**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп'ютерних систем**

«На правах рукопису»
УДК ____519.85____

До захисту допущено:
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2021 р.

# Магістерська дисертація

**на здобуття ступеня магістра**

**за освітньо-професійною програмою**

**«Системне програмування і спеціалізовані комп'ютерні системи»**

**зі спеціальності 123 «Комп'ютерна інженерія»**

**на тему: «Засоби аналізу регулярних структур лінійної складності»**

Виконав:
студент II курсу, групи КВ-01мп
Керімов Хікмет Немат огли _____

Науковий керівник:
Доцент кафедри СПСКС, к.т.н., с.н.с,
Тесленко Олександр Кирилович _____
_____

Рецензент:
_____
_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп‘ютерних систем**

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп‘ютерна інженерія»

Освітньо-професійна програма «Системне програмування і спеціалізовані комп‘ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2020 р.

**ЗАВДАННЯ**
**на магістерську дисертацію студенту**

**Керімову Хікмету Немат огли**

1. Тема дисертації «Засоби аналізу регулярних структур лінійної складності», науковий керівник дисертації Тесленко Олександр Кирилович, доцент кафедри СПСКС, к.т.н., с.н.с, затверджені наказом по університету від «5» 11. 2021 р. №3682-С

2. Термін подання студентом дисертації: 10.12.2021

3. Об’єкт дослідження:

- алгоритми для формування конструктивних модулів, та операції підстановки які реалізуються на структурах із цих модулів.

4. Вихідні дані:

- засоби аналізу регулярних структур лінійної складності.

5. Перелік завдань, які потрібно розробити:

- аналіз існуючих досліджень щодо регулярних структур лінійної складності;
- дослідження алгоритмів та засобів, які використовуються при аналізі даних структур.

6. Перелік графічного (ілюстративного) матеріалу:

- презентація.

7. Перелік публікацій:

- IV Міжнародна Науково-Практична Конференція «Теоретичні Та Практичні Аспекти Розвитку Науки» (м. Львів, 23-24 листопада 2021 року);
- міжнародна науково-практична конференція «Наука, освіта, технології, інновації: тенденції, виклики, перспективи» (м. Полтава, 30 листопада 2021 року).

9. Дата видачі завдання 7.10.2020

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів магістерської дисертації | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою МД | 01.11.2020 | |
| 2. | Розроблення та узгодження технічного завдання | 14.02.2021 | |
| 3. | Аналіз існуючих рішень | 13.03.2021 | |
| 4. | Підготовка матеріалів першого розділу МД | 18.05.2021 | |
| 5. | Розроблення програмного забезпечення | 16.09.2021 | |
| 6. | Відлагодження програмного продукту | 14.10.2021 | |
| 7. | Підготовка матеріалів другого розділу МД | 20.10.2021 | |
| 9. | Оформлення документації МД | 30.11.2021 | |
| 10. | Проходження попереднього захисту | 01.12.2021 | |

Студент Хікмет КЕРІМОВ

Науковий керівник Олександр ТЕСЛЕНКО

# РЕФЕРАТ

**Актуальність теми.** На сьогодні зросла актуальність застосування для комп’ютерних пристроїв базових перетворень інформації та відповідних блоків з їх параметричним налаштуванням. До таких перетворень належать підстановки, які застосовуються у різних розділах математики, а також у практичних розробках. Дослідження підстановок у математиці мають досить значні результати, на відміну від комп’ютерної інженерії, де реалізація досліджена меншою мірою.

**Мета роботи:** розробка програмного засобу для аналізу регулярних структур; аналіз існуючих досліджень щодо регулярних структур лінійної складності; дослідження алгоритмів та засобів, які використовуються при аналізі даних структур; порівняння засобів аналізу, та реалізація оптимального засобу для аналізу.

**Об’єктом дослідження** є алгоритми для формування конструктивних модулів, та операції підстановки які реалізуються на структурах із цих модулів.

**Предметом дослідження** є аналіз регулярних структур лінійної складності, створення відповідних засобів та їх аналіз з існуючими засобами.

**Методи дослідження.** В роботі використовуються методи математичного моделювання, методи оптимізації та засоби комбінаторики.

**Наукова новизна.** Модифікація одного з розглянутих алгоритмів та подальший його розвиток для формування таблиць виходів та станів конструктивних модулів, зокрема засобами мови Python.

**Практична цінність** отриманих в роботі результатів полягає в тому, що було створено простий для розуміння, і відповідно простий для впровадження засіб для аналізу регулярних структур лінійної складності за допомогою можливостей бібліотеки мови Python - numpy. Вона використовується в основному для

виконання математичних операцій (в нашому випадку реалізація підстановок), та для роботи з багатовимірними масивами.

**Особистий внесок магістранта.** Дослідження та реалізація засобів аналізу регулярних структур лінійної складності засобами мови Python.

**Апробація результатів дисертації.** Основні положення і результати роботи були представлені на конференціях:

- IV Міжнародна Науково-Практична Конференція «Теоретичні Та Практичні Аспекти Розвитку Науки» (м. Львів, 23-24 листопада 2021 року);
- міжнародна науково-практична конференція «Наука, освіта, технології, інновації: тенденції, виклики, перспективи» (м. Полтава, 30 листопада 2021 року).

**Публікації.** Основні наукові результати дисертації опубліковані у двох публікаціях.

**Структура та обсяг роботи.** Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У *вступі* надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їх впровадження.

У *першому* розділі розглянуто існуючі дослідження щодо регулярних структур лінійної складності, та засоби які були використані для їх аналізу. Коротко були розглянуті основні види регулярних структур та їх властивості.

У *другому* розділі розглянуто алгоритми, які застосовуються для аналізу регулярних структур, та засоби комбінаторики, які вони застосовують, а саме підстановки.

У *третьому* розділі розглянуто засоби для реалізації алгоритмів з другого розділу  та засоби для здійснення підстановок. З них було обрано засіб для оптимального аналізу регулярних структур лінійної складності.

У *четвертому* розділі розглянуто власне реалізацію обраного засобу для аналізу регулярних структур лінійної складності.

У *висновках* представлені результати проведеної роботи.

Робота виконана на 80 аркушах, містить 53 посилання на використані джерела інформації, 26 рисунків і 15 таблиць.

**Ключові слова:** перестановки, регулярні структури, конструктивний модуль, алгоритми, Python.

## ABSTRACT

**Theme urgency.** Today, the application of basic information transformations and corresponding blocks with their parametric settings for computer devices has increased. Such transformations include substitutions that are used in various sections of mathematics, as well as in practical developments. Research on substitutions in mathematics has quite significant results, in contrast to computer engineering, where the implementation is less studied.

**Research objective:** development of software for analysis of regular structures; analysis of existing research on regular structures of linear complexity; research of algorithms and means used in the analysis of these structures; comparison of analysis tools, and implementation of the optimal tool for analysis.

**Object of research** are algorithms for the formation of structural modules, and substitution operations that are implemented on the structures of these modules.

**Subject of research** is the analysis of regular structures of linear complexity, the creation of appropriate tools and their analysis with existing tools.

**Research methods.** The thesis uses methods of mathematical modeling, optimization methods and combinatorics.

**Scientific novelty.** Modification of one of the considered algorithms and its subsequent development for the formation of tables of outputs and states of structural modules, in particular by means of the Python language.

**Practical value** of the results obtained in this work is that it was created easy to understand, and therefore easy to implement a tool for analyzing regular structures of linear complexity using the capabilities of the Python language library - numpy. It is used mainly to perform mathematical operations (in our case, the implementation of substitutions), and to work with multidimensional arrays.

**Personal contribution of the student of the master's degree.** Research and implementation of tools for analysis of regular structures of linear complexity using Python language.

**Approbation.** The main provisions and results of the work were presented at conferences:

- IV International Scientific and Practical Conference "Theoretical and Practical Aspects of Science Development" (Lviv, November 23-24, 2021);
- International scientific-practical conference "Science, education, technology, innovation: trends, challenges, prospects" (Poltava, November 30, 2021).

**Publications.** The main scientific results of the dissertation are published in two publications.

**Structure and content of the thesis.** The master thesis consists of the introduction, four chapters and conclusions.

The *introduction* provides a general description of the work, assesses the current state of the problem, substantiates the relevance of research, formulates the purpose and objectives of research, shows the scientific novelty of the results and the practical value of the work, provides information on testing results and their implementation.

The *first* chapter discusses the existing research on regular structures of linear complexity, and the tools that were used for their analysis. The main types of regular structures and their properties were briefly considered.

The *second* chapter discusses the algorithms used to analyze regular structures and the combinatorics tools they use, namely substitutions.

In the *third* chapter the means for realization of algorithms from the second chapter and means for realization of substitutions are considered. From them the means for the optimum analysis of regular structures of linear complexity was chosen.

The *fourth* chapter considers the actual implementation of the selected tool for the analysis of regular structures of linear complexity.

In the *conclusions* the general conclusions on the presented thesis are given; the obtained results are analyzed.

The thesis is presented on 80 pages, contains 53 references to used information sources, 26 figures, and 15 tables.

**Keywords**: permutations, regular structures, constructive module, algorithms, Python.

ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

КМ - конструктивний модуль.

ОККМ - одновимірний каскад конструктивних модулів.

NumPy (Numerical Python) – бібліотека мови програмування Python для роботи з масивами.

PRNG (Pseudorandom number generator) – генератор псевдовипадкових чисел.

# ВСТУП

Станом на сьогодні зросла актуальність використання різних перетворень інформації для комп'ютерних пристроїв. Одним із таких перетворень являються підстановки, які досить широко застосовуються в різних розділах математики, та інших науках. Але в комп'ютерній інженерії перестановки досліджені меншою мірою.

Для реалізації та аналізу підстановок були використані комбінаційні схеми, зокрема регулярні структури. Метою роботи є аналіз існуючих досліджень щодо регулярних структур лінійної складності та відповідне дослідження алгоритмів та засобів, які використовуються при аналізі таких структур і порівняння засобів аналізу.

Науковою новизною даної роботи, є модифікація одного з розглянутих алгоритмів, який використовувався для формування таблиць станів та виходів конструктивних модулів. На структурах із цих модулів власне і реалізуються операції перестановок.

Отримані в роботі результати, мають досить практичну цінність, адже створений програмний засіб, є досить простим для розуміння, та відповідно простим для майбутнього впровадження для аналізу регулярних структур лінійної складності. Це зумовлено використанням мови Python, яка є однією з простих та популярних мов програмування на сьогоднішній день. Зокрема, для реалізації перестановок, та роботи з генератором випадкових чисел, були досліджені засоби стандартної бібліотеки Python із засобами бібліотеки NumPy, яка застосовується для роботи з даними.

Основні положення і результати наукової роботи були представлені на двох конференціях:

- IV Міжнародна Науково-Практична Конференція «Теоретичні Та Практичні Аспекти Розвитку Науки» (м. Львів, 23-24 листопада 2021 року)

– міжнародна науково-практична конференція «Наука, освіта, технології, інновації: тенденції, виклики, перспективи» (м. Полтава, 30 листопада 2021 року).

# 1 АНАЛІЗ ІСНУЮЧИХ ДОСЛІДЖЕНЬ РЕГУЛЯРНИХ СТРУКТУР ЛІНІЙНОЇ СКЛАДНОСТІ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

## 1.1 Причини розгляду аналізу регулярних структур

На сьогодні зросла актуальність застосування для комп’ютерних пристроїв базових перетворень інформації та відповідних блоків з їх параметричним налаштуванням. До таких перетворень належать підстановки, які застосовуються майже в кожній галузі математики та в багатьох інших областях науки. В інформатиці вони використовуються для аналізу алгоритмів сортування; у квантовій фізиці для опису станів частинок; а в біології — для опису послідовностей РНК. У свою чергу, в математиці дослідження підстановок мають досить значні результати, тоді як у комп’ютерної інженерії, реалізація досліджена меншою мірою.

Максимальна продуктивність апаратної реалізації спеціалізованих розрахунків забезпечується за допомогою використання комбінаційних схем. У загальному випадку складність комбінаційних схем зростає експоненційно зі зростанням кількості розрядів k, що у свою чергу, може обмежити ефективну реалізацію довільних підстановок у більш великих масштабах. Задача розробки ефективної структури для реалізації підстановок зводиться, з одного боку, до аналізу властивостей підстановок, що можуть призвести до спрощення їх апаратної реалізації, а з іншого боку – до пошуку простої технологічної структури, що дає змогу реалізувати відповідні підстановки. За таку структуру можна прийняти одну з регулярних структур - одновимірні каскади конструктивних модулів (ОККМ) [1].

## 1.2 Огляд принципів роботи засобу аналізу регулярних структур лінійної складності

Для створення засобу аналізу регулярних структур лінійної складності, необхідно розробити програмне забезпечення для:

– формування таблиць виходів та станів;

– реалізації відображень;

– формування автоматів для реалізації обернених підстановок.

На рисунку 1.1 зображено приклад скінченного автомату.

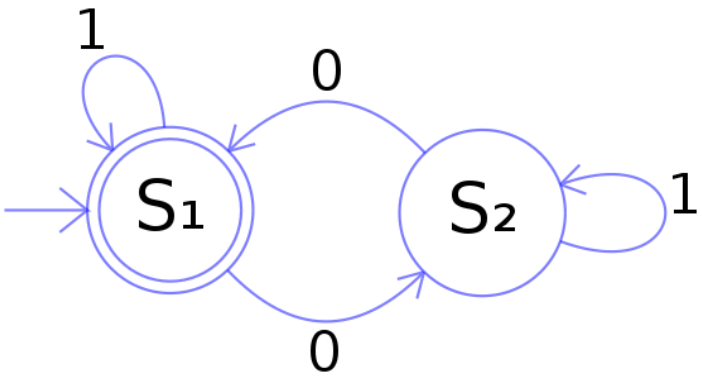


Рисунок 1.1 – Приклад скінченного автомату

Для формування таблиць й автоматів використовуються операції підстановки, які виконуються власне на структурах цих модулів. У роботі [2] був створений програмний засіб для реалізації цих пунктів, і він написаний мовою програмування C#. В рамках цієї роботи було вирішено скористатися засобами мови програмування Python, адже вона є більш легкою для реалізації, подальшого читання, підтримки та експортування на інші платформи, у порівнянні з C#. Та це не є єдиною перевагою мови Python, і більш детально аналіз даних мов програмування наведено у третьому розділі даної роботи.

## 1.3 Аналіз існуючих здобутків в дослідженнях регулярних структур лінійної складності

Методи апаратної реалізації підстановок обмеженої розрядності досліджувались у роботах [3] - [5], зокрема в [4], [5] розглянуто результати реалізації на програмованих логічних інтегральних схемах (ПЛІС), як довільних 8-розрядних підстановок, так і класу 8-розрядних підстановок зі спеціальними характеристиками. У роботах [6], [7] розглянуто просту програмну реалізацію довільних підстановок обмеженої розрядності, яка використовується у криптографічних перетвореннях. В розглянутих роботах досліджувався підхід «підстановки-реалізація», тобто формувався той чи інший клас підстановок, а потім визначалась реалізація підстановок цього класу. На рисунку 1.2 зображено візуалізацію роботи функції Фейстеля (функція F) стандарту DES, в той час як на рисунку 1.3, зображено візуалізацію роботи симетричного алгоритму блочного шифрування стандарту AES, який прийшов на заміну DES.

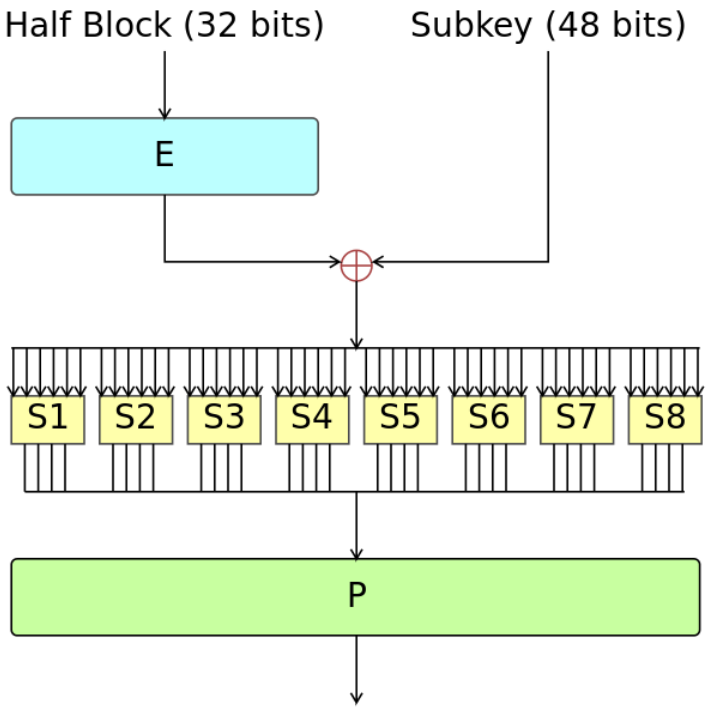


Рисунок 1.2 - Візуалізацію роботи функції Фейстеля (функція F) стандарту DES

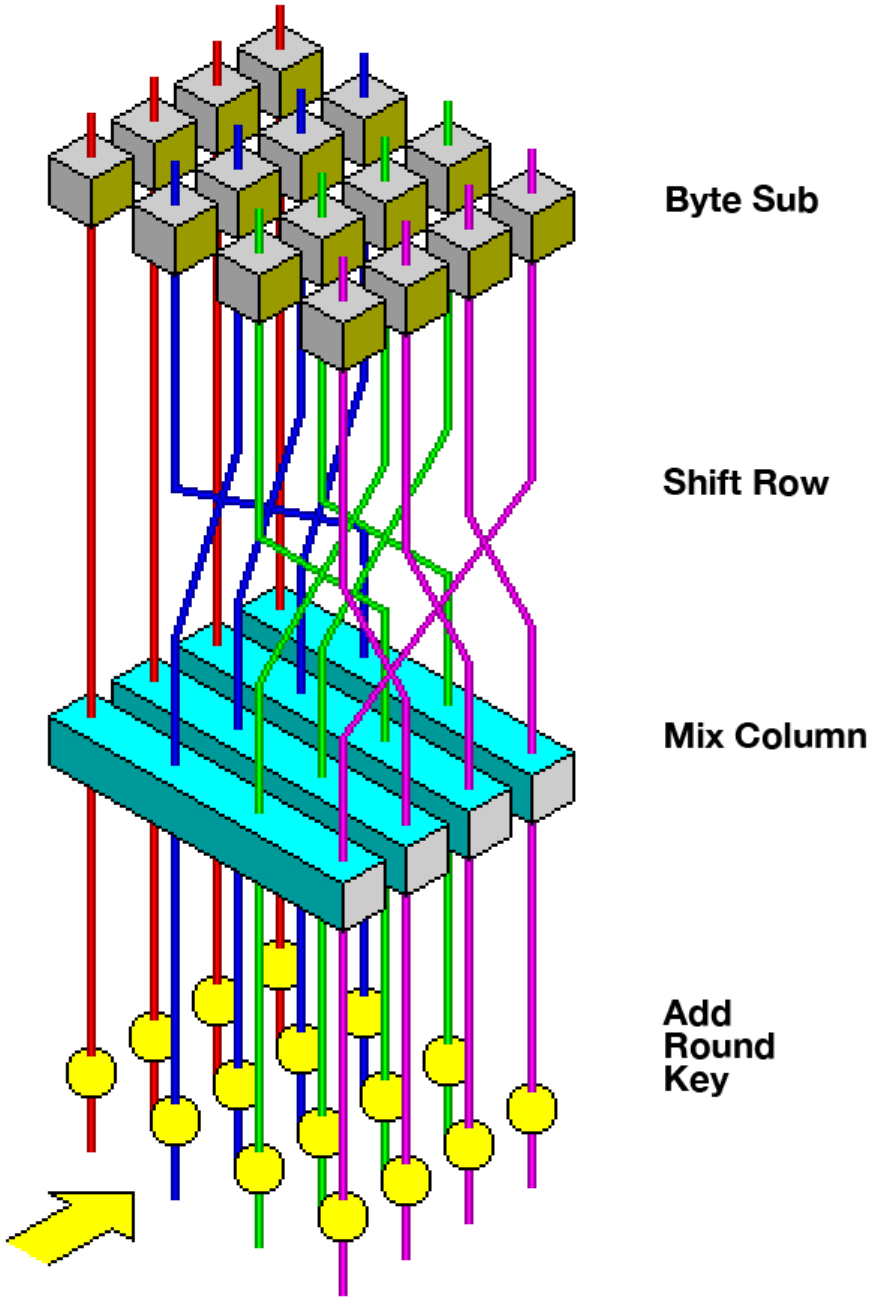


Рисунок 1.3 - Візуалізація роботи симетричного алгоритму блочного шифрування стандарту AES

В [8] було запропоновано інший підхід до реалізації підстановок, який можна охарактеризувати як «реалізація-підстановки».

Очевидно, що на ОККМ теоретично можна реалізовувати підстановки довільної розрядності. При цьому виникають наступні задачі: які повинні бути

конструктивні модулі (КМ) та скільки і які підстановки можна реалізувати на ОККМ відповідного класу. В загальному випадку ці задачі не вирішені.

В роботі [9] показано, що на найпростіших однонаправлених регулярних ОККМ шляхом зміни КМ можна реалізувати 48 різних підстановок довільної розрядності (степені $2^b$) ($b > 1$). В [10] показано, що на найпростіших двонаправлених регулярних ОККМ можна реалізувати 850 різних підстановок степені $2^b$ ($b > 1$). Значний приріст кількості різних підстановок при незначному ускладненні ОККМ дозволяє прогнозувати ефективність подальших ускладнень. Деталі проведених досліджень були наведені в наступних розділах роботи.

## 1.4 Обґрунтування теми дипломного проєкту

Як бачимо, перестановки досить широко застосовуються в різних розділах науки, і являються їх важливою складовою. Один із розглянутих засобів аналізу регулярних структур був реалізований досить давно, тому програмні засоби використані у ньому частково втратили свою актуальність. Саме тому, в рамках даної роботи, були реалізовані засоби з використанням актуальних та досить потужних програмних засобів. Python є досить простим у розумінні, написанні самого коду, та підтримці, що є перевагою у порівнянні з роботою [2].

## Висновки до розділу

У даному розділі було розглянуто існуючі здобутки у дослідженнях перестановок та сфер їх застосування. Також були окреслено здобутки у дослідженнях регулярних структур лінійної складності. У наступному розділі

будуть розглянуті власне регулярні структури, їх класифікація та алгоритми які застосовуються у роботі з ними.

# 2 АНАЛІЗ РЕГУЛЯРНИХ СТРУКТУР, АЛГОРИТМІВ І ЗАСОБІВ ДЛЯ ЇХ ФОРМУВАННЯ

## 2.1 Аналіз регулярних структур

Як було розглянуто у попередньому розділі для аналізу підстановок були використані комбінаційні схеми, а саме ОККМ. Ця комбінаційна структура лінійної складності в свою чергу, поділяється на наступні класи:

- найпростіших;
- простих;
- складних;
- однонаправлених (рис. 2.1);
- двонаправлених (рис. 2.2);
- регулярних каскадів.

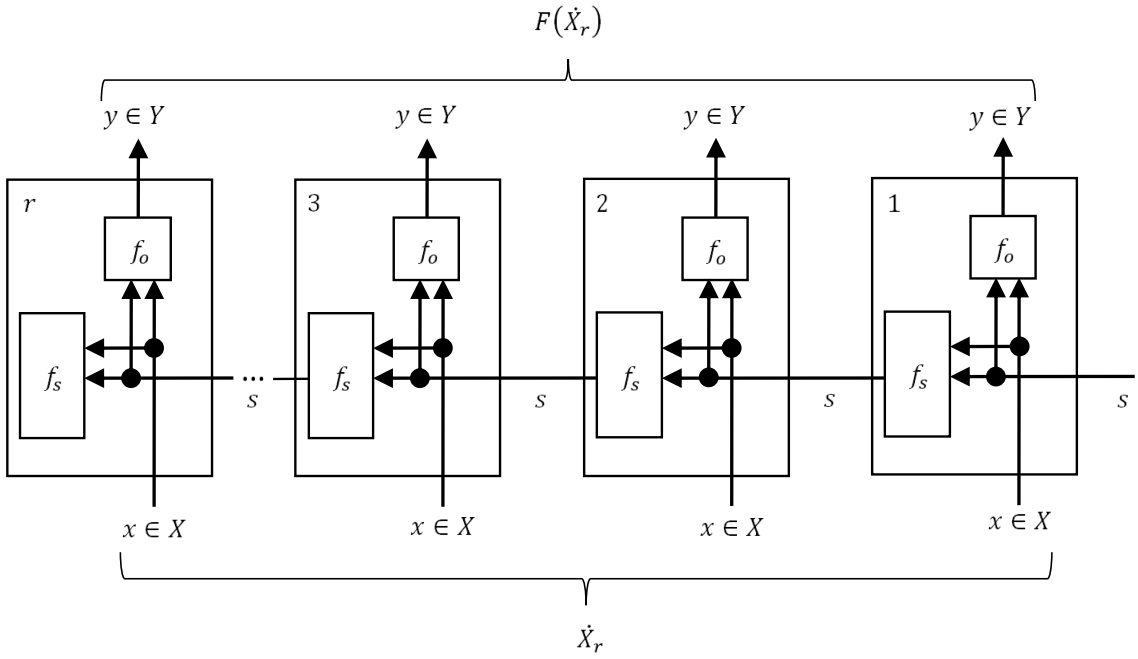


Рисунок 2.1 – Однонаправлений ОККМ

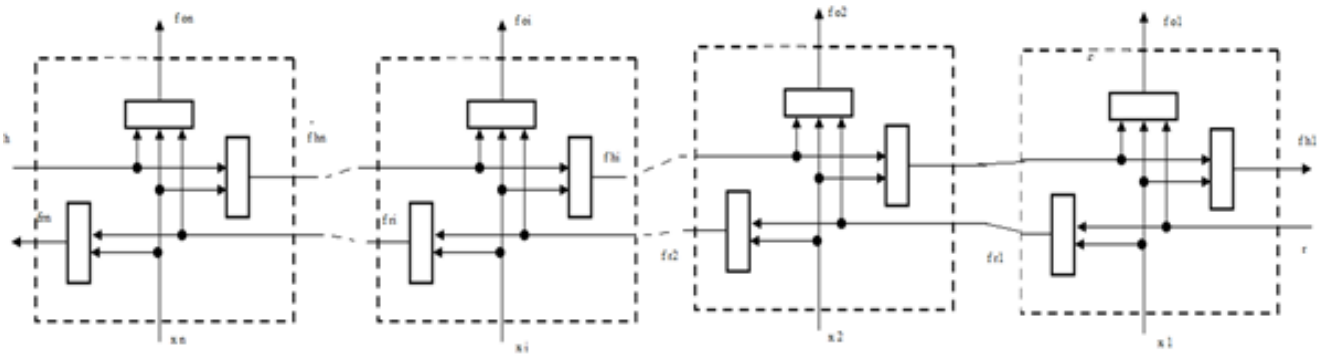

Рисунок 2.2 – Двонаправлений ОККМ

Розглянемо більш детально однонаправлений ОККМ. Конструктивний модуль цього каскаду складається з двох комбінаційних схем:

– перша реалізує значення сигналів на первинних виходах;

– друга реалізує значення сигналів на бокових виходах.

На входи обох комбінаційних схем поступають сигнали з первинних та бокових входів КМ. Очевидно, що такий клас ОККМ реалізує відображення вхідних даних, яке збігається з відображенням вхідних послідовностей відповідного цифрового автомату Мілі. На рисунку 2.3 зображено приклад простого автомату з одним входом та одним виходом.

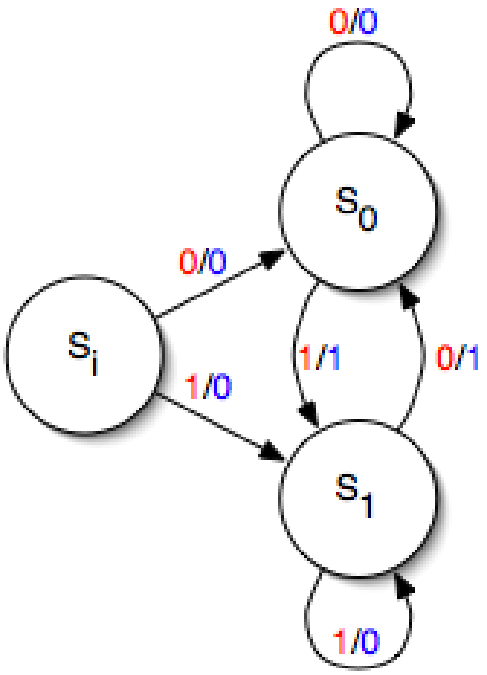


Рисунок 2.3 – Простий автомат Мілі

Це дозволяє використовувати теорію цифрових автоматів [11] для вирішення наступних задач:

- визначення комбінаційних схем для формування сигналів на виходах КМ та реалізації на ОККМ підстановок довільної розрядності (бієктивних відображень);
- визначення умов, при яких різні ОККМ реалізують однакові підстановки;
- визначення кількості різних підстановок, які можуть бути реалізовані на ОККМ вибраного класу;
- визначення обернених КМ, які б забезпечували реалізацію обернених підстановок при заданих прямих КМ.

У свою чергу автомати поділяються на декілька класів, що зображено на рисунку 2.4.

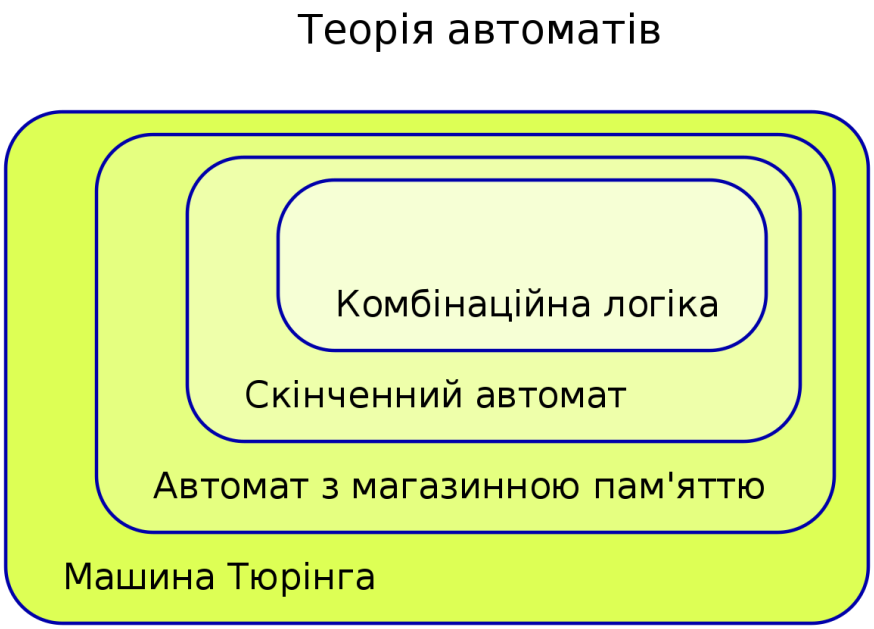


Рисунок 2.4 – Класифікація автоматів

Будемо вважати, що на первинні (не бокові) входи кожного конструктивного модуля подаються розряди двійкових кодів із монотонним зростанням номерів, а на первинних (не бокових) виходах кожного конструктивного модуля формуються значення розрядів результатів із тими самими номерами. Особливостями ОККМ є

теоретична необмеженість розрядності вхідних і вихідних даних (шляхом нарощування кількості конструктивних модулів), тобто на ОККМ реалізуються масові перетворювачі інформації з лінійною залежністю складності реалізації від кількості розрядів.

Каскади називають простими, якщо кількість бокових входів і виходів з кожного боку дорівнює одиниці. Коли до кожного модуля ОККМ подається один розряд вхідних даних, на виході модуля формується один розряд результату, і такий каскади називають найпростішими.

Практичну цінність має реалізація масових повних підстановок, тобто підстановок, у значеннях розрядності аргументів яких, пробігає певний ряд натуральних чисел. Структура ОККМ дозволяє збільшувати розрядність вхідних даних шляхом нарощування кількості модулів, але при цьому необхідно забезпечувати реалізацію повних підстановок на кожному етапі такого нарощування. При цьому можливі наступні два варіанти. За першим з них новий модуль може підключатись до каскаду тільки з якоїсь однієї сторони – або зі сторони молодших розрядів, або зі сторони старших розрядів. Для подальшого аналізу це не має суттєвого значення, тому будемо вважати, що нові модулі будуть підключатися до каскаду зі сторони старших розрядів. За другим варіантом, нові КМ можуть підключатись як зі сторони старших, так і зі сторони молодших розрядів. У випадку найпростіших ОККМ будемо вважати, що значення розрядності пробігає ряд натуральних чисел (n=1, 2, 3,…). Доречним є наступне формулювання методики побудови та вивчення властивостей n-розрядного ОККМ: (n – 1) - розрядний ОККМ розглядається як окремий (n – 1) розрядний конструктивний модуль (до певної міри ігноруючи внутрішню структуру), до якого, в свою чергу, підключається найпростіший однорозрядний КМ. Це дозволяє використати результати досліджень з реалізації повних підстановок на двохмодульному каскаді, які були розглянуті в роботі [12] та суть яких полягає в наступному.

Простий двомодульний каскад зображений на рис. 2.5. Позначимо $X_{1,t}$ кортеж (впорядковану послідовність) аргументів ($x_1$, $x_2$, …, $x_n$), які надходять на первинні входи правого модуля ОККМ (молодші розряди аргументу), відповідно $X_{t+1,n} \rightarrow (x_{t+1}, x_{t+2}, \ldots, x_n)$ - кортеж аргументів, що надходять на первинні входи лівого модуля (старші розряди аргументу), $F_{1,t}$ - кортеж функцій ($f_1$, $f_2$, …, $f_n$), що реалізуються на первинних виходах правого модуля ОККМ, аналогічно $F_{t+1,n} \rightarrow (f_{t+1}, f_{t+2}, \ldots, f_n)$ кортеж функцій, що реалізуються на первинних виходах лівого модуля ОККМ, $S_{1,t}$ – множина кодів значень аргументів із кортежа $X_{1,t}$, $S_{t+1,n}$ - множина кодів значень аргументів $X_{t+1,n}$.

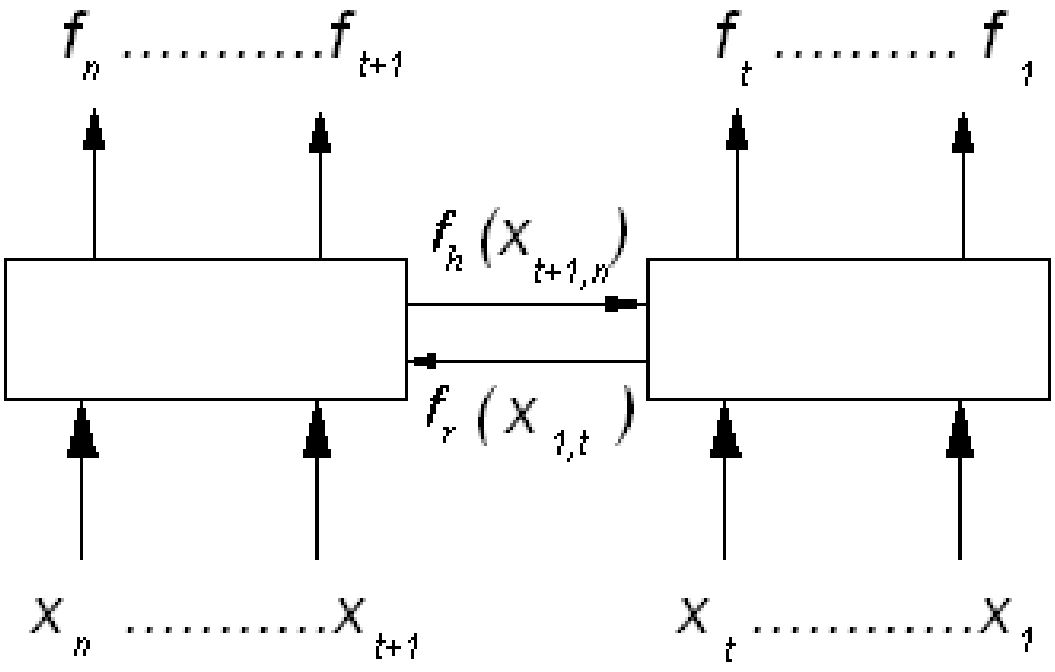


Рисунок 2.5 – Двомодульний каскад

Значення логічної функції $f_r(X_{1,t})$ на боковому виході правого модуля поділяє множину $S_{1,t}$ на дві підмножини $S_a$ і $S_b$ таким чином, що для будь-яких значень аргументів $a_1, a_2 \in S_a (a_1 \neq a_2)$ значення функції $f_r(a_1) \neq f_r(a_2)$, відповідно $F_{t+1,n}(a_1, X_{t+1,n}) = F_{t+1,n}(a_2, X_{t+1,n})$, для будь-яких $b, b_2 \in S_b (b_1 \neq b_2)$ $f_r(b) \neq f_r(b_2)$, відповідно $F_{t+1,n}(b, X_{t+1,n}) =$

$F_{t+1,n}(b_2, X_{t+1,n})$, а $F_{t+1,n}(a_1, X_{t+1,n}) \neq F_{t+1,n}(b_1, X_{t+1,n})$ $(f_r(a_1) \neq f_r(b_1))$. Згідно з наведеним, два кортежі логічних функцій будемо вважати однаковими, якщо виконуватимуться наступні умови:

- вони містять однакову кількість функцій;
- функції кортежу залежать від одних і тих самих змінних;
- функції з однаковими позиціями в кортежі рівні між собою.

Аналогічно позначимо $S_c$ і $S_d$ підмножини кодів аргументів із $X_{t+1,n}$, на які розбивається множина $S_{t+1,n}$ за значенням логічної функції $f_h(X_{t+1}, n)$. Шляхом декартового множення можна утворити наступні множини кодів аргументів із X: $S_{ca} = S_c \times S_a$, $S_{da} = S_d \times S_a, S_{cb} = S_c \times S_b$ та $S_{db} = S_d \times S_b$. Утворені множини попарно не мають спільних елементів, а їх об'єднання дорівнює $S_{1,n}$ − множині всіх кодів аргументів із X.

Парою підстановки будемо називати впорядковану пару кодів - коду аргументу із $S_{1,t}$ ( або $S_{t+1,n}$) та коду значень функцій (далі - коду функцій) із кортежу $F_{1,t}$ ( або $F_{t+1,n}$). Нехай $P_{ca}^{\wedge}$ − множина пар підстановки з кодами функцій із $F_{1,t}(X_{1,t}, c)$, коли аргументи із $X_{1,t}$ приймають всі можливі коди із множини $S_a$, а $T_{ca}^{\wedge}$ − множина пар підстановки із кодами функцій із $F_{t+1,n}(a, X_{t+1,n})$. Коли аргументи із $X_{t+1,n}$ приймають всі можливі коди із множини $S_C$. Аналогічно визначимо множини пар підстановки $P_{da}^{\wedge}, T_{da}^{\wedge}, P_{cb}^{\wedge}, T_{cb}^{\wedge}$ та $P_{db}^{\wedge}, T_{db}^{\wedge}$ Указані множини пар підстановки однозначно визначають повну підстановку та кортеж функцій F(X), які реалізує двомодульний каскад. Надалі множини $P^{\wedge}$ та $T^{\wedge}$ з однаковим комплектом літерних індексів будемо називати спорідненими.

Як показано у роботі [12], якщо кортеж функцій F(X) відтворює повну підстановку, то коди функцій пар підстановок, будуть різними, якщо вони містяться в одній і тій же множині пар підстановки. Це надає можливість сформувати множини $P_{ca}, T_{ca}, P_{da}, T_{da}, P_{cb}$, $T_{cb}$ та $P_{db}, T_{db}$ на базі множин пар підстановки $P_{ca}^{\wedge}, T_{ca}^{\wedge}, P_{da}^{\wedge}, T_{da}^{\wedge}, P_{cb}^{\wedge}, T_{cb}^{\wedge}$ та $P_{db}^{\wedge}, T_{db}^{\wedge}$ в яких елементами будуть лише коди функцій.

Для того щоб кортеж функцій, який реалізується простим двомодульним каскадом, відтворював повну підстановку, необхідно, щоб сукупність множини значень функцій P (або T) розбивались на дві пари таким чином, щоб множини, які належать до однієї пари не мали спільних елементів і були доповненням одна до одної в множині $S_{1,t}$ ( або $S_{t+1,n}$), тобто згідно з [12] забезпечувалась балансність та ортогональність логічних функцій із кортежів $F_{1,t}$ та $F_{t+1,n}$.

Можливі три варіанти розбиття чотирьох множин кодів функцій на дві пари, що необхідно для забезпечення балансності:

– перший варіант (позначимо його а) - $(P_{ca}, P_{cb}), (P_{da}, P_{db})$;
– другий варіант (позначимо його - б) - $(P_{Ca}, P_{da}), (P_{cb}, P_{db})$;
– третій варіант (позначимо його - в) $-(P_{ca}, P_{db}), (P_{da}, P_{cb})$.

У кожному із цих варіантів виділимо два випадки наявності спільних елементів, що важливо для забезпечення ортогональності. У першому з них серед чотирьох множин $P_{ca}, P_{cb}, P_{da}, P_{db}$ кодів функцій існують три множини, які попарно мають спільні елементи (тобто, обов'язково існує множина, яка має спільні елементи з двома іншими). Такі типи конструктивних модулів будемо позначати цифрою 3. В другому випадку серед множин $P_{ca}, P_{cb}, P_{da}, P_{db}$ попарно спільні елементи можуть мати тільки дві множини (тобто будь-яка з множин може мати спільні елементи лише з однією з інших). Такі типи конструктивних модулів будемо позначати цифрою 2.

Таким чином, можна виділити наступні типи конструктивних модулів простого двомодульного каскаду: 3а, 3б, 3в, 2а, 2б, 2в. Характеристики даних модулів було розглянуто в роботі [1]. Загальні властивості типів модулів двомодульного ОККМ наведені в табл. 2.1.

Таблиця 2.1 - Властивості типів модулів двомодульного ОККМ

| | Ознака входження множин в пару за значенням коду на бокових входах та виходах КМ | | | |
|---|---|---|---|---|
| Тип модуля | За однаковим кодом на вході | За однаковим кодом на виході | За різними кодами на вході й виході | Бокова функція |
| Тип 3а | Множини пари не мають спільних елементів | Множини пари можуть мати спільні елементи | Множини пари можуть мати спільні елементи | Довільна |
| Тип 3б | Множини пари можуть мати спільні елементи | Множини пари не мають спільних елементів | Множини пари можуть мати спільні елементи | Рівномірна |
| Тип 3в | Множини пари можуть мати спільні елементи | Множини пари можуть мати спільні елементи | Множини пари не мають спільних елементів | Довільна |
| Тип 2а | Множини пари не мають спільних елементів | Множини пари співпадають | Множини пари не мають спільних елементів | Довільна |
| Тип 2б | Множини пари не мають спільних елементів | Множини пари не мають спільних елементів | Множини пари співпадають | Рівномірна |
| Тип 2в | Множини пари співпадають | Множини пари не мають спільних елементів | Множини пари не мають спільних елементів | Рівномірна |

На основі наведених результатів розглянемо ітераційний процес формування n-розрядного ОККМ (рис. 2.6), створеного з найпростіших модулів за першим варіантом.

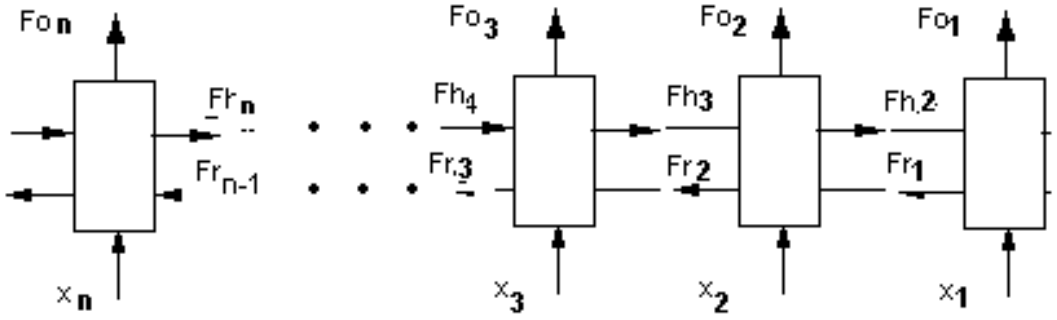


Рисунок 2.6 – Найпростіший багатомодульний ОККМ

Нехай $n = 1$, тоді відповідно до першого варіанту перший модуль ОККМ не має бокових виводів в сторону молодших розрядів. Функція на боковому виході першого КМ в сторону старших розрядів залежить від однієї змінної, тому можливі лише чотири варіанти формування множин $S_0$ та $S_1$:

- $S_0 = \{0\}, S_1 = \{1\}, f_{r1} = x$;
- $S_0 = \{1\}, S_1 = \{0\}, f_{r1} = \text{not } x$;
- $S_0 = \{0,1\}, S_1 = \emptyset, f_{r1} = 0$;
- $S_0 = \emptyset, S_1 = \{0,1\}, f_{r1} = 1$.

При конкретному значенні змінної на боковому вході першого КМ (0 або 1) на первинному виході модуля реалізується функція змінної x, яка в свою чергу може бути тотожньою, інверсією, константою - 0 або 1. Перелік всіх можливих варіантів поданий в таблиці 2.2, де $f_{01}(0, x)$ - функція на первинному виході при значенні 0 змінної на боковому вході, $f_{01}(1, x)$ - функція на первинному виході при значенні 1 змінної на боковому вході.

Таблиця 2.2 – Варіанти формування н-розрядного ОККМ за першим варіантом

| $f_{01}(0, x)$ | $f_{01}(1, x)$ | $f_{r1}(x)$ | Тип модуля |
|---|---|---|---|
| 0, 1 | 0, 1 | 0, 1 | - |
| 0 (1) | 0 (1) | x, $\overline{x}$ | - |
| 0 (1) | 1 (0) | x, $\overline{x}$ | 2в |
| 0, 1 | x, $\overline{x}$ | 0, 1 | - |
| 0, 1 | x, $\overline{x}$ | x, $\overline{x}$ | - |
| x, $\overline{x}$ | 0, 1 | 0, 1 | - |
| x, $\overline{x}$ | 0, 1 | x, $\overline{x}$ | - |
| x, $\overline{x}$ | x, $\overline{x}$ | 0, 1 | 2а1 |
| x($\overline{x}$) | x($\overline{x}$) | x, $\overline{x}$ | 2а2 |
| x($\overline{x}$) | ($\overline{x}$) x | x, $\overline{x}$ | 2б |

Із всіх можливих комбінацій функцій на боковому та первинному виході згідно з [12] необхідно виключити комбінації, в яких обидві функції є константами. Таким чином можливі лише наступні типи найпростіших КМ – 2а1, 2а2, 2б та 2в.

Відмітимо, що у випадку модуля типу 2а2 відсутня залежність функції на первинному виході від значення на боковому вході, а у випадку типу 2а1 – функція на боковому виході модуля є константою. Крім того, модулі типу 2в не реалізують підстановку при n=1. Такий тип модулів може бути використаний в наступних ітераціях. Визначимо перемикальні функції, які реалізують модулі відповідних типів:

- тип 2а1:
  - $f_{01}(f_{h2}, x_1) = x_1, f_{r1}(x_1) = 0$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1, f_{r1}(x_1) = 0$
  - $f_{01}(f_{h2}, x_1) = x_1 \oplus f_{h2}, f_{r1}(x_1) = 0$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1 \oplus f_{h2}, f_{r1}(x_1) = 0$
  - $f_{01}(f_{h2}, x_1) = x_1, f_{r1}(x_1) = 1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1, f_{r1}(x_1) = 1$
  - $f_{01}(f_{h2}, x_1) = x_1 \oplus f_{h2}, f_{r1}(x_1) = 1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1 \oplus f_{h2}, f_{r1}(x_1) = 1$
- тип 2а2:
  - $f_{01}(f_{h2}, x_1) = x_1, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = x_1, f_{r1}(x_1) = \text{not } x_1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1, f_{r1}(x_1) = \text{not } x_1$
- тип 2б:
  - $f_{01}(f_{h2}, x_1) = x_1 \oplus f_{h2}, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1 \oplus f_{h2}, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = x_1 \oplus f_{h2}, f_{r1}(x_1) = \text{not } x_1$
  - $f_{01}(f_{h2}, x_1) = \text{not } x_1 \oplus f_{h2}, f_{r1}(x_1) = \text{not } x_1$
- тип 2в:
  - $f_{01}(f_{h2}, x_1) = f_{h2}, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = \text{not } f_{h2}, f_{r1}(x_1) = x_1$
  - $f_{01}(f_{h2}, x_1) = f_{h2}, f_{r1}(x_1) = \text{not } x_1$

- $f_{01}(f_{h2}, x_1) = \text{not } f_{h2}, f_{r1}(x_1) = \text{not } x_1$

Варто зазначити, що кількість модулів кожного з типів повністю узгоджується з теоретичними розрахунками, наведеними у роботі [1].

Розглянемо другий варіант побудови ОККМ. Нехай $n = 2$, перша ітерація. Розглянемо логічні функції найпростішого КМ при його підключенні до ОККМ $-f_{02}(f_{h3}, x_2, f_{r1}), f_{h2}(f_{h3}, x_2), f_{r2}(x_2, f_{r1})$. Якщо зафіксувати конкретне значення $f_{h3}$, то фактично отримаємо старший модуль двомодульного каскаду. Таким чином, найпростіший КМ має два типи в сторону молодших розрядів, які позначимо $L_0$ та $L_1$. Оскільки функції $fo_2(0, x_2, f_{r1})$, та $fo_2(1, x_2, f_{r1})$, (відповідно $f_{h2}(0, x_2)$, та $f_{h2}(1, x_2)\big)$ не залежні одна від одної, то як $L_0$, так і $L_1$ можуть бути будь-якими із наведених вище 12 типів. Для реалізації повної підстановки, згідно з [1], необхідно і достатньо, щоб будь-який із типів $L_0$ та $L_1$ був сумісний із типом першого модуля. При цьому функція $f_{r2}(x_2, f_{r1})$ може бути довільною. Варто зазначити, що двомодульний каскад реалізує дві повні підстановки (при $f_{h3} = 0$, та $f_{h3} = 1$). Характер функції $f_{r2}(x_2, f_{r1})$ має значення при визначенні типу «об'єднаного КМ» із першого та другого модулів. Очевидно, що тип може бути лише 2а, 3а та 2б.

Друга та наступні ітерації виконуються аналогічно. Загалом можливим типом «об'єднаного КМ» є тип 3а, який обмежує можливість підключення наступних КМ лише типами 2а, тому важливим є визначення умов, при виконанні яких «об'єднаний КМ» буде мати типи 2а та 2б. Напрямок вирішення цієї задачі полягає у наступному. Кожна з двох підстановок «об'єднаного КМ» є в свою чергу деякою перестановкою елементів із множини $S_{1,t}$. Дві перестановки в свою чергу утворюють підстановку, яка характеризується своїми циклами. Тому, якщо функція $fr_i$ на $(i-1)$-ій ітерації приймає однакові значення в межах будь-якого циклу, то «об'єднаний КМ» буде належати до типу 2а. Якщо кількість елементів будь-якого циклу парна, а функція $f_{ri}$ на ( i - 1)-ій ітерації приймає різні значення на половині елементів будь-якого циклу, то «об'єднаний КМ» буде належати до типу 2б.

На відміну від попереднього варіанту, функція $f_{r2}(x_2, f_{r1})$ другого варіанту побудови ОККМ не може бути довільною. При фіксуванні значень $f_{ri}$ отримаємо типи в сторону старших розрядів - $H_0$ та $H_1$.

КМ в цьому випадку характеризується чотирма типами - $H_0$, $H_1$, $L_0$, $L_1$, наприклад 2а1, 2а2, 2б, 2а1. В цьому плані ОККМ у цілому може розглядатися як один модуль зі своїми правосторонніми та лівосторонніми типами. Новий модуль може підключатися ло каскаду як зі сторони молодших розрядів, так і зі сторони старших розрядів.

При підключенні модуля до другого модуля або до каскада кожний лівосторонній тип модуля має бути сумісним 3 кожним правостороннім типом попереднього модуля чи каскаду. Згідно з [1] це є необхідною і достатньою умовою для відтворення новим ОККМ повних підстановок при будь-яких значеннях на правому боковому вході першого модуля та лівому боковому вході останнього модуля.

Шляхом повного перебору всіх можливих функцій (як на первинних, так і на бокових виходах) було доведено існування 75 різних комбінацій правосторонніх та лівосторонніх типів найпростіших ОККМ, які можуть бути використані для реалізації повної підстановки. При побудові найпростіших ОККМ із вищерозглянутих модулів необхідно дотримуватись сумісності типів. Згідно з попередніми твердженнями сумісність типів забезпечує реалізацію повних підстановок за допомогою каскаду найпростіших ОККМ.

У загальному випадку кожний КМ каскаду на первинних виходах реалізує m булевих функцій, які представляють m поточних розрядів підстановки. На первинних виходах КМ булеві функції залежать від:

- m булевих змінних, які репрезентують m поточних розрядів аргументу підстановки та подаються на первинні входи КМ;
- $k_r$ булевих змінних, які подаються на $k_r$ правих бокових входів КМ;
- $k_l$ булевих змінних, які подаються на $k_l$ лівих бокових входів КМ.

Окрім того, на лівих бокових виходах кожний КМ реалізує $k_r$ булевих функцій, які залежать від m первинних булевих змінних та $k_r$ змінних на правих бокових входах. На правих бокових виходах КМ реалізуються $k_l$ булевих функцій, які залежать від первинних булевих змінних та $k_{ll}$ змінних на лівих бокових входах.

Якщо всі КМ каскаду однакові, то такий каскад називають регулярним. Якщо $k_r = k_l = 1$, то ОККМ називають, простим. Якщо в простому ОККМ для всіх КМ m = 1, то такий каскад називають найпростішим. Якщо $k_r \neq 0$ та $k_l \neq 0$, то такий ОККМ називають двонаправленим. Якщо $k_r = 0$ або $k_l = 0$, то такий ОККМ називають однонаправленим, і відповідно якщо $k_r = k_l = 0$, то такий каскад називають тривіальним [1]. Зауважимо, що в алгоритмах симетричних криптографічних перетворень DES, AES, ГОСТ 28147-89 для формування багаторозрядних повних підстановок фактично використовуються наступні ОККМ:

- тривіальні регулярні (m = 8, AES);
- тривіальні нерегулярні (m = 4, ГОСТ);
- прості двонаправлені нерегулярні ОККМ (m = 4, DES).

В [1] на основі аналізу каскаду із двох КМ були визначені властивості КМ, які необхідні і достатні для реалізації каскадом повної підстановки та визначено 6 типів структур КМ позначених як 2а, 3а, 2б, 3б, 2в та 3в. В [3] проаналізовані методи реалізації багаторозрядних повних підстановок на простих ОККМ як ітеративного процесу, де на кожному кроці ітерації ОККМ доповнюється одним КМ. Було показано, що для реалізації багаторозрядних повних підстановок на кожному кроці ітерації достатньо, щоб КМ який підключається до ОККМ, мав тип (2а,2а) у відповідний бік. Далі проводиться дослідження структур найпростіших регулярних двонаправлених ОККМ з метою встановлення кількісних характеристик підстановок, які реалізуються такими ОККМ.

Наступним кроком розглянемо двонаправлений ОККМ [10]. Значення сигналу на первинному виході КМ $f_o$ залежить від значень сигналів на первинному вході x та на лівому та правому бокових входах h, r. Значення сигналу на лівому

боковому виході $f_r$ залежить від значення первинної змінної x та від значення сигналу на правому боковому вході r. Відповідно значення на правому боковому виході $f_h$ залежить від значення первинної змінної x та від значення на лівому боковому вході h. Оскільки всі КМ каскаду є однаковими, то для опису кожного КМ і, як наслідок, каскаду в цілому, необхідно застосовувати одну логічну функцію трьох змінних $f_o(x,h,r)$ та дві логічні функції двох змінних $f_r(x,r)$ та $f_h(x,h)$. Таким чином, кількість різних КМ і, відповідно ОККМ, дорівнює $2^{16}$, а кількість різних повних підстановок – не більшою за $2^{18}$ .

В залежності від комбінації функцій на первинному та бокових виходах КМ характеризується комбінацією лівих та правих типів, які визначають його властивості щодо реалізації на ОККМ повних підстановок. В табл. 2.3 наведені можливі типи КМ при m=1, де використані наступні позначення:

- 0, 1 – функції є будь-якими константами (або 0, або 1);
- x, $\overline{x}$ - функції є будь- якими не константами (або x, або $\overline{x}$);
- 0 (1) 0 (1) – функції є одинакові константи;
- 0 (1) 1 (0) – функції є різні константи;
- x ($\overline{x}$) x () - функції є одинакові не константи;
- x ($\overline{x}$) $\overline{x}$ (x) - функції є різні не константи.

Таблиця 2.3 - Можливі типи КМ при m=1

| $f_o$ (x,0,0) | $f_o$ (x,1,0) | $f_r$ (x,0) | ЛТ0 |
|---|---|---|---|
| $f_o$ (x,0,1) | $f_o$ (x,1,1) | $f_r$ (x,1) | ЛТ1 |
| $f_o$ (x,0,0) | $f_o$ (x,0,1) | $f_h$ (x,0) | ПТ0 |
| $f_o$ (x,1,0) | $f_o$ (x,1,1) | $f_h$ (x,1) | ПТ1 |
| 1 | 2 | 3 | 4 |
| 0, 1 | 0, 1 | 0, 1 | - |
| 0 (1) | 0 (1) | x, $\overline{x}$ | - |
| 0 (1) | 1 (0) | x, $\overline{x}$ | Тип 2в |
| 0, 1 | x, $\overline{x}$ | 0, 1 | - |
| 0, 1 | x, $\overline{x}$ | x, $\overline{x}$ | - |
| x, $\overline{x}$ | 0, 1 | 0, 1 | - |
| x, $\overline{x}$ | 0, 1 | x, $\overline{x}$ | - |
| x, $\overline{x}$ | x, $\overline{x}$ | 0, 1 | Тип 2а1 |

Продовження таблиці 2.3

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| x($\overline{x}$) | x($\overline{x}$) | x, $\overline{x}$ | Тип 2а2 |
| x($\overline{x}$) | $\overline{x}$(x) | x, $\overline{x}$ | Тип 2б |

Отже конструктивні модулі можна позначать наступним чином - (ЛТ0, ЛТ1, ПТ0 ПТ1), де:

- ЛТ0 - лівосторонній тип, за умови, що на правий боковий вхід подається 0;
- ЛТ1 - лівосторонній тип, за умови, що на правий боковий вхід подається 1;
- ПТ0 - правосторонній тип, за умови, що на лівий боковий вхід подається 0;
- ПТ1 – правосторонній тип, за умови, що на лівий боковий вхід подається 1, наприклад (2а1, 2а2, 2б, 2а1).

Модулі найпростіших ОККМ (m = 1) мають наступні властивості:

- якщо один з типів 2в, то всі інші – також 2в;
- із можливих 34 комбінацій типів 2а1,2а2 та 2б не існують наступні 8 комбінацій типів:
  - (2а2,2а2,2а2,2б),
  - (2а2,2а2,2б,2а2),
  - (2а2,2б,2а2,2а2),
  - (2б,2а2,2а2,2а2),
  - (2б,2б,2б,2а2),
  - (2б,2б,2а2,2б),
  - (2б,2а2,2б,2б),
  - (2а2,2б,2б,2б)

Таким чином, кількість різних функцій $f_o(x,r,h)$ зменшилась до 10, оскільки при двох із них реалізуються тривіальні підстановки (функція $f_o(x,r,h)$ не залежить від r та h), а при 4-х – реалізуються підстановки однонаправленого каскаду (функція

$f_o(x,r,h)$ не залежить від r або h). Згідно зі сказаним попередньо для реалізації повних підстановок на двонаправленому простому каскаді достатньо, щоб кожний КМ мав тип (2а, 2а) принаймні в один бік. Це означає, що при обох значеннях змінної на боковому вході r (або h) КМ повинен реалізовувати повні підстановки на множині $Q_m$ всіх можливих значень булевих змінних. При m=1 таких підстановок лише дві – x та not x. Враховуючи значення бокових змінних, виходить, що такі КМ можуть реалізувати 16 різних первинних функцій, що представлені в табл. 2.4.

Таблиця 2.4 - Перелік первинних функцій КМ при m=1

| № п/п | Повна ДНФ $f_o(x,r,h)$ | Можлива мінімізація | Коментар |
|---|---|---|---|
| 1 | $(\overline{x}\wedge\overline{h}\wedge\overline{r})\vee(\overline{x}\wedge\overline{h}\wedge r)\vee(\overline{x}\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $\overline{x}$ | Тривіальна підстановка |
| 2 | $(x\wedge\overline{h}\wedge\overline{r})\vee(x\wedge\overline{h}\wedge r)\vee(x\wedge h\wedge\overline{r})\vee(x\wedge h\wedge r)$ | x | Тривіальна підстановка |
| 3 | $(x\wedge\overline{h}\wedge\overline{r})\vee(x\wedge\overline{h}\wedge r)\vee(\overline{x}\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $(x\wedge\overline{h})\vee(\overline{x}\wedge h)$ | Не залежить від r – односторонній каскад |
| 4 | $(\overline{x}\wedge\overline{h}\wedge\overline{r})\vee(\overline{x}\wedge\overline{h}\wedge r)\vee(x\wedge h\wedge\overline{r})\vee(x\wedge h\wedge r)$ | $(x\wedge h)\vee(\overline{x}\wedge\overline{h})$ | Не залежить від r – односторонній каскад |
| 5 | $(x\wedge\overline{h}\wedge\overline{r})\vee(\overline{x}\wedge\overline{h}\wedge r)\vee(x\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $(x\wedge\overline{r})\vee(\overline{x}\wedge r)$ | Не залежить від h – односторонній каскад |
| 6 | $(\overline{x}\wedge\overline{h}\wedge\overline{r})\vee(x\wedge\overline{h}\wedge r)\vee(\overline{x}\wedge h\wedge\overline{r})\vee(x\wedge h\wedge r)$ | $(x\wedge r)\vee(\overline{x}\wedge\overline{r})$ | Не залежить від h – односторонній каскад |
| 7 | $(x\wedge\overline{h}\wedge\overline{r})\vee(\overline{x}\wedge\overline{h}\wedge r)\vee(\overline{x}\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $x+(\overline{h}\wedge\overline{r})+1$ | Спряжена з функціями 8, 9, 10 |
| 8 | $(\overline{x}\wedge\overline{h}\wedge\overline{r})\vee(x\wedge\overline{h}\wedge r)\vee(\overline{x}\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $x+(\overline{h}\wedge r)+1$ | Спряжена з функціями 7, 9, 10 |
| 9 | $(\overline{x}\wedge\overline{h}\wedge\overline{r})\vee(\overline{x}\wedge\overline{h}\wedge r)\vee(x\wedge h\wedge\overline{r})\vee(\overline{x}\wedge h\wedge r)$ | $x+(h\wedge\overline{r})+1$ | Спряжена з функціями 7, 8, 10 |

Продовження таблиці 2.4

| № п/п | Повна ДНФ $f_o(x,r,h)$ | Можлива мінімізація | Коментар |
|---|---|---|---|
| 10 | $(\bar{x}\wedge\bar{h}\wedge\bar{r})\vee(\bar{x}\wedge\bar{h}\wedge r)\vee(\bar{x}\wedge h\wedge\bar{r})\vee(x\wedge h\wedge r)$ | $x+(h\wedge r)+1$ | Спряжена з функціями 7, 8, 9 |
| 11 | $(\bar{x}\wedge\bar{h}\wedge\bar{r})\vee(x\wedge\bar{h}\wedge r)\vee(x\wedge h\wedge\bar{r})\vee(x\wedge h\wedge r)$ | $x+(\bar{h}\wedge\bar{r})$ | Спряжена з функціями 12, 13, 14 |
| 12 | $(x\wedge\bar{h}\wedge\bar{r})\vee(\bar{x}\wedge\bar{h}\wedge r)\vee(x\wedge h\wedge\bar{r})\vee(x\wedge h\wedge r)$ | $x+(\bar{h}\wedge r)$ | Спряжена з функціями 11, 13, 14 |
| 13 | $(x\wedge\bar{h}\wedge\bar{r})\vee(x\wedge\bar{h}\wedge r)\vee(\bar{x}\wedge h\wedge\bar{r})\vee(x\wedge h\wedge r)$ | $x+(h\wedge\bar{r})$ | Спряжена з функціями 11, 12, 14 |
| 14 | $(x\wedge\bar{h}\wedge\bar{r})\vee(x\wedge\bar{h}\wedge r)\vee(x\wedge h\wedge\bar{r})\vee(\bar{x}\wedge h\wedge r)$ | $x+(h\wedge r)$ | Спряжена з функціями 11, 12, 13 |
| 15. | $(x\wedge\bar{h}\wedge\bar{r})\vee(\bar{x}\wedge\bar{h}\wedge r)\vee(\bar{x}\wedge h\wedge\bar{r})\vee(x\wedge h\wedge r)$ | $x+h+r$ | Спряжена з функціями 15 та 16 |
| 16 | $(\bar{x}\wedge\bar{h}\wedge\bar{r})\vee(x\wedge\bar{h}\wedge r)\vee(x\wedge h\wedge\bar{r})\vee(\bar{x}\wedge h\wedge r)$ | $x+h+r+1$ | Спряжена з функціями 15 та 16 |

Подальше уточнення кількості різних підстановок полягає в обмежені можливих комбінацій бокових функцій та значень бокових змінних. Такі обмеження, в першу чергу, полягають у забезпеченні реалізації повних перестановок при будь-якій кількості КМ в ОККМ, що безумовно може мати місце лише за наявності або обох лівосторонніх або обох правосторонніх типів 2а. У таблицях 2.5 та 2.6 показано всі можливі типи для КМ з вибраними функціями $f_o(x,r,h)$ і виділено КМ, які не мають вказаної властивості.

Таблиця 2.5 - Можливі типи КМ з первинними функціями, які належать до першого та другого класів

| $f_h(0, x)$ | $f_h(1, x)$ | $f_r(0, x)$ | $f_r(1, x)$ | Типи КМ при $f_o(x,r,h)= x+(h\wedge r)$, або $f_o(x,r,h)= x+(h\wedge r)+1$ |
|---|---|---|---|---|
| const | const | const | const | 2a1, 2a1, 2a1, 2a1 |

Продовження таблиці 2.5

| $f_h(0, x)$ | $f_h(1, x)$ | $f_r(0, x)$ | $f_r(1, x)$ | Типи КМ при $f_o(x,r,h)= x+(h\wedge r)$, або $f_o(x,r,h)= x+(h\wedge r)+1$ |
|---|---|---|---|---|
| const | const | const | x, $\overline{x}$ | 2a1, 2a1, 2a1, 2б |
| const | const | x, $\overline{x}$ | const | 2a1, 2a1, 2a2, 2a1 |
| const | const | x, $\overline{x}$ | x, $\overline{x}$ | 2a1, 2a1, 2a2, 2б |
| const | x, $\overline{x}$ | const | const | 2a1, 2б, 2a1, 2a1 |
| const | x, $\overline{x}$ | const | x, $\overline{x}$ | 2a1, 2б, 2a1, 2б |
| const | x, $\overline{x}$ | x, $\overline{x}$ | const | 2a1, 2б, 2a2, 2a1 |
| const | x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | 2a1, 2б, 2a2, 2б |
| x, $\overline{x}$ | const | const | const | 2a2, 2a1, 2a1, 2a1 |
| x, $\overline{x}$ | const | const | x, $\overline{x}$ | 2a2, 2a1, 2a1, 2б |
| x, $\overline{x}$ | const | x, $\overline{x}$ | const | 2a2, 2a1, 2a2, 2a1 |
| x, $\overline{x}$ | const | x, $\overline{x}$ | x, $\overline{x}$ | 2a1, 2a1, 2a2, 2б |
| x, $\overline{x}$ | x, $\overline{x}$ | const | const | 2a2, 2б, 2a1, 2a1 |
| x, $\overline{x}$ | x, $\overline{x}$ | const | x, $\overline{x}$ | 2a2, 2б, 2a1, 2б |
| x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | const | 2a2, 2б, 2a2, 2a1 |
| x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | 2a2, 2б, 2a2, 2б |

Таблиця 2.6 - Можливі типи КМ з первинними функціями,

які належать до третього класу.

| $f_h(0, x)$ | $f_h(1, x)$ | $f_r(0, x)$ | $f_r(1, x)$ | Типи КМ при $f_o(x,r,h)= x+h+r$ |
|---|---|---|---|---|
| const | const | const | const | 2a1, 2a1, 2a1, 2a1 |
| const | const | const | x, $\overline{x}$ | 2a1, 2a1, 2a1, 2б |
| const | const | x, $\overline{x}$ | const | 2a1, 2a1, 2б, 2a1 |
| const | const | x, $\overline{x}$ | x, $\overline{x}$ | 2a1, 2a1, 2б, 2б |
| const | x, $\overline{x}$ | const | const | 2a1, 2б, 2a1, 2a1 |
| const | x, $\overline{x}$ | const | x, $\overline{x}$ | 2a1, 2б, 2a1, 2б |
| const | x, $\overline{x}$ | x, $\overline{x}$ | const | 2a1, 2б, 2б, 2a1 |
| const | x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | 2a1, 2б, 2б, 2б |
| x, $\overline{x}$ | const | const | const | 2б, 2a1, 2a1, 2a1 |
| x, $\overline{x}$ | const | const | x, $\overline{x}$ | 2б, 2a1, 2a1, 2б |
| x, $\overline{x}$ | const | x, $\overline{x}$ | const | 2б, 2a1, 2б, 2a1 |
| x, $\overline{x}$ | const | x, $\overline{x}$ | x, $\overline{x}$ | 2б, 2a1, 2б, 2б |
| x, $\overline{x}$ | x, $\overline{x}$ | const | const | 2б, 2б, 2a1, 2a1 |
| x, $\overline{x}$ | x, $\overline{x}$ | const | x, $\overline{x}$ | 2б, 2б, 2a1, 2б |
| x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | const | 2б, 2б, 2б, 2a1 |
| x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | x, $\overline{x}$ | 2б, 2б, 2б, 2б |

Варто зазначити, що до цієї групи належать підстановки, які реалізуються на таких ОККМ, в яких принаймні в одному із напрямів, значення на бокових виходах КМ незмінне, і дорівнює значенню змінної налагодження, наприклад, при $f_r(x,r) = r$. До другої групи належить 190 підстановок, які реалізуються при умові, що в відповідних ОККМ, принаймні в одному із напрямів, значення на бокових виходах КМ незмінне та дорівнює протилежному значенню змінної налагодження. До третьої групи належить 576 підстановок, які можна вважати дійсними підстановками двонаправлених каскадів.

Експериментальні дослідження також показали існування КМ, при з'єднанні яких в ОККМ повні підстановки реалізуються тільки при відповідних значеннях розрядності. Це стосується КМ з типами (2б, 2б, 2б, 2б) або (2в, 2в, 2в, 2в). Так ОККМ на КМ з $f_o(x,r,h)= x+h+r$ і $f_r(x,r) = x$ $f_h(x,h)= x$ або $f_h(x,h)=$not x реалізує повні підстановки при значенні n, які відповідають елементам наступних арифметичних прогресій - $n=3+3\times j$ або $n=4+3\times j$ ($j=0,1,\ldots$), що додає ще 8 оригінальних повних підстановок з указаною розрядністю. КМ з типами (2в,2в,2в,2в) дають можливість реалізовувати ще 28 повних оригінальних підстановок при парних значеннях n або при значеннях n, які відповідають елементам наступних арифметичних прогресій - $n=2+3\times j$ та $3+3\times j$. Загальна кількість різних КМ та відповідно двонаправлених ОККМ, за допомогою яких реалізуються вказані підстановки дорівнює 350.

## 2.2 Огляд алгоритму та його реалізації для формування конструктивних модулів

Реалізація автоматом (відповідно у нашому випадку ОККМ) підстановок означає реалізацію бієктивного відображення ($\dot{X}_r \rightarrow \dot{Y}_r$) при будь якому значенні r, тобто різним послідовностям $\dot{X}_r$ відповідають різні послідовності $\dot{Y}_r$.

Для усунення двозначностей підстановки довільної розрядності (у нашому випадку, бієктивні відображення), які реалізуються автоматом (або ОККМ) в подальшому будемо позначати $F(\dot{X}_r)$ .

Якщо дві послідовності символів відрізняються принаймні одним символом, вони вважаються різними. Якщо автоматне відображення не є бієктивним при деякому значенні r, то воно буде не бієктивним при будь-яких $r_1 > r$.

Станом з втратами називають стан $s_l$ такий, для якого існують $x_1, x_2 \in X$, де $x_1 \neq x_2$ і $f_o(s_l, x_1) = f_o(s_l, x_2)$. З урахуванням цього наводиться наступна теорема.

Теорема 1. Автоматне відображення $f_A: \dot{X}_r \to \dot{Y}_r$ бієктивне на $\dot{X}$ тоді і тільки тоді, коли автомат A не містить станів з втратами, які досягаються з початкового стану $s_0$ за r кроків [13][14].

Теорема 2. Для того, щоб ОККМ реалізував підстановку $F(\dot{X}_r)$ необхідно і достатньо, щоб при будь-якому сигналі на бокових входах КМ, на первинних (не бокових) виходах комбінаційна схема реалізувала будь-яку підстановку значень сигналів на первинних входах.

Розрядність підстановки $F(\dot{X}_r)$ дорівнює $b = r \log n$. Степінь підстановки у такому випадку дорівнює $n^r$.

Нехай $p(s)$ – деяка перестановка елементів множини S, а $p^{-1}(s)$ – обернена перестановка, тобто $p(p^{-1}(s)) = p^{-1}(p(s)) = s$.

Теорема 3. Якщо для будь-якого автомату A створити автомат $A_1$ з тими самими алфавітами X, Y, S і з наступними функціями $v_o(x, s) = f_o(x, p(s))$, $v_s(x, s) = p^{-1}(f_s(x, p(s)))$, та початковим станом $s_{0_A} = p^{-1}(s_0)$, то автомати A та $A_1$ будуть еквівалентними.

Доведення. Нехай на вхід автоматів A та $A_1$ подається символ x. Автомат $A_1$ формує символ виходу $v_o(x, s) = f_o(x, p(s))$, та стан $v_s(x, s) = p^{-1}(f_s(x, p(s)))$. На першому такті початковий стан автомата $A_1$ – $s_{0_A} = p^{-1}(s_0)$. Маємо,

$$v_o(x, s) = f_o\big(x, p(s)\big) = f_o\left(x, p\big(s_{0_A}\big)\right) = f_o\big(x, p^{-1}(s_0)\big) = f_o(x, s) \qquad (1)$$

і при цьому автомат $A_1$ перейде в стан,

$$p^{-1}\left(f_s\left(x, p\left(s_{0_A}\right)\right)\right) = p^{-1}\left(f_s\left(x, p\left(p^{-1}(s_0)\right)\right)\right) = p^{-1}\left(f_s(x, s_0)\right) \qquad (2)$$

Для будь-якої кількості наступних вхідних символів аналогічно. Таким чином обидва автомати реалізують одне і те ж відображення, тобто є еквівалентними [11].

Таким чином, в класі ОККМ що розглядається, існує m! ОККМ, які реалізують одну і ту ж підстановку $F(\dot{X}_r)$, що важливо при оцінках кількості різних підстановок.

У загальному випадку кожний КМ каскаду на первинних виходах реалізує m булевих функцій, які представляють m поточних розрядів підстановки. Базовий алгоритм для формування таблиць виходів полягає в наступному. Вважається, що рядки таблиці виходів позначені станами. З множини виходів генерується випадковий символ, за допомогою генератора випадкових чисел. З генератора одержується черговий вихідний символ. Відбувається пошук рядка таблиці, де цей символ відсутній і символ додається в даний рядок. Якщо одержаний з генератора символ існує у всіх рядках, то він пропускається. Алгоритм завершується після заповнення всіх рядків. Для формування таблиці виходів використовується наступний базовий алгоритм. Знову за допомогою генератора випадкових чисел, для генерації символу з множини станів. Одержані стани по черзі записуються у вільні комірки таблиці станів. Алгоритм завершується після заповнення всіх клітинок таблиці станів. Якщо необхідно отримати приведений автомат, то виконується необхідний аналіз. У випадку, якщо автомат не приведений, то формування таблиці переходів повторюється. Реалізацію алгоритмів для генерування таблиць переходів і виходів прямого відображення мовою C# наведено у фрагменті лістингу програми 2.1.

```
for (int i = 0; i < M; i++)
        {
                StateMatrix.Add(new List<int>());
                OutputMatrix.Add(new List<int>());
                for (int j = 0; j < N; j++)
                        {
                                StateMatrix[i].Add(_random.Next(M));
                        }

                 while (OutputMatrix[i].Count < N)
                        {
                                int y = _random.Next(N);
                                 if (!OutputMatrix[i].Contains(y))
                                        {
                                                OutputMatrix[i].Add(y);
                                        }
                        }
        }
```

Фрагмент лістингу програми 2.1 – Реалізація алгоритмів формування таблиць виходів та станів автомата

Розглянемо приклад автомата для m=12 та n=8. В комірці таблиці виходів (табл. 2.7) знаходиться один із символів вихідного алгоритму.

Таблиця 2.7 – Таблиця виходів автомата для реалізації $F(\dot{X}_r)$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $y_1$ | $y_4$ | $y_6$ | $y_7$ | $y_2$ | $y_5$ | $y_3$ | $y_8$ |
| $s_2$ | $y_6$ | $y_3$ | $y_4$ | $y_8$ | $y_5$ | $y_7$ | $y_2$ | $y_1$ |
| $s_3$ | $y_4$ | $y_8$ | $y_3$ | $y_6$ | $y_2$ | $y_5$ | $y_1$ | $y_7$ |
| $s_4$ | $y_1$ | $y_3$ | $y_8$ | $y_4$ | $y_5$ | $y_7$ | $y_6$ | $y_2$ |
| $s_5$ | $y_1$ | $y_7$ | $y_6$ | $y_3$ | $y_2$ | $y_5$ | $y_4$ | $y_8$ |
| $s_6$ | $y_3$ | $y_1$ | $y_6$ | $y_5$ | $y_4$ | $y_8$ | $y_7$ | $y_2$ |
| $s_7$ | $y_4$ | $y_7$ | $y_3$ | $y_1$ | $y_6$ | $y_2$ | $y_5$ | $y_8$ |
| $s_8$ | $y_4$ | $y_1$ | $y_8$ | $y_5$ | $y_2$ | $y_6$ | $y_3$ | $y_7$ |
| $s_9$ | $y_5$ | $y_7$ | $y_4$ | $y_8$ | $y_2$ | $y_3$ | $y_6$ | $y_1$ |
| $s_{10}$ | $y_4$ | $y_1$ | $y_7$ | $y_8$ | $y_5$ | $y_2$ | $y_3$ | $y_6$ |
| $s_{11}$ | $y_6$ | $y_3$ | $y_2$ | $y_1$ | $y_5$ | $y_4$ | $y_8$ | $y_7$ |
| $s_{12}$ | $y_6$ | $y_2$ | $y_7$ | $y_1$ | $y_3$ | $y_8$ | $y_5$ | $y_4$ |

Відповідно у кожній комірці таблиці функції переходів (табл. 2.8) знаходиться один із станів.

Таблиця 2.8 – Таблиця переходів автомата для реалізації $F(\dot{X}_r)$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $s_1$ | $s_{12}$ | $s_5$ | $s_{11}$ | $s_4$ | $s_{11}$ | $s_5$ | $s_4$ |
| $s_2$ | $s_7$ | $s_{11}$ | $s_4$ | $s_4$ | $s_8$ | $s_{12}$ | $s_8$ | $s_3$ |
| $s_3$ | $s_{10}$ | $s_{12}$ | $s_1$ | $s_7$ | $s_{10}$ | $s_5$ | $s_9$ | $s_6$ |
| $s_4$ | $s_6$ | $s_7$ | $s_1$ | $s_5$ | $s_8$ | $s_8$ | $s_{10}$ | $s_7$ |
| $s_5$ | $s_{11}$ | $s_{12}$ | $s_5$ | $s_3$ | $s_4$ | $s_7$ | $s_{10}$ | $s_4$ |
| $s_6$ | $s_9$ | $s_7$ | $s_3$ | $s_9$ | $s_3$ | $s_6$ | $s_3$ | $s_6$ |
| $s_7$ | $s_8$ | $s_8$ | $s_8$ | $s_9$ | $s_{11}$ | $s_7$ | $s_7$ | $s_1$ |
| $s_8$ | $s_{10}$ | $s_2$ | $s_{11}$ | $s_5$ | $s_6$ | $s_1$ | $s_4$ | $s_{12}$ |
| $s_9$ | $s_7$ | $s_2$ | $s_8$ | $s_6$ | $s_2$ | $s_{12}$ | $s_{12}$ | $s_{10}$ |
| $s_{10}$ | $s_9$ | $s_{10}$ | $s_{12}$ | $s_{11}$ | $s_{12}$ | $s_7$ | $s_2$ | $s_9$ |
| $s_{11}$ | $s_9$ | $s_{11}$ | $s_{12}$ | $s_1$ | $s_{10}$ | $s_9$ | $s_1$ | $s_{11}$ |
| $s_{12}$ | $s_4$ | $s_{12}$ | $s_9$ | $s_4$ | $s_5$ | $s_6$ | $s_5$ | $s_5$ |

Нумерація рядків таблиць переходів та виходів здійснюється зверху вниз починаючи з умовного позначення (номера) першого стану і закінчуючи останнім. Нумерація стовпчиків таблиць переходів та виходів здійснюється аналогічно, зліва направо починаючи з першого символу вхідного алфавіту і відповідно закінчуючи останнім.

У наступному фрагменті лістингу програми 2.2 наведено реалізацію відображень $\dot{X}_r \to \dot{Y}_r$ мовою C#.

```
int state = InitialState;
var outputs = new List<int>();
for (int i = 0; i < r; i++)
{
        outputs.Add(OutputMatrix[state][input[i]]);
        state = StateMatrix[state][input[i]];
```

```
}
return outputs;
```

Фрагмент лістингу програми 2.2 – Реалізація алгоритму відображень $\dot{X}_r \rightarrow \dot{Y}_r$ мовою C#

Розглянемо більш детально термін «бієкція». Технічно, перестановка множини S визначається як бієкція від S до себе [15], [16]. Тобто це функція від S до S, для якої кожен елемент зустрічається рівно один раз як значення зображення. Це пов’язано з перегрупуванням елементів S, в якому кожен елемент s замінюється відповідним f(s).

У математиці бієкція, або бієктивною функцією [17], називають як відповідність один до одного або як зворотна функція — це функція між елементами двох наборів, де кожен елемент однієї множини поєднується точно з одним елементом іншої множини, і кожен елемент іншого набору в парі точно з одним елементом першого набору. Непарних елементів немає. З точки зору математики, бієктивна функція f: X → Y є взаємним (ін’єктивним і сур’єктивним) відображенням множини X на множину Y.

На рисунках 2.7 зображено приклад відображення, яке є лише ін’єктивним.

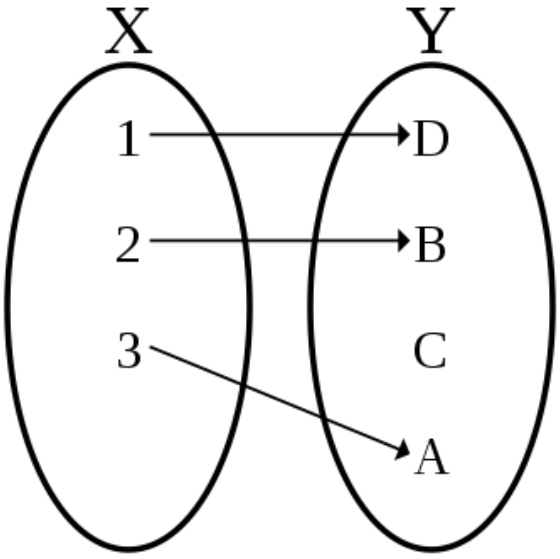


Рисунок 2.7 – Приклад ін’єктивного відображення

Відповідно на наступному рисунку 2.8 зображено бієктивне відображення, яке є ін’єктивним та сур’єктвиним водночас.

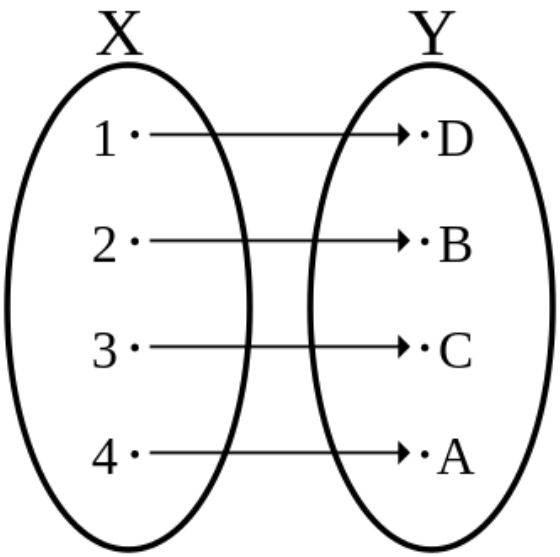


Рисунок 2.8 – Приклад бієктивного відображення (одночасно сюр'єктивне та ін'єктивне)

Бієкція від множини X до множини Y має функцію, обернену від Y до X. Якщо X і Y скінченні множини, то існування бієкції означає, що вони мають однакову кількість елементів. Для нескінченних множин картина складніша, що призводить до поняття кардинального числа – способу розрізняти різні розміри нескінченних множин.

Бієктивна функція від множини до себе також називається перестановкою, а множина всіх перестановок цієї множини, у свою чергу, утворює симетричну групу.

Бієктивні функції важливі для багатьох областей математики, включаючи визначення ізоморфізму, гомеоморфізму, дифеоморфізму, груп перестановок і проективних карт.

Функція f: R → R є бієктивною тоді і тільки тоді, коли її графік перетинає кожну горизонтальну та вертикальну прямі рівно один раз. Якщо X є множиною, то бієктивні функції від X до себе, разом з операцією функціональної композиції (),

утворюють групу, симетричну групу X, яка позначається по-різному S (X), SX або X! (X факторіал). Бієкції зберігають ступінь множин: для підмножини A області зі степенем | A | і підмножина B піддомену з потужністю | B | мають такі рівняння:

$$| f (A) | = | A | \text{ та } | f - 1 (B) | = | B | \qquad (3)$$

Якщо X і Y — скінченні множини з однаковою потужністю, а f: X → Y, то наступне еквівалентно:

- f є бієкцією;
- f є ін’єкцією;
- f є сур’єкцією (рис 2.9).

Для кінцевої множини S існує бієкція між множиною можливих повних порядків елементів і набором проекцій від S до S. Тобто кількість перестановок елементів S дорівнює кількості загальних порядків цієї множини, а саме n !.

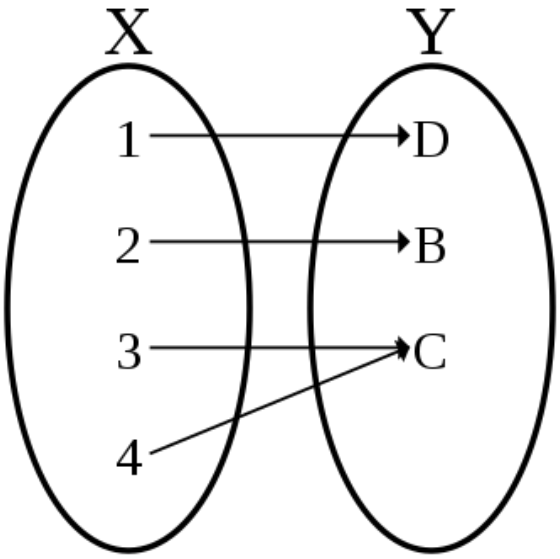


Рисунок 2.9 - Приклад сур’єктивного відображення

Також варто зазначити, що відображення може не бути ні сур’єктвним, ані ін’єктивним. Приклад такого відображення наведено на рисунку 2.10.

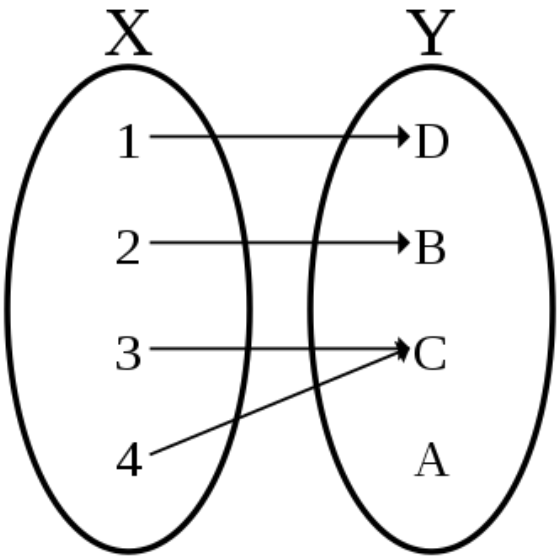


Рисунок 2.10 – Приклад не ін’єктивного та не сур’єтивного відображення

В комп’ютерній інженерії застосування прямих підстановок в багатьох випадках супроводжується використання обернених підстановок. Обернене бієктивне відображення $\dot{Y}_r \to \dot{X}_r$ (обернена підстановка $F^{-1}(\dot{Y}_r)$) визначається як $F^{-1}(F(\dot{X}_r)) = \dot{X}_r$ та реалізується оберненим автоматом. Обернений автомат визначається на основі прямого автомата. Функція виходів генеруються за наступним алгоритмом. Відбувається перестановка символів таблиці функції виходів прямого окремо для кожного стану s. У таблиці виходів (таблиця 2.9) комірку, що відповідає символу вихідного алфавіту $y_a$ зліва направо вписується символ вхідного алфавіту $x_b$, який в таблиці виходів функції прямої підстановки відповідав за поточний символ $y_a$, причому a і b можуть мати будь-яке значення, а також збігатися.

Функція переходів (таблиця 2.10) створюється на основі функції виходів оберненого автомата та функції станів прямого. Відбувається перестановка станів таблиці переходів функції прямої підстановки окремо для кожного рядка. У комірку, що відповідає символу вихідного алфавіту $y_a$ зліва направо вписується стан $s_b$, що знаходиться в тих же координатах таблиці переходів, що і відповідний вихідному символу $y_a$ вхідний символ $x_b$ в таблиці виходів оберненої підстановки, причому a і b можуть мати будь-яке значення, а також збігатися. Розглянемо

фрагмент лістингу програми 2.3 для генерування таблиць переходів і виходів оберненого автомата мовою C#:

```
InitialState = directMachine.InitialState;
for (int i = 0; i < M; i++)
{
        StateMatrix.Add(new List<int>());
        OutputMatrix.Add(new List<int>());
        for (int j = 0; j < N; j++)
        {
                for (int k = 0; k < N; k++)
                {
                if (directMachine.OutputMatrix[i][k] == j)
                        {
                        OutputMatrix[i].Add(k);
                        StateMatrix[i].Add(
                        directMachine.StateMatrix[i][k]);
                        break;
                        }
                }
        }
}
```

Фрагмент лістингу програми 2.3 - Реалізація алгоритмів формування таблиць виходів та станів оберненого автомата

Алгоритм формування оберненого бієктивного відображення $\dot{Y}_r \to \dot{X}_r$ повністю збігається з алгоритмом формування прямого, який було наведено раніше. Різниця полягає у використанні відповідних таблиць.

Таблиця 2.9 - Таблиця виходів автомата для реалізації $F^{-1}(\dot{Y}_r)$ оберненого автомата

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $x_1$ | $x_5$ | $x_7$ | $x_2$ | $x_6$ | $x_3$ | $x_4$ | $x_8$ |
| $s_2$ | $x_8$ | $x_7$ | $x_2$ | $x_3$ | $x_5$ | $x_1$ | $x_6$ | $x_4$ |
| $s_3$ | $x_7$ | $x_5$ | $x_3$ | $x_1$ | $x_6$ | $x_4$ | $x_8$ | $x_2$ |
| $s_4$ | $x_1$ | $x_8$ | $x_2$ | $x_4$ | $x_5$ | $x_7$ | $x_6$ | $x_3$ |
| $s_5$ | $x_1$ | $x_5$ | $x_4$ | $x_7$ | $x_6$ | $x_3$ | $x_2$ | $x_8$ |
| $s_6$ | $x_2$ | $x_8$ | $x_1$ | $x_5$ | $x_4$ | $x_3$ | $x_7$ | $x_6$ |
| $s_7$ | $x_4$ | $x_6$ | $x_3$ | $x_1$ | $x_7$ | $x_5$ | $x_2$ | $x_8$ |

Продовження таблиці 2.9

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_8$ | $x_2$ | $x_5$ | $x_7$ | $x_1$ | $x_4$ | $x_6$ | $x_8$ | $x_3$ |
| $s_9$ | $x_8$ | $x_5$ | $x_6$ | $x_3$ | $x_1$ | $x_7$ | $x_2$ | $x_4$ |
| $s_{10}$ | $x_2$ | $x_6$ | $x_7$ | $x_1$ | $x_5$ | $x_8$ | $x_3$ | $x_4$ |
| $s_{11}$ | $x_4$ | $x_3$ | $x_2$ | $x_6$ | $x_5$ | $x_1$ | $x_8$ | $x_7$ |
| $s_{12}$ | $x_4$ | $x_2$ | $x_5$ | $x_8$ | $x_7$ | $x_1$ | $x_3$ | $x_6$ |

Таблиця 2.10 - Таблиця переходів автомата для реалізації $F^{-1}(\dot{Y}_r)$ оберненого автомата

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $s_1$ | $s_4$ | $s_5$ | $s_{12}$ | $s_{11}$ | $s_5$ | $s_{11}$ | $s_4$ |
| $s_2$ | $s_3$ | $s_8$ | $s_{11}$ | $s_4$ | $s_8$ | $s_7$ | $s_{12}$ | $s_4$ |
| $s_3$ | $s_9$ | $s_{10}$ | $s_1$ | $s_{10}$ | $s_5$ | $s_7$ | $s_6$ | $s_{12}$ |
| $s_4$ | $s_6$ | $s_7$ | $s_7$ | $s_5$ | $s_8$ | $s_{10}$ | $s_8$ | $s_1$ |
| $s_5$ | $s_{11}$ | $s_4$ | $s_3$ | $s_{10}$ | $s_7$ | $s_5$ | $s_{12}$ | $s_4$ |
| $s_6$ | $s_7$ | $s_6$ | $s_9$ | $s_3$ | $s_9$ | $s_3$ | $s_3$ | $s_6$ |
| $s_7$ | $s_9$ | $s_7$ | $s_8$ | $s_8$ | $s_7$ | $s_{11}$ | $s_8$ | $s_1$ |
| $s_8$ | $s_2$ | $s_6$ | $s_4$ | $s_{10}$ | $s_5$ | $s_1$ | $s_{12}$ | $s_{11}$ |
| $s_9$ | $s_{10}$ | $s_2$ | $s_{12}$ | $s_8$ | $s_7$ | $s_{12}$ | $s_2$ | $s_6$ |
| $s_{10}$ | $s_{10}$ | $s_7$ | $s_2$ | $s_9$ | $s_{12}$ | $s_9$ | $s_{12}$ | $s_{11}$ |
| $s_{11}$ | $s_1$ | $s_{12}$ | $s_{11}$ | $s_9$ | $s_{10}$ | $s_9$ | $s_{11}$ | $s_1$ |
| $s_{12}$ | $s_4$ | $s_{12}$ | $s_5$ | $s_5$ | $s_5$ | $s_4$ | $s_9$ | $s_6$ |

## 2.3 Аналіз алгоритму та його реалізації для формування конструктивних модулів

При заповненні таблиць автоматів, розглянуті алгоритми, заповнювали наступний елемент рядка, порівнюючи з іншими елементами у рядку. Дану операцію можна спростити, використавши перестановки. Розглянемо ці операції та їх властивості більш детально.

У математиці перестановка множини — це, розташування її членів у послідовність або лінійний порядок, або, якщо множина вже впорядкована, перегрупування її елементів. Слово «перестановка» також відноситься до акту або процесу зміни лінійного порядку впорядкованої множини [18].

Перестановки відрізняються від комбінацій, які є виділеннями деяких членів набору незалежно від порядку. Наприклад, записані у вигляді кортежів, існує шість перестановок множини {1, 2, 3}, а саме (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2) та (3, 2, 1). Це всі можливі упорядкування цього триелементного набору. Анаграми слів, літери яких відрізняються, також є перестановками: букви вже впорядковані в оригінальному слові, а анаграма — це переупорядкування букв. Вивчення перестановок скінченних множин є важливою темою в області комбінаторики та теорії груп.

Перестановки використовуються майже в кожній галузі математики та в багатьох інших галузях науки. В інформатиці вони використовуються для аналізу алгоритмів сортування; у квантовій фізиці для опису станів частинок; а в біології — для опису послідовностей РНК.

Кількість перестановок n різних об’єктів є n факторіальним, зазвичай записується як n!, що означає добуток усіх натуральних чисел, менших або рівних n. Сукупність усіх перестановок множини утворює групу, яка називається симетричною групою множини. Групова операція — це композиція (виконання двох заданих перестановок поспіль), що призводить до іншої перегрупування.

Оскільки властивості перестановок не залежать від природи елементів множини, часто для вивчення перестановок розглядаються перестановки.

У елементарній комбінаториці k-перестановки, або часткові перестановки, являють собою впорядковане розташування k різних елементів, вибраних із множини. Коли k дорівнює розміру множини, це перестановки множини.

Розглянемо алгоритми для виконання повних перестановок, та їх ключові особливості:

- лексикографічне впорядкування є узагальненням алфавітного порядку словників на послідовності впорядкованих символів або, елементів повністю впорядкованого набору;
- алгоритм Стейнхауза–Джонсона–Троттера [19] генерує упорядкування для всіх перестановок даної послідовності з властивістю, що будь-які дві послідовні перестановки на її виході відрізняються, міняючи місцями два сусідніх значення;
- алгоритм купи [20] генерує всі можливі перестановки з n об'єктів;
- алгоритм перестановки зірок Ерліха [21]: на кожному кроці перший запис перестановки обмінюється більш пізнім записом;
- алгоритм зміни префікса Закса [22]: на кожному кроці префікс поточної перестановки змінюється на зворотний, щоб отримати наступну перестановку;
- алгоритм Савади-Вільямса [23]: кожна перестановка відрізняється від попередньої або циклічним зсувом вліво на одну позицію, або обміном перших двох записів;
- алгоритм Корбетта [24]: кожна перестановка відрізняється від попередньої циклічним зсувом деякого префікса вліво на одну позицію;
- одноколійне впорядкуванням [25]: кожен стовпець є циклічним зсувом інших стовпців;

– одноколійний код Грея [26]: кожен стовпець є циклічним зсувом інших стовпців, плюс будь-які дві послідовні перестановки відрізняються лише однією або двома транспозиціями.

На рисунку 2.11 зображено упорядкування всіх перестановок довжини n=4 за допомогою попередньо розглянутих алгоритмів. Перестановки позначені кольором, де червоний – 1, жовтий – 2, зелений – 3, та синій – 4.

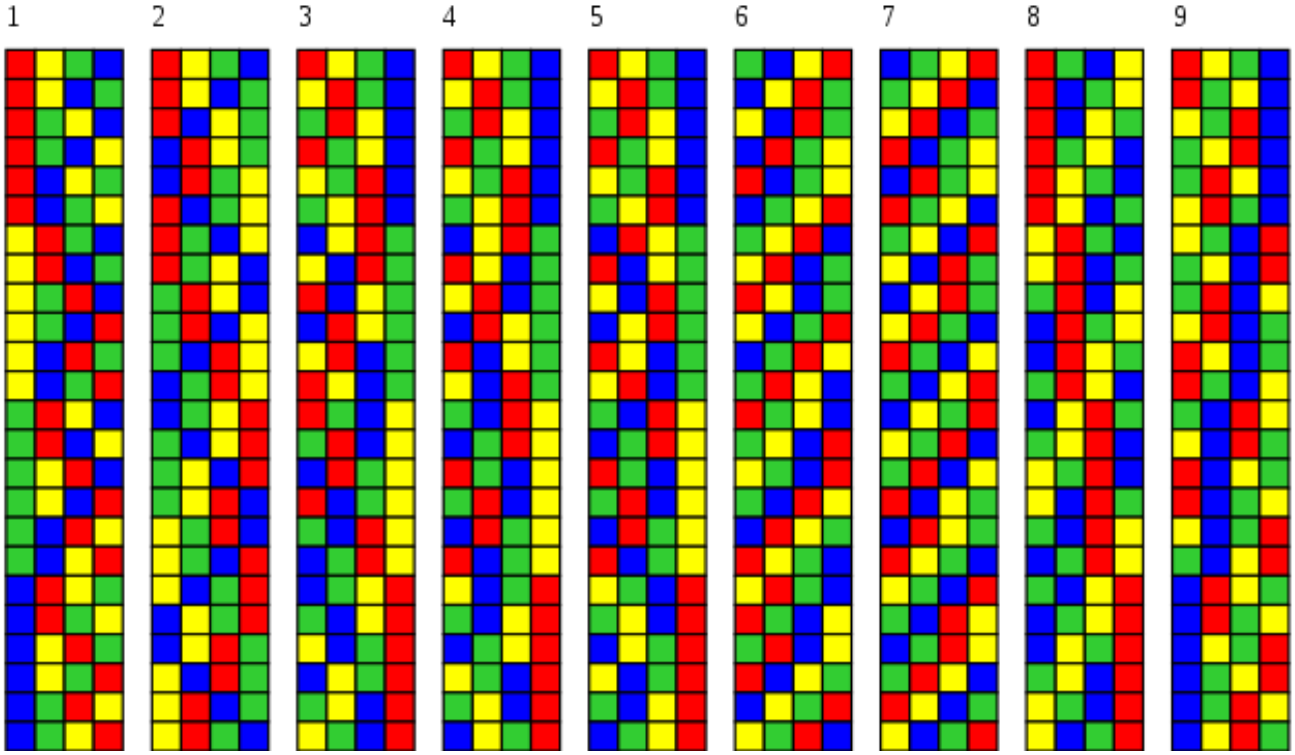


Рисунок 2.11 – Візуалізація перестановок довжини n=4 за допомогою попередньо розглянутих алгоритмів

Також у роботі були використані засоби для генерації випадкових чисел. За допомогою цього процесу, генератор випадкових чисел генерує відповідно послідовність чисел або символів, які неможливо передбачити краще, ніж згенерувавши випадково. Це означає, що конкретна послідовність результатів міститиме певні закономірності, які можна визначити ретроспективно, але неможливо буде передбачити. Справжні генератори випадкових чисел можуть бути апаратними генераторами випадкових чисел, які генерують випадкові числа, де

кожне покоління є функцією поточного значення атрибута фізичного середовища, яке постійно змінюється таким чином, що знову ж таки практично неможливо змоделювати. Це було б відмінністю від так званих «генерацій випадкових чисел», які виконуються генераторами псевдовипадкових чисел, які в свою чергу генерують числа, які лише виглядають випадковими, але насправді вони є заздалегідь визначеними – ці покоління можна відтворити, просто знаючи стан цього генератора. Взагалі, генератори випадкових чисел застосовуються в наступних областях:

- криптографії;
- азартних іграх;
- статистичній вибірці;
- комп’ютерному моделюванні;
- повністю рандомізованому дизайні.

Як правило, у програмах безпеки, основною функцією яких є непередбачуваність, апаратні генератори, як правило, віддають перевагу псевдовипадковим алгоритмам, де це можливо.

Генерування псевдовипадкових чисел є важливим і поширеним завданням комп’ютерного програмування. У той час як криптографія та певні числові алгоритми вимагають дуже високого ступеня видимої випадковості, багато інших операцій вимагають лише невеликого ступеня непередбачуваності. Кілька простих прикладів можуть полягати в тому, щоб дати користувачеві «випадкову цитату дня» або визначити, як керований комп’ютером супротивник може рухатися в комп’ютерній грі. Слабші форми випадковості використовуються в алгоритмах хешування та при створенні алгоритмів пошуку та сортування.

## Висновки до розділу

У даному розділі було розглянуто та проаналізовано основні характеристики конструктивних модулів, та досліджені існуючі алгоритми для формування конструктивних модулів. Також були проаналізовані операції підстановки які реалізуються на структурах із цих модулів. У наступному розділі будуть розглянуті та проаналізовані засоби для реалізації алгоритмів, та операцій підстановки, які реалізуються у цих структурах.

# 3 ПОРІВНЯННЯ ТЕХНОЛОГІЙ АНАЛІЗУ РЕГУЛЯРНИХ СТРУКТУР ЛІНІЙНОЇ СКЛАДНОСТІ

## 3.1 Обґрунтування обраного типу прикладного програмного забезпечення для розробки програмних засобів аналізу регулярних структур

Як вже було зазначено у попередніх розділах, для даного дипломного проєкту було вирішено використовувати мову програмування - Python. Згідно досить відомому індексу популярності мови програмування (PopularitY of Programming Language - PYPL), мова Python залишається лідером (рис. 3.1) [27].

**Worldwide**, Dec 2021 compared to a year ago:

| Rank | Change | Language | Share | Trend |
|---|---|---|---|---|
| 1 | | Python | 30.21 % | -0.5 % |
| 2 | | Java | 17.82 % | +1.3 % |
| 3 | | JavaScript | 9.16 % | +0.6 % |
| 4 | | C# | 7.53 % | +1.0 % |
| 5 | | C/C++ | 6.82 % | +0.6 % |

Рисунок 3.1 – Рейтинг мов програмування станом на грудень 2021 року за індексом PYPL

На рисунку 3.2 візуалізовано рейтинг популярності програмування C# та Python починаючи з 2005 року за індексом PYPL.

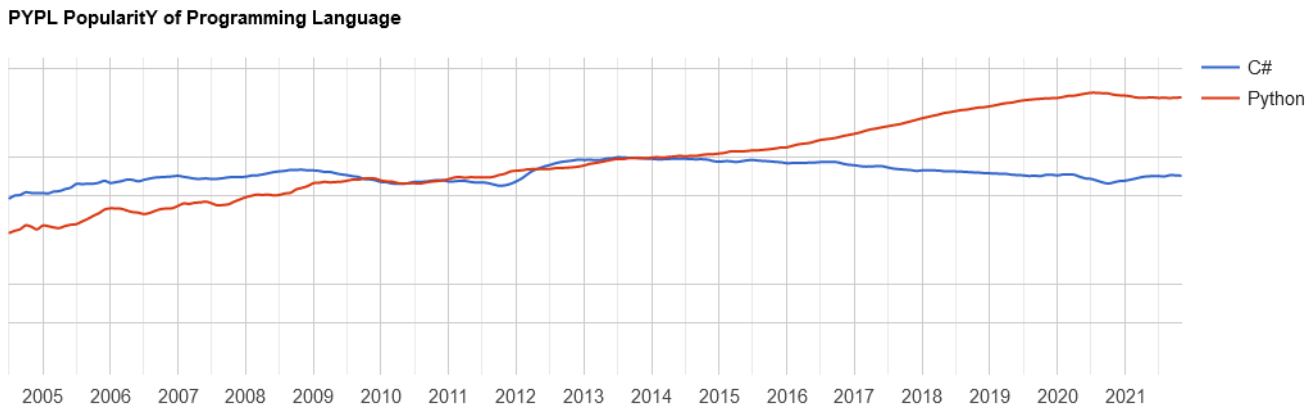


Рисунок 3.2 – Рейтинг мов програмування C# та Python за індексом PYPL

Та це не єдиний відомий рейтинг де мова Python займає перше місце. Згідно індексу спільноти програмування TIOBE (рис. 3.3) Python вийшов на першу сходинку у порівнянні з минулорічним рейтингом [28].

| Dec 2021 | Dec 2020 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 3 | ▲ | Python | 12.90% | +0.69% |
| 2 | 1 | ▼ | C | 11.80% | -4.69% |
| 3 | 2 | ▼ | Java | 10.12% | -2.41% |
| 4 | 4 | | C++ | 7.73% | +0.82% |
| 5 | 5 | | C# | 6.40% | +2.21% |

Рисунок 3.3 - Рейтинг мов програмування станом на грудень 2021 року за індексом TIOBE

На рисунку 3.4 візуалізовано рейтинг популярності цих мов починаючи з 2001 року за індексом TIOBE.

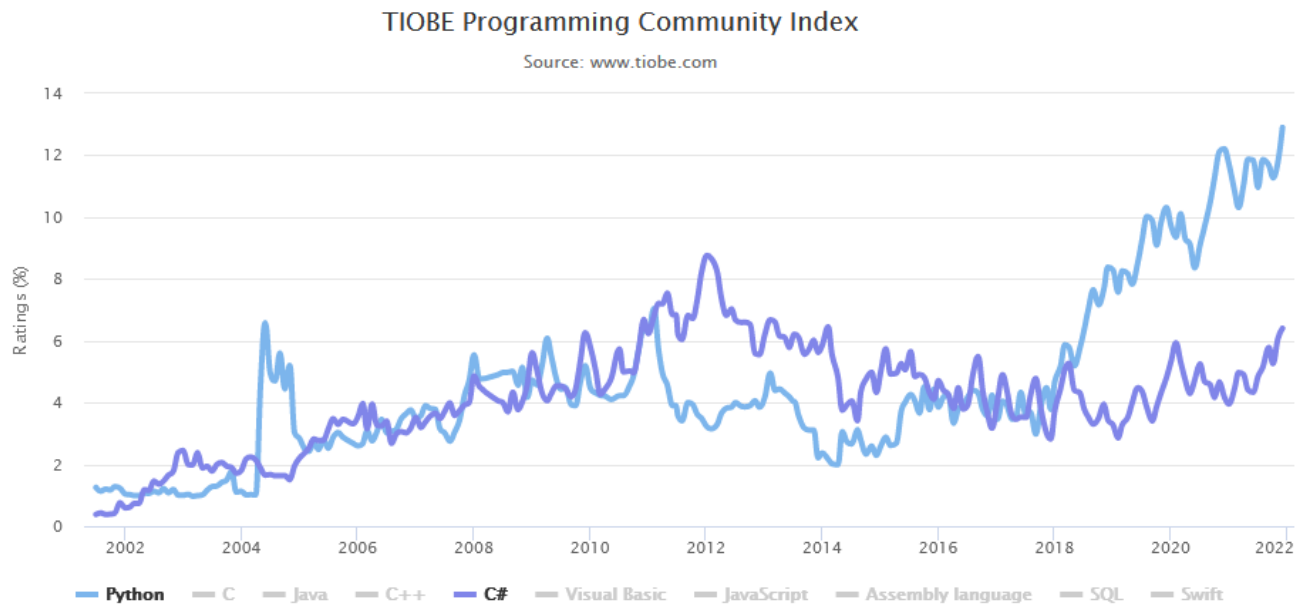


Рисунок 3.4 – Рейтинг мов програмування C# та Python за індексом TIOBE

Розглянемо більш детально різницю між мовами програмування C# та Python. Насамперед обидві мови є одними з популярних мов програмування 2021 року. Обидві засновані на концепціях ООП, прості у вивченні та написанні коду, а також пропонують швидку розробку та високу продуктивність. Перш ніж, окреслити їх відмінності, коротко оглянемо кожну з них, щоб відповідно точно оцінити їх відмінності [29].

C# — потужна мова, яка відповідає традиційним конструкціям C і C++, і водночас є більш легшою для вивчення та сучаснішою. Будучи розробленою Microsoft, ця об’єктно-орієнтована мова програмування також має багато спільного з Java. Код написаний на C# може бути скомпільований на різних платформах і має безліч досить потужних функцій, таких як:

- інтеграція з .NET Framework;
- компонентно-орієнтований;
- структурована мова високого рівня;
- сучасний синтаксис;
- легкість у навчанні;
- багата стандартна бібліотека;

– автоматичний збір сміття.

C# інтегрується з потужною платформою .NET. Окрім того, якщо людина знає мову Java і хочете перейти на .NET, вивчення C# може дати необхідний імпульс. Розглянемо власне переваги мови C#:

– проста, надійна і масштабована;
– код, безпечний для роботи з типами;
– не дозволяє небезпечне переведення;
– швидка компіляція та час виконання;
– структурована мова програмування;
– підтримує сумісність мов.

Як і C#, Python є мовою програмування загального призначення. У більшості своїх функцій він слідує C і Java. Мова є портативною і легкою для вивчення, в той же час мова має високорівневі можливості програмування, і має наступні особливості:

– підтримує як об'єктно-орієнтоване програмування, так і функціональне та структурне програмування;
– проста у написанні коду, читанні, підтримці та портативності(можливості запуску на інших платформах);
– підтримує автоматичний збір сміття;
– досить велика стандартна бібліотека, яка включає інтерфейси ОС, інструменти для роботи з веб-сервісами;
– безкоштовне використання та поширення, Python розроблено за ліцензією з відкритим кодом;
– багата стандартна бібліотека, яка є портативною і сумісна з різними платформами, такими як Windows, Mac або Unix;
– підходить для мережевих програм, які використовують декілька протоколів.

Безпосередній аналіз мов наведено в таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз мов програмування C# та Python

| C# | Python |
|---|---|
| Розроблено Microsoft. Поставляється з ліцензією. | Розробка та розповсюдження з відкритим кодом, навіть для комерційного використання. |
| На основі концепцій ООП. | Підтримує декілька парадигм програмування (ООП, процедурне) |
| Статично типізований. Компілятор видаватиме помилки за неправильне приведення типів. | Динамічний типізація. Немає необхідності в оголошенні змінних. |
| Підтримує роботу на .NET Framework. | Може бути інтегрований з Java (JVM), .NET, C і JavaScript. |
| Більш організований і послідовний синтаксис і формат. | Простий, зручний для читання та написання коду, не містить занадто багато символів або форматів. |
| Більш статична мова. Все потрібно побудувати (скомпілювати), а потім запустити. | Зменшує цілий крок у циклі розробки, оскільки все динамічно, вибирається під час виконання. |
| Відсутній інтерпретатор. | Інтерактивний інтерпретатор для легкого написання програм. |
| Завдяки платформі Common Language Infrastructure (CLI) C# працює швидше і забезпечує кращу продуктивність. | Робота з розробки йде швидше, але в порівнянні з C# продуктивності повільніша. |
| Підтримка бібліотеки є хорошою і базується на платформі .NET. | Велика стандартна бібліотека. Багато коду можна використовувати повторно, що полегшує роботу розробникам. |
| Багатопоточність досить проста за допомогою платформи .NET. | Через глобальне блокування інтерпретатора (GIL) багатопотокова робота вимагає використання декількох процесів. |

Мова Python залишається популярною вже досить багато часу, і за цей час встигла показати себе майже у кожній сфері створення програмного забезпечення. Вона є простою для вивчення, що стало однією з причин цього розвитку. Та зараз Python є одним з провідних інструментів у веброзробці, та для роботи з даними згідно опитувань цього року [30] від компанії JetBrains (рис. 3.5).

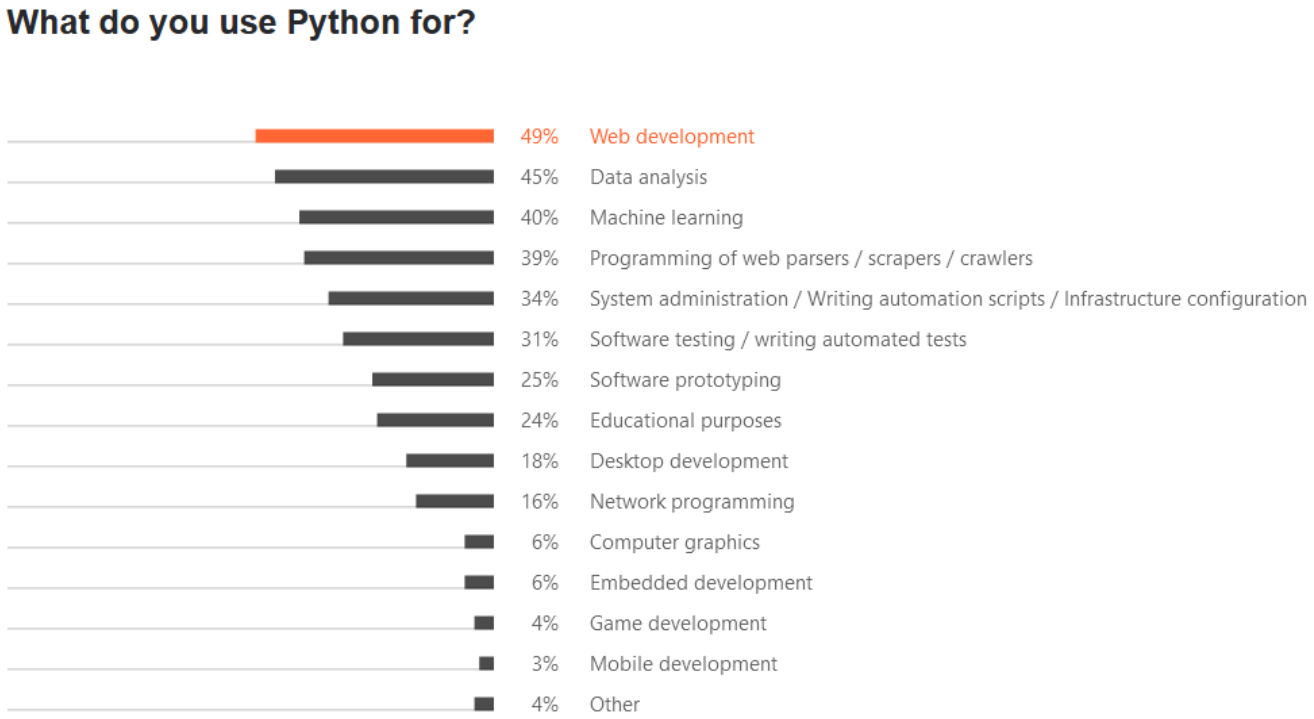


Рисунок 3.5 – Рейтинг напрямків, у яких найбільше застосовується Python

Як бачимо з таблиці 3.1, мова Python програє мові C#, лише в продуктивності будучи інтерпретованою мовою. Для вирішення цієї проблеми було використано бібліотеку NumPy – засіб для роботи з даними. На рисунку 3.6 можемо бачити рейтинг бібліотек [31], які застосовуються для аналізу даних.

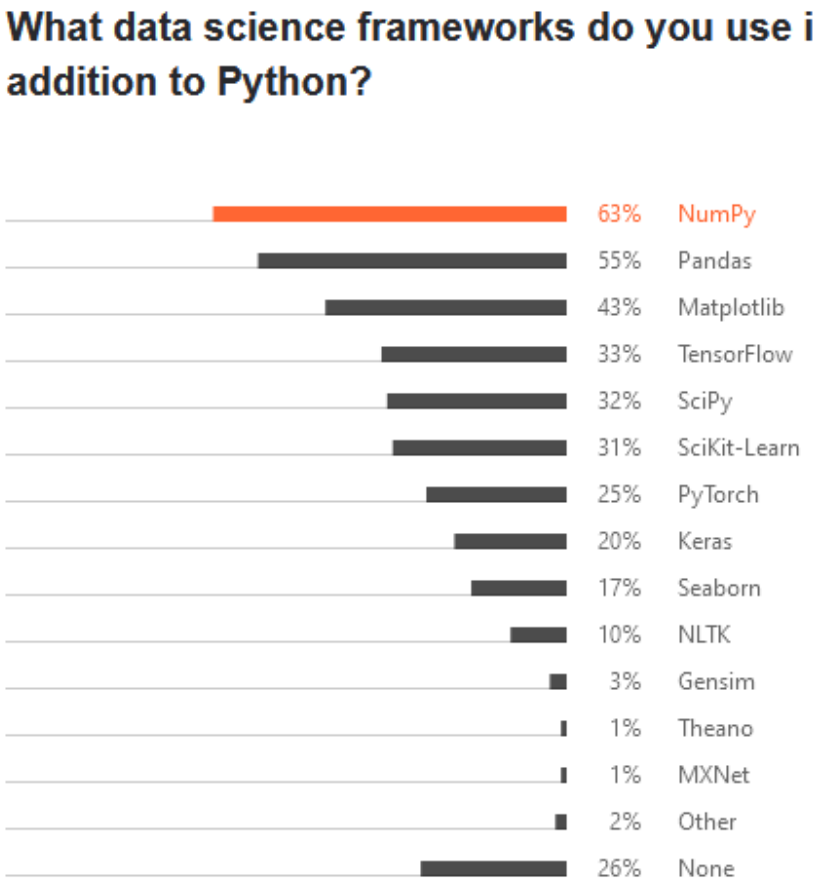


Рисунок 3.6 – Рейтинг бібліотек, для роботи з даними

Цей рейтинг очолює бібліотека NumPy, яка застосовувалася при створенні засобів аналізу регулярних структур. Розглянемо більш детально дану бібліотеку. В першу чергу, це фундаментальний пакет для наукових обчислень на Python [32].

Ця бібліотека Python, надає багатовимірний об’єкт масиву, різні похідні об’єкти (такі як замасковані масиви та матриці), а також набір підпрограм для швидких операцій з масивами, включаючи:

– математичні;
– логічні;
– маніпуляції з формою;
– сортування;
– вибір;
– введення/виводу;
– дискретні перетворення Фур'є;

- операції базової лінійної алгебри;
- основні статистичні операції;
- випадкове моделювання.

В основі пакету NumPy лежить об'єкт ndarray. Він інкапсулює n-вимірні масиви однорідних типів даних, при цьому багато операцій виконуються в скомпільованому коді для підвищення продуктивності.

Існує кілька важливих відмінностей між масивами NumPy і стандартними списками Python:

- масиви NumPy мають фіксований розмір під час створення, на відміну від списків Python (які можуть динамічно зростати). Зміна розміру ndarray створить новий масив і видалить оригінал;
- усі елементи в масиві NumPy повинні мати один тип даних і, таким чином, мати однаковий розмір пам’яті. Виняток: можна мати масиви об’єктів (Python, включаючи NumPy), що дозволяє використовувати масиви елементів різного розміру;
- масиви NumPy полегшують розширені математичні та інші типи операцій над великою кількістю даних. Як правило, такі операції виконуються ефективніше і з меншою кількістю коду, ніж це можливо з використанням вбудованих послідовностей Python;
- зростаюча кількість науково-математичних пакетів на основі Python використовує масиви NumPy; хоча вони зазвичай підтримують введення списків Python, вони перетворюють такі вхідні дані в масиви NumPy перед обробкою, і вони часто виводять масиви NumPy. Іншими словами, для того, щоб ефективно використовувати більшу частину (можливо, навіть більшість) сучасного науково-математичного програмного забезпечення на основі Python, просто знати, як використовувати вбудовані типи списки Python, недостатньо – потрібно також знати, як використовувати масиви NumPy.

У наукових обчисленнях особливо важливі моменти про розмір і швидкість послідовності. Як простий приклад розглянемо випадок множення кожного елемента в одновимірній послідовності на відповідний елемент в іншій послідовності такої ж довжини. Якщо дані зберігаються в двох списках Python, a і b, ми можемо повторити кожен елемент (фрагмент лістингу програми 3.1):

```
c = []
for i in range(len(a)):
        c.append(a[i]*b[i])
```

Фрагмент лістингу програми 3.1 – Множення елементів одновимірного масиву засобами стандартної бібліотеки Python

Це дає правильну відповідь, але якщо a і b містять мільйони чисел у кожному, це не буде зовсім ефективно, використовуючи цикли в Python. Те саме завдання можна було б виконати набагато швидше мовою C, написавши (для наочності ми нехтуємо оголошеннями змінних та ініціалізаціями, виділенням пам’яті тощо) наступний фрагмент лістингу програми 3.2.

```
for (i = 0; i < rows; i++): {
        c[i] = a[i]*b[i];
}
```

Фрагмент лістингу програми 3.2 - Множення елементів однакових одновимірних масивів засобами мови C

Це заощаджує всі накладні витрати, пов’язані з інтерпретацією коду Python та маніпулюванням об’єктами Python, але за рахунок переваг, отриманих від кодування на цій мові. Крім того, необхідна робота з написанням коду збільшується зі збільшенням розмірності наших даних. У випадку двохвимірного масиву, наприклад, код C (скорочений, як і раніше) розширюється до фрагменту лістингу програми 3.3.

```
for (i = 0; i < rows; i++): {
        for (j = 0; j < columns; j++): {
                c[i][j] = a[i][j]*b[i][j];
```

```
                    }
                }
```

Фрагмент лістингу програми 3.3 - Множення елементів однакових двовимірних масивів засобами мови С

NumPy дає переваги обох мов: поелементні операції є «режимом за замовчуванням», коли задіяний об’єкт ndarray, але поелементна операція швидко виконується попередньо скомпільованим кодом С. Наступний фрагмент лістингу програми 3.4 написаний засобами бібліотеки NumPy

```
c = a * b
```

Фрагмент лістингу програми 3.4 – Множення елементів однакових n-вимірних масивів засобами бібліотеки NumPy

виконує те саме, що і попередні приклади, зі швидкістю, близькою до С, але з простотою коду, яку ми очікуємо від чогось на основі Python. Цей останній приклад ілюструє дві функції NumPy, які є основою більшої частини його потужності: векторизація і трансляція.

Векторизація описує відсутність будь-якого явного циклу, індексування в коді - ці речі відбуваються, звичайно, просто «за кадром» в оптимізованому, попередньо скомпільованому коді С. Векторизований код має багато переваг, серед яких:

- векторизований код є більш стислим і легшим для читання;
- менше рядків коду зазвичай означає менше помилок;
- код більше нагадує стандартні математичні позначення (що полегшує, як правило, правильне кодування математичних конструкцій);
- векторизація призводить до більш «Pythonic» (чистого) коду. Без векторизації код був би ускладнений неефективними і важкими для читанням циклами for.

Трансляція — це термін, який використовується для опису неявної поелементної поведінки операцій. Загалом, у NumPy всі операції, не тільки

арифметичні, а й логічні, порозрядні, функціональні, поводяться таким чином неявно, тобто вони транслюються. Більше того, у наведеному вище фрагменті лістингу 3.1, a і b можуть бути багатовимірними масивами однакової форми, або скаляром і масивом, або навіть двома масивами з різними формами, за умови, що менший масив «розширюється» до форми більшого таким чином, щоб отримана трансляція була однозначною.

## 3.2 Порівняння засобів стандартної бібліотеки Python із засобами бібліотеки NumPy

Розглянемо різницю основних засобів стандартних бібліотек Python із засобами NumPy, які були використані в рамках цієї роботи:

- порівняння масивів;
- порівняння роботи з масивами;
- порівняння засобів перестановки;
- порівняння засобів генерування випадкових чисел.

### 3.2.1 Порівняння структури масивів реалізованих засобами мови Python із засобами бібліотеки NumPy

Масиви є одними з важливих структурних елементів, які використовувалися для аналізу регулярних структур лінійної складності. В мові Python для роботи з масивами використовується такий тип даних як список. Відповідно двовимірний масив це список списків. Та в NumPy список називається масивом. В обох випадках

список в Pythoni та масив в NumPy є змінними і є можливість індексувати певні елементи [33]. Різниця цих типів масивів [34] наведено на рисунку 3.7.

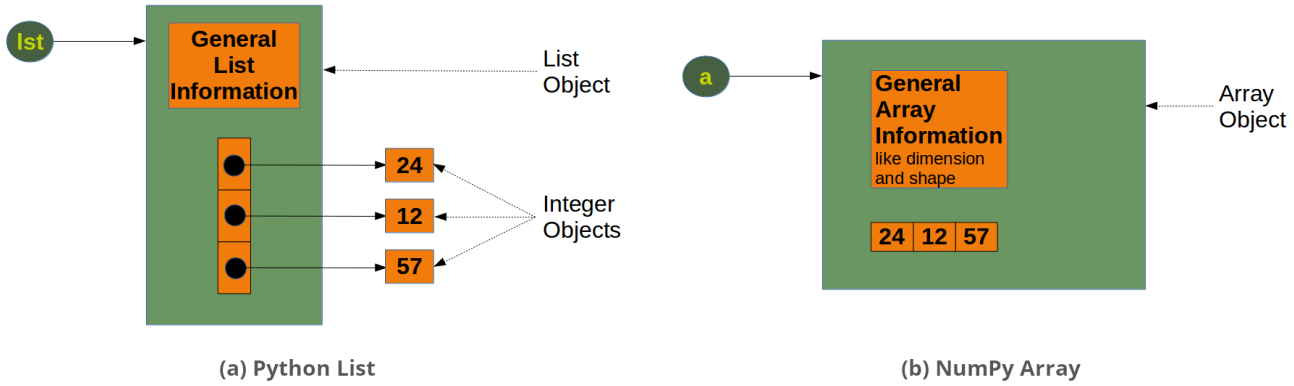


Рисунок 3.7 – Різниця масивів реалізованих засобами стандартної бібліотеки Python із бібліотекою NumPy

Щодо різниці між цими структурами, то NumPy має наступні переваги, над реалізацією у Python:

– можна виконувати математичні операції над масивом
– масив займає менше місця в пам’яті
– використання масиву є більш швидким ніж використання списку.

### 3.2.2 Порівняння роботи над масивами засобами мови Python із засобами бібліотеки NumPy

В [35] було розглянуто різницю цих засобів на прикладі операції множення матриць. Найпростіший і зрозумілий спосіб зробити добуток 2 матриць у Python та будь-якій іншій мові програмування — це використання 3 вкладених циклів for. Розглянемо фрагмент лістингу програми 3.5, де було імпортовано необхідні бібліотеки, та оголошено функції для виміру часу.

```
import time
import numpy as np
import random as rn

def timer(func):
	def wrapper(*args, **kwargs):
		before = time.time()
		result = func(*args, **kwargs)
		after = time.time()
		return after - before, result
		return wrapper

def generate(size, range_):
	arr = [[[rn.randrange(*range_) for _ in range(size)] for _ in range(size)] for _ in range(2)]
	return arr
```

Фрагмент лістингу програми 3.5 – Налаштування бібліотек та функцій для виміру часу

У даному фрагменті таймер виступає обгорткою для вимірювання часу виконання функцій, і генерейт генерує матрицю заповнену за допомогою генератора випадкових чисел. Функції із реалізацією множення матриць наведено у фрагменті лістингу програми 3.6.

```
@timer
def python_implementation(arr1, arr2):
	result = [[0 for _ in range(len(arr1))] for _ in range(len(arr2[0]))]
	for i in range(len(arr1)):
		for j in range(len(arr2[0])):
			for k in range(len(arr2[0])):
				result[i][j] += arr1[i][k] * arr2[k][j]
	return result

@timer
def numpy_implementation(arr1, arr2):
	return np.array(arr1).dot(arr2)
```

Фрагмент лістингу програми 3.6 – Функції множення двох матриць засобами стандартної бібліотеки Python і засобами бібліотеки NumPy

Протестуємо виконання цих функцій, і перемножимо дві матриці розміром 500x500 (фрагменті лістингу 3.7).

```
if __name__ == '__main__':
 data = generate(size=500, range_=(1, 100))
 numpy_time_taken, numpy_result = numpy_implementation(*data)
```

```
python_time_taken, python_result = python_implementation(*data)
print(f"Час витрачений засобами бібліотеки numpy:\t{numpy_time_taken} секунд.")
print(f"Час витрачений стандартними засобами python:\t{python_time_taken} секунд.")
```

Фрагмент лістингу програми 3.7 – Тестування виконання попередньо розглянутих функцій

З рисунку 3.8 бачимо що дана операція виконалася швидше більш в 100 раз засобами бібліотеки NumPy, ніж засобами стандартної бібліотек мови Python.

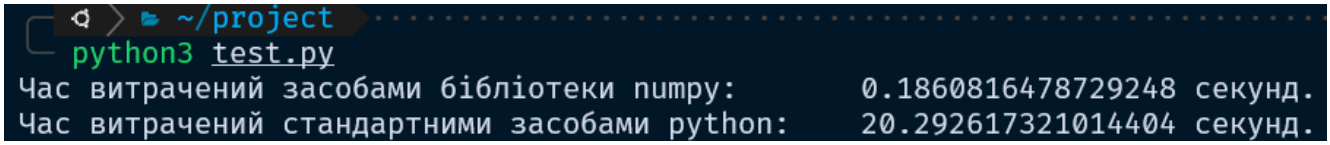


Рисунок 3.8 – Результати роботи програми порівняння

Варто зазначити що у розглянутій роботі [35], бібліотека NumPy впоралася із завданням швидше в 320 разів, що може свідчити, про відповідний розвиток мови Python, і її оптимізації із кожною новою версією.

### 3.2.3 Порівняння операцій перестановок реалізованих засобами мови Python із засобами бібліотеки NumPy

Наступним кроком порівняємо засоби перестановок стандартної бібліотеки itertools мови Python, із відповідними засобами бібліотеки NumPy. Бібліотека itertools [36] реалізує ряд структурних блоків ітераторів, натхнених конструкціями з APL, Haskell і SML. Кожен з них був перероблений у форму, придатну саме для Python. Модуль стандартизує основний набір швидких, ефективних для пам'яті інструментів, які корисні самі по собі, або в комбінації. Разом вони утворюють «алгебру ітераторів», що дозволяє конструювати спеціалізовані інструменти коротко й ефективно на чистому Python.

Розглянемо принцип роботи функції permutations [37] із цієї бібліотеки. Як результат вона повертає послідовні r перестановки довжини елементів у ітерації. Якщо r не вказано або має значення None, тоді r за замовчуванням відповідає довжині ітеративного елемента, і генеруються всі можливі повнорозмірні перестановки. Кортежі перестановки випускаються в лексикографічному порядку відповідно до порядку введення ітерації.

Отже, якщо вхідний ітератор відсортований, кортежі комбінацій будуть створені в відсортованій послідовності. Елементи вважаються унікальними на основі їхнього положення, а не цінності. Таким чином, якщо вхідні елементи є унікальними, у кожній перестановці не буде повторюваних значень.

Перейдемо до модуля Random [38] бібліотеки NumPy. Підпрограми випадкових чисел Numpy створюють псевдовипадкові числа, використовуючи комбінації BitGenerator для створення послідовностей і Generator для використання цих послідовностей для вибірки з різних статистичних розподілів:

- BitGenerators: об’єкти, які генерують випадкові числа. Зазвичай це цілі слова без знака, заповнені послідовностями з 32 або 64 випадкових бітів.
- Генератори: об’єкти, які перетворюють послідовності випадкових бітів з BitGenerator у послідовності чисел, які відповідають певному розподілу ймовірностей (наприклад, рівномірний, нормальний або біноміальний) у межах заданого інтервалу.

Починаючи з Numpy версії 1.17.0, генератор можна ініціалізувати за допомогою кількох різних генераторів BitGenerator. Він відкриває багато різних розподілів ймовірностей. Контекст щодо оновлених процедур випадкових чисел Numpy представлені у NEP 19 [39].

Застарілі підпрограми випадкових чисел RandomState все ще доступні, але обмежені одним BitGenerator. Для зручності та зворотньої сумісності методи одного екземпляру RandomState імпортуються в простір імен numpy.random. Функція permutations бібліотеки виконує довільну перестановку послідовності або повернення зміненого діапазону. На вхід функція може приймати число або масив. Якщо на вхід подається ціле число, тоді функція здійснює перестановку у масив

розміром x за допомогою np.arange(x). Якщо на вхід було подано масив, тоді створює його копію, і переставляє елементи випадковим шляхом. Відповідно як результат повертається масив.

### 3.2.4 Порівняння генераторів випадкових чисел реалізованих засобами мови Python із засобами бібліотеки NumPy

Розглянемо модулі для генерації випадкових чисел. Стандартна бібліотека мови Python використовує для цього модуль Random [40]. Цей модуль реалізує генератори псевдовипадкових чисел для різних розподілів. Для цілих чисел є рівномірний вибір із діапазону. Для послідовностей існує рівномірний вибір випадкового елемента, функція для створення випадкової перестановки списку на місці та функція випадкової вибірки без заміни. Також даний модуль використовується для роботи з лініями реальних чисел, і має функції для обчислення:

- рівномірного,
- нормального (гауссового),
- логнормального,
- негативного експоненціального,
- гамма- і бета-розподілів.

Для генерування розподілів кутів доступний розподіл фон Мізеса. Майже всі функції модуля залежать від базової функції random(), яка генерує випадкове число з плаваючою точкою, рівномірно у напіввідкритому діапазоні [0.0, 1.0). Python використовує алгоритм «Вихор Мерсена» як генератор ядру. Розглянемо більш детально цей алгоритм.

Вихор Мерсена — це генератор псевдовипадкових чисел (PRNG) [41]. Це, безумовно, найбільш широко використовуваний PRNG загального призначення.

Його назва походить від того факту, що його довжина періоду обрана як просте число Мерсенна. Генератор був розроблений у 1997 році Такудзі Нішімурою [42] та Макото Мацумотою. Він був розроблений спеціально для виправлення більшості недоліків, виявлених у старому PRNG. Найпоширеніша версія алгоритму Вихору Мерсена заснована на простому числі Мерсенна $2^{19937}-1$. Його стандартна реалізація MT19937, використовує 32-бітну довжину слова. Існує ще одна реалізація (з п'ятьма варіантами [43]), яка використовує 64-бітну довжину слова (MT19937-64), і вона в свою чергу створює іншу послідовність.

Вихор Мерсена використовується як PRNG за замовчуванням в наступних програмних засобах:

- мови програмування:
  - Dyalog APL;
  - IDL;
  - R;
  - Ruby;
  - Free Pascal;
  - PHP;
  - Python (також використовується в NumPy, хоча за замовчуванням змінено на PCG64, з версії 1.17) ;
  - Julia;
  - CMU Common Lisp;
  - Embeddable Common Lisp;
  - Steel Bank Common Lisp.
- Бібліотеки та програмне забезпечення Linux:
  - GLib;
  - GNU Multiple Precision Arithmetic Library;
  - GNU Octave;
  - GNU Scientific Library.
- Інші програмні засоби:

- Microsoft Excel;
- GAUSS;
- gretl;
- Stata;
- SageMath;
- Scilab;
- Maple;
- MATLAB.

Він також доступний у Apache Commons, у стандартному C++ (починаючи з версії C++11), та у програмному засобі Mathematica. Реалізації доповнень надаються в багатьох бібліотеках програм, включаючи бібліотеки Boost C++, бібліотеку CUDA та числову бібліотеку NAG. Переваги даного алгоритму полягають у наступному:

- для всіх варіантів, окрім CryptMT, дозвільна ліцензія та використання без патентів;
- проходить численні тести на статистичну випадковість, включаючи тести Діхарда і більшість тестів TestU01 [44] ;
- дуже довгий період $2^{19937}-1$. Варто зазначити, що хоча тривалий період не є гарантією якості в генераторі випадкових чисел, короткі періоди, такі як 232, поширені в багатьох старих програмних пакетах, можуть бути проблематичними;
- k-розподілене з 32-бітовою точністю для кожного $1 \leq k \leq 623$;
- реалізації зазвичай створюють випадкові числа швидше, ніж справжні випадкові методи. Дослідження показало, що Вихор Мерсена створює 64-розрядні випадкові числа з плаваючою комою приблизно в двадцять разів швидше, ніж апаратно реалізований набір інструкцій RDRAND на основі процесора [45];

Також варто окреслити основні недоліки цього алгоритму:

- відносно великий буфер стану - 2,5 КБ, якщо не використовується варіант TinyMT;
- середня пропускна здатність за сучасними стандартами, якщо не використовується варіант SFMT[46];
- показує дві очевидні помилки (лінійна складність) у Crush і BigCrush у пакеті TestU01. Тест, як і Вихор Мерсена, заснований на F2-алгебрі [44]. Є ряд інших генераторів, які пройшли всі тести (і численні генератори, які вийшли з ладу);
- кілька екземплярів, які відрізняються лише початковим значенням (але не іншими параметрами), як правило, не підходять для моделювання Монте-Карло, що вимагає незалежних генераторів випадкових чисел, хоча існує метод вибору кількох наборів значень параметрів [47], [48];
- може знадобитися багато часу, щоб почати генерувати вихідні дані, які проходять тести на випадковість, якщо початковий стан не дуже випадковий, особливо якщо початковий стан має багато нулів. Наслідком цього є те, що два екземпляри генератора, запущені з майже однаковими початковими станами, зазвичай видають майже однакову послідовність протягом багатьох ітерацій, перш ніж врешті-решт розходяться. Оновлення алгоритму Мерсена у 2002 році покращило ініціалізацію, так що розпочинати з такого стану дуже малоймовірно [49], у роботі [50] стверджено що версія GPU (MTGP) є кращою;
- містить підпослідовності з більшою кількістю нулів, ніж одиниць. Це додає погану властивість дифузії, що ускладнює відновлення з багатонульових станів;
- не є криптографічно захищеним, якщо не використовується варіант CryptMT. Причина в тому, що спостереження за достатньою кількістю ітерацій (624 у випадку MT19937, оскільки це розмір вектора стану, з якого створюються майбутні ітерації) дозволяє передбачити всі майбутні ітерації.

Власне для генерування випадкового числа у чистому Python використовується функція random.randrange() [51]. Функція повертає випадково вибраний елемент із заданого діапазону, який задається для цієї функції – start, end, step. Шаблон позиційного аргументу відповідає шаблону range(). Аргументи ключових слів не слід використовувати, оскільки функція може використовувати їх несподівано.

Функція для генерації випадкових у бібліотеці NumPy належить до попередньо розглянутого модуля Random. Функція [52] повертає випадкові цілі числа від нижньої межі (включно) до верхньої (виключно) або, якщо вказано параметр endpoint = True, то кінцева точка включається в проміжок з якого буде згенеровано випадкове число.

## Висновки до розділу

У поточному розділі було розглянуто основні програмні засоби, які були використані в рамках даної роботи. Наведено порівняльний аналіз між обраними засобами та програмними засобами використаними в іншій роботі. Наведено основні відмінності, між засобами стандартної бібліотеки Python та бібліотеки NumPy. У наступному розгляді було протестовано створений програмний засіб засобами бібліотеки NumPy.

# 4 ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ Й АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 4.1 Тестування роботи програмних засобів

Розглянемо створені програмні засоби для аналізу регулярних структур лінійної складності. У фрагменті лістингу програми 4.1 наведено імпортування потрібних бібліотек та оголошення змінних, масивів і їх заповнення пустими значеннями засобами бібліотеки NumPy.

```
from datetime import datetime
from pathlib import Path

import numpy as np

now = datetime.now()

m_size = 12
n_size = 8

state_list = np.empty([m_size, n_size], dtype=int)
output_list = np.empty([m_size, n_size], dtype=int)
start_row = np.arange(1, n_size + 1)
rng = np.random.default_rng()
subdir = now.strftime('%d%m%Y-%H%M%S')
parent_dir = Path(__file__).parent.resolve()

path = Path(parent_dir/subdir)
path.mkdir(parents=True, exist_ok=True)
```

Фрагмент лістингу програми 4.1 – Імпорт бібліотек та оголошення змінних і масивів

У цьому фрагменті також налаштовується директорія для збереження матриць у файл, для подальшого аналізу. У фрагменті лістингу 4.2 наведено реалізацію алгоритму для формування таблиць станів та виходів КМ.

```
for i_row in range(0, m_size, 1):
        for j_col in range(0, n_size, 1):
                state_list[i_row] = rng.permutation(start_row)
                output_list[i_row][j_col] = rng.integers(1, m_size, endpoint=True)
```

Фрагмент лістингу програми 4.2 – Реалізація алгоритму формування таблиць станів та виходів КМ

У фрагменті лістингу 4.3 реалізовано збереження вихідних таблиць у файл за допомогою функції бібліотеки NumPy - savetxt() [53].

```
np.savetxt('{0}/output_matrix.txt'.format(path), state_list, fmt='%-2d', header='Таблиця виходів автомата')
np.savetxt('{0}/state_matrix.txt'.format(path), output_list, fmt='%-2d', header='Таблиця переходів автомата')
```

Фрагмент лістингу програми 4.3 – Збереження вихідних таблиць у файл

Відповідно на рисунках 4.1 та 4.2 зображено вміст цих файлів.

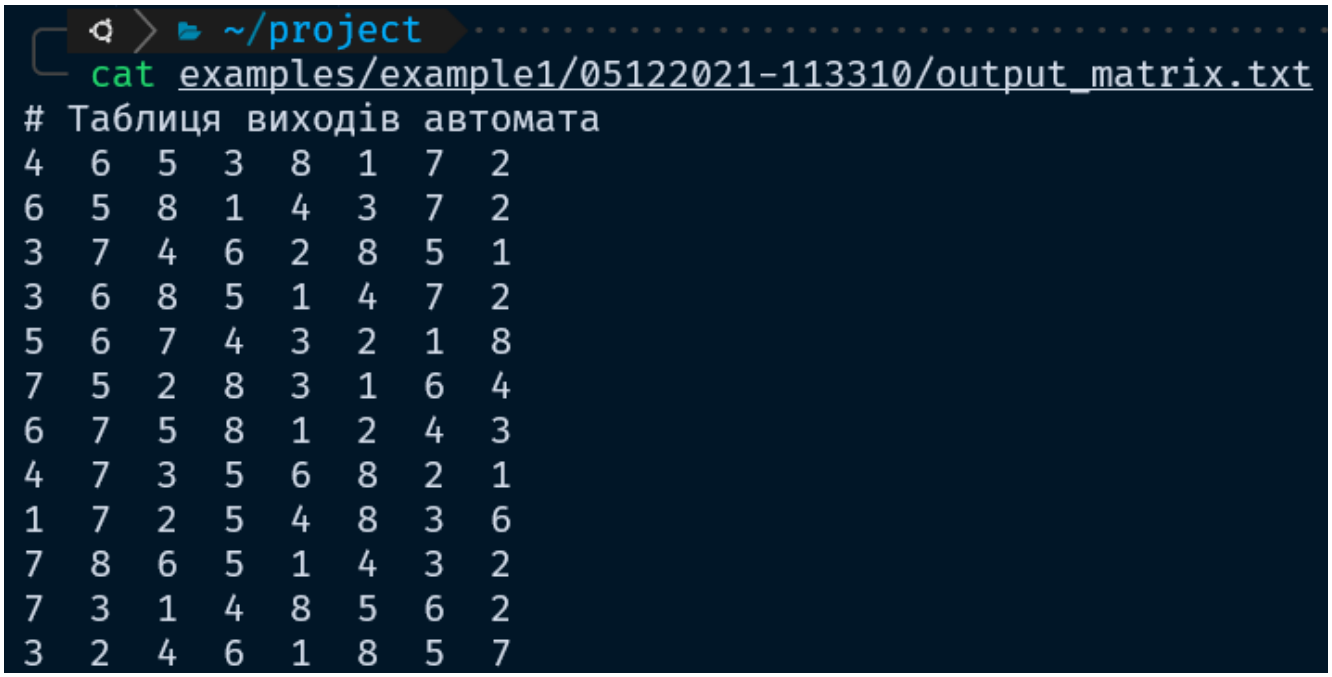


Рисунок 4.1 – Виведення змісту файлу з таблицею виходів автомата

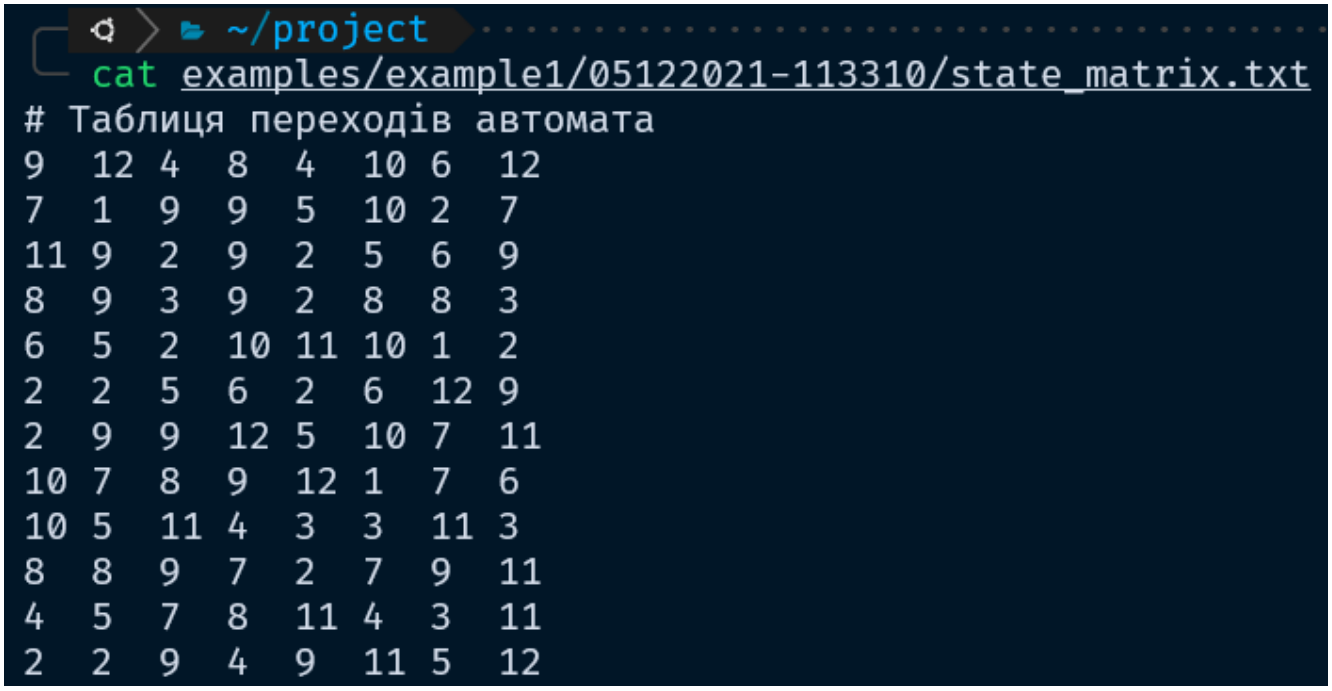


Рисунок 4.2 – Виведення змісту файлу з таблицею станів автомата

Вміст даних файлів набуває наступного вигляду в таблиці переходів (табл. 4.1) та таблиці станів (табл. 4.2) автомату.

Таблиця 4.1 - Таблиця переходів автомата для реалізації $F(\dot{X}_r)$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $y_4$ | $y_6$ | $y_5$ | $y_3$ | $y_8$ | $y_1$ | $y_7$ | $y_2$ |
| $s_2$ | $y_6$ | $y_5$ | $y_8$ | $y_1$ | $y_4$ | $y_3$ | $y_7$ | $y_2$ |
| $s_3$ | $y_3$ | $y_7$ | $y_4$ | $y_6$ | $y_2$ | $y_8$ | $y_5$ | $y_1$ |
| $s_4$ | $y_3$ | $y_6$ | $y_8$ | $y_5$ | $y_1$ | $y_4$ | $y_7$ | $y_2$ |
| $s_5$ | $y_5$ | $y_6$ | $y_7$ | $y_4$ | $y_3$ | $y_2$ | $y_1$ | $y_8$ |
| $s_6$ | $y_7$ | $y_5$ | $y_2$ | $y_8$ | $y_3$ | $y_1$ | $y_6$ | $y_4$ |
| $s_7$ | $y_6$ | $y_7$ | $y_5$ | $y_8$ | $y_1$ | $y_2$ | $y_4$ | $y_4$ |
| $s_8$ | $y_4$ | $y_7$ | $y_3$ | $y_5$ | $y_6$ | $y_8$ | $y_2$ | $y_1$ |
| $s_9$ | $y_1$ | $y_7$ | $y_2$ | $y_5$ | $y_4$ | $y_8$ | $y_3$ | $y_6$ |
| $s_{10}$ | $y_7$ | $y_8$ | $y_6$ | $y_5$ | $y_1$ | $y_4$ | $y_3$ | $y_2$ |
| $s_{11}$ | $y_7$ | $y_3$ | $y_1$ | $y_4$ | $y_8$ | $y_5$ | $y_6$ | $y_2$ |

Продовження таблиці 4.1

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_{12}$ | $y_3$ | $y_2$ | $y_4$ | $y_6$ | $y_1$ | $y_8$ | $y_5$ | $y_7$ |

Таблиця 4.2 - Таблиця станів автомата для реалізації $F(\dot{X}_r)$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $s_9$ | $s_{12}$ | $s_4$ | $s_8$ | $s_4$ | $s_{10}$ | $s_6$ | $s_{12}$ |
| $s_2$ | $s_7$ | $s_1$ | $s_9$ | $s_9$ | $s_5$ | $s_{10}$ | $s_2$ | $s_7$ |
| $s_3$ | $s_{11}$ | $s_9$ | $s_2$ | $s_9$ | $s_2$ | $s_5$ | $s_6$ | $s_9$ |
| $s_4$ | $s_8$ | $s_9$ | $s_3$ | $s_9$ | $s_2$ | $s_8$ | $s_8$ | $s_3$ |
| $s_5$ | $s_6$ | $s_5$ | $s_2$ | $s_{10}$ | $s_{11}$ | $s_{10}$ | $s_1$ | $s_2$ |
| $s_6$ | $s_2$ | $s_2$ | $s_5$ | $s_6$ | $s_2$ | $s_6$ | $s_{12}$ | $s_9$ |
| $s_7$ | $s_2$ | $s_9$ | $s_9$ | $s_{12}$ | $s_5$ | $s_{10}$ | $s_7$ | $s_{11}$ |
| $s_8$ | $s_{10}$ | $s_7$ | $s_8$ | $s_9$ | $s_{12}$ | $s_1$ | $s_7$ | $s_6$ |
| $s_9$ | $s_{10}$ | $s_5$ | $s_{11}$ | $s_4$ | $s_3$ | $s_3$ | $s_{11}$ | $s_3$ |
| $s_{10}$ | $s_8$ | $s_8$ | $s_9$ | $s_7$ | $s_2$ | $s_7$ | $s_9$ | $s_{11}$ |
| $s_{11}$ | $s_4$ | $s_5$ | $s_7$ | $s_8$ | $s_{11}$ | $s_4$ | $s_3$ | $s_{11}$ |
| $s_{12}$ | $s_2$ | $s_2$ | $s_9$ | $s_4$ | $s_9$ | $s_{11}$ | $s_5$ | $s_{12}$ |

У наступному фрагменті лістингу програми 4.4 реалізовано відображення $\dot{X}_r \rightarrow \dot{Y}_r$.

```
out_list = np.empty([m_size, n_size], dtype=int)
state = rng.integers(0, m_size)

for i_row in range(0, m_size, 1):
        for j_col in range(0, n_size, 1):
                out_list[i_row][j_col] = output_list[state][j_col]
                state = state_list[state][i_row]
```

Фрагмент лістингу програми 4.4 – Реалізація відображень

У фрагменті лістингу 4.5 зображено реалізацію алгоритму для генерування таблиць переходів і станів оберненого автомата.

```
invert_state_list = np.empty([m_size, n_size], dtype=int)
invert_output_list = np.empty([m_size, n_size], dtype=int)

for i_row in range(0, m_size, 1):
        for j_col in range(0, n_size, 1):
                for k in range(0, n_size, 1):
                        if output_list[i_row][k] == j_col:
                                invert_output_list[i_row][j_col] = k
                                invert_state_list[i_row][j_col] = state_list[i_row][k]
```

Фрагмент лістингу програми 4.5 – Реалізація алгоритму формування таблиць станів та виходів оберненого автомата

Реалізовані таблиці відповідно також були збережені у файл для подальшого аналізу. Фрагмент відповідного лістингу програми наведено у фрагменті 4.6. На рисунках 4.3 та 4.4 виведено вміст цих файлів.

```
np.savetxt('{0}/invert_output_matrix.txt'.format(path), invert_state_list, fmt='%-2d', header='Таблиця виходів оберненого автомата')
np.savetxt('{0}/invert_state_matrix.txt'.format(path), invert_output_list, fmt='%-2d', header='Таблиця переходів оберненого автомата')
```

Фрагмент лістингу програми 4.6 – Збереження вихідних таблиць оберненого автомата у файл

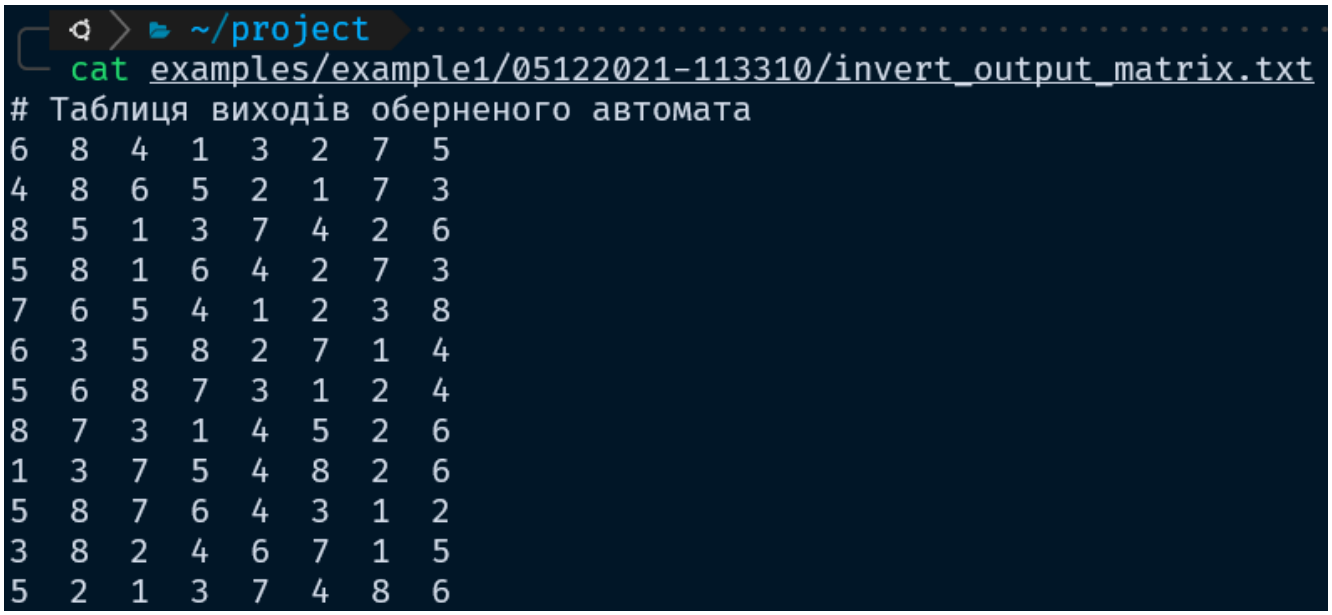


Рисунок 4.3 – Виведення змісту файлу з таблицею виходів оберненого автомата

```
~/project
cat examples/example1/05122021-113310/invert_state_matrix.txt
# Таблиця станів оберненого автомата
10 12 8  9  4  12 6  4
9  7  10 5  1  7  2  9
9  2  11 2  6  9  9  5
2  3  8  8  9  9  8  3
1  10 11 10 6  5  2  2
6  5  2  9  2  12 2  6
5  10 11 7  9  2  9  12
6  7  8  10 9  12 7  1
10 11 11 3  4  3  5  3
2  11 9  7  7  9  8  8
7  11 5  8  4  3  4  11
9  2  2  9  5  4  12 11
```

Рисунок 4.4 – Виведення змісту файлу з таблицею станів оберненого автомата

Аналогічно, вміст цих файлів набуває наступного вигляду в таблиці переходів (табл. 4.1) та таблиці станів (табл. 4.1) оберненого автомату.

Таблиця 4.3 - Таблиця переходів автомата для реалізації $F^{-1}(\dot{Y}_r)$

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $x_6$ | $x_8$ | $x_4$ | $x_1$ | $x_3$ | $x_2$ | $x_7$ | $x_5$ |
| $s_2$ | $x_4$ | $x_8$ | $x_6$ | $x_5$ | $x_2$ | $x_1$ | $x_7$ | $x_3$ |
| $s_3$ | $x_8$ | $x_5$ | $x_1$ | $x_3$ | $x_7$ | $x_4$ | $x_2$ | $x_6$ |
| $s_4$ | $x_5$ | $x_8$ | $x_1$ | $x_6$ | $x_4$ | $x_2$ | $x_7$ | $x_3$ |
| $s_5$ | $x_7$ | $x_6$ | $x_5$ | $x_4$ | $x_1$ | $x_2$ | $x_3$ | $x_8$ |
| $s_6$ | $x_6$ | $x_3$ | $x_5$ | $x_8$ | $x_2$ | $x_7$ | $x_1$ | $x_4$ |
| $s_7$ | $x_5$ | $x_6$ | $x_8$ | $x_7$ | $x_3$ | $x_1$ | $x_2$ | $x_4$ |
| $s_8$ | $x_8$ | $x_7$ | $x_3$ | $x_1$ | $x_4$ | $x_5$ | $x_2$ | $x_6$ |
| $s_9$ | $x_1$ | $x_3$ | $x_7$ | $x_5$ | $x_4$ | $x_8$ | $x_2$ | $x_6$ |
| $s_{10}$ | $x_5$ | $x_8$ | $x_7$ | $x_6$ | $x_4$ | $x_3$ | $x_1$ | $x_2$ |
| $s_{11}$ | $x_3$ | $x_8$ | $x_2$ | $x_4$ | $x_6$ | $x_7$ | $x_1$ | $x_5$ |
| $s_{12}$ | $x_5$ | $x_2$ | $x_1$ | $x_3$ | $x_7$ | $x_4$ | $x_8$ | $x_6$ |

Таблиця 4.4 - Таблиця станів автомата для реалізації $F^{-1}(\dot{Y}_r)$

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | $s_{10}$ | $s_{12}$ | $s_8$ | $s_9$ | $s_4$ | $s_{12}$ | $s_6$ | $s_4$ |
| $s_2$ | $s_9$ | $s_7$ | $s_{10}$ | $s_5$ | $s_1$ | $s_7$ | $s_2$ | $s_9$ |
| $s_3$ | $s_9$ | $s_2$ | $s_{11}$ | $s_2$ | $s_6$ | $s_9$ | $s_9$ | $s_5$ |
| $s_4$ | $s_2$ | $s_3$ | $s_8$ | $s_8$ | $s_9$ | $s_9$ | $s_8$ | $s_3$ |
| $s_5$ | $s_1$ | $s_{10}$ | $s_{11}$ | $s_{10}$ | $s_6$ | $s_5$ | $s_2$ | $s_2$ |
| $s_6$ | $s_6$ | $s_5$ | $s_2$ | $s_9$ | $s_2$ | $s_{12}$ | $s_2$ | $s_6$ |
| $s_7$ | $s_5$ | $s_{10}$ | $s_{11}$ | $s_7$ | $s_9$ | $s_2$ | $s_9$ | $s_{12}$ |
| $s_8$ | $s_6$ | $s_7$ | $s_8$ | $s_{10}$ | $s_9$ | $s_{12}$ | $s_7$ | $s_1$ |
| $s_9$ | $s_{10}$ | $s_{11}$ | $s_{11}$ | $s_3$ | $s_4$ | $s_3$ | $s_5$ | $s_3$ |
| $s_{10}$ | $s_2$ | $s_{11}$ | $s_9$ | $s_7$ | $s_7$ | $s_9$ | $s_8$ | $s_8$ |
| $s_{11}$ | $s_7$ | $s_{11}$ | $s_5$ | $s_8$ | $s_4$ | $s_3$ | $s_4$ | $s_{11}$ |
| $s_{12}$ | $s_9$ | $s_2$ | $s_2$ | $s_9$ | $s_5$ | $s_4$ | $s_{12}$ | $s_{11}$ |

Висновки до розділу

У даному розділі було реалізовано програмний засіб для аналізу регулярних структур лінійної складності засобами мови Python з використанням засобів бібліотеки NumPy. Безпосередньо були створені алгоритми для формування таблиць переходів та станів звичайного та оберненого автоматів.

# ВИСНОВКИ

Головною метою даної магістерської дисертації було створення та дослідження засобів аналізу регулярних структур.

У першому розділі були розглянуті сфери застосування перестановок, та здобутки у їх дослідженнях, окреслені здобутки у дослідженнях регулярних структур лінійної складності.

Були опрацьовані роботи із дослідженням регулярних структур лінійної складності. Проаналізовано існуючі алгоритми для формування конструктивних модулів, та операції підстановок, які реалізуються на цих структурах.

У рамках даної роботи також були проаналізовані засоби стандартної бібліотеки Python із засобами бібліотеки NumPy, яка застосовується для роботи з даними. Було розглянуто основні характеристики цих бібліотек, їх переваги та недоліки. Окреслено основні відмінності методів кожного із засобів, для використання в аналізі регулярних структур. Варто зазначити, що опис засобів стандартної бібліотеки мови Python є досить повним, в той час як документація бібліотеки NumPy має лише короткий опис по застосовуванню тих чи інших модулів.

Науковою новизною даної роботи, є модифікація одного з розглянутих алгоритмів, який використовувався для формування таблиць станів та виходів конструктивних модулів. Результати наукової роботи були представлені на двох конференціях:

- IV Міжнародна Науково-Практична Конференція «Теоретичні Та Практичні Аспекти Розвитку Науки» (м. Львів, 23-24 листопада 2021 року);
- міжнародна науково-практична конференція «Наука, освіта, технології, інновації: тенденції, виклики, перспективи» (м. Полтава, 30 листопада 2021 року).

Даний програмний засіб також має напрямки для подальшого розвитку. Можна зробити даний засіб більш мобільним шляхом розгортання даної системи в хмарних сервісах у разі потреби для розрахунків даних у великих об'ємах, що з часом буде необхідністю.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. В. П. Тарасенко, О.К. Тесленко, О. Ю. Яновська Реалізація повних підстановок за допомогою багатомодульного каскаду найпростіших конструктивних модулів. Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні, випуск 2 (17), 2008.
2. Тесленко О. К., Бондарчук М. Ю. Реалізація підстановок довільної розрядності в одному із класів лінійних структур. Вісник сучасних інформаційних технологій. 3, 1 (Лют 2020), 406–417.
3. Peng, Li, "The Generation of (n,n(n−1),n−1) Permutation Group Codes for Communication Systems", Communications IEEE Transactions on, vol. 67, no. 7, pp. 4535-4549, 2019.
4. Boss E., Grosso V., Tim Guneysu T., et al. Strong 8-bit sboxes with efficient masking in hardware // J. Cryptographic Engineering. 2017. No.7(2). P.149–165.
5. Д. Б. Фомин, Д. И. Трифонов Об аппаратной реализации одного класса байтовых подстановок ПДМ. Приложение, 2019, № 12. страницы 134–137.
6. Р. Олійников, І. Горбенко, О. Казимиров, В. Руженцев, Ю. Горбенко Принципи побудови і основні властивості нового національного стандарту блокового шифрування України захист інформації, том 17, №2, квітень-червень 2015 с.142-157.
7. "Advanced Encryption Standard (AES)" (PDF). Federal Information Processing Standards. 26 November 2001.
8. В. П. Тарасенко, О. К. Тесленко, О. Ю. Яновська Проблеми апаратної реалізації підстановок Наукові записки УНДІЗ, №2, 2007, с 52-58.
9. В. П. Тарасенко, О. К. Тесленко, О. Ю. Яновська Властивості повних підстановок, які реалізуються найпростішим однонаправленим регулярним ОККМ Радіоелектронні і комп’ютерні системи. №6, 2010, с.123-128.
10. В. П. Тарасенко, О. К. Тесленко, О. Ю. Яновська Можливості найпростіших двонаправлених регулярних одновимірних каскадів конструктивних модулів

щодо реалізації різних повних підстановок. Радіоелектронні і комп'ютерні системи, 2012, №7(59), с. 147-153.

11. Глушков В.М. Синтез цифрових автоматов. Москва (1967).
12. Михайлюк А. Ю., Тарасенко В. П., Тесленко А. К. "Анализ эффективности применения нетрадиционных элементарных функций шифрования". Матеріали Третьої міжнародної науково-практичної конференції «Безпека інформації в інформаційно-телекомунікаційних системах» К.2000. с. 161-168.
13. Karandashov M. V. Issledovanie biektivnykh avtomatnykh otobrazhenii na kol'tse vychetov po moduliu $2^k$[Research bijective automaton mappings on the ring of residues modulo $2^k$]. Komp'iuternye nauki i informatsionnye tekhnologii: materialy Mezhdunar. nauch. konf. [Computer Science and Information Technologies: Proc. Intern. Sci. Conf.]. Saratov, Publ. Center "Nauka", 2014, pp. 148–152.
14. Gill A. Introduction to the theory of finite-state machines. New York, Toronto, Ontario, London, McGraw-Hill Book Co., Inc., 1962. 207 p. (Russ. ed. : Gill A. Vvedenie v teoriiu konechnykh avtomatov. Moscow, Nauka, 1966. 272 p.)
15. McCoy, Neal H. Introduction To Modern Algebra, Revised Edition, Boston: Allyn and Bacon, 1968.
16. Nering, Evar D., Linear Algebra and Matrix Theory (2nd ed.), New York: Wiley, 1970.
17. "Bijection", Encyclopedia of Mathematics, EMS Press, 2001 [1994].
18. Webster's Seventh New Collegiate Dictionary, Springfield: G. & C. Merriam Company, 1969.
19. Dershowitz, Nachum, "A simplified loop-free algorithm for generating permutations", Nordisk Tidskr. Informationsbehandling (BIT), 1975, 15 (2): 158–164.
20. Heap, B. R., "Permutations by Interchanges". The Computer Journal, 1963, 6 (3): 293–298.

21. Knuth, Donald Generating All Tuples and Permutations, The Art of Computer Programming, 4, Addison–Wesley, 2005.
22. Zaks, S. "A new algorithm for generation of permutations". BIT Numerical Mathematics, 1984, 24 (2): 196–204.
23. Sawada, Joe; Williams, Aaron. "A Hamilton path for the sigma-tau problem". Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018. New Orleans, Louisiana: Society for Industrial and Applied Mathematics (SIAM), 2018. pp. 568–575.
24. Corbett, P. F. "Rotator graphs: An efficient topology for point-to-point multiprocessor networks". IEEE Transactions on Parallel and Distributed Systems, 1995. 3 (5): 622–626.
25. Arndt J. Permutations. In: Matters Computational. Springer, Berlin, Heidelberg, 2011.
26. Arndt J. Gray codes for strings with restrictions. In: Matters Computational. Springer, Berlin, Heidelberg, 2011.
27. PYPL PopularitY of Programming Language: URL: https://pypl.github.io/PYPL.html (дата звернення: 01.12.2021)
28. index | TIOBE - The Software Quality Company: URL: https://www.tiobe.com/tiobe-index/ (дата звернення 02.12.2021)
29. C# vs Python: Head to Head Comparison [Updated]: URL: https://hackr.io/blog/c-sharp-vs-python (дата звернення: 10.09.2021)
30. Python Programming - The State of Developer Ecosystem in 2021 Infographic | JetBrains: Developer Tools for Professionals and Teams: URL: https://www.jetbrains.com/lp/devecosystem-2021/python/#Python_what-do-you-use-python-for (дата звернення: 30.11.2021)
31. Python Programming - The State of Developer Ecosystem in 2021 Infographic | JetBrains: Developer Tools for Professionals and Teams: URL: https://www.jetbrains.com/lp/devecosystem-2021/python/#Python_what-data-science-frameworks-do-you-use-in-addition-to-python (дата звернення: 30.11.2021)

32. What is NumPy? — NumPy v1.21 Manual: URL: https://numpy.org/doc/stable/user/whatisnumpy.html (дата звернення: 25.11.2021)
33. Difference Between Python List and NumPy Array | by Leonie M Windari | Python in Plain English: URL: https://python.plainenglish.io/python-list-vs-numpy-array-whats-the-difference-7308cd4b52f6 (дата звернення: 20.08.2021)
34. NumPy: URL: https://devopedia.org/numpy (дата звернення: 27.11.2021)
35. Beating NumPy performance speed by extending Python with C | by Yan Yanchii | Analytics Vidhya | Medium: URL: https://medium.com/analytics-vidhya/beating-numpy-performance-by-extending-python-with-c-c9b644ee2ca8 (дата звернення: 30.09.2021)
36. itertools — Functions creating iterators for efficient looping — Python 3.10.1 documentation: URL: https://docs.python.org/3/library/itertools.html#module-itertools (дата звернення: 15.02.2021)
37. itertools — Functions creating iterators for efficient looping — Python 3.10.1 documentation: URL: https://docs.python.org/3/library/itertools.html#itertools.permutations (дата звернення: 15.02.2021)
38. Random Generator — NumPy v1.21 Manual: URL: https://numpy.org/doc/stable/reference/random/generator.html (дата звернення: 13.09.2021)
39. NEP 19 — Random number generator policy — NumPy Enhancement Proposals: URL: https://numpy.org/neps/nep-0019-rng-policy.html (дата звернення: 03.12.2021)
40. random — Generate pseudo-random numbers — Python 3.10.1 documentation: URL: https://docs.python.org/3/library/random.html#module-random (дата звернення: 15.02.2021)
41. E.g. Marsland S. Machine Learning (CRC Press), §4.1.1, 2011.

42. Matsumoto, M.; Nishimura, T. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator" (PDF). ACM Transactions on Modeling and Computer Simulation, 1998. 8 (1): 3–30.
43. John Savard. "The Mersenne Twister". "A subsequent paper, published in the year 2000, gave five additional forms of the Mersenne Twister with period 2^19937-1. All five were designed to be implemented with 64-bit arithmetic instead of 32-bit arithmetic."
44. P. L'Ecuyer and R. Simard, "TestU01: "A C library for empirical testing of random number generators", ACM Transactions on Mathematical Software, 33, 4, Article 22 (August 2007).
45. Route, Matthew. "Radio-flaring Ultracool Dwarf Population Synthesis". The Astrophysical Journal, August 10, 2017. 845 (1)
46. "SIMD-oriented Fast Mersenne Twister (SFMT): twice faster than Mersenne Twister". Japan Society for the Promotion of Science.
47. Makoto Matsumoto; Takuji Nishimura. "Dynamic Creation of Pseudorandom Number Generators" (PDF).
48. Hiroshi Haramoto; Makoto Matsumoto; Takuji Nishimura; François Panneton; Pierre L'Ecuyer. "Efficient Jump Ahead for F2-Linear Random Number Generators" (PDF).
49. mt19937ar: Mersenne Twister with improved initialization: URL: http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/MT2002/emt19937ar.html (дата звернення: 03.12.2021)
50. Fog, Agner. "Pseudo-Random Number Generators for Vector Processors and Multicore Processors". Journal of Modern Applied Statistical Methods, 1 May 2015. 14 (1): 308–334.
51. random — Generate pseudo-random numbers — Python 3.10.1 documentation: URL: https://docs.python.org/3/library/random.html#random.randrange (дата звернення: 20.02.2021)

52. numpy.random.Generator.integers — NumPy v1.21 Manual: URL: https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.integers.html#numpy.random.Generator.integers (дата звернення: 27.10.2021)
53. numpy.savetxt — NumPy v1.21 Manual: URL: https://numpy.org/doc/stable/reference/generated/numpy.savetxt.html (дата звернення: 30.10.2021)