогични модели.

В следващия раздел 6.3 ще опишем някои полиномиални алгоритми за проверка дали даден автоматен или линеен език се съдържа в групов език с разрешим проблем за равенство на думите [23, 180]. В основата на тези алгоритми са търсенето на цикли в диаграмите на преходите, моделиращи крайните автомати, разпознаващи съответните езици. Тази идея би могла да се използва и при разрешаването на открития все още за науката въпрос за намиране на полиномиален алгоритъм, проверяващ дали произволен безконтекстен език се съдържа в групов език с разрешим проблем за равенство на думите. Моделът, който ще опишем има за цел да ни доближи до решаването на тази задача.

В [22][3] е описан алгоритъм, работещ за време $O(n^2)$ и проверяващ дали дадена последователност от символи принадлежи на произволен безконтекстен език. Алгоритъмът се основава на едно матрично представяне на безконтекстните езици и е докладван на конгреса по размити множества и интелигентни технологии EUFIT'93 [171]. Моделът, който ще разгледаме тук в частния случай, когато граматиката е в нормална форма на Чомски, на практика е нова гледна точка на споменатия по-горе алгоритъм.

6.2.2. Крайни ориентирани графи и безконтекстни езици (нов модел)

За улеснение само в този раздел ще използваме дефинициите и означенията приети в [138]. Пораждаща граматика е наредената тройка $\Gamma = (N, \Sigma, \Pi)$, където N, Σ и Π са съответно крайни множествата от нетерминални символи, терминални символи и правила (сравнете с дефиницията на стр. 121 от раздел 6.1.2). За всяко $A \in N$ разглеждаме езика

$$L(\Gamma, A) = \{\alpha \in \Sigma^* \mid A \models \alpha\}.$$

С други думи с всяка граматика, разглеждаме множеството от езици

$$\mathfrak{L} = \{L \subseteq \Sigma^* \mid L = L(\Gamma, A),\ A \in N\}.$$

Очевидно $|\mathfrak{L}| \leq |N|$.

Под *диаграма на преходите* ще разбираме всеки краен ориентиран граф, дъгите на когото са маркирани с елемент на дадена полугрупа, който ще наричаме *етикет* на дъгата [123, стр. 48]. С други думи под диаграма на преходите ще разбираме наредената четворка $H = (V, R, S, l)$, където (V, R) е ориентиран граф (с възможно кратни ребра) с множество от върхове V и мултимножество от дъги $R \subseteq V \times V = \{(v_1, v_2) \mid v_1, v_2 \in V\}$; S е полугрупа, чийто елементи ще наричаме етикети, а l е изображение от R в S, което ще наричаме *маркиращо изображение*. Ако $\pi = p_1\ p_2\ \cdots\ p_k$ е път в H, $p_i \in R$, $i = 1, 2, \ldots k$, то по дефиниция

$$l(p_1\ p_2\ \cdots\ p_k) = l(p_1)l(p_2)\ldots l(p_k),$$

[3] Оригинала на тази статия е на руски език. По-късно публикацията е преведена на английски език от издателство Springer в [170].

т.е. ако π е път в дадена диаграма на преходите, то *етикет на пътя*, $l(\pi)$ е произведението на етикетите на дъгите, които образуват пътя. Отделните множители се вземат по ред, съответстващ на реда на етикетите на дъгите по пътя π.

Ако P е множество от пътища в диаграмата на преходите H, то по дефиниция

$$l(P) = \bigcup_{\pi \in P} l(\pi) = \{\omega \in S \mid \exists \pi \in P : l(\pi) = \omega\}$$

Нека $\Gamma = (N, \Sigma, \Pi)$ е безконтекстна граматика. За всяко $A \in N$ и всяко $x \in \Sigma$ дефинираме новите елементи, съответно A' и x', такива че $A', x' \notin N \cup \Sigma$. Полагаме:

$$N' = \{A' \mid A \in N\}$$

и

$$\Sigma' = \{x' \mid x \in \Sigma\}.$$

Разглеждаме моноида T с множество от образуващи

$$N \cup \Sigma \cup N' \cup \Sigma'$$

и пораждащи равенства:

$$A'A = \varepsilon, \qquad x'x = \varepsilon, \tag{6.2.1}$$

където $A' \in N'$, $A \in N$, $x' \in \Sigma'$ и $x \in \Sigma$, а ε е *празната дума* (единицата в моноида T) и нека

$$R = \Sigma^* \times T = \{(\omega, z) \mid \omega \in \Sigma^*, z \in T\}.$$

В R въвеждаме операцията $\circ$ по следния начин: ако $(\omega_1, z_1), (\omega_2, z_2) \in R$, то

$$(\omega_1, z_1) \circ (\omega_2, z_2) = (\omega_1\omega_2, z_2z_1). \tag{6.2.2}$$

Не е трудно да се забележи, че R с така въведената операция е моноид с единица $(\varepsilon, \varepsilon)$.

Полагаме по дефиниция

$$(x')' = x, \quad (A')' = A$$

и ако $\omega = z_1z_2 \dots z_k \in T$, то ω' ще означава думата

$$\omega' = (z_1z_2 \dots z_k)' = z_k'z_{k-1}' \dots z_1'.$$

По дефиниция

$$\varepsilon' = \varepsilon.$$

Дефиниция 6.2.1. *Нека Γ е произволна безконтекстна граматика. Построяваме диаграмата на преходите $H(\Gamma)$ с множество от върхове*

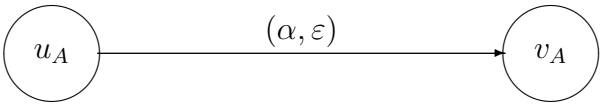
$$V = \{u_A, v_A \mid A \in N\}$$

и множество от дъги

$$E \subseteq V \times V,$$

получени по следния начин:

1. *За всяко правило $A \to \alpha$, където $A \in N$, $\alpha \in \Sigma^*$ съществува дъга с начало u_A, край v_A и с етикет (α, ε) (виж фиг. 6.1);*



Фигура 6.1. Правило $A \to \alpha$, $A \in N$, $\alpha \in \Sigma^*$ и съответната дъга от диаграмата на преходите $H(\Gamma)$.

2. *За всяко правило $A \to \alpha B z$, където $A, B \in N$, $\alpha \in \Sigma^*$, $z \in (N \cup \Sigma)^*$, съществува дъга с начало u_A, край u_B и с етикет (α, z);*

3. *За всяко правило $A \to z_1 B_1 \alpha B_2 z_2$, където $A, B_1, B_2 \in N$, $\alpha \in \Sigma^*$, $z_1, z_2 \in (N \cup \Sigma)^*$, съществува дъга с начало v_{B_1}, край u_{B_2} и с етикет $(\alpha, B_2' \alpha')$;*

4. *За всяко правило $A \to z B \alpha$, където $A, B \in N$, $\alpha \in \Sigma^*$, $z \in (N \cup \Sigma)^*$ съществува дъга с начало v_B, край v_A и с етикет (α, α');*

5. *Не съществуват други дъги в $H(\Gamma)$, освен описаните в точки 1,2,3,4.*

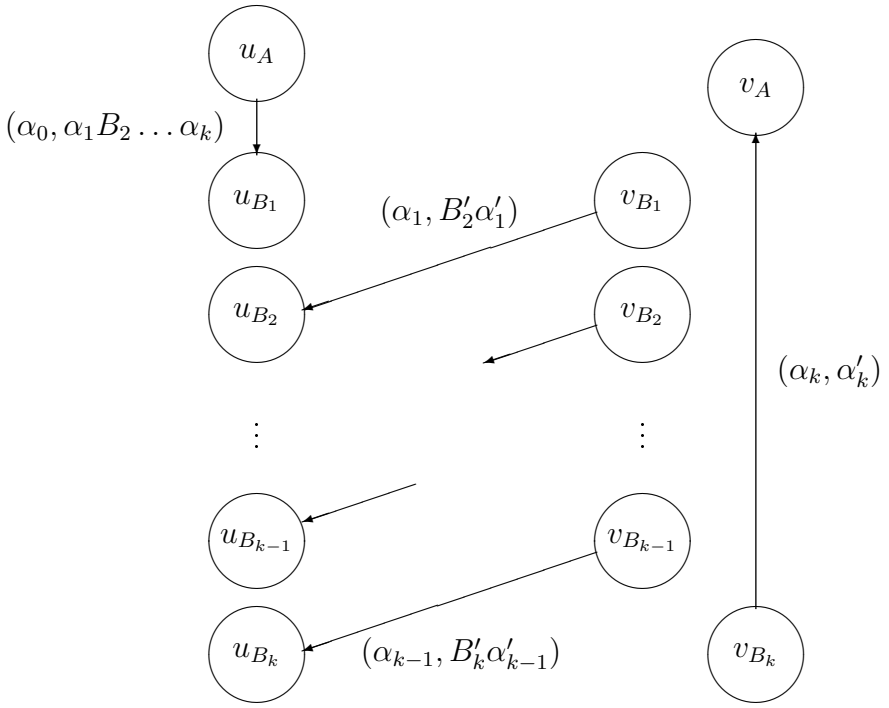
Нека в Γ съществува правилото

$$A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \dots \alpha_{k-1} B_k \alpha_k,$$

където

$$\alpha_i \in \Sigma^* \ (i = 0, 1, \dots k), \ A, B_j \in N \ (j = 1, 2, \dots, k).$$

Тогава съответната част от диаграмата на преходите $H(\Gamma)$ е изобразена на фигура 6.2.



Фигура 6.2. Правило $A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \dots \alpha_{k-1} B_k \alpha_k$ и съответната част от диаграмата на преходите $H(\Gamma)$.

Дефиниция 6.2.2. *Пътят $\pi \in H(\Gamma)$ ще наричаме **правилен**, ако:*

1. *за всяко $A \in N$ дъгите с начало u_A и край v_A (ако съществуват такива) са правилни пътища;*

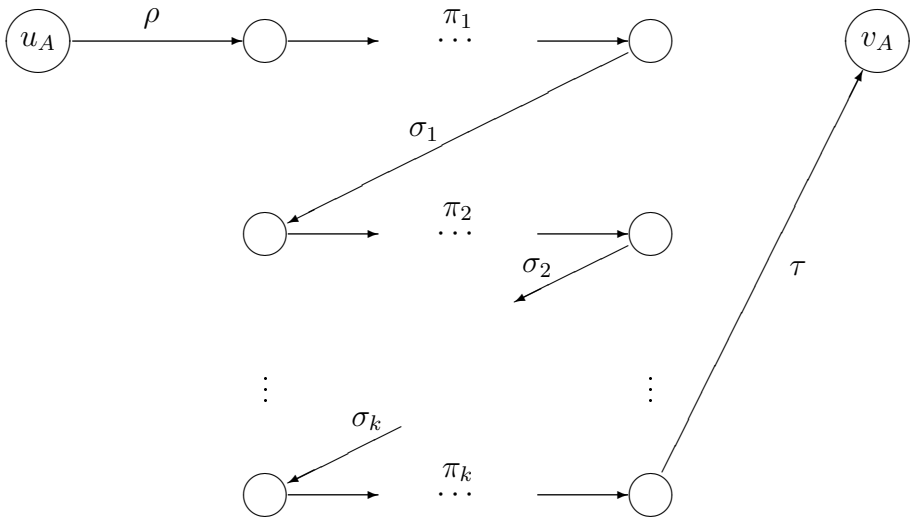
2. *ако $\pi_1, \pi_2, \ldots, \pi_k$ са правилни пътища и нека да съществуват:*

   - *дъга ρ с начало u_A, за някое $A \in N$ и край, съвпадащ с началото на π_1;*
   - *за всяко $i = 1, 2, \ldots, k-1$ съществуват дъги σ_i с начало, съвпадащо с края на π_i и край, съвпадащ с началото на π_{i+1};*
   - *дъга τ с начало, съвпадащо с края на π_k и край v_A,*

   такива, че $l(\rho\pi_1\sigma_1\pi_2\sigma_2 \ldots \sigma_{k-1}\pi_k\tau) = (\alpha, \varepsilon)$, за някое $\alpha \in \Sigma^*$.

   Тогава пътят $\pi = \rho\pi_1\sigma_1\pi_2\sigma_2 \ldots \sigma_{k-1}\pi_k\tau$ е правилен (виж фигура 6.3).

3. *Не съществуват други правилни пътища в $H(\Gamma)$, освен описаните в точки 1 и 2.*



Фигура 6.3. Правилен път в $H(\Gamma)$.

В сила е следната теорема:

Теорема 6.2.3. *Нека $\Gamma = (N, \Sigma, \Pi)$ е безконтекстна граматика и нека $A \in N$. Тогава $\alpha \in L(\Gamma, A)$ тогава и само тогава, когато съществува правилен път в $H(\Gamma)$ с начало u_A, край v_A и етикет (α, ε).*

Доказателство. Необходимост. Нека $\alpha \in L(\Gamma, A)$. Тогава съществува извод $A \models \alpha$. Доказателството ще извършим чрез индукция по дължината на извода.

Нека да съществува правило $A \to \alpha$. Тогава, съгласно дефиниция 6.2.1 точка 1, съществува дъга $u_A v_A$ с етикет (α, ε).

Допускаме, че за всяко $A \in N$ и за всяко $\alpha \in L(\Gamma, A)$ за които съществува извод $A \models \alpha$ с дължина не по-голяма от t, съществува правилен път с начало u_A, край v_A и етикет (α, ε).

Нека $\alpha \in L(\Gamma, A)$ и съществува извод $A \models \alpha$ с дължина $t+1$. Нека $A \to \alpha_0 B_1 \alpha_1 \ldots B_k \alpha_k$ е първото правило на този извод, където $\alpha_i \in \Sigma^*$, $B_j \in N$, $i = 0, 1, \ldots k$, $j = 1, 2, \ldots k$. Тогава съществува извод $A \to \alpha_0 B_1 \alpha_1 \ldots B_k \alpha_k \models \alpha_0 \beta_1 \alpha_1 \ldots \beta_k \alpha_k =$

α, където $\beta_i \in L(\Gamma, B_i)$ и за всяко $i = 1, 2, \ldots k$ съществуват изводи $B_i \models \beta_i$ с дължина по-малка или равна на t.

Съгласно индукционното предположение за всяко $i = 1, 2, \ldots, k$ съществува правилен път π_i с начало u_{B_i}, край v_{B_i} и етикет (β_i, ε).

За първото правило $A \to \alpha_0 B_1 \alpha_1 \ldots B_k \alpha_k$ при извода на α имаме:

а) Съгласно дефиниция 6.2.1, точка 2 съществува дъга ρ с начало u_A, край u_{B_1} и с етикет $(\alpha_0, \alpha_1 B_2 \ldots B_k \alpha_k)$.

б) Съгласно дефиниция 6.2.1, точка 3 за всяко $i = 1, 2, \ldots k-1$ съществува дъга σ_i с начало v_{B_i}, край $u_{B_{i+1}}$ и с етикет $(\alpha_i, B'_{i+1}\alpha'_i)$.

в) Съгласно дефиниция 6.2.1, точка 4 съществува дъга τ с начало v_{B_k}, край v_A и с етикет (α_k, α'_k).

За пътя $\pi = \rho\pi_1\sigma_1\pi_2 \ldots \sigma_{k-1}\pi_k\tau \in H(\Gamma)$ е изпълнено:

$$l(\pi) =$$

$$= (\alpha_0, \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k) \circ (\beta_1, \varepsilon) \circ (\alpha_1, B'_2\alpha'_1) \circ (\beta_2, \varepsilon) \circ \cdots \circ (\alpha_{k-1}, B'_k\alpha'_{k-1}) \circ (\alpha_k, \alpha'_k) =$$

$$= (\alpha_0\beta_1\alpha_1 \ldots \beta_k\alpha_k \ , \ \alpha'_k B'_k \alpha'_{k-1} \ldots B'_2\alpha'_1\alpha_1 B_2 \ldots \alpha_{k-1} B_k \alpha_k) =$$

$$= (\alpha, \varepsilon)$$

Тъй като $\pi_1, \pi_2 \ldots \pi_k$ са правилни пътища, то съгласно дефиниция 6.2.2 следва, че π е правилен път в $H(\Gamma)$ с начало u_A, край v_A и етикет (α, ε). Необходимостта е доказана.

Достатъчност. Нека π да е правилен път в $H(\Gamma)$ с начало u_A, край v_A и етикет (α, ε). Ако дължината на π е равна на 1, то π е дъга, а това е възможно тогава и само тогава, когато съществува правило $A \to \alpha$, т.е. $\alpha \in L(\Gamma, A)$.

Допускаме, че за всяко $A \in N$ и за всички правилни пътища с начало u_A, край v_A, етикет (α, ε) и дължина по-малка или равна на t е изпълнено $\alpha \in L(\Gamma, A)$. Нека π да е правилен път с начало u_A, край v_A, етикет (α, ε) и дължина $t+1 > 1$. Тъй като π е правилен път, то съществува $B_1 \in N$ така, че първата дъга ρ в пътя π е с начало u_A и край u_{B_1}. Следователно в Γ съществува правило $A \to \alpha_0 B_1 \alpha_1 B_2 \ldots B_k \alpha_k$ така, че $l(u_A u_{B_1}) = (\alpha_0, \alpha_1 B_2 \ldots B_k \alpha_k)$ за някои $B_i \in N$, $\alpha_j \in \Sigma^*$, $i = 2, 3, \ldots, k$, $j = 0, 1, \ldots, k$.

Съгласно дефиниция 6.2.2, π има вида $\pi = \rho\pi_1\sigma_1\pi_2 \ldots \sigma_{s-1}\pi_s\tau$, където π_i са правилни пътища с начало u_{C_i} и край v_{C_i} за някои $C_i \in N$ (очевидно $C_1 = B_1$), $i = 1, 2, \ldots, s$, дъги $\sigma_i = v_{C_i} u_{C_{i+1}}$ с начало v_{C_i} и край $u_{C_{i+1}}$, $i = 1, 2, \ldots, s-1$, а дъгата $\tau = v_{C_s} v_A$ има начало v_{C_s} и край v_A. Нека $l(\sigma_i) = (\gamma_i, C'_{i+1}\gamma'_i)$, $\gamma_i \in \Sigma^*$, $i = 1, 2, \ldots, s-1$, $l(v_{C_s} v_A) = (\gamma_s, \gamma'_s)$, $\gamma \in \Sigma^*$. Тогава имаме:

$$l(\pi) = l(u_A u_{C_1}) \circ l(\pi_1) \circ l(v_{C_1} u_{C_2}) \circ l(\pi_2) \circ \ldots \circ l(v_{C_{s-1}} u_{C_s}) \circ l(\pi_s) \circ l(v_{C_s} v_A) =$$

$$= (\omega, \gamma'_s C'_s \gamma_{s-1} \ldots C'_2 \gamma'_1 \alpha_1 B_2 \ldots B_k \alpha'_k),$$

където $\omega = \alpha$ и $\gamma'_s C'_s \gamma'_{s-1} \ldots C'_2 \gamma'_1 \alpha_1 B_2 \ldots B_k \alpha_k = \varepsilon$. Имайки предвид (6.2.1) лесно се вижда, че това е възможно тогава и само тогава, когато $s = k$, $C_i = B_i$ при $i = 2, 3, \ldots, k$ и $\gamma_j = \alpha_j$ при $j = 1, 2, \ldots, k$. Следователно $\pi = \rho\pi_1\sigma_1 \ldots \sigma_{k-1}\pi_k\tau$, където π_i са правилни пътища с начало съответно u_{B_i} и край v_{B_i} $(i = 1, 2, \ldots, k)$; σ_i са дъги с начало v_{B_i} и край $u_{B_{i+1}}$ $(i = 1, 2, \ldots, k-1)$; ρ е дъга с начало u_A и край u_{B_1}; τ е дъга с начало v_{B_k} и край v_A и съществуват $\beta_i \in \Sigma^*$ $(i = 1, 2, \ldots, k)$, такива, че $l(\pi_i) = (\beta_i, \varepsilon)$. От индукционното предположение имаме $\beta_i \in L(\Gamma, B_i)$, т.е. съществуват изводи $B_i \models \beta_i$ за всяко $i = 1, 2, \ldots, k$. Но тогава ще имаме извода:

$$A \to \alpha_0 B_1 \alpha_1 B_2 \ldots B_k \alpha_k \models \alpha_0\beta_1\alpha_1\beta_2 \ldots \beta_k\alpha_k = \alpha,$$

откъдето следва, че $\alpha \in L(\Gamma, A)$. Теоремата е доказана.

□

6.2.3. Графово представяне на граматики в нормална форма на Чомски

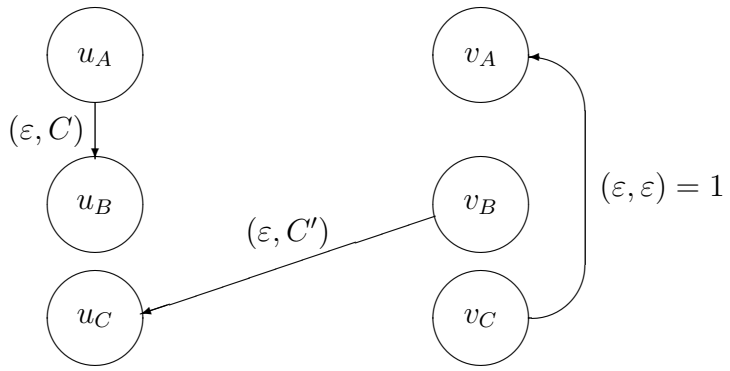
Дефиниция 6.2.4. *Безконтекстната граматика* Γ *се нарича* **граматика в нормална форма на Чомски**, *ако всички правила са един от следните два вида:*

1. $A \to BC$, *където* A, B *и* C *са нетерминални символи;*
2. $A \to x$, *където* A *е нетерминален, а* x *терминален символ.*

Както е добре известно[19, 123, 155], за всяка безконтекстна граматика съществува еквивалентна на нея граматика в нормална форма на Чомски, пораждаща същия език.

Описаният в предния раздел модел е особено полезен, когато се прилага към безконтекстни граматики от специален вид и в частност към безконтекстни граматики в нормална форма на Чомски. Така например в [170, 171] описваме алгоритъм, работещ за време $O(n^2)$ и разпознаващ дали дадена дума принадлежи на езикq породен от дадена граматика в нормална форма на Чомски. В тази връзка са и разглежданията в този раздел, като се наблегне преди всичко на факта, че в този случай се получава значително опростено представяне на граматиката с помощта на крайни ориентирани графи.

Ако дадено правило в граматиката Γ има вида $A \to BC$, т.е. е от вида 1 на дефиниция 6.2.4, то съответният фрагмент от диаграмата на преходите $H(\Gamma)$ ще има вида, показан на фигура 6.4.



Фигура 6.4. Правило $A \to BC$ и съответната част от диаграмата на преходите $H(\Gamma)$.

6.3. Включване на регулярни и линейни езици в групови езици

6.3.1. Постановка на задачата

Съдържанието на този раздел е естествено продължение, съществено подобрение и обобщение на някои резултати от първата докторска дисертация на автора защитена успешно през 1992 година пред специализиран научен съвет в град Киев, Украйна (тогава СССР) под научното ръководство на проф. (в момента академик) Анатолий Василиевич Анисимов.

В този раздел ще се спрем по-подробно на някои алгебрични характеристики на формалните езици и граматики [138, 149]. Връзката между теорията на формалните езици и граматики и теория на групите подробно е развита в [98]. Списък

от нерешени до момента на издаване на публикацията проблеми, които са в тясна връзка с резултатите от настоящия раздел са дадени в [16] (превод на английски език [116]).

При обобщените описания на алгоритмите съществено ще използваме теоретико-множествени операции. В този смисъл описаните в глава 3, по-конкретно в раздел 3.3 алгоритми с използване на побитови операции биха били от полза при конкретната реализация.

Дефиниция 6.3.1. *Нека*

$$\Sigma = X \cup X^{-1} = \left\{x_1, x_2, ..., x_m, x_1^{-1}, x_2^{-1}, ..., x_m^{-1}\right\} \tag{6.3.1}$$

е крайна азбука и нека със Σ^* *да сме означили свободния моноид над* Σ. *Нека* G *да е група с множество от образуващи* Σ, *множество от пораждащи съотношения* Θ, *единичен елемент* e *и с разрешим проблем на равенство на думите. Тогава множеството от думи*

$$\mathfrak{L}(G) = \{\omega \in \Sigma^* \mid \omega \equiv e \ (mod\, G)\} \subseteq \Sigma^* \tag{6.3.2}$$

ще наричаме ***групов език, задаващ групата*** G. *Групата* G *се задава от безконтекстен език, ако съответният групов език* $\mathfrak{L}(G)$ *е безконтекстен. Групата* G *в този случай се нарича* ***безконтекстна група***.

Понятието групов език е въведено от А. В. Анисимов в [5] (превод на английски език [89]). В тази статия А. В. Анисимов доказва, че $\mathfrak{L}(G)$ е автоматен език тогава и само тогава, когато групата G е крайна (виж също [98, Теорема 5.17]).

Малко по-различна дефиниция на понятието групов език е дадено в [120], а именно автоматен език, чийто синтактичен моноид е крайна група. В дефиниция 6.3.1 се допускат безконтекстни езици, които не са автоматни. Ние ще се придържаме към дефиницията формулирана от А. В. Анисимов и цитирана в редица авторитетни източници [98].

В [6, 90] А. В. Анисимов доказва, че проблемът за определяне на еднозначността на крайноавтоматните изображения е частен случай на проблема за проверка дали даден безконтекстен език е подмножество на групов език. Така естествено възниква въпросът за намиране на полиномиален алгоритъм, проверяващ включването на даден безконтекстен език в групов език. В този раздел ние ще решим този проблем за най-разпространените частни случаи на безконтекстните езици, а именно автоматните и линейни езици. Автоматните езици ще задаваме с помощта на краен автомат, а линейните езици – с помощта на линейни граматики. И в двата случая ще намерим крайни множества, такива че всяко от тях се съдържа в груповия език $\mathfrak{L}(G)$ тогава и само тогава, когато съответният безконтекстен език се съдържа в $\mathfrak{L}(G)$. В резултат ще получим полиномиални алгоритми, проверяващи включването на произволен автоматен или съответно линеен език в групов език с разрешим проблем за равенство на думите.

До края на главата, G ще означава крайно породена група с разрешим проблем за равенство на думите, а $\Gamma = \langle \Sigma, N, A_1, \Pi \rangle$ ще означава безконтекстна граматика, пораждаща езика L, т.е $L = L(\Gamma)$, където N е множеството от нетерминални символи,

$$\Sigma = \{x_1, x_2, ..., x_m, x_1^{-1}, x_2^{-1}, ..., x_m^{-1}\}$$

е терминалната азбука, Π е множеството от правила, а $A_1 \in N$ е началният символ. Означаваме с r константата от известната "uvwxy" теорема (*pumping lemma*) за

дадения безконтекстен език L [123, Теорема 7.18] (виж също [18, Теорема 4.19], [19, Теорема 3.4.3] или [114, Лема 3.1.1]).

Разглеждаме следните множества:

$\Omega_1 = \{\omega \in L \mid |\omega| \leq r\}$;

$\Omega_2 = \left\{uwvw^{-1} \;\middle|\; |uwv| \leq r,\ uv \neq \varepsilon,\ \exists A \in N : A \overset{*}{\Rightarrow} uAv, A \overset{*}{\Rightarrow} w\right\}$;

$W_1 = \Omega_1 \cup \Omega_2$.

Следващата теорема е доказана в [6], като тази статия е преведена на английски език в [90].

Теорема 6.3.2. (A. V. Anisimov [6, 90]) *Съобразявайки се с въведените по-горе означения, $L \subseteq \mathfrak{L}(G)$ тогава и само тогава, когато $W_1 = \Omega_1 \cup \Omega_2 \subseteq \mathfrak{L}(G)$.* □

Теорема 6.3.2 дава алгоритъм, проверяващ дали включването $L \subseteq \mathfrak{L}(G)$ е истина. За съжаление този алгоритъм не е полиномиален. В следващите два раздела на тази глава, ние ще покажем, че ако L е автоматен или линеен език, то алгоритъмът на Анисимов може да се преработи така, че да стане полиномиален.

6.3.2. Включване на автоматни езици в групови езици

В този раздел L ще означава произволен автоматен език. Следователно съществува (детерминиран или недетерминиран) краен автомат A, където

$$A = (Q, \Sigma, \delta, q_1, Z) \tag{6.3.3}$$

такъв, че

$$L = L(A) = \{\omega \in \Sigma^* \mid \delta(q_1, \omega) \cap Z \neq \emptyset\},$$

където:

$Q = \{q_1, q_2, \ldots, q_n\}$ е множество от състоянията;

$\Sigma = \left\{x_1, x_2, ..., x_m, x_1^{-1}, x_2^{-1}, ..., x_m^{-1}\right\}$ е входната азбука;

δ е функция на преходите;

$q_1 \in Q$ е начално състояние на автомата;

$Z \subseteq Q$ е множеството от заключителни състояния.

Нека $H_A = (Q, R, \Sigma^*, l_A)$ да е диаграмата на преходите на автомата (6.3.3) (дефиниция на понятието диаграма на преходите е дадена на стр. 126). Нека G да е група с разрешим проблем за равенство на думите, множество от образуващи $\Sigma = \{x_1, x_2, ..., x_m, x_1^{-1}, x_2^{-1}, ..., x_m^{-1}\}$ и единичен елемент e. Нека

$$H_G = (Q, R, G, l_G)$$

да е диаграма на преходите с множества от върхове и дъги съвпадащи със съответните множества в диаграмата на преходите H_A, но етикетите на дъгите разглеждаме като елементи на групата G.

Разглеждаме полупръстена

$$F_G = (\mathcal{P}(G), \cup, \cdot, \phi, \{e\}),$$

където $\mathcal{P}(G)$ е множествата от всички подмножества на G, включително и празното. Операциите в F_G са съответно операциите обединение и произведение на подмножества на групата G, т.е. ако $A, B \subseteq G$, то по дефиниция $A \cdot B = \{ab \mid a \in A, b \in B\}$. Единичен елемент в F_G е множеството $\{e\}$ съдържащо единствено единицата e на G, а нулев елемент в F_G е празното множество ϕ.

Разглеждаме множествата $X, Y \in F_G$. В полупръстена F_G дефинираме следната бинарна операция:

$$X \star Y = \left\{ xyx^{-1} \mid x \in X, y \in Y \right\} \tag{6.3.4}$$

Разглеждаме следните множества от пътища в H_G:

P_{ij} – множеството от всички пътища $\pi \in H_G$ с начало върхът $q_i \in Q$ и край върхът $q_j \in Q$, $1 \leq i, j \leq n$;

$\widehat{P_{ij}}$ – множеството от всички пътища $\pi \in H_G$ с начало върхът $q_i \in Q$, край върхът $q_j \in Q$, $1 \leq i, j \leq n$ и в които всички върхове са различни с едно възможно изключение, ако $q_i = q_j$. $\widehat{P_{ij}} \subseteq P_{ij}$;

P_{iZ} – множеството от всички пътища $\pi \in H_G$ с начало върхът $q_i \in Q$ и край – елемент на множеството от заключителни състояния Z, $1 \leq i \leq n$;

$\widehat{P_{iZ}}$ – множеството от всички пътища $\pi \in H_G$ с начало върхът $q_i \in Q$, край – елемент на множеството от заключителни състояния Z, $1 \leq i \leq n$ и в които всички върхове са различни с едно възможно изключение, ако началото и краят съвпадат. $\widehat{P_{iZ}} \subseteq P_{iZ}$;

O_i – множеството от всички цикли $\pi \in H_G$, минаващи през върха $q_i \in Q$, $1 \leq i \leq n$ и в които всички върхове са различни. $O_i = \widehat{P_{ii}}$.

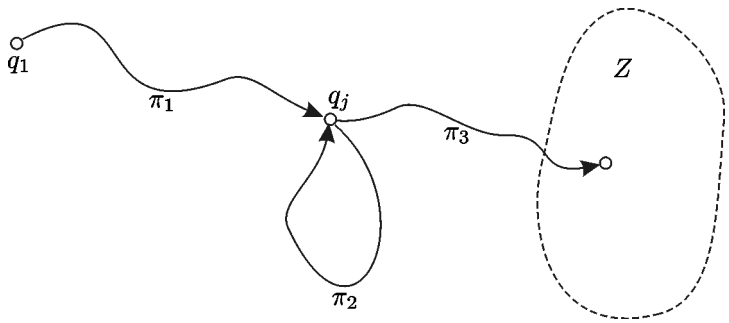
Очевидно $L \subseteq \mathfrak{L}(G)$ тогава и само тогава, когато $l_G(P_{1Z}) = \{e\}$.

Разглеждаме и множествата:

$\Omega_3 = \left\{ l_G(\pi) \;\middle|\; \pi \in \widehat{P_{1Z}} \right\}$;

$\Omega_4 = \left\{ uvu^{-1} \right\}$, където u и v са такива, че съществува връх $q_j \in Q$ и пътища $\pi_1 \in \widehat{P_{1j}}$, $\pi_2 \in O_j$, $\pi_3 \in P_{jZ}$, така както е показано на фигура 6.5 и за които е изпълнено $u = l_G(\pi_1)$, $v = l_G(\pi_2)$;

$W_2 = \Omega_3 \cup \Omega_4 \in F_G$.



Фигура 6.5

Дефинираме следните множества от пътища $\mathcal{K}_{ij}^k$ в H_G, където $i, j \in \{1, 2, ..., n\}$, $k \in \{0, 1, ..., n\}$, $n = |Q|$ по следния начин:

$$\mathcal{K}_{ij}^0 = \{ \rho \mid \rho = (q_i, q_j) \in R \} \tag{6.3.5}$$

$$\mathcal{K}_{ij}^{k} = \mathcal{K}_{ij}^{k-1} \cup \mathcal{K}_{ik}^{k-1}\mathcal{K}_{kj}^{k-1} \tag{6.3.6}$$

Лесно е да се забележи, че за всяко $k \in \{0, 1, \ldots, n\}$ никой от пътищата от $\mathcal{K}_{ij}^{k}$ не минава през върха q_s ако $s > k$ и q_s не е начало или край на пътя. По-точно от дефиницията следва, че $\mathcal{K}_{ij}^{k}$ се състои от всички пътища (и само от тях) с начало върхът $q_i \in Q$, край върхът $q_j \in Q$, и в пътя няма междинен връх q_s при $s \geq k$ и от всички пътища, които могат да се представят като произведение на два пътя π_1 и π_2, такива че π_1 е с начало q_i и край q_k, а π_2 е с начало q_k и край q_j, като никой от тези пътища π_1 и π_2 не съдържат междинен връх q_s, където $s \geq k$.

Разглеждаме следните елементи на полупръстена F_G:

$\Omega_5 = \{l_G(\pi) \mid \pi \in \mathcal{K}_{1t}, q_t \in Z\}$;

$\Omega_6 = \left\{l_G(\pi_1) \star l_G(\pi_2) \mid q_j \in Q, q_t \in Z : \mathcal{K}_{jt}^{n} \neq \phi, \pi_1 \in \mathcal{K}_{1j}^{n}, \pi_2 \in \mathcal{K}_{jj}^{n}\right\}$, където $''\star''$ е дефинираната с помощта на формулата (6.3.4) операция;

$W_3 = \Omega_5 \cup \Omega_6 \in F_G$.

Не е трудно да се види, че

$$\Omega_3 \subseteq \Omega_5 \quad \text{и} \quad \Omega_4 \subseteq \Omega_6. \tag{6.3.7}$$

Тъй като в $\mathcal{K}_{ij}^{k}$ е възможно да има пътища с повтарящи се вътрешни върхове, то в общия случай $\Omega_3 \neq \Omega_5 \quad \text{and} \quad \Omega_4 \neq \Omega_6$.

Теорема 6.3.3. *Нека L да е автоматен език и нека $L = L(A)$, където A е дефинираният с (6.3.3) краен автомат. Тогава, съгласно въведените по-горе означения, следващите условия са еквивалентни:*

(i) $L \subseteq \mathfrak{L}(G)$;
(ii) $W_1 = \Omega_1 \cup \Omega_2 = \{e\}$;
(iii) $W_2 = \Omega_3 \cup \Omega_4 = \{e\}$;
(iv) $W_3 = \Omega_5 \cup \Omega_6 = \{e\}$.

Доказателство. Тъй като автоматните езици са частен случай на безконтекстните езици, то еквивалентността на условия (i) и (ii) е даказано от А. В. Анисимов в [6, 90] (Теорема 6.3.2). Тъй като съгласно (6.3.7) $W_2 \subseteq W_3$, то от $W_3 = \{e\}$ следва $W_2 = \{e\}$. И така ние доказваме, че от (iv) следва (iii). За да докажем теоремата остава да докажем, че от (iii) следва (i) и от (i) следва (iv).

От (iii) следва (i): Нека $W_2 = \Omega_3 \cup \Omega_4 = \{e\}$ и нека $\omega \in L$. Тогава съществува път $\pi \in P_{1Z}$, такъв че $l_A(\pi) = \omega$.

Ако в π няма повтарящи се вътрешни върхове, то $l_G(\pi) \in l_G(\widehat{P_{1Z}}) = \Omega_3 = \{e\}$ и следователно $\omega \in \mathfrak{L}(G)$.

Нека π съдържа междинни повтарящи се върхове. С други думи съществува, $q_j \in Q$ така че π може да се представи във вида $\pi = \pi_1\pi_2\pi_3$, където $\pi_1 \in \widehat{P_{1j}}, \pi_2 \in O_j, \pi_3 \in P_{jZ}$ (виж фиг. 6.5) и $l_G(\pi_1)l_G(\pi_2)(l_G(\pi_1))^{-1} \in \Omega_4 = \{e\}$. Следователно, $l_G(\pi_1)l_G(\pi_2) = l_G(\pi_1)$, откъдето $l_G(\pi_1)l_G(\pi_2)l_G(\pi_3) = l_G(\pi_1)l_G(\pi_3)$. Следователно $l_G(\pi_1\pi_2\pi_3) = l_G(\pi_1\pi_3)$. Тъй като $\pi_2 \in O_j$, то дължината на π_2 е по-голяма или равна на 1. Следователно в H_G съществува път $\pi' = \pi_1\pi_3$ с по-малка дължина от дължината на π, завършващ в заключително състояние и за който ако $l(\pi') = \omega'$, то разглеждайки думите ω и ω' като елементи на групата G ще е изпълнено $\omega = \omega'$. Този процес на редукция може да продължи краен брой пъти, тъй като дължината на думата ω е крайна. На края на този процес ще получим път в H_G без повтарящи

се междинни върхове и с етикет равен на ω в групата G. Но $l_G(\widehat{P_{1Z}}) = \Omega_3 = \{e\}$. Следователно $\omega = e$ в групата G и следователно $L \subseteq \mathfrak{L}(G)$.

От (i) Следва (iv): Нека $L \subseteq \mathfrak{L}(G)$, т.е. $l_G(P_{1Z}) = \{e\}$. От $\Omega_5 \subseteq l_G(P_{1Z})$ следва, че $\Omega_5 = \{e\}$.

Нека $z \in \Omega_6$. Тогава z може да се представи във вида $z = uvu^{-1}$, където $u \in l_G(\mathcal{K}^n_{1j})$, $v \in l_G(\mathcal{K}^n_{jj})$ за някое цяло число j така, че да съществува път $\pi_3 \in P_{jZ}$ и нека $l_G(\pi_3) = w$. Очевидно съществуват път $\pi_1 \in P_{1j}$ и път $\pi_2 \in O_j$ такива, че $u = l_g(\pi_1)$ and $v = l_g(\pi_2)$. Полагаме $\pi' = \pi_1\pi_2\pi_3 \in P_{1Z}$ и $\pi'' = \pi_1\pi_3 \in P_{1Z}$. Тъй като $L \subseteq \mathfrak{L}(G)$, то $l_G(\pi') = l_G(\pi'') = e$, т.е. $uvw = uw$, откъдето $uvu^{-1} = e$. Следователно $z = e$ и тъй като z е произволна дума от Ω_6, то $\Omega_6 = \{e\}$. Теоремата е доказана. □

Следващия алгоритъм се основава на еквивалентните условия (i) и (iv) от теорема 6.3.3. За улеснение на описанието под $i \in Z$ ще разбираме $q_i \in Z$ и под g^k_{ij} ще разбираме множеството $l_G(\mathcal{K}^k_{ij})$. В този случай k в g^k_{ij} ще означава индекс, а не степенен показател.

Алгоритъм 6.3.4. *Проверява включването $L \subseteq \mathfrak{L}(G)$ при зададени автоматен език L и групов език $\mathfrak{L}(G)$, където G е група с разрашим проблем за равенство на думите.*

Вход: $g^0_{ij} = l_G(\mathcal{K}^0_{ij}), \quad i, j = 1, 2, ..., n$

Изход: Булева променлива T, която получава стойност **True** ако $L \subseteq \mathfrak{L}(G)$ и стойност **False** в противен случай. Алгоритъма спира веднага след получаване на стойност $T :=$ **False**.

Begin
1. $T :=$ **True**;
2. For $1 \leq k \leq n$ Do
3. For $1 \leq i, j \leq n$ Do
4. $g^k_{ij} := g^{k-1}_{ij} \cup g^{k-1}_{ik} g^{k-1}_{kj}$;
5. End Do;
6. End Do;
7. For $j \in Z$ Do
8. If $g^n_{1j} \neq \phi$ and $g^n_{1j} \neq \{e\}$ Then
9. Begin $T :=$ **False**; Halt; End;
10. End Do;
11. For $1 \leq j \leq n$ Do
12. For $t \in Z$ Do
13. If $g^n_{jt} \neq \phi$ and $g^n_{1j} \neq \phi$ and $g^n_{jj} \neq \phi$ Then
14. If $g^n_{1j} \star g^n_{jj} \neq \{e\}$ Then
15. Begin $T :=$ **False**; Halt; End;
16. End Do;
17. End Do;
End.

Теорема 6.3.5. *Алгоритъм 6.3.4 коректно проверява включването $L \subseteq \mathfrak{L}(G)$, където L е автоматен език, разпознаван от краен автомат с n състояния, $\mathfrak{L}(G)$ е групов език, който се определя от групата G с разрешим проблем за равенство на думите.*

Алгоритъм 6.3.4 изпълнява не повече от $O(n^3)$ пъти теоретико-множествените операции "$\cup$" и "$\cdot$" и не повече от $O(n^2)$ пъти операцията "$\star$", дефинирана с помощта на равенството (6.3.4) в полупръстена F_G.

Доказателство. Съгласно теорема 6.3.3 и съобразявайки се с аксиомите в полупръстена F_G (чийто операции по същество са теоретико-множествени операции), то лесно се вижда, че в редове 9 и 15 на алгоритъм 6.3.4 булевата променлива T приема стойност **False** тогава и само тогава, когато L не се съдържа в $\mathfrak{L}(G)$. В противен случай T получава стойност **True**. Следователно алгоритъмът коректно проверява дали се изпълнява включването $L \subseteq \mathfrak{L}(G)$.

Лесно е да се види, че ред 4 в алгоритъм 6.3.4 се изпълнява не повече от n^3 пъти. Операциите "$\cup$" и "$\cdot$" в полупръстена F_G се изпълняват по веднъж всяка една от тях по време на всяка итерация. Линии 13 и 14 се изпълняват не повече от n^2 пъти всяка от тях. Следователно алгоритъм 6.3.4 изпълнява не повече от $O(n^3)$ теоретико-множествените операции "$\cup$" и "$\cdot$" и не повече от $O(n^2)$ пъти операцията "$\star$" дефинирана с (6.3.4) в полупръстена F_G. Теоремата е доказана.

□

Следствие 6.3.6. *Ако операциите "$\cup$", "$\cdot$" и "$\star$" в полупръстена F_G могат да бъдат реализирани за полиномиално време, то алгоритъм 6.3.4 е полиномиален.*

□

6.3.3. Включване на линейни езици в групови езици

Безконтекстната граматика $\Gamma = \langle \Sigma, N, A_1, \Pi \rangle$ се нарича *линейна*, ако всички правила в Π имат вида $A_i \to \alpha A_j \beta$ или $A_i \to \alpha$, където $A_i, A_j \in N$ са нетерминални символи, $1 \leq i, j \leq n$, $\alpha, \beta \in \Sigma^*$ са думи съставени само от терминални символи, или евентуално се допуска някоя от тях да е празната. Езикът L се нарича *линеен*, ако съществува линейна граматика Γ, такава че $L = L(\Gamma, A_1)$.

Нека S да е произволен моноид с единица 1. Разглеждаме множеството

$$U_S = S \times S = \{(x, y) | x, y \in S\}.$$

За всеки (x, y), $(z, t) \in U_S$ дефинираме операцията "$\diamond$" както следва

$$(x, y) \diamond (z, t) = (xz, ty). \tag{6.3.8}$$

Лесно се вижда, че тази операция $\diamond$ е асоциативна и U_S и с тази операция е моноид с единица (1,1). Ако S е група, то и U_S е група, при това ако $a = (x, y) \in U_S$, то обратният елемент на a ще бъде $a^{-1} = (x^{-1}, y^{-1})$.

Дефинираме изображенията f_l, f_r и f_d от U_S в S както следва:

$$f_l(x, y) = x \tag{6.3.9}$$

$$f_r(x, y) = y \tag{6.3.10}$$

$$f_d(x, y) = xy \tag{6.3.11}$$

Очевидно

$$f_d(x, y) = f_l(x, y) f_r(x, y).$$

В този раздел ще разглеждаме линейната граматика

$$\Gamma = \langle \Sigma, N, A_1, \Pi \rangle, \tag{6.3.12}$$

където:

$\Sigma = \left\{x_1, x_2, ..., x_m, x_1^{-1}, x_2^{-1}, ..., x_m^{-1}\right\}$ е множеството от терминални символи (входна азбука);

$N = \{A_1, A_2, ..., A_n\}$ е множеството от терминални символи;
$A_1 \in N$ е началният символ;
П е множеството от правила.
Разглеждаме следната диаграма на преходите:

$$H_\Gamma = (V, R, U_{\Sigma^*}, l_\Gamma) \tag{6.3.13}$$

с множество от върхове $V = N \cup \{A_{n+1}\}$, където $A_{n+1} \notin N$, а U_{Σ^*} е дефинираният по-горе моноид с множество от елементи $\{(\alpha, \beta) | \alpha, \beta \in \Sigma^*\}$ и операция "$\diamond$". Множеството от дъги R в H_Γ образуваме по следния начин:

а) Ако в П съществува правило $A_i \to \alpha A_j \beta$, където $A_i, A_j \in N$, то в R съществува дъга с начало A_i, край A_j и етикет (α, β);

б) Ако в П съществува правило $A_i \to \alpha$, където $A_i \in N$, $\alpha \in \Sigma^*$, то в R съществува дъга с начало A_i, край A_{n+1} и етикет (α, ε). В случая с ε сме означили празната дума;

в) Не съществуват други дъги в R, освен описаните в подточки а) и б).

Нека G да е група с множество от образуващи

$$\Sigma = \{x_1, x_2, ..., x_m, x_1^{-1}, ..., x_m^{-1}\},$$

с множество от пораждащи съотношения Θ, единица e и с разрешим проблем за равенство на думите. Нека U_G да е групата, получена както е показано на страница 137. Разглеждаме диаграмата на преходите

$$H_U = (V, R, U_G, l_U), \tag{6.3.14}$$

където множеството от върхове V и множеството от дъги R съвпада със съответните множества в диаграмата на преходите H_Γ съгласно (6.3.13), а етикетите разглеждаме като елементи на групата U_G.

Разглеждаме следните множество от пътища в H_U:

D_{ij} – множеството от всички пътища $\pi \in H_U$ с начало върхът $A_i \in V$ и край върхът $A_j \in V$, $1 \le i \le n$, $1 \le j \le n+1$;

$\widehat{D_{ij}}$ – множеството от всички пътища $\pi \in H_U$ с начало върхът $A_i \in V$, край върхът $A_j \in V$, $1 \le i \le n$, $1 \le j \le n+1$ и без повтарящи се вътрешни върхове (допуска се началото и краят да съвпадат, т.е. допуска се да е изпълнено $A_i = A_j$). Очевидно $\widehat{D_{ij}} \subseteq D_{ij}$;

C_i – множеството от всички пътища $\pi \in H_U$ с край, съвпадащ с началото $A_i \in V$, $1 \le i \le n$, т.е. цикли, съдържащи върха A_i и без повтарящи се върхове. Очевидно $C_i = \widehat{D_{ii}}$.

Лема 6.3.7. *Нека $\Gamma = \langle \Sigma, N, A_1, \Pi \rangle$ да е линейна граматика и H_Γ да е диаграмата на преходите, съгласно (6.3.13). Нека P_Γ да е множеството от всички пътища $\pi \in H_\Gamma$ с начало върхът A_1 и край върхът A_{n+1}. Тогава*

$$L = L(\Gamma) = f_d(l_\Gamma(P_\Gamma)).$$

Доказателство – непосредствено.

□

Следствие 6.3.8. *Нека L да е линеен език, породен от линейната граматика (6.3.12) и нека G да е крайнопородена група с разрашим проблем за равенство на думите. Нека множеството от образуващи да е $\Sigma = \{x_1, x_2, ..., x_m, x_1^{-1}, ..., x_m^{-1}\}$. Тогава*

$$L \subseteq \mathfrak{L}(G) \Longleftrightarrow f_d(l_U(D_{1\,n+1})) = \{e\}.$$

□

Както и в раздел 6.3.2, стр. 133, разглеждаме полупръстените $F_G = (\mathcal{P}(G), \cup, \cdot, \phi, \{e\})$ и $F_U = (\mathcal{P}(U_G), \cup, \diamond, \phi, \{(e,e)\})$. Дефинираните с помощта на равенствата (6.3.9), (6.3.10) и (6.3.11) изображения f_l, fr и f_d по естествен начин могат да бъдат продължени до изображения от F_U в F_G.

Нека $X, Y, Z \in F_G$. В F_G въвеждаме операцията:

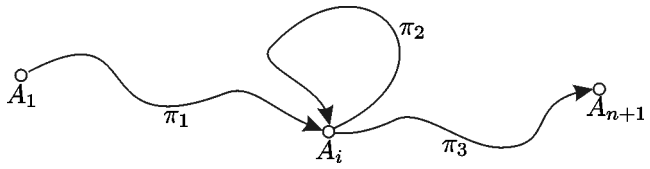
$$\langle X, Y, Z\rangle = \left\{xyzy^{-1} \mid x \in X, y \in Y, z \in Z\right\} \tag{6.3.15}$$

Разглеждаме следните елементи на полупръстена F_G:

$\Omega_7 = \left\{ f_d(l_U(\pi)) \;\middle|\; \pi \in \widehat{D_{1\,n+1}} \right\}$;

$\Omega_8 = \{\langle f_l(l_U(\pi_2)), f_d(l_U(\pi_3)), f_r(l_U(\pi_2))\rangle\}$, където π_2 и π_3 са такива пътища, че за някое $i \in \{1, 2, \ldots, n\}$ е изпълнено $\pi_2 \in C_i$, $\pi_3 \in \widehat{D_{i\,n+1}}$ и съществува път $\pi_1 \in D_{1i}$ (фиг. 6.6);

$W_4 = \Omega_7 \cup \Omega_8$.



Фигура 6.6

Не е трудно да се забележи, че

$$\Omega_7 \subseteq \Omega_1 \quad \text{и} \quad \Omega_8 \subseteq \Omega_2 \Longrightarrow W_4 \subseteq W_1 \tag{6.3.16}$$

и в общия случай $\Omega_7 \neq \Omega_1 \quad$ и $\quad \Omega_8 \neq \Omega_2$.

Както и в раздел 6.3.2 (виж (6.3.5) и (6.3.6)), дефинираме множествата от пътища $\mathcal{K}_{ij}^k$ в H_U, където $i \in \{1, 2, ..., n\}$, $j \in \{1, 2, ..., n+1\}$, $k \in \{0, 1, ..., n\}$, $n = |N|$, $V = N \cup \{A_{n+1}\}$, $N = \{A_1, A_2, ..., A_n\}$ е множество от нетерминални символи в линейната граматика Γ, $A_{n+1} \notin N$.

$$\mathcal{K}_{ij}^0 = \{\rho \mid \rho = (A_i, A_j) \in R\} \tag{6.3.17}$$

$$\mathcal{K}_{ij}^k = \mathcal{K}_{ij}^{k-1} \cup \mathcal{K}_{ik}^{k-1}\mathcal{K}_{kj}^{k-1} \tag{6.3.18}$$

Нека $g_{ij}^k = l_U(\mathcal{K}_{ij}^k) \in F_U$, където k е индекс.

Разглеждаме следните елементи на полупръстена F_G:

$\Omega_9 = f_d(g_{1\,n+1}^n)$;

$\Omega_{10} = \left\{\langle f_l(g_{ii}^n), f_d(g_{i\,n+1}^n), f_r(g_{ii}^n)\rangle \;\middle|\; 1 \leq i \leq n, \mathcal{K}_{1i}^n \neq \emptyset, \mathcal{K}_{ii}^n \neq \emptyset, \mathcal{K}_{i\,n+1}^n \neq \emptyset\right\}$;

$W_5 = \Omega_9 \cup \Omega_{10}$.

Лесно се забелязва, че

$$\Omega_7 \subseteq \Omega_9 \quad \text{и} \quad \Omega_8 \subseteq \Omega_{10} \Longrightarrow W_4 \subseteq W_5. \tag{6.3.19}$$

Тъй като в $\mathcal{K}_{ij}^k$ е възможно съществуването на повтарящи се междинни върхове, то в общия случай $\Omega_7 \neq \Omega_9$ и $\Omega_8 \neq \Omega_{10}$.

Теорема 6.3.9. *Нека L да е линеен език и нека $L = L(\Gamma)$, където Γ е дефинираната с (6.3.12) линейна граматика. Тогава съгласно въведените по-горе означения, следните условия са еквивалентни:*

(i) $L \subseteq \mathfrak{L}(G)$;
(ii) $W_1 = \Omega_1 \cup \Omega_2 = \{e\}$;
(iii) $W_4 = \Omega_7 \cup \Omega_8 = \{e\}$;
(iv) $W_5 = \Omega_9 \cup \Omega_{10} = \{e\}$.

Доказателство. Еквивалентността на условия (i) и (ii) е доказана от Анатолий Василиевич Анисимов в [6, 90] (виж теорема 6.3.2). Както бе отбелязано по-горе (виж 6.3.16), $W_4 \subseteq W_1$ и следователно от $W_1 = \{e\}$ следва $W_4 = \{e\}$, т.е. от (ii) следва (iii). От (6.3.19) следва, че (iv) влече (iii). За да докажем теоремата е достатъчно да докажем, че от (iii) следва (i) и от (i) следва (iv).

От (iii) следва (i): Нека $W_4 = \Omega_7 \cup \Omega_8 = \{e\}$ и нека $\omega \in L$. Съгласно лема 6.3.7 $\omega \in f_d(l_\Gamma(P_\Gamma))$. Следователно ω може да бъде представена по следния начин $\omega = \omega_1\omega_2$, където (ω_1, ω_2) е етикет на път в H_Γ с начален връх A_1 и краен връх A_{n+1} и нека π да е съответният му път в диаграмата на преходите H_U. Тогава е изпълнено $f_d(l_U(\pi)) \equiv \omega \pmod G$.

Ако $\pi \in \widehat{D_{1\,n+1}}$, тогава $f_d(l_U(\pi)) \in f_d(l_U(\widehat{D_{1\,n+1}})) = \Omega_7 \subseteq \{e\}$ и следователно $\omega \in \mathfrak{L}(G)$.

Допускаме, че π съдържа повтарящи се междинни върхове. Тогава π може да бъде записан във вида $\pi = \pi_1\pi_2\pi_3$, където $\pi_1 \in D_{1j}$, $\pi_2 \in C_j$ и $\pi_3 \in \widehat{D_{j\,n+1}}$, за някое $j \in \{1, 2, \ldots, n\}$. Нека $l_U(\pi_1) = (a_1, b_1)$, $l_U(\pi_2) = (a_2, b_2)$ и $l_U(\pi_3) = (a_3, b_3)$. Тогава

$$f_d(l_U(\pi)) = f_d((a_1, b_1) \diamond (a_2, b_2) \diamond (a_3, b_3)) = f_d(a_1a_2a_3, b_3b_2b_1) = a_1a_2a_3b_3b_2b_1.$$

Но $a_2a_3b_3b_2(a_3b_3)^{-1} \in \Omega_8$ и следователно $a_2a_3b_3b_2(a_3b_3)^{-1} = e$, т.е. $a_1a_2a_3b_3b_2b_1 = a_1a_3b_3b_1$. Лесно е да се види, че (a_1a_3, b_3b_1) е етикетът на пътя $\pi_1\pi_3$, който се получава от π, премахвайки цикъла π_2. По този начин премахваме всички цикли в π. Тъй като думата ω е крайна, то след краен брой стъпки, ще получим път $\pi' \in \widehat{D_{1\,n+1}}$ такъв, че $f_d(l_u(\pi')) = f_d(l_U(\pi)) \equiv \omega \pmod G$. Но $f_d(l_U(\pi')) \in \Omega_7 = \{e\}$. Следователно $f_d(l_U(D_{1\,n+1})) = \{e\}$ и съгласно следствие 6.3.8 ще имаме $L \subseteq \mathfrak{L}(G)$.

От (i) следва (iv): Нека $L \subseteq \mathcal{M}$. Тогава съгласно следствие 6.3.8 ще имаме $f_d(l_U(D_{1\,n+1})) = \{e\}$. Очевидно $\mathcal{K}_{1\,n+1}^n \subseteq D_{1\,n+1}$, от където следва, че $\Omega_9 = \{e\}$.

Нека $z \in \Omega_{10}$. Тогава $z = uvwv^{-1}$, където $u \in f_l(g_{ii}^n)$, $v = f_d(g_{i\,n+1}^n)$, $w = f_r(g_{ii}^n)$ и следователно съществува път $\pi_1 \in D_{1i}$, $\pi_2 \in C_i$, $\pi_3 \in D_{i\,n+1}$ за някое $i \in \{1, 2, \ldots, n\}$, така че $(u, w) = l_u(\pi_2)$ и $v = v_1v_2$, където $(v_1, v_2) = l_u(\pi_3)$. Нека $l_U(\pi_1) = (x, y)$. Разглеждаме пътищата $\pi' = \pi_1\pi_2\pi_3 \in D_{1\,n+1}$ и $\pi'' = \pi_1\pi_3 \in D_{1\,n+1}$, за които получаваме:

$l_U(\pi') = l_U(\pi_1\pi_2\pi_3) = (x, y) \diamond (u, w) \diamond (v_1, v_2) = (xuv_1, v_2wy)$
$l_U(\pi'') = l_U(\pi_1\pi_3) = (x, y) \diamond (v_1, v_2) = (xv_1, v_2y)$

Съгласно следствие 6.3.8, $xuvwy = xvy = e$, от където следва, че $uvwv^{-1} = e$, т.е. $z = e$. Тъй като z е произволен елемент от Ω_{10}, то $\Omega_{10} = \{e\}$. Теоремата е доказана.

□

Следващият алгоритъм се основава на еквивалентности (i) и (iv) от 6.3.9.

Алгоритъм 6.3.10. *Проверява включването $L \subseteq \mathfrak{L}(G)$ за линейния език L и груповия език $\mathfrak{L}(G)$, където G е група с разрешим проблем за равенство на думите.*

Вход: $g_{ij}^0 = l_U(\mathcal{K}_{ij}^0)$, $i = 1, 2, ..., n$, $j = 1, 2, ..., n, n+1$

Изход: Булева променлива T, която получава стойност **True**, ако $L \subseteq \mathfrak{L}(G)$ и стойност **False** в противен случай. Алгоритъмът завършва своята работа веднага след получаване на стойност $T :=$ **False**.

begin
1. $T :=$ **True** ;
2. For $1 \leq k \leq n$ Do
3. For $1 \leq i \leq n$ and $1 \leq j \leq n+1$ Do
4. $g_{ij}^k := g_{ij}^{k-1} \cup g_{ik}^{k-1} \diamond g_{kj}^{k-1}$;
5. End Do;
6. End Do;
7. If $g_{1\ n+1}^n \neq \phi$ и $f_d(g_{1\ n+1}^n) \neq \{e\}$ Then
8. Begin $T :=$ **False**; Halt; End;
9. For $1 \leq i \leq n$ Do
10. If $g_{1i}^n \neq \phi$ and $g_{ii}^n \neq \phi$ and $g_{i\ n+1}^n \neq \phi$ Then
11. If $\langle f_l(g_{ii}^n), f_d(g_{i\ n+1}^n), f_r(g_{ii}^n)\rangle \neq \{e\}$ Then
12. Begin $T :=$ **False**; Halt; End;
13. End Do;
End.

Теорема 6.3.11. *Алгоритъм 6.3.10 проверява включването $L \subseteq \mathfrak{L}(G)$, където L е линеен език, породен от линейна граматика с n нетерминални символа, G е група с разрешим проблем на равенството на думите, $\mathfrak{L}(G)$ е съответният групов език.*

Алгоритъм 6.3.10 изпълнява не повече от $O(n^3)$ операции "$\cup$" and "$\diamond$" в полупръстена F_U и не повече от $O(n^2)$ операции "$\langle\ ,\ ,\ \rangle$" в полупръстена F_G, където операцията "$\langle\ ,\ ,\ \rangle$" е дефинирана с помощта на формула (6.3.15).

Доказателство. Аналогично на доказателството на теорема 6.3.5.

□

Следствие 6.3.12. *Ако операциите "$\cup$" и "$\diamond$" в полупръстена F_U и операцията "$\langle\ ,\ ,\ \rangle$" в полупръстена F_G могат да бъдат реализирани за полиномиално време, то алгоритъм 6.3.4 е полиномиален.*

6.4. Заключение към шеста глава

В настоящата глава продължихме да използваме теорията на графите за решаване на задачи от други области на математиката. В случая сме разгледали някои приложения на ориентирани графи в теоретичната информатика и по-специално в теорията на формалните езици и граматики. Подобен подход не е нов, известни са много теоретико-графови модели в тази област. Ние тук предлагаме

някои нови модели, като целта е решаването на един открит за науката проблем за намирането на полиномиален алгоритъм за проверка дали произволен безконтекстен език се съдържа в зададен групов език. Този проблем бе формулиран в края на миналия век от известния украински математик проф. Анатолий Василиевич Анисимов.

Целта на раздел 6.1 бе по занимателен начин да въведем основните дефиниции и означения от теорията на формалните езици и граматики, описвайки с езика и свойствата на понятията от тази област решението на известната в програмистките среди главоблъсканица за Ханойските кули.

В раздел 6.2 е предложен още един модел за описание на произволен безконтекстен език с помощта на диаграма на преходите (краен ориентиран граф, чийто дъги са маркирани с елементи на дадена полугрупа). Този модел е особено полезен при граматики в нормална форма на Чомски.

В раздел 6.3 се доближихме в голяма степен до пълното решение на проблема на А. В. Анисимов, като го решихме за важните от практическа гледна точка частни случаи, а именно за автоматните и за линейните езици. Съдържанието на този раздел е естествено продължение, съществено подобрение и обобщение на някои резултати от първата докторска дисертация на автора.

Заключение

В настоящия дисертационен труд бе направен опит за систематизиране на някои от постиженията на автора в качествата му на учен и преподавател по дискретна математика и програмиране.

Считаме, че поставените цели и задачи са изпълнени до голяма степен успешно, като постигнатите резултати са сравнени със световните постижения в тази област. Доказателство на този факт са публикациите на автора в реномирани научни издания с импакт фактор.

Приноси. По мнение на автора, основните научни, научно-приложни и методически приноси са:

- Разработеният в раздел 1.2 бърз алгоритъм, работещ за време $O(n)$ за сортиране на n цели числа;

- Доказаната на стр. 17 теорема 1.3.5, което ни дава основание да въведем понятието *канонична бинарна матрица*. Намирането на каноничните матрици в дадено множество от бинарни матрици е равносилно на получаването на точно по един представител от всеки клас на еквивалентност относно релацията на еквивалентност "$\sim$", където $A \sim B$ тогава и само тогава, когато A се получава от B чрез разместване на някои от редовете и/или стълбовете на B;

- В раздел 1.3 дефинираме понятието *полуканонична бинарна матрица*, разглеждаме множеството Λ_n^k, състоящо се от всички $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб и предлагаме алгоритъм за получаване на полуканоничните матрици от Λ_n^k без да се обхождат всички $n \times n$ бинарни матрици;

- Получените числови редици (1.3.5), (1.3.6), (1.3.7) и (1.3.8), които в последствие са отбелязани в енциклопедията от числови последователности [164], съответно под номерата A229161 [79], A229162 [80], A229163 [81] и A229164 [82].

- Направеният в глава 2 математически модел на различните тъкачни структури (*сплитки*). За целта е дефинирана релация на еквивалентност в множеството $\mathcal{Q}_n$ от всички $n \times n$ бинарни матрици, имащи поне една единица и поне една нула във всеки ред и всеки стълб;

- Програмната реализация на операциите && (покомпонентна конюнкция), || (покомпонентна дизюнкция), ! (покомпонентно отрицание), * (логическо произведение) и < (линейна наредба) в множеството $\mathfrak{B}_n$ от всички $n \times n$ бинарни матрици, представени по два начина: стандартно и с помощта на наредени n-торки от цели неотрицателни числа. Доказахме, че второто представяне, като се използват побитови операции, води до по-бързи и икономисващи оперативна памет алгоритми, реализиращи горепосочените операции;

- Намирането на мощността на дефинираните в раздел 2.2 фактормножества $\overline{\mathcal{Q}_n}$ (т.е. общ брой сплитки с повтор равен на n), $\overline{\mathcal{M}_n}$ (т.е броят на всички самоогледални сплитки с повтор равен на n) и $\overline{\mathcal{R}_n}$ (т.е. броят на всички ротационно устойчиви сплитки с повтор равен на n) за някои стойности на естественото число $n \geq 2$;

- Изследвани са някои комбинаторни проблеми, свързани с популярната главоблъсканица Судоку;
- От методическа гледна точка свързан с преподаването на дисциплината програмиране, е описан и подробно анализиран един алгоритъм, използващ теоретико-множествени операции за решаване на произволно Судоку;
- Описанието и реализацията на клас „множество“ с използуване на побитови операции и предефиниране на оператори (методически принос);
- Направен е математически модел на произволно психологическо изследване с личностни въпросници. Предложена е примерна компютърна реализация на този модел;
- Друго доказателство на теоремата на I. Good и J. Grook [115] (теорема 4.2.5);
- С използването на теоретико-множествените операции е описан алгоритъм за получаване на всички $n \times n$ бинарни матрици с точно две единици на всеки ред и всеки стълб при зададено естествено число $n \geq 2$ (методически принос);
- Конструктивно доказателство на формулата на В. Е. Тараканов (теорема 4.3.4);
- Доказаната теорема 4.4.3 утвърждаваща, че стойностите на функцията $\mu(n,k) = \left|\Lambda^k_{n/\sim}\right|$ при $n = k+1$ съвпадат с числата на Фибоначи, където с Λ^k_n означаваме множеството от всички квадратни $n \times n$ бинарни матрици с точно k единици на всеки ред и всеки стълб, а $\sim$ е релация на еквивалентност, такава че две матрици са еквивалентни по между си, ако едната се получава от другата след разместване на редовете и/или стълбовете.
- Теорема 4.5.1 с приложения в k-значната логика;
- Дефинирането на множеството Ψ_n от всички $(2n) \times n$ матрици, всеки ред на които представлява пермутация на елементите на $[n] = \{1, 2, \ldots, n\}$ и доказателството, че съществува биективно изображение от Ψ_n в множеството Σ_{n^2} от всички $n^2 \times n^2$ S-пермутационни матрици, при което двойка непресичащи се матрици от Ψ_n да съответстват на двойка непресичащи се матрици от Σ_{n^2} (теорема 4.6.6);
- Разгледаните в раздел 4.7 алгоритми за получаване на случайни обекти (методически принос);
- Теореми 4.8.1, 5.2.9 и 5.2.10, даващи различни формули за пресмятане броя на всевъзможните взаимно непресичащи се (disjoint) двойки от S-пермутационни матрици. Интересен подход в случая е използването на апарат от теория на графите;
- Преброяване на всички взаимно непресичащи се двойки от $n^2 \times n^2$ S-пермутационни матрици при $n = 2$ и $n = 3$. За целта са описани всички 2×2 и 3×3 биполярни графи, разглеждани с точност до изоморфизъм;
- Описание на решението на задачата за Ханойските кули, използвайки езика на безконтекстни граматики и констуирайки магазинен автомат (методически принос);

- Описаното в раздел 6.2 представяне на безконтекстните граматики с помощта на крайни ориентирани графи и което се различава от известните и разпространени до момента модели;

- Разгледан е проблемът на А. В. Анисимов за намиране на полиномиален алгоритъм, проверяващ дали произволен безконтекстен език се съдържа в даден групов език. Проблемът е решен в дисертационния труд за автоматните и линейните езици.

Публикации на автора по темата на дисертационния труд.

а) **Самостоятелни публикации в специализирани научни издания в чужбина:**

1. P[170] K. Yordzhev (Iordzhev), An n^2 Algorithm for Recognition of Context-free Languages. *Cybernetics and Systems Analysis*, 29, No.6 (1993) 922–927. http://link.springer.com/article/10.1007%2FBF01122746 (последно посетен на 15.06.2014) (Impact F.)

2. P[171] K. Yordzhev (Iordjev), On a Matrix Recognition Algorithm for Context-free Languages. in *First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, Aachen, Germany, September 7-10, 1993 v. 3, 1204–1210.

3. P[175] K. Yordzhev, An Entertaining Example of Using the Concepts of Context-Free Grammar and Pushdown Automation. *Open Journal of Discrete Mathematics*, 2 (2012), 105-108, http://www.scirp.org/journal/PaperInformation.aspx?PaperID=21127 (последно посетен на 15.06.2014) (Impact F.)

4. P[176] K. Yordzhev, Some Combinatorial Problems on Binary Matrices in Programming Courses. *Informational Technologies in Education*, № 12, (2012) 39-43. http://ite.kspu.edu/en/issue-12/p-39-43 (посетен на 15.06.2014)

5. P[177] K. Yordzhev, Random Permutations, Random Sudoku Matrices and Randomized Algorithms. *International Journal of Mathematical Sciences and Engineering Applications (IJMSEA)*, ISSN 0973-9424, Vol. 6, No. VI (2012), pp. 291–302. http://www.ascent-journals.com/ijmsea_contents_Vol6No6.html (последно посетен на 15.06.2014) (Impact F.)

6. P[178] K. Yordzhev, Bipartite Graphs Related to Mutually Disjoint S-permutation Matrices. *ISRN Discrete Mathematics*, vol. 2012, Article ID 384068, 18 pages, 2012. http://www.hindawi.com/journals/isrn.discrete.mathematics/2012/384068/ (последно посетен на 15.06.2014)

7. P[179] K. Yordzhev, A Representation of Context-free Grammars with the Help of Finite Digraphs. *American Journal of Applied Mathematics*. Vol. 1, No. 1 (2013) pp. 8–11. http://article.sciencepublishinggroup.com/pdf/10.11648.j.ajam.20130101.12.pdf (последно посетен на 15.06.2014)

8. P[180] K. Yordzhev, Inclusion of Regular and Linear Languages in Group Languages. *International Journal of Mathematical Sciences and Engineering Applications (IJMSEA)*, ISSN 0973-9424, Vol. 7 No. I (2013), pp. 323–336. http://www.ascent-journals.com/ijmsea_contents_Vol7No1.html (последно посетен на 15.06.2014) (Impact F.)

9. P[181] K. Yordzhev, Bitwise Operations Related to a Combinatorial Problem on Binary Matrices. *I. J. Modern Education and Computer Science*, 4 (2013) 19-24. http://www.mecs-press.org/ijmecs/ijmecs-v5-n4/IJMECS-V5-N4-3.pdf (последно посетен на 15.06.2014)

10. P[182] K. Yordzhev, On the Number of Disjoint Pairs of S-permutation Matrices. *Discrete Applied Mathematics*, 161 (2013), 3072–3079. www.sciencedirect.com/science/article/pii/S0166218X13002850 (последно посетен на 15.06.2014) (Impact F.)

11. P[183] K. Yordzhev, The Bitwise Operations Related to a Fast Sorting Algorithm. *International Journal of Advanced Computer Science and Applications*, Vol. 4, No. 9 (2013) 103-107. http://thesai.org/Downloads/Volume4No9/Paper_17-The_Bitwise_Operations_Related_to_a_Fast_Sorting.pdf (последно посетен на 15.06.2014) (Impact F.)

12. P[184] K. Yordzhev, On an Algorithm for Obtaining All Binary Matrices of Special Class Releted to V. E. Tarakanov's Formula. *Journal of Mathematical Sciences and Applications*, 1, no. 2 (2013): 36-38. http://pubs.sciepub.com/jmsa/1/2/5/jmsa-1-2-5.pdf (последно посетен на 15.06.2014)

13. P[185] K. Yordzhev, On an Algorithm for Isomorphism-Free Generations of Combinatorial Objects. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, Vol. 2, No. 6 (2013) 215–220. http://www.ijettcs.org/Volume2Issue6/IJETTCS-2013-12-21-080.pdf (последно посетен на 15.06.2014) (Impact F.)

14. P[186] K. Yordzhev, Fibonacci sequence related to a combinatorial problem on binary matrices. *American Journal Mathematics and Sciences (AJMS)*, ISSN 2250 3102, Vol. 3, No. 1 (2014), 79–83. http://ajms.yolasite.com/resources/12.Fibonacci.pdf (последно посетен на 15.06.2014) (*preprint* arXiv:1305.6790 http://arxiv.org/pdf/1305.6790v2.pdf (последно посетен на 15.06.2014))

15. P[188] K. Yordzhev, Factor-set of binary matrices and Fibonacci numbers. *Applied Mathematics and Computation*, Vol. 236, (2014), 235–238. http://www.sciencedirect.com/science/article/pii/S0096300314004354 (последно посетен на 15.06.2014) (Impact F.)

16. P[189] K. Yordzhev, On the probability of two randomly generated S-permutation matrices to be disjoint. *Statistics & Probability Letters*, Vol. 91 (2014). http://www.sciencedirect.com/science/article/pii/S0167715214001370 (посетен на 15.06.2014) (Impact F.)

17. P[22] К. Йорджев, Алгоритм распознавания контекстно-свободных языков за время n^2. *Кибернетика и системны анализ*, 6 (1993), 159-164. (Impact F.)

18. P[23] К. Йорджев, Еще о проблеме включения регулярных и линейных языков в групповые языки. International Conference "*Algebra, Logic & Discrete Mathematics*", Niš, Yugoslavia, April 14–16, 1995, *FILOMAT (Niš)* 9:3 (1995), 699–710. (Impact F.)

19. P[25] К. Йорджев, Системы счисления и комбинаторика. в *Проблемы и перспективы физико-математического и технического образования*. сборник материалов Всероссийской научно-практической конференции, отв. ред. Т.С.

Мамонтова, Ишим: Изд-во ИГПИ им. П.П. Ершова, 2013, 106–114. http://www.igpi-ishim.ru/novosti-fiziko-matematicheskiy-fakultet/ (последно посетен на 15.06.2014)

б) **Публикации в съавторство в специализирани научни издания в чужбина:**

20. P[148] I. Peneva, K. Yordzhev, Adnan Sharaf Ali, The Adaptation of Translation Psychological Test as a Necessary Condition for Ensuring the Reliability of Scientific Research. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Volume 2, Issue 4, July 2013, 557-560. http://www.ijesit.com/Volume%202/Issue%204/IJESIT201304_71.pdf (последно посетен на 15.06.2014) (Impact F.)

21. P[191] K. Yordzhev, H. Kostadinova, On Some Entertaining Applications of the Concept of Set in Computer Science Course, *Informational Technologies in Education*. № 10 (2011), 24–29. http://ite.ksu.ks.ua/en/issue-10/p-24-29 (последно посетен на 15.06.2014)

22. P[192] K. Yordzhev, H. Kostadinova, Mathematical Modeling in the Textile Industry. *Bulletin of Mathematical Sciences & Applications*, Vol. 1, No. 1 (2012), 20 –35, http://www.bmsa.us/admin/uploads/00Czn3.pdf (последно посетен на 15.06.2014)

23. P[193] K. Yordzhev, I. Peneva, Computer Administering of the Psychological Investigations – Set-Relational Representation, *Open Journal of Applied Sciences*, Vol 2, No 2, (2012) 110-114. http://www.scirp.org/journal/PaperInformation.aspx?PaperID=20331 (последно посетен на 15.06.2014) (Impact F.)

в) **Самостоятелни публикации в специализирани научни издания у нас:**

24. P[172] K. Yordzhev, On an equivalence relation in the set of the permutation matrices. in *Discrete Mathematics and Applications*, SWU "N. Rilski", Blagoevgrad, Bulgaria, 2004, 77–87.

25. P[173] K. Yordzhev, On a Class of Binary Matrices. *Mathematics and Educations in Mathematics*, v.37 (2008), 245–250. http://www.math.bas.bg/smb/2008_PK/2008/pdf/245-250.pdf (последно посетен на 15.06.2014)

26. P[174] K. Yordzhev, An Example for the Use of Bitwise Operations in programming. *Mathematics and education in mathematics*, 38 (2009), 196–202. http://www.math.bas.bg/smb/2009_PK/tom_2009/pdf/196-202.pdf (последно посетен на 15.06.2014)

27. P[187] K. Yordzhev, On some numerical characteristics of a bipartite graph. *Mathematics and education in mathematics*, 43 (2014), 101–104. http://www.math.bas.bg/smb/2014_PK/tom_2014/pdf/150-153.pdf (последно посетен на 15.06.2014)

28. P[24] К. Йорджев, Комбинаторни задачи над бинарни матрици. *Математика и математическо образование*, 24 (1995), 288–296.

г) **Публикации в съавторство в специализирани научни издания у нас:**

29. P[131] H. Kostadinova, K. Yordzhev, A Representation of Binary Matrices. in *Mathematics and education in mathematics*, 39, (2010), 198–206. http://www.math.bas.bg/smb/2010_PK/tom/pdf/198-206.pdf (последно посетен на 15.06.2014)

30. P[132] H. Kostadinova, K. Yordzhev, An Entertaining Example for the Usage of Bitwise Operations in Programming. *Mathematics and natural science*, v. 1, SWU "N. Rilski", (2011), 159-168. http://www.fmns.swu.bg/FMNS2011_Volume_1.pdf (последно посетен на 15.06.2014)

31. P[136] D. Kovachev, K. Yordzhev, Binary Matrices and Some Combinatorial Applications of the Theory of k-valued Functions. *Mathematics and Educations in Mathematics*, v. 34 (2005), 118–123.

32. P[137] D. Kovachev, K. Yordzhev, On Finding a Particular Class of Combinatorial Identities. *Mathematics and Natural Science*, v.1, 2009, 50-54. http://www.fmns.swu.bg/Volume_1.pdf (последно посетен на 15.06.2014)

33. P[147] I. Peneva, K. Gaidarov, K. Yordzhev, Computer Administering of Psychological Tests. in *Mathematics and Natural Sciences*, v.1, SWU N. Rilsky, Blagoevgrad, Bulgaria, 2009, 129-135. http://www.fmns.swu.bg/Fmns2009.html (последно посетен на 15.06.2014)

34. P[153] G. Praskova, I. Petrov, K. Yordzhev, I. Peneva, Online Generation of Psychological tests. in *Mathematics and natural science-2013*, Volume 1, SWU N. Rilsky, Blagoevgrad, Bulgaria, 2013, 235-240. http://www.fmns.swu.bg/FMNS2013-Volume_1.pdf (посетен на 15.06.2014)

35. P[190] K. Yordzhev, H. Kostadinova, Mathematical Modeling of the Weaving Structure Design. in *Mathematics and education in mathematics*, 39 (2010). http://www.math.bas.bg/smb/2010_PK/tom/pdf/212-220.pdf (последно посетен на 15.06.2014)

36. P[194] K. Yordzhev, I. Peneva, B. Kirilieva-Shivarova, A relational Model of Personality Psychological Tests. in *Mathematics and Natural Sciences*, v.1, 2009, 69-77. http://www.fmns.swu.bg/Fmns2009.html (последно посетен на 15.06.2014)

37. P[195] K. Yordzhev (Iordjev), S. Stefanov, On Some Applications of the Consept of Set in Computer Science Course. in *Mathematics and education in mathematics*, v.32 (2003), 249-252.

38. P[26] К. Йорджев, Д. Ковачев, Един клас от комбинаторни тъждества. *Математика*, 1 (2009), 9–12.

39. P[27] К. Йорджев, Х. Костадинова, Приложение на математически методи в сплиткознанието за получаване на количествени оценки на многообразието на тъкачните сплитки. *Текстил и облекло*, 1, (2011), 7–10.

40. P[28] К. Йорджев, И. Пенева, Относно необходимостта от разширяване практиката на компютърното тестиране в психодиагностичните изследвания в България. в *Съюз на учените - Благоевград, Годишник*, том 6, част 1, (2012), 364–371.

41. Р[29] К. Йорджев, И. Статулов, Математическо моделиране и количествена оценка на първичните тъкачни сплитки. *Текстил и облекло*, 10, (1999), 18–20.

42. Р[30] И. Калчев, К. Йорджев, В. Въчков, *Стохастични измервателни системи*, Технически Университет, София, 2007.

43. Р[47] И. Пенева, К. Йорджев, Интернет тестирането най-новата тенденция в психодиагностичните изследвания. *Trakia journal of sciences*, Volume 7, 2009, 155-159. http://tk.uni-sz.bg/files/M10_et_al.pdf (последно посетен на 15.06.2014)

44. Р[77] С. Щраков, К. Йорджев, М. Тодорова, *Ръководство за решаване на задачи по дискретна математика*. Благоевград, ЮЗУ "Н. Рилски", 2004.

Материалите от дисертацията са публикувани в 44 печатни работи, от които 2 книги Р[30, 77], 20 статии в рецензируеми и индексирани списания издавани в чужбина Р[22, 148, 170, 175–186, 188, 189, 191–193], 5 статии в списания издавани в България Р[26–29, 47] и 17 в сборници с доклади от международни или с международно участие научни конференции у нас и в чужбина Р[23–25, 131, 132, 136, 137, 147, 153, 171–174, 187, 190, 194, 195].

По отношение на използвания език 33 от работите са на английски език Р[131, 132, 136, 137, 147, 148, 153, 170–188, 188, 189, 189–195], 3 на руски език Р[22, 23, 25] и 8 са на български език Р[24, 26–30, 47, 77].

Степен на достоверност и апробация на резултатите. Основните резултати от дисертацията са докладвани на следните научни форуми:

а) **Научни форуми в чужбина:**

1. European Congress on Fuzzy and Intelligent Technologies (EUFIT'93), Aachen, Germany, September 7–10, 1993. Представена е работата Р[171].

2. International Conference "*Algebra, Logic & Discrete Mathematics*", Niš, Югославия, 14–16 Април, 1995. Представена е работата Р[23].

3. Всероссийская научно-практическая конференция с международным участием "*Проблемы и перспективы физико-математического и технического образования*", Ишим, Россия, 19-20 ноября 2013. Представена е работата Р[25].

б) **Международни или с международно участие научни форуми в страната:**

4. Двадесет и четвърта пролетна конференция на СМБ, Април 1995. Представена е работата Р[24].

5. Thirty Second Spring Conference of the Union of Bulgarian Mathematicians, April 2003. Представена е работата Р[195].

6. 7-th International Conference on Discrete Mathematics and Applications (ICDMA7), Bansko, June 17–20, 2004. Представена е работата Р[172].

7. Thirty Fourth Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, April 2005. Представена е работата Р[136].

8. Thirty Seventh Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, April 2–6, 2008. Представена е работата Р[173].

9. Thirty Eighth Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, April 1–5, 2009. Представена е работата Р[174].

10. Third International Scientific Conference – FMNS2009, 3 – 7 June 2009. Представена е работата Р[137].

11. Third International Scientific Conference – FMNS2009, 3 – 7 June 2009. Представена е работата Р[147].

12. Third International Scientific Conference – FMNS2009, 3 – 7 June 2009. Представена е работата Р[194].

13. Thirty Ninth Spring Conference of the Union of Bulgarian Mathematicians, Albena, April 6—10, 2010. Представена е работата Р[131].

14. Thirty Ninth Spring Conference of the Union of Bulgarian Mathematicians, Albena, April 6—10, 2010. Представена е работата Р[190].

15. Fourth International Scientific Conference – FMNS2011, 8 – 11 June 2011, Представена е работата Р[132].

16. Fifth International Scientific Conference – FMNS2013, 12 – 16 June 2013. Представена е работата Р[153].

17. Forty Third Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, April 1–5, 2014. Представена е работата Р[187].

Личен принос на автора. Подготовката за публикуване на получените резултати се проведе съвместно със съавторите на публикациите, при което приносът на дисертанта бе определящ. Всички представени в дисертацията резултати са получени лично от автора.

24 от публикациите Р[22–25, 170–189] са самостоятелни.

В монографията от трима автори Р[30], първа глава (стр. 6 – 74) е разработена самостоятелно от К. Йорджев.

Отделните глави на учебника Р[77] са написани, както следва:

Славчо Щраков - глава 4;

Красимир Йорджев - глава 1, частите 1.6, 1.7 и 1.8; глава 2, част 2.5 и глава 3;

Маргарита Тодорова - глава 1, частите 1.1, 1.2, 1.3, 1.4 и 1.5; глава 2, частите 2.1, 2.2, 2.3 и 2.4.

Публикацията [29] е подготвена съвместно с доц. д-р И. Статулов, инженер, специалист по текстилна техника и технологии, формулирал някои нерешени до съответният момент проблеми от тази област. Тяхното решаване посредством математическо моделиране с използване на теория на групите е направено от К. Йорджев. По-късно тази разработка бе продължена и доразвита успешно от автора на дисертационния труд с помощта на докторант Хр. Костадинова.

Работите Р[27, 131, 132, 190–192] са подготвени съвместно с докторант Христина Костадинова, като К. Йорджев бе научен ръководител на успешно защитената докторска дисертация. Постановката на задачите, както и методите и алгоритмите за тяхното решаване са формулирани от научния ръководител като техническото изпълнение бе извършено съвместно от двамата автори.

Работите P[28, 47, 147, 148, 193, 194] са направени съвместно с Ивелина Пенева, докторант и в последствие доктор по психология. Математическите модели и тяхната компютърна реализация са осъществени от К. Йорджев, като съавторката бе изключително полезна със своите експертни оценки и консултации от областта на психологията. При оформянето на публикациите P[147, 148, 194] от същата област, двамата автори бяхме подпомогнати съответно от проф. д-р К. Гайдаров, докторант Adnan Sharaf Ali от република Йемен и Б. Кирилиева-Шиварова, студент, магистърска степен на обучение.

Задачите, които са решени в работите P[136, 137] са формулирани от доц. д-р Д. Ковачев, а тяхното решение е направено съвместно от двамата автори.

Идеята за публикациите P[26, 195] е на К. Йорджев, а разработките са оформени съвместно от двамата съавтори съответно К. Йорджев и Д. Ковачев и К. Йорджев и С. Стефанов.

Публикацията P[153] е компилация от части на две магистърски дипломни работи разработени съответно от Гергана Праскова и Иван Петров под научното ръководство на К. Йорджев и научното консултиране на И. Пенева.

Импакт фактор. Резултатите от дисертационния труд са публикувани от автора в следните списания с импакт фактор:

1. *Applied Mathematics and Computation*, издателство "Elsevier", Impact Factor: 2012: 1.349 © Thomson Reuters Journal Citation Reports 2013, 5-Year Impact Factor: 1.454 – публикация P[188].
   Справка:
   www.journals.elsevier.com/applied-mathematics-and-computation (последно посетен на 15.06.2014)

2. *Discrete Applied Mathematics*, издателство "Elsevier", Impact Factor 2012: 0.718 © Thomson Reuters Journal Citation Reports 2013, 5-Year Impact Factor: 0.802 – публикация P[182].
   Справка:
   www.journals.elsevier.com/discrete-applied-mathematics (последно посетен на 15.06.2014)

3. *Statistics & Probability Letters*, издателство "Elsevier", Impact Factor 2012: 0.531 © Thomson Reuters Journal Citation Reports 2013, 5-Year Impact Factor: 0.615 – публикация P[189].
   Справка:
   www.journals.elsevier.com/statistics-and-probability-letters (последно посетен на 15.06.2014)

4. *FILOMAT (Niš)*, издателство "University of Nis", Impact Factor 2012: 0.714 © Thomson Reuters Journal Citation Reports 2013, – публикация P[23].
   Справка:
   http://www.journals4free.com/link.jsp?l=12798904 (последно посетен на 15.06.2014)

5. *Open Journal of Discrete Mathematics*, издателство "Scientific Research Publishing Inc. (SCIRP)", Impact Factor 0.15 based on the ISI Web of Knowledge – публикация P[175]
   Справка:
   http://www.scirp.org/journal/Indexing.aspx?JournalID=586 (последно посетен на 15.06.2014)

6. *Open Journal of Applied Sciences*, издателство "Scientific Research Publishing Inc. (SCIRP)", Impact Factor 0.08 based on the ISI Web of Knowledge – публикация P[193]
   Справка:

   http://www.scirp.org/journal/Indexing.aspx?JournalID=1003 (посетен на 15.06.2014)

7. *International Journal of Modern Education and Computer Science*, издателство "Modern Education & Computer Science Publisher" Impact Factor 0.13 based on the ISI Web of Knowledge – публикация P[181]
   Справка:
   http://www.mecs-press.org/ijmecs/indexing.html (посетен на 15.06.2014)

8. *International Journal of Mathematical Sciences and Engineering Applications*, издателство "Ascent Publication House", Impact Factor: 0.3025 – публикация P[177]
   Справка:
   http://www.ascent-journals.com/ijmsea_Impact_factor.html (посетен на 15.06.2014)

9. *International Journal of Mathematical Sciences and Engineering Applications*, издателство "Ascent Publication House", Impact Factor: 0.3025 – публикация P[180]
   Справка:
   http://www.ascent-journals.com/ijmsea_Impact_factor.html (посетен на 15.06.2014)

10. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, ISRA Journal Impact Factor: 2.524 – публикация P[185]
    Справка:
    http://www.israjif.org/ISSN2278-6856.html (посетен на 15.06.2014)

11. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, ISRA Journal Impact Factor: 1.753 – публикация P[148]
    Справка:
    http://www.israjif.org/ISSN2319-5967.html) (посетен на 15.06.2014)

12. *International Journal of Advanced Computer Science and Applications (IJACSA)*, издателство "The Science and Information (SAI) Organization", 2012 Impact Factor = 1.324 – публикация P[183]
    Справка:
    http://thesai.org/Publications/ImpactFactor (посетен на 15.06.2014)

13. *Кибернетика и системный анализ*, издателство "Институт кибернетики им. В.М. Глушкова НАН Украины", Според "Российского индекса научного цитирования (ИФ РИНЦ)", "НАУЧНАЯ ЭЛЕКТРОННАЯ БИБЛИОТЕКА", г. Москва http://elibrary.ru (посетен на 15.06.2014), ИФ РИНЦ=0,035 – публикация P[22].

Забелязани цитирания. (общо 39 цитирания, от които 16 в чужбина и 23 в България)

- Р[30] И. Калчев, К. Йорджев, В. Въчков, *Стохастични измервателни системи*, Технически Университет, София, 2007.

  1. N. Petrov, I. Peneva, Notes on the Reliability in Social Organizations. International Journal of Engineering Science and Innovative Technology (IJESIT), Volume 2, Issue 4, 2013, 163–166.
  2. Н. Петров, Абстрактност и конкретност на философската категория "надеждност". Наука, образование, култура, бр. 3, 2013, 17-27.
  3. И. Пенева, Надеждността в психологическите измервания. Наука, образование, култура, бр. 3, 2013, 87–92.
  4. И. Пенева, Приложение на системния и вероятностния подходи при прогнозиране на риска в социалните организации. Научна конференция с международно участие „Психологията – традиции и перспективи", ЮЗУ „Н. Рилски", Благоевград, 2013.
  5. С. Стоянова , Ив. Пенева, Методическо ръководство за провеждане на емпирични психологически изследвания. УИ към ЮЗУ „Н. Рилски", Благоевград, 2014.
  6. И. Пенева, И. Кръстев, Личност и асертивност в бизнеса. УИ към ЮЗУ „Н. Рилски", Благоевград, 2014.
  7. Н. Петров, К. Ангелова, С. Павлов, Уравнение за надеждностната устойчивост на фона на единството на микросвета. Наука, образование, култура, бр. 4, 2014, 51–65.

- Р[47] И. Пенева, К. Йорджев, Интернет тестирането—най-новата тенденция в психодиагностичните изследвания. *Trakia journal of sciences*, Volume 7, 2009, 155-159.

  1. Х. Костадинова, Г. Тотков, М. Райкова, Към автоматизирано генериране на тестове по Блум. Mathematics and education in mathematics, v. 39, 2011, 413-422.
  2. M. V. Todorova, P. R. Armyanov, Runtime Verification of Computer Programs and its Application in Programming Education. GSTF Journal of Mathematics, Statistics and Operations Research, Vol.1 No.1, July 2012, 105–110.
  3. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- Р[29] К. Йорджев, И. Статулов, Математическо моделиране и количествена оценка на първичните тъкачни сплитки. *Текстил и облекло*, 10, (1999), 18–20.

  1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- Р[77] С. Щраков, К. Йорджев, М. Тодорова, *Ръководство за решаване на задачи по дискретна математика*. Благоевград, ЮЗУ "Н. Рилски", 2004.

1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.
2. С. Стоянова , Ив. Пенева, Методическо ръководство за провеждане на емпирични психологически изследвания. УИ към ЮЗУ „Н. Рилски", Благоевград, 2014.

- P[147] I. Peneva, K. Gaidarov, K. Yordzhev, Computer Administering of Psychological Tests. in *Mathematics and Natural Sciences*, v.1, SWU N. Rilsky, Blagoevgrad, Bulgaria, 2009, 129-135.

  1. M. V. Todorova, P. R. Armyanov, Runtime Verification of Computer Programs and its Application in Programming Education. GSTF Journal of Mathematics, Statistics and Operations Research, Vol.1 No.1, July 2012, 105-110.

- P[170] K. Yordzhev (Iordzhev), An n^2 Algorithm for Recognition of Context-free Languages. *Cybernetics and Systems Analysis*, 29, No.6 (1993) 922–927.

  1. Kazimierz Gazek. Linear algebra over semirings. In A Guide to the Literature on Semirings and their Applications in Mathematics and Information Sciences, pages 25–42. Springer Netherlands, 2002.
  2. Kazimierz Gazek. Selected applications of semirings. In A Guide to the Literature on Semirings and their Applications in Mathematics and Information Sciences, pages 67–87. Springer Netherlands, 2002.

- P[172] K. Yordzhev, On an equivalence relation in the set of the permutation matrices. in *Discrete Mathematics and Applications*, SWU "N. Rilski", Blagoevgrad, Bulgaria, 2004, 77–87.

  1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- P[174] K. Yordzhev, An Example for the Use of Bitwise Operations in programming. in *Mathematics and education in mathematics*, 38 (2009), 196–202.

  1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- P[177] K. Yordzhev, *Random Permutations, Random Sudoku Matrices and Randomized Algorithms*. International Journal of Mathematical Sciences and Engineering Applications, Vol. 6, No. VI (2012), pp. 291–302.

  1. N. Petrov, I. Peneva, Notes on the Reliability in Social Organizations. International Journal of Engineering Science and Innovative Technology (IJESIT), Volume 2, Issue 4, 2013, 163–166.
  2. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.
  3. Н. Петров, Абстрактност и конкретност на филосовската категория "надеждност". Наука, образование, култура, бр. 3, 2013, 17-27.

4. И. Пенева, Приложение на системния и вероятностния подходи при прогнозиране на риска в социалните организации. Научна конференция с международно участие „Психологията – традиции и перспективи", ЮЗУ „Н. Рилски", Благоевград, 2013.

- P[180] K. Yordzhev, Inclusion of Regular and Linear Languages in Group Languages. *International J. of Math. Sci. & Engg. Appls. (IJMSEA)* ISSN 0973-9424, Vol. 7 No. I (January, 2013), pp. 323–336.

1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- P[186] K. Yordzhev, Fibonacci sequence related to a combinatorial problem on binary matrices. *American Journal Mathematics and Sciences (AJMS)*, ISSN 2250 3102, Vol. 3, No. 1 (2014), 79–83. http://ajms.yolasite.com/resources/12.Fibonacci.pdf (последно посетен на 15.06.2014)
(*preprint* arXiv:1305.6790 http://arxiv.org/pdf/1305.6790v2.pdf (последно посетен на 15.06.2014))

1. A229161 – Number of $n \times n$ binary matrices with exactly 2 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229161 (последно посетен на 15.06.2014)
2. A229162 – Number of $n \times n$ binary matrices with exactly 3 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229162 (последно посетен на 15.06.2014)
3. A229163 – Number of $n \times n$ binary matrices with exactly 4 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229163 (последно посетен на 15.06.2014)
4. A229164 – Number of $n \times n$ binary matrices with exactly 5 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229164 (последно посетен на 15.06.2014)

- P[185] K. Yordzhev, On an Algorithm for Isomorphism-Free Generations of Combinatorial Objects. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, Vol. 2, No. 6 (2013) 215–220.
http://www.ijettcs.org/Volume2Issue6/IJETTCS-2013-12-21-080.pdf (последно посетен на 15.06.2014)

1. A229161 – Number of $n \times n$ binary matrices with exactly 2 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229161 (последно посетен на 15.06.2014)
2. A229162 – Number of $n \times n$ binary matrices with exactly 3 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229162 (последно посетен на 15.06.2014)

3. A229163 – Number of $n \times n$ binary matrices with exactly 4 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229163 (последно посетен на 15.06.2014)
4. A229164 – Number of $n \times n$ binary matrices with exactly 5 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in The On-Line Encyclopedia of Integer Sequences® (OEIS®) http://oeis.org/A229164 (последно посетен на 15.06.2014)

- P[193] K. Yordzhev, I. Peneva, Computer Administering of the Psychological Investigations – Set-Relational Representation, *Open Journal of Applied Sciences*, Vol 2, No 2, (2012) 110-114.

  1. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.
  2. Аднан Шараф Али, Обучението по информатика и компютърни науки в съвременното общество. Наука, образование, култура, бр. 4, 2014, 115–120.

- P[194] K. Yordzhev, I. Peneva, B. Kirilieva-Shivarova, A relational Model of Personality Psychological Tests. in *Mathematics and Natural Sciences*, v.1, 2009, 69-77.

  1. M. Raykova, H. Kostadinova, G. Totkov, Adaptive test system based on revised Bloom's taxonomy. Proceedings of the 12th International Conference on Computer Systems and Technologies, Viena, 2011, pp. 504-509.
  2. M. V. Todorova, P. R. Armyanov, Runtime Verification of Computer Programs and its Application in Programming Education. GSTF Journal of Mathematics, Statistics and Operations Research, Vol.1 No.1, July 2012, 105–110.
  3. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

- P[195] K. Yordzhev, S. Stefanov, On Some Applications of the Consept of Set in Computer Science Course. in *Mathematics and education in mathematics*, v.32 (2003), 249-252.

  1. M. Palahanova,The Concept Set in the Programming. Mathematics and Natural Sciences, v.1, 2009, 211-216
  2. Х. Костадинова, Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра. Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

Обучение на докторанти. Авторът на дисертационния труд е

а) **Научен ръководител на:**

1. **Христина Александрова Костадинова**, "*Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра*", дата на защита – 14.05.2013 г., ЮЗУ "Н. Рилски".

2. **Аднан Шараф Али Юсеф**, Република Йемен, "*Междууниверситетски обмен на информация в Република Йемен*", заповед за зачисляване 1316 / 05.06.2013 г., ЮЗУ "Н. Рилски".

3. **Станчо Вълканов Павлов**, "*Актуални проблеми на математическата теория на катастрофите – компютърна симулация*", заповед за зачисляване 3320 / 18.12.2012 г., ЮЗУ "Н. Рилски".

б) **Научен консултант на:**

4. **Ивелина Пенева Енчева**, "*Специфика на асертивността при студенти*", дата на защита 16.11.2012 г., ЮЗУ "Н. Рилски".

Библиография

1. П. Азълов, *Информатика Езикът C++ в примери и задачи за 9-10 клас.* София, Просвета, 2005.

2. П. Азълов, *Обектно ориентирано програмиране. Структури от данни и STL.* София, Сиела, 2008.

3. П. Азълов, Ф. Златарова *Информатика за 9 клас профилирана подготовка.* София, Просвета, 2001.

4. П. Азълов, Ф. Златарова, М. Тодорова *Информатика за 10 клас профилирана подготовка.* София, Просвета, 2003.

5. А. В. Анисимов, О групповых языках. *Кибернетика*, 4 (1971) 18–24.

6. А. В. Анисимов, Полугрупповые конечно-автоматные отображения. *Кибернетика*, 5 (1981), 1–7.

7. А. В. Анисимов, *Рекурсивные преобразователи информации.* Киев, Вища школа, 1987.

8. А. В. Анисимов, *Информатика, творчество, рекурсия.* Киев, Наукова думка, 1988.

9. В. И. Баранов, Б. С. Стечкин, *Экстремальные номбинаторные задачи и их приложения.* Москва, Наука, 1989

10. Г. И. Борзунов, *Шерстяная промишленост-обзорная информащия.* том 3, Москва, ЦНИИ ИТЭИЛП, 1983.

11. Л. Бурлачук, С. Морозов *Соварь-справочник по психодиагностике.* Санкт-Петербург, Питер, 2002.

12. Б. Бухбергер, Ж. Калме, Э. Калтофен, Дж. Коллинз, М. Лауэр, Ж. Лафон, Р. Лоос, М. Минньотт, Й. Нойбюзер, А. Норман, Ф. Уинклер, Я. ван Хюльзен *Компьютерная алгебра: Символьные и алгебраические вычисления.* Москва, Мир, 1986.— (Пер. с англ./Под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса, 1982 and 1983 by Springer-Verlag/Wien)

13. И. Буюклиев, *Алгоритмични подходи за изследване на линейни кодове.* Дисертация за присъждане на научна степен "доктор на математическите науки", БАН, 2008.

14. И. Буюклиев, Алгоритми за класификация на комбинаторни обекти с отхвърляне на изоморфните в процеса на генериране. *Математика и математическо образование*, 38 (2009), 51–60.

15. С. В. Глушаков, А. В. Коваль, С. В. Смирнов, *Язык программирования C++.* Харьков, Фолио, 2001.

16. Р. И. Григорчук, В. В. Некрашевич, В. И. Сущанский, Автоматы, динамические системы и группы. *Труды математического института имени В.А. Стеклова*, т. 231 (2000) 134—214.

17. Г. Давидков *Развлекателен софтуер с логическа насоченост в мрежов вариант.* Дипломна работа, бакалавърска степен, ЮЗУ "Н. Рилски", 2010.

18. Й. Денев, Р. Павлов, Я. Деметрович, *Дискретна математика.* София, Наука и изкуство, 1984.

19. Й. Д. Денев, С. Щраков, *Дискретна математика.* Благоевград, ЮЗУ "Н. Рилски", 1995.

20. С. Додунеков, К. Чакърян, *Задачи по теория на числата.* София, Регалия 6, 1999.

21. Г. Имам, *Компютърно администриране на психологически тестове.* Дипломна работа, бакалавърска степен, ЮЗУ "Н. Рилски", 2010.

22. К. Йорджев, Алгоритм распознавания контекстно-свободных языков за время n^2. *Кибернетика и системный анализ*, 6 (1993), 159-164

23. К. Йорджев, Еще о проблеме включения регулярных и линейных языков в групповые языки. International Conference "*Algebra, Logic & Discrete Mathematics*", Niš, Yugoslavia, April 14–16, 1995, *FILOMAT (Niš)* 9:3 (1995),699–710.

24. К. Йорджев, Комбинаторни задачи над бинарни матрици. *Математика и математическо образование*, 24 (1995), 288–296.

25. К. Йорджев, Системы счисления и комбинаторика. в *Проблемы и перспективы физико-математического и технического образования.* сборник материалов Всероссийской научно-практической конференции, отв. ред. Т.С. Мамонтова, Ишим: Изд-во ИГПИ им. П.П. Ершова, 2013, 106–114. http://www.igpi-ishim.ru/novosti-fiziko-matematicheskiy-fakultet/ (последно посетен на 15.06.2014)

26. К. Йорджев, Д. Ковачев, Един клас от комбинаторни тъждества. *Математика*, 1 (2009), 9–12.

27. К. Йорджев, Х. Костадинова, Приложение на математически методи в сплиткознанието за получаване на количествени оценки на многообразието на тъкачните сплитки. *Текстил и облекло*, 1, (2011), 7–10.

28. К. Йорджев, И. Пенева, Относно необходимостта от разширяване практиката на компютърното тестиране в психодиагностичните изследвания в България. в *Съюз на учените - Благоевград, Годишник*, том 6, част 1, (2012), 364–371.

29. К. Йорджев, И. Статулов, Математическо моделиране и количествена оценка на първичните тъкачни сплитки. *Текстил и облекло*, 10, (1999), 18–20.

30. И. Калчев, К. Йорджев, В. Въчков, *Стохастични измервателни системи*, Технически Университет, София, 2007.

31. А. В. Картузов, *Программирование на языке JAVA.* http://bookz.ru/authors/kartuzov-av/kartuzovav01.html (последно посетен на 15.06.2014)

32. Х. Костадинова, *Автоматизирано създаване на адаптивни е-курсове и комбинаторни приложения на матричната алгебра.* Дисертация, доктор, ЮЗУ "Н. Рилски", 2013.

33. Х. Крушков, *Практическо ръководство по програмиране на C++.* Пловдив, Макрос,2006.

34. А. Г. Курош, *Курс высшей алгебры.* Москва, Наука, 1975.

35. О. С. Леонтьева, *Новая книга СУДОКУ. СУПЕР головоломки.* Москва, РИПОЛ классик, 2006.

36. К. Манев, *Увод в дискретната математика.* София, КЛМН, 2003.

37. О. В. Мельников, В. Н. Ремесленников, В. А. Романков, Л. А. Скорняков, И. П. Шестяков, *Общая алгебра.* Москва, Наука, 1990.

38. И. Мирчев, *Графи - оптимизационни алгоритми в мрежи.* ЮЗУ "Неофит Рилски", Благоевград, 2001.

39. И. Мирчев, *Теория на числата.* ЮЗУ "Неофит Рилски", Благоевград, 1995.

40. П. Наков, П. Добриков *Програмиране =++ Алгоритми.* Трето издание, София, 2005. ISBN 954-8905-16-X.

41. С. Наков и колектив, *Въведение в програмирането с Java.* НАРС, 2008.

42. А. Наследов, *Математические методы психологического исследования. Анализ и интерпретация данных.* Санкт-Петербург. Речь, 2004.

43. Р. Павлов, *Математическа лингвистика.* София, Народна просвета, 1982.

44. Р. Павлов, С. Радев, С. Щраков, *Математически основи на информатиката.* ЮЗУ "Неофит Рилски", Благоевград, 1996.

45. И. Пенева, *Проектиране на софтуер за компютърно администриране на психологически въпросници.* Дипломна работа, магистърска степен, ЮЗУ "Н. Рилски", 2009.

46. И. Пенева, *Специфика на асертивността при студенти.* Дисертация, доктор, ЮЗУ "Н. Рилски", 2012.

47. И. Пенева, К. Йорджев, Интернет тестирането — най-новата тенденция в психодиагностичните изследвания. *Trakia journal of sciences*, Volume 7, 2009, 155-159. http://tk.uni-sz.bg/files/M10_et_al.pdf (последно посетен на 15.06.2014)

48. Ю. Пенева *Бази от данни.* Първа част. София, Регалия 6, 2004.

49. Ю. Пенева, Г. Тупаров *Бази от данни.* Втора част. София, Регалия 6, 2004.

50. И. Петров, *Компютърно администриране на личностни психологически тестове - част II.* Дипломна работа, магистърска степен, ЮЗУ "Н. Рилски", 2013.

51. Н. Петров *Вероятност, независимост, информационно общество.* Ямбол, изд. "Жельо Учков", 2008. ISBN 978-954-931-007-6

52. Б. С. Послед, *Borland C++ Builder: Разработка приложений баз данных.* Киев, ДиаСофтЮП, 2003.

53. Г. Праскова, *Компютърно администриране на личностни психологически тестове - част I.* Дипломна работа, магистърска степен, ЮЗУ "Н. Рилски", 2013.

54. Е. Л. Романов, *Практикум по программированию на C++.* Санкт Петербург, БХВ, 2004.

55. В. Н. Сачков, В. Е. Тараканов *Комбинаторика неотрицательных матриц.* Научное издательство ТПВ, Москва, 2000. (English translation: V. N. Sachkov, V. E. Tarakanov, *Combinatorics of Nonnegative Matrices.* Translations of Mathematical Monographs, American Mathematical Society, 2002.)

56. Бл. Сендов, В. Попов, *Числени методи.* Втора част, София, Наука и изкуство, 1978.

57. Е. Сидоренко, *Методы математическои обработки в психологии.* Санкт Петербург, Речь, 2000.

58. И. М. Соболь, *Метод Монте-Карло.* Москва, Наука, 1968.

59. Н. Стаменкова, *Психологически измервания.* София, Псидо, 2005.

60. И. В. Статулов Множествена оценка на тъкачната сплитка. *Текстилна промишленост*, 3 (1973).

61. Л. Стефанова, *Статистическа обработка с език за програмиране Java.* Дипломна работа, бакалавърска степен, ЮЗУ "Н. Рилски", 2012.

62. Р. Стоянова, Г. Гочев, *Програмиране на Pascal с 99 примерни програми.* Paraflow, Sofia, 1994.

63. С. Стоянова, *Основи на психологическите измервания. Адаптация на тест.* Благоевград, ЮЗУ "Н. Рилски", 2007.

64. С. Стоянова , И. Пенева, *Методическо ръководство за провеждане на емпирични психологически изследвания.* Благоевград, ЮЗУ "Н. Рилски", 2014.

65. В. Е. Тараканов Комбинаторные задачи на бинарных матрицах. *Комбинаторный анализ*, Москва, изд-во МГУ, 1980, вып.5, 4-15.

66. В. Е. Тараканов, *Комбинаторные задачи и (0,1)-матрицы.* Москва, Наука, 1985.

67. М. Тодорова, *Програмиране на C++.* Част I, София, Сиела, 2002.

68. М. Тодорова, *Програмиране на C++.* част II, София, Сиела, 2002.

69. М. Тодорова, *Обектно-ориентирано програмиране на базата на езика C++.* СИЕЛА СОФТ ЕНД ПУБЛИШИНГ, София, 2011, ISBN 978-954-28-0909-8.

70. М. Тодорова, *Структури от данни и програмиране на езика C++*, СИЕЛА СОФТ ЕНД ПУБЛИШИНГ, София, 2011, ISBN 978-954-28-0990-6.

71. Л. Тотина, *Матрично представяне на решението на един комбинаторен проблем от теория на групите*. Дипломна работа, бакалавърска степен, ЮЗУ "Н. Рилски", 2011

72. Уикипедия, *Судоку*. http://bg.wikipedia.org/wiki/%D0%A1%D1%83%D0%B4%D0%BE%D0%BA%D1%83 (последно посетен на 15.06.2014)

73. Н. Хадживанов *Избрани етюди по теория на графите*. София, Везни-4, 2012.

74. А. Д. Хомоненко, В. М. Циганков, М. Г. Мальцев, *Базы данных: Учебник для ВУЗ*. Санкт-Петербург, КОРОНА принт, 2004.

75. А. Д. Хомоненко, С. Е. Ададуров, *Работа с базами данных в C++ Builder*. Санкт-Петербург, „БХБ-Петербург", 2006.

76. В. С. Шевелев Редуцированные латинские прямоугольники и квадратные матрицы с одинаковыми суммами в стоках и столбцах. *Дискретная математика*, том 4, вып. 1, 1992, 91-110.

77. С. Щраков, К. Йорджев, М. Тодорова, *Ръководство за решаване на задачи по дискретна математика*. Благоевград, ЮЗУ "Н. Рилски", 2004.

78. А. Л. Эфрос, *Физика и геометрия безпорядка*. Москва, Наука, 1982.

79. A229161 – Number of $n \times n$ binary matrices with exactly 2 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in *The On-Line Encyclopedia of Integer Sequences® (OEIS®)* http://oeis.org/A229161 (последно посетен на 15.06.2014)

80. A229162 – Number of $n \times n$ binary matrices with exactly 3 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in *The On-Line Encyclopedia of Integer Sequences® (OEIS®)* http://oeis.org/A229162 (последно посетен на 15.06.2014)

81. A229163 – Number of $n \times n$ binary matrices with exactly 4 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in *The On-Line Encyclopedia of Integer Sequences® (OEIS®)* http://oeis.org/A229163 (последно посетен на 15.06.2014)

82. A229164 – Number of $n \times n$ binary matrices with exactly 5 ones in each row and column, and with rows and columns in lexicographically nondecreasing order. in *The On-Line Encyclopedia of Integer Sequences® (OEIS®)* http://oeis.org/A229164 (последно посетен на 15.06.2014)

83. A.V. Aho, J.D. Ullman, *The theory of parsing, translation and computing*, volume 1,2. Prentice-Hall, 1972. (русский перевод: А. Ахо, Дж. Ульман, *Теория синтаксического анализа, перевода и компиляции* Том 1,2. Москва, Мир, 1978.)

84. M. Aigner, *Combinatorial Theory*, Springer-Verlag, 1979. (русский перевод: М. Айгнер *Комбинаторная теория*. Москва, Мир, 1982.)

85. J.-P. Allouche, F. Dress, Tours de Hanoï et automates. *Informatique théorique et applications*, 24(1), 1990, 1–15. (in French).

86. J.-P. Allouche, J. Shallit, *Automatic sequences. Theory, applications, generalizations*. University press, Cambridge, 2003.

87. H. Anand, V. C. Dumir, H. Gupta, A combinatorial distribution problem. *Duke Math. J.* 33 (1966), 757-769.

88. A. Anastasi, S. Urbina *Psychological testing*. Prentice-Hall, 1997.

89. A. V. Anisimov, Group languages. *Cybernetics and Systems Analysis*, 7 (1971), pp. 594–601.

90. A. V. Anisimov, Finite-automaton semigroup mappings, *Cybernetics and Systems Analysis*, 17 (1981), pp. 571–578.

91. K. Arnold, J. Goslin, D. Holmes, *The Java Programming Language*. Sun Microsystems,2000. (превод на български: К. Арнолд, Дж. Гослинг, Д. Холмс, *Програмният език JAVA*. ИнфоДАР, 2001.)

92. J. Arsac, *Jeux et casse - tête a programmer*. BORDAS, Paris, 1985. (in French).

93. K. Atanassov, V. Atanassova, A. Shannon, J. Tumer, *New Visual Perspectives on Fibonacci Numbers*. World Scientific, 2002.

94. R.A. Bailey, P.J. Cameron, R. Connelly, Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and hamming codes. *Amer. Math. Monthly*, 115, (2008), 383–404.

95. F. Baker, Computer technology in test construction and processing. In I.R.Line (Ed), *Educational measurement*, 3rd ed., New York, American council in Education. Macmillan, 1989, 409-428.

96. J. Buther, (Ed.), *Computerized psychological itssessment: A practitioner's guide*. New York, Basic Books, 1987.

97. O. Bogopolski, *Introduction to Group Theory*. European Mathematical Society, Zurich, 2008.

98. I. Chiswell. *A course in formal languages, automata and groups*. Springer-Verlag, London, 2009.

99. E. F. Codd. A relational model of data for large shared data banks. *Communication of the Association for Computing Machinery*, 13(6):377–387, 1970.

100. W. J. Collins, *Data structures and the standard template library*. McGraw-Hill, 2002.

101. C. W. Curtis, I. Rainer, *Representation Theory of Finite Groups and Associative Algebras*. Wiley-Interscience, 1962.

102. D. J. Dahl, E. W. Dijkstra, C. A. R. Hoare, *Structured Programming*. Academic Press Inc., 1972.

103. G. Dahl, Permutation matrices related to sudoku. *Linear Algebra and its Applications*, 430, (2009), 2457–2463.

104. J. Daintith, R. D. Nelson, *The Penguin Dictionary of Mathemathics*. Penguin books, 1989.

105. J. Darlington, A Synthesis of Several Sorting Algorithms. *Acta Informatica* Volume 11, Number 1, December 18, 1978, 1–30.

106. S. R. Davis, *C++ for dummies*. IDG Books Worldwide, 2000.

107. R. Diestel, *Graph Theory*, Springer-Verlag Heidelberg, New York, 1997, 2000, 2006.

108. B. Felgenhauer, F. Jarvis, *Enumerating possible sudoku grids* (2005). http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf (последно посетен на 15.06.2014)

109. D. Flanagan, *JAVA in a nutshell*. O'Reilly, 2002. (русский перевод: Д. Флэнеган *Java. Справочник.* Санкт Петербург, Символ-Плюс, 2004.)

110. R. Fontana, Fraction of permutations. an application to sudoku, *Journal of Statistical Planning and Inference*, 141 (2011) 3697–3704.

111. M. R. Garey, D. S. Jonson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, 1979. (русский перевод: М. Гэри, Д. Джонсон, *Вычислительные машины и труднорешаемые задачи*. Москва, Мир, 1982.)

112. J. E. Gentle. *Numerical Linear Algebra for Applications in Statistics*. Springer-Verlag, 1998.

113. J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 2003.

114. S. Ginsburg, *The mathematical theory of context-free languages*. Mc Graw-Hill, 1966. (русский перевод: С. Гинсбург, *Математическая теория контекстно-свободных языков*. Москва, Мир, 1970.)

115. I. Good, J. Grook, The enumeration of arrays and generalization related to contingency tables. *Discrete Mathematics*, 19 (1977), 23–45.

116. R. I. Grigorchuk, V. V. Nekrashevich, V. I. Sushchanskii, Automata, dynamical systems, and groups, *Proceedings of the Steklov Institute of Mathematics*, 231 (2000), pp. 128–203.

117. H. Gupta, G. L. Nath, Enumeration of stochastic cubes. *Notices of the Amer. Math. Soc.* 19 (1972) A–568.

118. M. Hall, *The Theory of Groups*. New York, 1959.

119. F. Harary, *Graph Theory*, Addison-Wesley, Massachusetts, 1998.

120. P.-C. Héam, On the complexity of computing the profinite closure of a rational language, *Theoretical Computer Science*, 412 (2011), pp. 5808 – 5813.

121. M. J. Hernandez. *Database Design for Mere Mortals.* Addison Wesley, 2 edition, 2003.

122. M. L. Hetland, *Python Algorithms: Mastering Basic Algorithms in the Python Language*. Apress, 2010.

123. J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

124. C. S. Horstmann, *Computing concepts with C++ essentials*. John Wiley & Sons,1999. (превод на български: Кай Хорстман, *Принципи на програмирането със C++*. София, СОФТЕХ, 2000.)

125. J. Hromkovic, *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*. Springer, 2006.

126. B. Huppert, *Enlishe Gruppen*. Springer, 1967.

127. K. Jensen, N. Wirth, *Pascal User Manual and Report*. 3rd ed., Springer-Verlag, 1985.

128. B. W. Kernigan, D. M Ritchie, *The C Programming Language*. AT&T Bell Laboratories, 1998.

129. D. E. Knuth, *The Art of Computer Programming. Volume 3: Sorting and Searching*, Second Edition, Addison-Wesley, 1998.

130. T. Koshy, *Fibonacci and Lucas Numbers with Applications*. John Wiley & Sons, 2011.

131. H. Kostadinova, K. Yordzhev, A Representation of Binary Matrices. in *Mathematics and education in mathematics*, 39, (2010), 198–206. http://www.math.bas.bg/smb/2010_PK/tom/pdf/198-206.pdf (последно посетен на 15.06.2014)

132. H. Kostadinova, K. Yordzhev, An Entertaining Example for the Usage of Bitwise Operations in Programming. *Mathematics and natural science*, v. 1, SWU "N. Rilski", (2011), 159-168. http://www.fmns.swu.bg/FMNS2011_Volume_1.pdf (последно посетен на 15.06.2014)

133. D. Kovachev, On the Number of Discrete Functions with a given Range. *General Algebra and Applications*, Proceedings of the "5th Workshop on General Algebra edited by K. Deneke and H.-J. Vogel, Potsdam 2000, 125–134

134. D. Kovachev, On the Number of Some k-valued Function of n Variables. *Mathematics and Education in Mathemathics*, v. 30, (2001), 176–181.

135. D. Kovachev, I. D. Gyudzhenov, On the Number of k-Valued Functions with Given Range of Their Subfunctions, *Discrete Mathematics and Applications*, Proceedings of the sixth international conference, August 31 - September 2, 2001, Bansko, Bulgaria, Edited by: Sl. Shtrakov, K. Denecke, South-West University, Blagoevgrad, 2002, 119–124.

136. D. Kovachev, K. Yordzhev, Binary Matrices and Some Combinatorial Applications of the Theory of k-valued Functions. *Mathematics and Educations in Mathematics*, v. 34 (2005), 118–123.

137. D. Kovachev, K. Yordzhev, On Finding a Particular Class of Combinatorial Identities. *Mathematics and Natural Science*, v.1, 2009, 50-54. http://www.fmns.swu.bg/Volume_1.pdf (последно посетен на 15.06.2014)

138. G. Lallement, *Semigroups and combinatorial applications*, John Wiley & Sons, New York-Chichester-Brisbane, 1979. (русский перевод: Ж. Лаллеман, *Полугруппы и комбинаторные приложения*. Москва, Мир, 1985.)

139. P. Lankaster, *Theory of Matrices*. Academic Press, New York, 1969. (русский перевод: П. Ланкастер *Теория матриц*, Москва, Наука, 1973)

140. R. Lischener, *STL Pocket Reference*. O'Reilly Media, 2004.

141. N. Y. Louis Lee, G. P. Goodwin, P. N. Johnson-Laird, *The psychological puzzle of Sudoku*. Thinking & Reasoning, 14 (4), (2008), 342 – 364.

142. D. Maier, *The theory of relational databases*. 1983.

143. J. McConnell, *Analysis of Algorithms*. Jones & Bartlett Publishers, 2009.

144. S. A. Miller. *Developmental research methods*. Prentice-Hall, Englewood Cliffs, NJ, 2 edition, 1998.

145. Mo Hui-Dong, Xu Ru-Gen, Sudoku Square — a New Design in Field Experiment. *Acta Agronomica Sinica*, 34(9): (2008), 1489–1493.

146. R. Motwani, P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

147. I. Peneva, K. Gaidarov, K. Yordzhev, Computer Administering of Psychological Tests. in *Mathematics and Natural Sciences*, v.1, SWU N. Rilsky, Blagoevgrad, Bulgaria, 2009, 129-135. http://www.fmns.swu.bg/Fmns2009.html (последно посетен на 15.06.2014)

148. I. Peneva, K. Yordzhev, Adnan Sharaf Ali, The Adaptation of Translation Psychological Test as a Necessary Condition for Ensuring the Reliability of Scientific Research. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Volume 2, Issue 4, July 2013, 557-560. http://www.ijesit.com/Volume%202/Issue%204/IJESIT201304_71.pdf (последно посетен на 15.06.2014)

149. D. Perrin, J.-E. Pin, *Infinite words. Automata, semigroups, logic and games*, Elsevier, 2004.

150. N. Petrov, *Probability Theory as a Systemic View of Nature and Society*. Trakia University, St. Zagora, 2009. ISBN 978-954-90893-4-9

151. N. Petrov, *Probability, Independence, Information Society*. Wydaw-nictwo Astra, Lodz, Poland, 2012.

152. N. Petrov, I. Peneva, Notes on the Reliability in Social Organizations. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Volume 2, Issue 4, 2013, 163–166.

153. G. Praskova, I. Petrov, K. Yordzhev, I. Peneva, Online Generation of Psychological tests. in *Mathematics and natural science-2013*, Volume 1, SWU N. Rilsky, Blagoevgrad, Bulgaria, 2013, 235-240. http://www.fmns.swu.bg/FMNS2013-Volume_1.pdf (последно посетен на 15.06.2014)

154. D. Price, *UCSD Pascal A Considerate Approach*. Prentice-Hall, 1983.

155. V. J. Rayward-Smith *A First Course in Formal Language Theory*. Blackwell, 1983. (русский перевод В. Дж. Рейуорд-Смит, *Теория формальных языков*. Москва, Радио и связь, 1988.)

156. H. Schildt, *Java 2 A Beginner's Guide*. McGraw-Hill, 2001. (превод на български: Х. Шилдт, *Java 2 – Ръководство на програмиста*, СофтПресс 2007.)

157. R. Sedgewick, *Algorithms in C++, Parts 1–4 Fundamentals, Data Structures, Sorting, Searching*. Third Edition, Addison Wesley Longman, 1999. (русский перевод Р. Седжвик, *Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск*. Киев, ДиаСофт, 2001.)

158. Ákos Seress, *Permutation Group Algorithms*. Cambridge University Press, 2003.

159. J.-P. Serre, *Linear Representations of Finite Groups*. Springer-Verlag, New York, 1977.

160. R. P. Stanley, *Enumerative combinatorics*. Wadword & Brooks, California, 1986. (русский перевод Р. Стенли, *Перечислительная комбинаторика*. Москва, Мир, 1990)

161. M. L. Stein, P. R. Stein, *Enumeration of stochastic matrices with integer elements*. Los Alamos Scientific Laboratory Report LA-4434, 1970.

162. M. Swami, K. Thulasirman, *Graphs, networks and algorithms*. John Wiley & Sons, 1981.

163. Tan Kiat Shi, W.-H. Steeb, Y. Hardy, *Symbolic C++: An Introduction to Computer Algebra using Object-Oriented Programming*. Springer, 2001. (русский перевод: К. Ш. Тан, В.-Х. Стиб, Й. Харди *Символьны C++: Введение в компьютерную алгебру с использованием объектно-ориентированного программирования*. Москва, Мир, 2001.)

164. *The On-Line Encyclopedia of Integer Sequences® (OEIS®)* http://oeis.org/ (последно посетен на 15.06.2014)

165. A. De Vos, *Reversible Computing: Fundamentals, Quantum Computing, and Applications*. Wiley, 2010.

166. N. Wirth, *Algorithms + data structures = programs*. Prentice Hall, 1976. (превод на български: *Н. Уирт*, Алгоритми + структури от данни = програми)

167. Wikipedia, *Sudoku*. http://en.wikipedia.org/wiki/Sudoku (последно посетен на 15.06.2014)

168. T. Yato *Complexity and Completeness of Finding Another Solution and Its Application to Puzzles*. Masters thesis, Univ. of Tokyo, Dept. of Information Science, 2003. http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.ps (последно посетен на 15.06.2014)

169. T. Yato, T. Seta Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. *IEICE Trans. Fundamentals*, E86-A (5), (2003), 1052—1060.

170. K. Yordzhev (Iordzhev), An n^2 Algorithm for Recognition of Context-free Languages. *Cybernetics and Systems Analysis*, 29, No.6 (1993) 922–927. http://link.springer.com/article/10.1007%2FBF01122746 (последно посетен на 15.06.2014)

171. K. Yordzhev (Iordjev), On a Matrix Recognition Algorithm for Context-free Languages. in *First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, Aachen, Germany, September 7-10, 1993 v. 3, 1204–1210.

172. K. Yordzhev, On an equivalence relation in the set of the permutation matrices. in *Discrete Mathematics and Applications*, SWU "N. Rilski", Blagoevgrad, Bulgaria, 2004, 77–87.

173. K. Yordzhev, On a Class of Binary Matrices. *Mathematics and Educations in Mathematics*, v.37 (2008), 245–250. http://www.math.bas.bg/smb/2008_PK/2008/pdf/245-250.pdf (последно посетен на 15.06.2014)

174. K. Yordzhev, An Example for the Use of Bitwise Operations in programming. *Mathematics and education in mathematics*, 38 (2009), 196–202. http://www.math.bas.bg/smb/2009_PK/tom_2009/pdf/196-202.pdf (последно посетен на 15.06.2014)

175. K. Yordzhev, An Entertaining Example of Using the Concepts of Context-Free Grammar and Pushdown Automation. *Open Journal of Discrete Mathematics*, 2 (2012) 105-108, http://www.scirp.org/journal/PaperInformation.aspx?PaperID=21127 (последно посетен на 15.06.2014)

176. K. Yordzhev, Some Combinatorial Problems on Binary Matrices in Programming Courses. *Informational Technologies in Education*, № 12, (2012) 39-43. http://ite.kspu.edu/en/issue-12/p-39-43 (последно посетен на 15.06.2014)

177. K. Yordzhev, Random Permutations, Random Sudoku Matrices and Randomized Algorithms. *International Journal of Mathematical Sciences and Engineering Applications (IJMSEA)*, ISSN 0973-9424, Vol. 6, No. VI (2012), pp. 291–302. http://www.ascent-journals.com/ijmsea_contents_Vol6No6.html (последно посетен на 15.06.2014)

178. K. Yordzhev, Bipartite Graphs Related to Mutually Disjoint S-permutation Matrices. *ISRN Discrete Mathematics*, vol. 2012, Article ID 384068, 18 pages, (2012). http://www.hindawi.com/journals/isrn.discrete.mathematics/2012/384068/ (последно посетен на 15.06.2014)

179. K. Yordzhev, A Representation of Context-free Grammars with the Help of Finite Digraphs. *American Journal of Applied Mathematics*. Vol. 1, No. 1 (2013) pp. 8-11. http://article.sciencepublishinggroup.com/pdf/10.11648.j.ajam.20130101.12.pdf (последно посетен на 15.06.2014)

180. K. Yordzhev, Inclusion of Regular and Linear Languages in Group Languages. *International Journal of Mathematical Sciences and Engineering Applications (IJMSEA)*, ISSN 0973-9424, Vol. 7 No. I (2013), pp. 323–336. http://www.ascent-journals.com/ijmsea_contents_Vol7No1.html (последно посетен на 15.06.2014)

181. K. Yordzhev, Bitwise Operations Related to a Combinatorial Problem on Binary Matrices. *I. J. Modern Education and Computer Science*, 4 (2013) 19-24 http://www.mecs-press.org/ijmecs/ijmecs-v5-n4/IJMECS-V5-N4-3.pdf (последно посетен на 15.06.2014)

182. K. Yordzhev, On the Number of Disjoint Pairs of S-permutation Matrices. *Discrete Applied Mathematics*, 161 (2013) 3072–3079 http://www.sciencedirect.com/science/article/pii/S0166218X13002850 (последно посетен на 15.06.2014)

183. K. Yordzhev, The Bitwise Operations Related to a Fast Sorting Algorithm. *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 4, No. 9 (2013) 103-107. http://thesai.org/Downloads/Volume4No9/Paper_17-The_Bitwise_Operations_Related_to_a_Fast_Sorting.pdf (последно посетен на 15.06.2014)

184. K. Yordzhev, On an Algorithm for Obtaining All Binary Matrices of Special Class Releted to V. E. Tarakanov's Formula. *Journal of Mathematical Sciences and Applications*, 1, No. 2 (2013): 36-38. http://pubs.sciepub.com/jmsa/1/2/5/jmsa-1-2-5.pdf (последно посетен на 15.06.2014)

185. K. Yordzhev, On an Algorithm for Isomorphism-Free Generations of Combinatorial Objects. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, Vol. 2, No. 6 (2013) 215–220. http://www.ijettcs.org/Volume2Issue6/IJETTCS-2013-12-21-080.pdf (последно посетен на 15.06.2014)

186. K. Yordzhev, Fibonacci sequence related to a combinatorial problem on binary matrices. *American Journal Mathematics and Sciences (AJMS)*, ISSN 2250 3102, Vol. 3, No. 1 (2014), 79–83. http://ajms.yolasite.com/resources/12.Fibonacci.pdf (последно посетен на 15.06.2014) (*preprint* arXiv:1305.6790, (2013) http://arxiv.org/pdf/1305.6790v2.pdf (последно посетен на 15.06.2014))

187. K. Yordzhev, On some numerical characteristics of a bipartite graph. *Mathematics and education in mathematics*, 43 (2014), 101–104. http://www.math.bas.bg/smb/2014_PK/tom_2014/pdf/150-153.pdf (последно посетен на 15.06.2014)

188. K. Yordzhev, Factor-set of binary matrices and Fibonacci numbers. *Applied Mathematics and Computation*, Vol. 236, (2014), 235–238. http://www.sciencedirect.com/science/article/pii/S0096300314004354 (последно посетен на 15.06.2014)

189. K. Yordzhev, On the probability of two randomly generated S-permutation matrices to be disjoint. *Statistics & Probability Letters*, Vol. 91 (2014). http://www.sciencedirect.com/science/article/pii/S0167715214001370 (последно посетен на 15.06.2014)

190. K. Yordzhev, H. Kostadinova, Mathematical Modeling of the Weaving Structure Design. in *Mathematics and education in mathematics*, 39 (2010). http://www.math.bas.bg/smb/2010_PK/tom/pdf/212-220.pdf (последно посетен на 15.06.2014)

191. K. Yordzhev, H. Kostadinova, On Some Entertaining Applications of the Concept of Set in Computer Science Course, *Informational Technologies in Education*. № 10 (2011), 24–29. http://ite.ksu.ks.ua/en/issue-10/p-24-29 (последно посетен на 15.06.2014)

192. K. Yordzhev, H. Kostadinova, Mathematical Modeling in the Textile Industry. *Bulletin of Mathematical Sciences & Applications*, Vol. 1, No. 1 (2012), 20 –35, http://www.bmsa.us/admin/uploads/00Czn3.pdf (последно посетен на 15.06.2014)

193. K. Yordzhev, I. Peneva, Computer Administering of the Psychological Investigations: Set-Relational Representation, *Open Journal of Applied Sciences*, Vol 2, No 2, (2012) 110-114. http://www.scirp.org/journal/PaperInformation.aspx?PaperID=20331 (последно посетен на 15.06.2014)

194. K. Yordzhev, I. Peneva, B. Kirilieva-Shivarova, A relational Model of Personality Psychological Tests. in *Mathematics and Natural Sciences*, v.1, 2009, 69-77. http://www.fmns.swu.bg/Fmns2009.html (последно посетен на 15.06.2014)

195. K. Yordzhev (Iordjev), S. Stefanov, On Some Applications of the Consept of Set in Computer Science Course. in *Mathematics and education in mathematics*, v.32 (2003), 249-252.