Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

«До захисту допущено»
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ДИПЛОМНИЙ ПРОЕКТ

освітньо-кваліфікаційного рівня "Спеціаліст"
зі спеціальності 7.05010201 "Комп'ютерні системи та мережі"
на тему:

ПРОГРАМНА СИСТЕМА РЕАЛІЗАЦІЇ ПОШУКУ ЗАЛИШКІВ ВЕЛИКОРОЗРЯДНИХ ЧИСЕЛ

Студент
групи КСМзс-51_____Ставнійчук О.М.
(підпис)

Керівник:
викладач _____Якименко І.З.
(підпис)

Нормоконтроль
к.т.н., доцент _____Васильків Н.М.
(підпис)

Консультант
з охорони праці
доцент _____Сапожник Г.В.
(підпис)

2012

Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії
спеціальність 7.05010201 – "Комп'ютерні системи та мережі"

"Затверджую"
завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА
Ставнійчука Олега Миколайовича

1. **Тема проекту:** "Програмна система реалізації пошуку залишків великорозрядних чисел" затверджена наказом університету № 475 від 14 жовтня 2012 р.

2. **Термін здачі студентом закінченого проекту** "___"_____20__ р.

3. **Вихідні дані для проекту:** Технічне завдання.

4. **Перелік задач, які мають бути вирішені:**
- провести аналіз існуючих рішень задачі, особливостей застосування в прикладгних задачах;
- проаналізувати теоретико-числові базиси, та встановити ефективність їх застосування для реалізації пошуку залишків велико розрядних чисел;
- розробити програмну систему реалізації пошуку залишків велико розрядних чисел;
- розробити математичне та алгоритмічне забезпечення для реалізації операції пошуку залишків;
- розробити структуру програмного забезпечення інтерактивного керування між окремими модулями;
- виконати тестування програмного сервісу.

5. **Перелік графічного матеріалу** (з точним вказанням обов'язкових креслень)
- Блок-схема алгоритму пошуку залишків великорозрядних чисел на основі використання розмежованої системи числення Радемахера-Крестенсона
- Блок-схема алгоритму пошуку залишків великорозрядних чисел на основі використання десяткової системи числення

– Застосування операції пошуку залишків великорозрядних чисел в системах захисту інформації. Схема структурна

– Взаємозв’язок функцій в бібліотеці опрацювання великорозрядних чисел. Схема структурна

6.Консультанти по проекту (із зазначенням розділів):

Розділ	Консультант	Підпис
Охорона праці	Сапожник Г.В.	

КАЛЕНДАРНИЙ ПЛАН

№	Назва розділів дипломного проекту	Термін виконання	Позначки керівника про виконання завдань
1	Аналіз методів опрацювання великорозрядних чисел в різних теоретико-числових базисах	15.09.2011 – 5.11.2011	
2	Теорія та алгоритми пошуку залишків великорозрядних чисел	6.11.2011 – 31.01.2012	
3	Організація інформаційного, програмного та технічного забезпечення	1.02.2012 – 14.04.2012	
4	Охорона праці	15.04.2012 – 23.04.2012	

Завдання прийняв до виконання _____
(підпис)

Керівник дипломного проекту _____
(підпис)

АНОТАЦІЯ

Робота виконана на ___ сторінках, з них ___ сторінок основного тексту, містить ___ рисунків, __таблиць, __ джерел посилань на __ сторінках, __ додатків.

Метою дипломного проекту є розробка програмної системи реалізації пошуку залишків великорозрядних чисел.

У дипломному проекті проаналізовано метод пошуку залишків з використанням десяткової системи числення та встановлено, що він характеризується великою часовою складністю. Тому, для зменшення складності, запропоновано метод з використанням розмежованої системи числення, що дозволило зменшити часову складність на 1-2 порядки. На основі математичних викладок, побудовано алгоритм пошуку залишків великорозрядних чисел та розроблено програмну систему.

Проаналізовано методи опрацювання великорозрядних чисел в різних теоретико-числових базисах. Розроблена математична та алгоритмічна основа методу залишків. Розроблено структуру програмного сервісу. Здійснено опис основних алгоритмів і процедур програмного сервісу. Розроблений програмний сервіс є гнучким та універсальним, тому в подальшому можна розширювати його функціональність.

Дипломний проект містить структурні схеми застосування операції пошуку залишків великорозрядних чисел в системах захисту інформації та взаємозв'язоку функцій в бібліотеці опрацювання великорозрядних чисел, блок-схеми алгоритму пошуку залишків великорозрядних чисел на основі використання розмежованої системи числення Радемахера-Крестенсона та десяткової системи числення , які подані в графічній частині.

Дипломний проект має практичну спрямованість і його результати плануються до впровадження.

ANNOTATION

The work performed at ___ pages, including ___ pages main text, contains ___ figures, ___ tables, ___ references sources on ___ pages, ___ applications.

In the thesis project analyzes the method of finding residues using the decimal number system and found that it is characterized by a large time complexity. Therefore, to reduce complexity, the method using the divided number system, which allowed to reduce the time complexity of 1-2 orders of magnitude. Based on mathematical calculations, the algorithm search for the remnants large numbers and developed a software system.

The methods of processing large numbers in various theoretical and numerical bases. The mathematical and algorithmic basis of the method of residues. The structure of software services.Done description of the basic algorithms and procedures for program services. Developed a software service is a flexible and versatile, so you can further expand its functionality.

The degree project contains block diagrams use search operation remains velykorozryadnyh numbers in the systems of information protection and interconnection of functions in the library working large numbers flowchart search algorithm remains lsrge numbers on the basis of divided Rademacher radix-Krestensona and decimal number system, submitted in graphical part.

The degree project is practical and the results are planned for implementation.

ЗМІСТ

Вступ

Знаходження залишків чисел великої розрядності (ЧВР) є важливою фундаментальною задачею теорії чисел, успішне вирішення якої дозволяє вдосконалити алгоритми широкого класу прикладних задач, особливо задач захисту інформаційних потоків в комп'ютерних системах з використанням асиметричної криптографії (алгоритмів RSA, Рабіна, Ель-Гамаля, з використанням математичних основ еліптичних кривих, електронного цифрового підпису, дослідження порядку еліптичної кривої за допомогою алгоритму Шуфа).

В зв'язку з цим актуальною задачею, яка розглядається в дипломному проекті, є розробка теоретичних основ пошуку залишків ЧВР з використанням теоретико-числових базисів Радемахера та Крестенсона, застосування яких дозволяє зменшити часову складність.

Найбільш розповсюдженим методом знаходження залишків є один з математичних алгоритмів, який ґрунтується на використанні десяткової системи числення, згідно якого для знаходження залишків ЧВР необхідно виконати ділення, виділити цілу частину від ділення, знайти добуток цілої частини на модуль та різницю ЧВР та знайденого добутку. Також можна декілька разів від ЧВР відняти модуль, поки різниця не стане меншою від'ємника. Оскільки числа, над якими виконуються операції, на кожному кроці зменшуються, то такий процес не може тривати нескінченно, а закінчиться через деяке число кроків. Дані алгоритми можна легко реалізувати програмним шляхом, але їх часова складність залишається великою, оскільки операція ділення є досить трудомісткою.

Іншим недоліком алгоритмів знаходження залишків ЧВР є послідовність виконання вище перелічених операції, тобто неможливість розпаралелення їх роботи.

Перспективним напрямком вдосконалення досліджуваних алгоритмів є розробка теорії їх реалізації на основі розмежованої системи числення залишкових класів.

Для зменшення часової складності потрібно розробляти високопродуктивні алгоритми знаходження залишків ЧВР за допомогою розмежованої системи числення та базису Крестенсона.

1 АНАЛІЗ МЕТОДІВ ОПРАЦЮВАННЯ ВЕЛИКОРОЗРЯДНИХ ЧИСЕЛ В РІЗНИХ ТЕОРЕТИКО-ЧИСЛОВИХ БАЗИСАХ

1.1Теоретико-числові базиси

1.1.1 Математичні основи теоретико-числових базисів.

Фундаментальною основою теоретико-числових базисів (ТЧБ) є теорія чисел, вища алгебра та теорія функцій комплексного змінного.

Значний вклад у розвиток математичних основ ТЧБ та їх практичного застосування для створення технічних засобів цифрового опрацювання сигналів (ЦОС) внесли І.Я. Акушський, Л.В. Варіченко, М.Г Карповський, В.Г. Лабунець, А.М Трахтман, Р.Г. Фараджев, Н. Ахмед і Н. Рао, Р. Аргевал та інші.

Особливо застосування потужного математичного апарату теорії чисел, системи діофантових рівнянь, полів Галуа (рисунок 1.1), залишкових класів та груп Абеля.

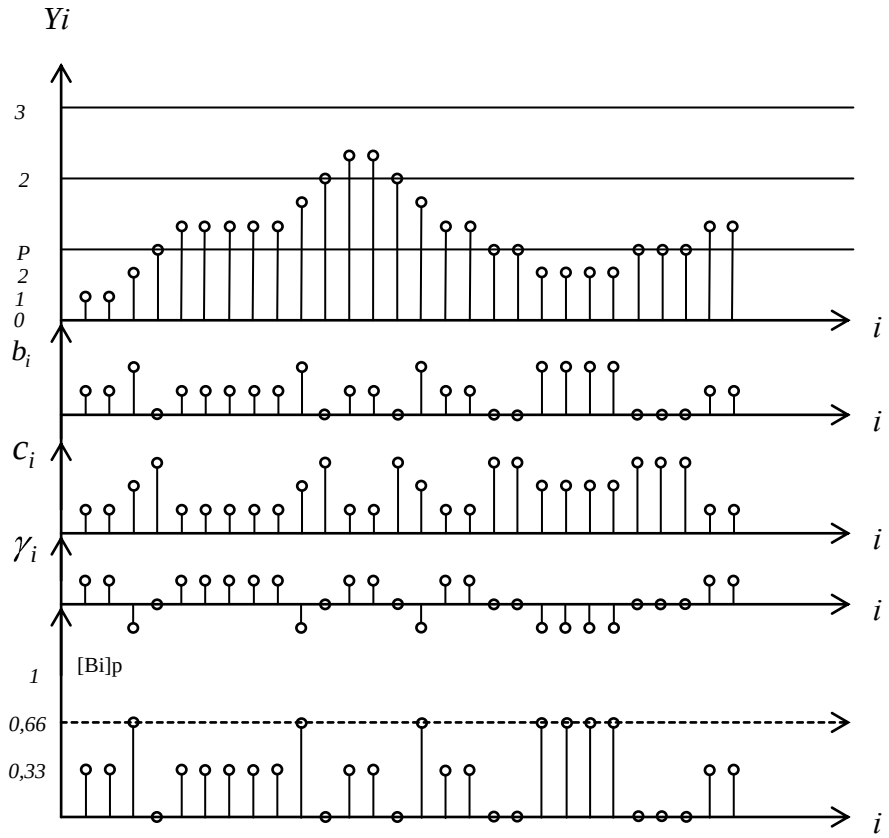


Рисунок 1.1 – Кодування решітчатої функції залишками по модулю у базисі Галуа.

Найбільш загальним поняттям ТЧБ є алгебраїчна система, яка визначається властивостями Абелевої групи.

Важливо, що Абелева група комутативна:

$$x * y = y * x$$

відносно операцій (*) додавання та множення.

Число Абелевих груп рівне числу довільних розкладів числа в суму невідємних доданків.

Додавання елементів у Абелевій групі виконується по модулю:

$$Z_i = x_i \otimes y_i (\mathrm{mod}\, P),$$

де P-ціле число.

Поля та кільця є алгебрами з двома бінарними операціями. Прикладом кілець є поля цілих Z, раціональних Q, дійсних R, комплексних C, та гіперкомплексних G чисел.

Множина значень дискретних відліків сигналу, представлено решітчатою функцією у Хемінговому просторі є підмножиною кільця цілих чисел або поля комплексних чисел.

Важливим типом кілець, які породжують ТЧБ, коди та системи числення є кільця F(x) поліномів від однієї змінної з коефіцієнтами з деякого поля.

$$F()x = a_0 + a, x + a_2 x^2 + ...a_m x^m = \sum_{i=0}^{m} a_i x^i ,$$

де a_i – коефіцієнти полінома, які належать полю P.

Глибока подібність властивостей кілець Z та F(x) дозволяють будувати алгебраїчні структури в одному кільці та використовувати їх рішення в іншому.

Виходячи з кільця Z, можна адекватно побудувати кільця класів залишків по модулю P та кінцеві поля Галуа GF(P).

Наприклад повна система залишків по модулю простого числа P утворює просте поле Галуа.

У кільці GF(P)[x] існують незвідні поліноми будь якої степені. Тому можна отримати різні розширення поля GF(P). Одним з популярних розширень що застосовуються у практиці цифрового опрацювання сигналів є складене поле Галуа $GF(P^n)$, n – ціле. Це поле належить до класу гіперкомплексних полів.

Складене поле $GF(P^n)$, володіючи подібним порядком, як і поле $GF(P)$ при суттєво менших значеннях P забезпечує паралельне виконання операцій над компонентами багаточлена, який є елементом поля $GF(P^n)$.

В теорії та практиці формування перетворення та цифрового опрацювання інформаційних потоків отримали кільця залишків по модулю цілого числа Zp.

Поняття “залишок” відноситься до теорії чисел і визначається лінійною формою $y = ap + b,$ яка відповідає порівнянню:

$$y \equiv b(\mathrm{mod}\ P)\ або\ b = res\ y(\mathrm{mod}\ P),$$

де y,a,b,p – цілі числа a – член числа y по модулю P:

$$a = \breve{E}\left[\frac{y}{p}\right],$$

де $\breve{E}[\ ]$ – цілочисельна функція з округленням до меншого цілого;

b – найменший невідємний залишок $a \le b \le p-1;$

$\equiv \mathrm{mod}$, res– відповідно символи порівняння, модуля та операції добування залишку.

Число залишків рівне P та рівне числу класів по mod P.

Існує чотири типи залишків (див. рисунок1.1)

b – найменші невідємні $0,1,2,...,P-1$;

c – найменші додатні $1,2,...,P-1,P$;

γ – найменші (по абсолютному значенню) $-(P-1)/2,...,-1,0,1,2,...,(P-1)/2$;

$[b]_p$ – нормовані по модулю

$$0/P \le [b]_p \le (P-1)/P:,\ [b]_p = b/P.$$

У базисах класифікованих залишків можливо побудувати чотири еквівалентні теоретичні системи, що відрізняються складністю перетворень та технічною реалізацією процесів.

Наприклад: $Y = 24_{(10)}, P = 5,$ тоді $b = res24(\mathrm{mod}5) = 4;$

$c = res24(\mathrm{mod}5)+1 = 5\ \ \gamma = b - P = 4-5 = -1;\ [b]_p = 4/5 = 0,8.$

Якщо число У представляється у 2-й степені числення, тобто $Y = 11001_{(2)}$, то нормалізований залишок $[b]_p$ записується у нормалізованій формі двійкового числа з фіксованою комою.

$$[b]_p = [4]_5 = 0,11001100..$$

При цьому операція додавання по модулю двох залишків в нормалізованій формі виконується по “mod 1” , тобто шляхом відкидання цілої частини отриманої суми числа $[b]_b$ записаних у 2-й системі числення з фіксованою комою.

Наприклад: $b_1 = 4; b_2 = 3; P = 5,$, отже

$b_0 = (b_1 + b_2) \bmod P = (4+3)(\mathrm{mod}5) = 2.$

Аналогічно: $[b_1]_5 = 0,110011_{(2)j} [b_2]_5 = 0,100110_{(2)},$

Звідси $[b_1]_5 = 0,110011$

$$[b_1]_5 = \frac{0{,}110011}{1{,}011001} \ (\mathrm{mod}1) = 0{,}11001_{(2)} = b_0.$$

Відновлення залишку модульної операції додавання нормалізованих значень $[b_i]_5$ виконується згідно виразу $b_j = \hat{E}([b_0]P),$, тобто $b_j = \hat{E}(0{,}011001{*}101) = 1{,}111101_{(2)} = 2$

Виконання операції піднесення до квадрату залишку b_i по модулю P виконується згідно виразу $b_{i+1} = res b_i^2 (\mathrm{mod} P)$ і потребує у цілочисельній формі послідовності виконання наступних операцій :

1) множення $b_i b_i = b_i^2;$

2) ділення $b_i^2;$ на P;

3) виділення залишку від ділення b_{i+1} шляхом відкидання цілої частини (рангу) у результаті не ділення $a, b_{i+1} \Rightarrow b_{i+1};$

У нормалізованій формі ці операції спрощуються і виконуються згідно виразу (1.1)

$$b_{i+1} = \hat{E}(res[b_i^2]_p \ \mathrm{mod}1) \qquad (1.1)$$

наступним чином :

1) Множення $b_i[1]_p = [b_i]_p$, де $[1]_p = b_i / P$;

2) Множення $b_i[1]_p = [b_{i+1}]_p;$

3) Множення з округленням до більшого цілого $\hat{E}(P[b_{i+1}]_p) = b_{i+1}$

Наприклад: $b_i = 5;\ P = 11,$ визначити $b_{i+1} = res 5^2 (\mathrm{mod}11) = 3.$

1) 5*0,09…=0,45…=$[5]_{11}$;

2) 5*0,45…=0,27…=$[5^2]_{11}$;

3) Ê(11*0,45) = Ê(2,99…) = 3 .

Таким чином операція ділення b_i^2 на модуль P у нормалізованій формі замінюється операцією відкидання цілої частини від результату перемноження $b_i[b_i]_p$ Очевидно, що нормалізоване значення $[1_p]$, при відомому P, що має місце у задачах криптографії, може бути визначене наперед і вибиратися з таблиці.

Теорія залишків базису Галуа також охоплює кільце цілих комплексних чисел $Z = a + bi; i = \sqrt{-1}$.

Причому в комплексній області залишок відповідає аргументу комплексного числа, який також описується лінійною формою по модулю

P= 2π, що слідує з формул Ейлера

$\omega = e^{-iZ} = \cos Z + i \sin Z,$

де $Z = Z_p(\cos\gamma + i\sin\gamma),$

Z_p – модуль комплексного числа, γ – аргумент.

$a = \sqrt[r]{-1} \in Z_p$ і в полі комплексних чисел $\sqrt[r]{-1} = e^{-\frac{2\pi}{p}} \in a$.

При цьому функції $\psi_a(t) = a^{at}$ утворюють ортогональний базис у просторі функцій, заданих на цілочисельному відрізку 0, T(a,p) -1. Базиси утворювані в такому випадку відрізняються тим, що дозволяють позбутися комплексних множень на числа виду $\exp\left(-i\frac{2\pi}{P}dt.\right)$. Вибираючи a=$2^k$, k=1,2,… отримують найбільше спрощення реалізації базисних операцій програмно-апаратними засобами.

З метою забезпечення високої точності опрацювання інформації у базисах полів Галуа, необхідно вибирати великі значення простих модулів P, що задовольняють Діофантовому рівнянню $2^q \equiv 1(2^q - 1),$ що приводить до арифметики по модулю $P = 2^k - 1$ і $P = 2^k + 1.$

Такими числами є відомі числа Мерсена $P = 2^q - 1,$ де q – просте число, і Ферма $F_n = 2^{2^n} + 1,$ де n –ціле число.

Числа Мерсена широко застосовуються у комплексних ТЧБ Фур'є-Галуа над полем GF(P^2) з числом точок $N = 2^q$.

Окремим випадком токого класу ТЧБ є базис "Крестенсона-Галуа.

Застосування чисел Ферма в якості модулів ТЧБ відоме під назвою перетворення Рейдера. Відомі також сукупності простих чисел, які використовуються у цифрових перетворювачах Фур'є-Шевілла.

У загальному, перетворення на базі чисел Мерсена найбільш прості в технічній реалізації у порівнянні з іншими арифметиками.

Загальними обмеженнями описаних перетворень, основаних на властивостях простих та розширених полів Галуа є :

1) Обмежені функціональні можливості та реалізація тільки перетворень Фур'є;

2) Особливі обмеження по числу точок перетворення та точності обчислень.

ТЧБ на основі сум полів Галуа.

При опрацюванні багатомірних сигналів та інформаційних потоків використовують перетворення над зваженими та прямими сумами полів Галуа.

У першому випадку теоретичною основою перетворень є Китайська теорема про залишки, з якої слідує:

Якщо $P_1P_2...P_n = P$ прості, то кільце Z_P класів залишків по модулю P ізоморфне прямій сумі полів $GF(P_i)$:

$$Z_p \approx GF(P_1) + GF(P_2) + ... + GF(P_k).$$

Прямий та зворотній ізоморфізм таких зважених полів заданий відображенням.

$\psi_1 : a \to (a_1, a_{2,}..., a_k)$,

Де aZ_p; $a \equiv a_i (\mathrm{mod}\, P_i)$; i Є $1,k$;

$\psi_1 : a \to (a_1, a_{2,}..., a_k) \to a$,

Де $a = M_1 M_1^{-1} a_1 + ... M_k M_k^{-1} Q_k$;

У другому випадку $P_1, P_2, \ldots, P_к$ – взаємно-прості , а всі $M_1^{-1} = 1$, і $\in 1, k$.

Викладені теоретичні засади утворення ТЧБ показують, що відомі базиси Крестернсона, Радемахера-Крестенсена та Крестензона-Галуа, які засновані на властивостях згаданої Китайської теореми про залишки.Своєю фундаментальною основою базуються на властивостях зважених та прямих сум полів Галуа.

Породжені такими ТЧБ системи числення та коди поля Галуа є фундаментальними теоретичними основами сучасної арифметики числень. Особливо це стосується системи числення залишкових класів базису Крестенсона та кодових систем Галуа.

1.2 Теоретико-числові базиси на основі кусково-постійних ортогональних функцій

До основних дискретних теоретико-числових базисів належать унітарні функції та коди, функції Хаара та розрядно-позиційні коди, дискретно-фазові функції та коди Лібова-Крейга, функції Радемахера та двійкові коди, функції Грея та коди Грея, функції Уолша, функції Ґалуа та кодові системи Ґалуа. Вибір кодової системи, базису або системи функцій залежить від задачі, властивостей інформаційного потоку, умов застосування даних та інш.

Зокрема, базисами для виконання дискретних ортогональних перетворень і дискретного подання одновимірного інформаційного потоку зі скінченною енергією, визначеного в просторі $L_2[a,b]$ на часовому інтервалі $T=[a,b]$, є повні ортонормовані системи функцій. Вейвлет-аналіз здійснюється на основі ортогональних або базисів Ріса. Повними ортонормованими системами функцій у просторі $L_2[a,b]$ є тригонометрична система та дискретні експоненціальні функції – як базис перетворення Фур'є, системи Уолша, Хаара, функції пилкоподібного базису, Віленкіна-Крестенсона та інші, проте актуальною

залишається задача визначення галузей, способів і методів їх ефективного застосування, формування та дослідження інших базисів. Визначення особливостей та ефективності застосування різних систем функцій для виконання дискретних перетворень і аналізу інформаційних потоків зумовлює необхідність дослідження властивостей цих систем, наведених у наступних викладках.

1.2.1 Унітарний базис

В якості вихідних у засобах перетворення форми інформації широкого застосування набули унітарні коди, розрядність бінарного подання слова яких відповідає повній шкалі квантування діапазону перетворення *N*. Здійснити перехід до ефективніших кодів із меншою розрядністю дозволяє аналітичне подання унітарних кодів і встановлення функціональних залежностей з іншими кодами чи системами кодування.

Для подання унітарних кодів використовуються унітарні функції:

$$Uni(m,\theta,i) = sign(\sin(2^m \pi(\theta + i \cdot 2^{-n}))), \qquad (1.2)$$

де $m = 0, 1, \ldots, n+1$ – порядок набору системи функцій;

$n = \log_2 N$; N – модуль цілочислових дискретних значень системи;

$\theta = t/T$; $(0 \le \theta < 1)$ – нормований параметр часу;

$\mathrm{T} = 2\pi$; t – потокове значення часу;

$0 \le t < 2\pi$; $i = 0,1,\ldots,2^{n-m+1} - 1$ – порядковий номер функції в наборі порядку m.

Набір нульового порядку $Uni(0,\theta,i)$ містить *2N* функцій (рисунок 1.2).

Властивості унітарних функцій:

1. Система з перших *N* унітарних функцій порядку *m* є лінійно незалежною, оскільки виконується достатня умова лінійної незалежності: ранг матриці *N* функцій дорівнює кількості функцій *N*. Наступні *N* функцій є лінійними комбінаціями *N* перших.

2. Унітарні функції не ортогональні, оскільки $\int_0^1 Uni(m,\theta,i)Uni(k,\theta,j)d\theta \neq 0$.

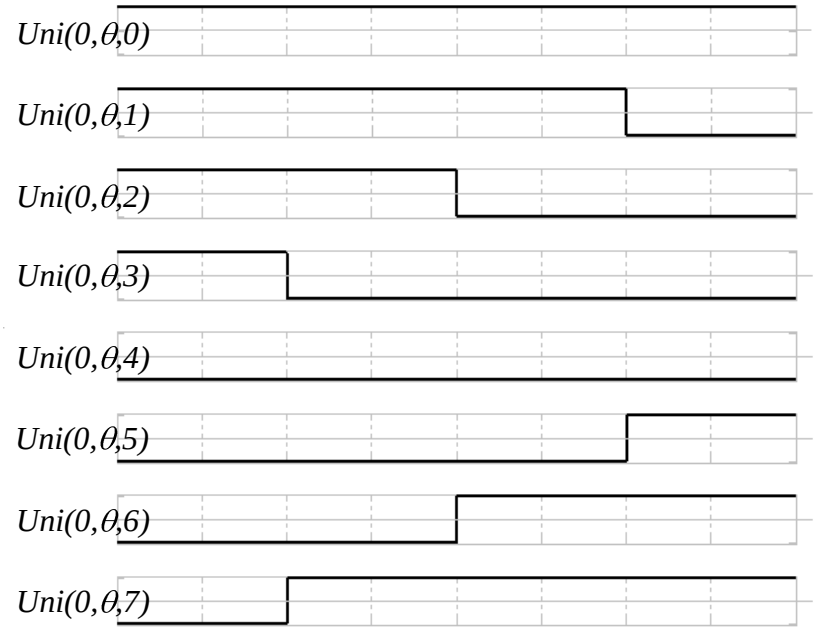


Рисунок 1.2 – Унітарні функції нульового порядку.

Неортогональність системи унітарних функцій зумовлює некомпактне пакування кодових елементів системи, що приводить до значної надлишковості інформаційних потоків. Внаслідок неортогональності та відсутності досліджень властивостей система не використовується як основа ТЧП.

Породжуючу кодову матрицю унітарного коду розміру $N \times N$ одержують при дискретизації з інтервалом $1/N$ за параметром часу перших $N=2^n$ із системи $2N$ унітарних функцій та здійсненні бінарної заміни значень функцій *1* на *0*, *–1* на *1* в точках $\theta_s = s/2^n$, $s=0,1,\ldots,2^n-1$, яка реалізується за допомогою операції:

$$u_i = (1 - Uni(0,\theta_s,2^n - 1 - i))/2, \qquad (1.3)$$

де $u_0, u_1, \ldots, u_i, \ldots, u_{2^n-1}$ – значення розрядів унітарного коду θ_s, $i=0,1,\ldots,2^n-1$.

Для прикладу, при $n=3$ восьми функціям відповідають такі елементи кодової матриці:

$$
\begin{array}{lcccccccc}
Uni(0,\theta,0) & \to & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Uni(0,\theta,1) & \to & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
Uni(0,\theta,2) & \to & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
Uni(0,\theta,3) & \to & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
Uni(0,\theta,4) & \to & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
Uni(0,\theta,5) & \to & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
Uni(0,\theta,6) & \to & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
Uni(0,\theta,7) & \to & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

Наведені властивості системи унітарних функцій дозволяють у наступних викладках визначити процедури перетворення до інших базисів. У ролі первинних при перетворенні форми інформації та при переході від *N*-розрядних унітарних до кодів із меншою розрядністю також використовуються розрядно-позиційні коди.

1.2.2 Базис Хаара

Основою розрядно-позиційних кодів є система функцій Хаара. Функції Хаара $Har(n,\theta,j)$ (рисунок 1.3) визначаються за формулою:

$$
Har(n,\theta,j)=\begin{cases} 2^{\frac{n-1}{2}}\, sign(\sin 2^n \pi\theta), & \dfrac{j}{2^{n-1}} \le \theta < \dfrac{j+1}{2^{n-1}}, \\ 0 \;\; при \;\; інших \;\; \theta \in [0,1), \end{cases} \qquad (1.4)
$$

де $n=0,1,...,\log_2 N$; $j=0,1,...,2^{n-1}-1$; ($j=0$ при $n=0$), $0\le\theta<1$.

При реалізації ТЧП використовуються наступні властивості системи Хаара.

1. Функції Хаара $\{Har(n,\theta,j)\}$ утворюють повну ортонормовану систему в просторі інтегровних із квадратом функцій $L_2[0,1)$, що дає можливість використовувати систему в якості базису для виконання ортогонального перетворення, яке є вейвлет-перетворенням.

2. На значній частині інтервалу визначення функції дорівнюють нулю, що дає можливість скоротити кількість арифметичних операцій при обчисленні перетворення. У результаті зменшується час обробки інформації.

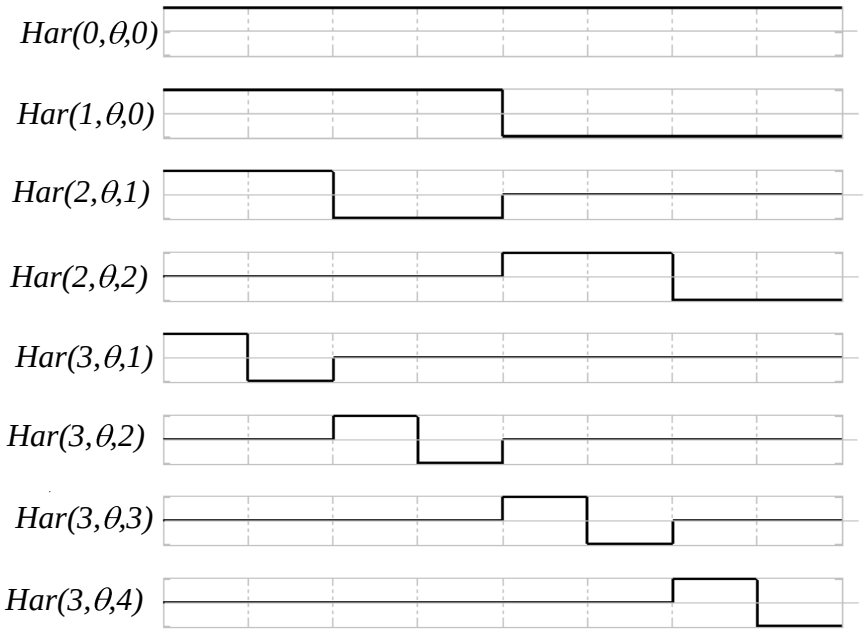


Рисунок 1.3 – Система функцій Хаара.

Ненормований базис Хаара $\{sign(\sin 2^n \pi\theta)\}$ (без нормуючого множника $2^{\frac{n-1}{2}}$), в якому функції набувають значень ±1, 0, є основою розрядно-позиційних кодів. Розрядно-позиційні коди застосовуються в засобах перетворення форми інформації, в якості проміжних при аналогово-цифровому перетворенні, в давачах переміщень, для ініціювання комірок пам'яті тощо.

Для встановлення аналітичних співвідношень зв'язку систем функцій, які лежать в основі перетворення даних із розрядно-позиційного коду та в розрядно-позиційний код, використовуються розрядно-позиційні функції:

$$RP(i,\theta) = \begin{cases} -1, \frac{i}{2^n} \le \theta \le \frac{i+1}{2^n}, \\ 1 \ \textit{при інших} \ \theta \in [0,1], \end{cases} \tag{1.5}$$

$n = 0,1,2,\ldots,\ i = 0,1,\ldots,2^n - 1.$

Дискретне подання 2^n розрядно-позиційних функцій та бінарна заміна значень функцій *1* на *0*, *–1* на *1*, яка подається за допомогою виразу:

$$p_i = (1 - RP(i,\theta_s))/2 = \frac{1}{2^{n/2}} Har(n+1,\theta_s,i-1) \tag{1.6}$$

де p_i – значення розрядів розрядно-позиційного коду,

породжує кодову матрицю:

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ \hdashline 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \end{array}$$

За умови простої реалізації перетворення інформації унітарному та розрядно-позиційному кодам властивий недолік необхідності використання повної N-розрядної шини кодового подання даних для кодування N дискретних повідомлень. Це зумовлює необхідність переходу до ефективніших методів кодування зі зменшеною розрядністю кодів до $n = \log_2 N$ в системах Радемахера та Грея.

В якості проміжних при переході до n-розрядних кодів використовуються коди Лібова-Крейга, які дозволяють у два рази зменшити розрядність коду та характеризуються властивістю абсолютного позиціонування.

1.2.3 Базис Лібова-Крейга

Залежність унітарних кодів з іншими встановлюється за допомогою системи дискретно-фазових функцій, що є основою кодів Лібова-Крейга.

Дискретно-фазові функції порядку m подаються згідно наступного аналітичного виразу:

$$Dyf(m, \theta, i) = sign(\sin(2^m \pi(\theta + i \cdot 2^{-n}))), \qquad (1.7)$$

де $i = 0, 1, \dots, 2^{n-m+1}-1$ – порядковий номер функції в наборі порядку m.

Графіки дискретно-фазових функцій першого порядку наведені на рисунку 1.4.

Властивості дискретно-фазових функцій:

1. Система з N дискретно-фазових функцій є лінійно залежною, оскільки частина функцій системи є лінійною комбінацією інших функцій системи:

$$Dyf(m,\theta, j+2^{n-m}) = -Dyf(m,\theta, j),$$

де $j = 0,1,\ldots,2^{n-m}-1.$

Внаслідок лінійної залежності перші N функцій не утворюють повної системи.

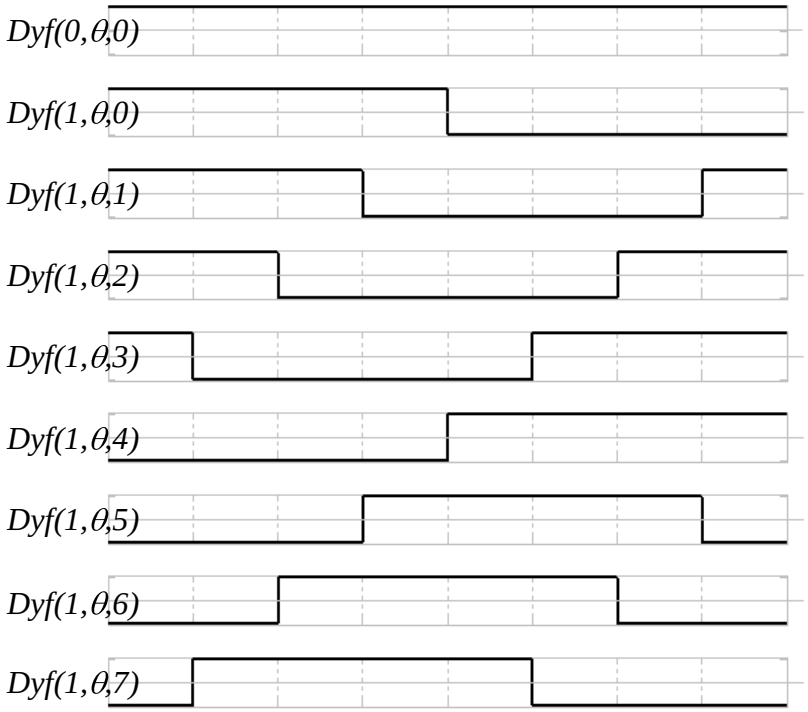


Рисунок 1.4 – Дискретно-фазові функції першого порядку.

2. Система є неортогональною, тому що $\int_0^1 Dyf(m,\theta,i)Dyf(m,\theta, j)d\theta \neq 0$.

Породжуючу кодову матрицю розміру $\frac{N}{2}\times N$ одержують за допомогою дискретизації перших $N=2^{n-1}$ дискретно-фазових функцій порядку m та здійснення бінарної заміни значень функцій 1 на 0, –1 на 1, яка реалізується за формулою:

$$d_j = (1 - Dyf(1,\theta_s,2^{n-1}-1-j))/2, \qquad (1.8)$$

де $d_0, d_1, ..., d_j, ..., d_{2^{n-1}-1}$ – значення розрядів коду Лібова-Крейга $\theta_s = \frac{s}{2^n}$, $s=0,1,\ldots,2^n-1$.

Наприклад, при N=8 елементи матриці відповідатимуть таким функціям:

$$\begin{array}{cccccccccc}
Dyf(1,\theta,0) & \rightarrow & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
Dyf(1,\theta,1) & \rightarrow & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
Dyf(1,\theta,2) & \rightarrow & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
Dyf(1,\theta,3) & \rightarrow & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
s & \rightarrow & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7.
\end{array}$$

Внаслідок неповноти, система не використовується як основа для ортогональних ТЧП. Дискретно-фазові функції розглядаються як перехідні та як основа творення базисів і систем функцій Радемахера, Грея, кодування даних в яких здійснюється з розрядністю $n = \log_2 N$ порівняно з N для унітарних кодів. Із цією метою в складі системи дискретно-фазових функцій виокремлюють дві підсистеми функцій виду $sign(\sin(2^n \pi\theta))$ та $sign(\cos(2^n \pi\theta))$.

1.2.4 Базис Радемахера

Екстракція *sin*-складових набору дискретно-фазових функцій утворює систему функцій Радемахера (рисунок 1.5).

$$Rad(n,\theta) = Dyf(n,\theta,0) = sign(\sin(2^n \pi\theta)) \qquad (1.9)$$

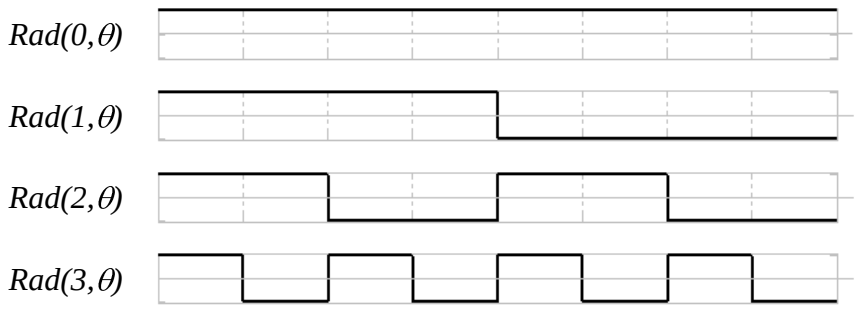


Рисунок 1.5 – Функції Радемахера.

Система Радемахера є основою двійкової системи числення.

Відповідність між значеннями функцій у точках $\theta_s = s/2^n$, $s=0,1,\ldots,2^n-1$ та їх поданням у двійковому коді $\theta_s = r_n r_{n-1} \ldots r_0$ встановлюється співвідношенням:

$$r_k = (1 - Rad(n-k, \theta_s))/2, \qquad (1.10)$$

де r_k – значення розрядів двійкового коду, $k = 0,1,\ldots,n$.

Наприклад, при $n=3$ чотирьом функціям відповідають такі елементи кодової матриці розміру 4×8

$$\begin{array}{cccccccccc}
Rad(0,\theta) & \rightarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Rad(1,\theta) & \rightarrow & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
Rad(2,\theta) & \rightarrow & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
Rad(3,\theta) & \rightarrow & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
s & \rightarrow & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7.
\end{array}$$

Наступні властивості системи Радемахера:

- функції Радемахера ортонормовані на відрізку $[0,1)$, оскільки:

$$\int_0^1 Rad(n,\theta) Rad(k,\theta) d\theta = 0 \text{ та } \int_0^1 Rad(n,\theta) Rad(n,\theta) d\theta = 1.$$

- система функцій Радемахера утворює в просторі інтегровних із квадратом функцій $L_2[0,1)$ неповну систему ортонормованих функцій, оскільки для довільного n не виконується означення повноти системи:

$$\int_0^1 Rad(n,\theta) Rad(1,\theta) Rad(2,\theta) d\theta = 0,$$

тобто існує функція $Rad(1,\theta)Rad(2,\theta)$, яка тотожно не дорівнює нулю на інтервалі $[0,1)$ та ортогональна до всіх функцій системи.

Неповнота системи Радемахера обмежує її застосування для подання інформаційних потоків на основі ортогональних перетворень. Одночасно із широким застосуванням, творенням за допомогою системи Радемахера двійковим кодам властивий недолік, що полягає в неоднозначності формування відліків суміжних кодів при міжрозрядному позиціонуванні. Уникнути такої вади дозволяє перехід до кодів Грея

1.2.5 Базис ортогональних функцій Грея

Екстракція *cos*-складових згідно кожного з порядків n набору дискретно-фазових функцій утворює систему функцій Грея. Система функцій Грея є підмножиною системи дискретно-фазових функцій:

$$Gry(0,\theta) = Dyf(0,\theta + 2^{-1},0);$$
$$Gry(m,\theta) = Dyf(m,\theta,2^{n-m-1}),\ m = 1,2,\ldots,n. \qquad (1.11)$$

Графіки функцій Грея наведено на рисунку 1.6.

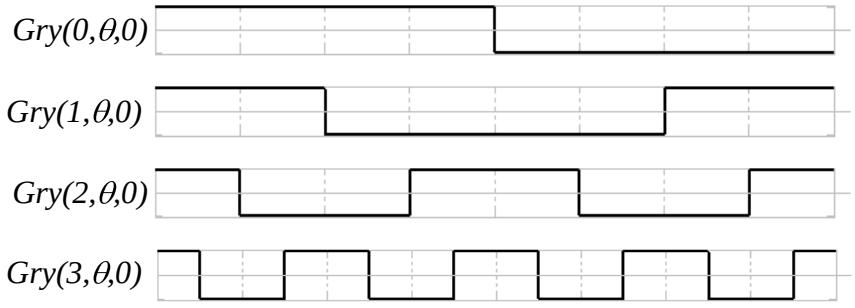


Рисунок 1.6 – Функції Грея.

Система функцій Грея є ортонормованою та неповною, що доводиться наступними викладками.

Відомо, що система Грея є складовою підсистемою функцій Уолша. Оскільки система функцій Уолша є ортонормованою, то і функції Грея, як її підсистема, є ортонормованими, тобто:

$$\int_0^1 Gry(n,\theta)Gry(k,\theta)d\theta = 0,\ \int_0^1 Gry(k,\theta)Gry(k,\theta)d\theta = 1.$$

Функції Грея утворюють у L_2 [0,1] неповну систему, оскільки існують функції, які тотожно не дорівнюють нулю та ортогональні до всіх функцій системи, зокрема, для довільного n:

$$\int_0^1 Gry(n,\theta)Rad(2,\theta)d\theta = 0.$$

Неповнота системи Грея обмежує її застосування для розкладання інформаційних потоків та реалізації ТЧП.

Розширення функціональних можливостей при реалізації системних функцій перетворення форми та обробки інформації забезпечує перехід до базису Уолша. Аналітичні залежності процедури переходу базуються на наступній властивості скінченних добутків функцій системи Грея.

Якщо $(k_1,\ldots, k_m)$ і $(l_1,\ldots, l_r)$ дві різні скінченні послідовності, то:

$$\int_0^1 [Gry(k_1,\theta)\ldots Gry(k_m,\theta)][Gry(l_1,\theta)\ldots Gry(l_r,\theta)]d\theta = 0. \quad (1.12)$$

Для доведення властивості необхідно перепозначити множники в підінтегральному виразі $Gry(j_1, \theta), Gry(j_2, \theta),\ldots, Gry(j_p, \theta)$ $(j_1<j_2<\ldots<j_p)$. Добуток пар функцій $Gry(j_k, \theta)Gry(j_k, \theta)=1$. Добуток інших множників $Gry(j_1, \theta)\ Gry(j_2, \theta)\ldots\ Gry(j_{p-1}, \theta)$ є частково-сталою функцією, кожний з інтервалів сталості якої можна поділити на парне число рівних підінтервалів, на яких $Gry(j_p,\theta)$ набуває почергово значення $+1$ і -1 або -1 і $+1$. Із врахуванням чого:

$$\int_0^1 [Gry(j_1,\theta)\ldots Gry(j_p,\theta)]d\theta = const\int_I Gry(j_p,\theta)d\theta = 0,$$

а тому буде рівним нулю значення інтегралу на інтервалі *[0;1)*. Тобто два різні добутки функцій системи є ортогональними, що і треба довести.

Система функцій Грея є основою кодів Грея. Відповідність між значеннями функцій у точках $\theta_s = s/2^n$, $s=0,1,\ldots,2^n-1$ та їх поданням у коді Грея $\theta_s = h_n h_{n-1} \ldots h_0$ встановлюється співвідношенням:

$$h_k = (1 - Gry(n-k-1, \theta_s))/2, \qquad (1.13)$$

де $k = 0,1,\ldots,n-1$.

Наприклад, чотирьом функціям відповідають елементи кодової матриці розміру 4×8

$$\begin{array}{cccccccccc}
Gry(0,\theta) & \rightarrow & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
Gry(1,\theta) & \rightarrow & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
Gry(2,\theta) & \rightarrow & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
Gry(3,\theta) & \rightarrow & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
s & \rightarrow & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7.
\end{array}$$

Кодування Грея дозволяє зменшити похибки формування відліків суміжних кодів унаслідок зміни тільки одного розряду порівняно із застосуванням двійкових кодів. Розширення функціональних можливостей при реалізації системних функцій перетворення форми та обробки інформації забезпечує базис Уолша.

1.2.6. Базис Уолша.

Властивості систем Радемахера, Грея та відповідних кодів визначають процедуру переходу в базис Уолша *Wal(i,θ),* $i = 0,1,\ldots,2^n - 1$, впорядкований за Уолшем, із системи Радемахера *Rad(n,θ)*. Функції Уолша *Wal(i,θ)* (рисунок 1.7) визначаються, як добуток функцій Радемахера:

$$Wal(i,\theta) = Rad(1,\theta)^{b_0}\, Rad(2,\theta)^{b_1} \ldots Rad(n,\theta)^{b_{n-1}} = \prod_{k=0}^{n-1} (Rad(k+1,\theta))^{b_k}, \qquad (1.14)$$

де $i = b_{n-1}b_{n-2}\ldots b_1 b_0$ – подання в коді Грея порядкового номера функції $Wal(i,\theta)$.

Система функцій Уолша є ортонормованою, повною та мультиплікативною.

Функції Уолша ортонормовані на інтервалі $0 \le \theta \le 1$

$$\int_0^1 Wal(k,\theta)Wal(i,\theta)d\theta = \begin{cases} 1 \ \ при \ \ k = i, \\ 0 \ \ при \ \ k \ne i. \end{cases}$$

Функції Уолша утворюють мультиплікативну систему

$$Wal(k,\theta)Wal(i,\theta) = Wal(k \oplus i,\theta).$$

Системи Радемахера та Грея є підсистемами функцій Уолша:

$$Wal\,(2^n{-}1,\theta) = Rad(n,\theta),$$
$$Wal\,(2^n,\theta) = Gry(n,\theta).$$

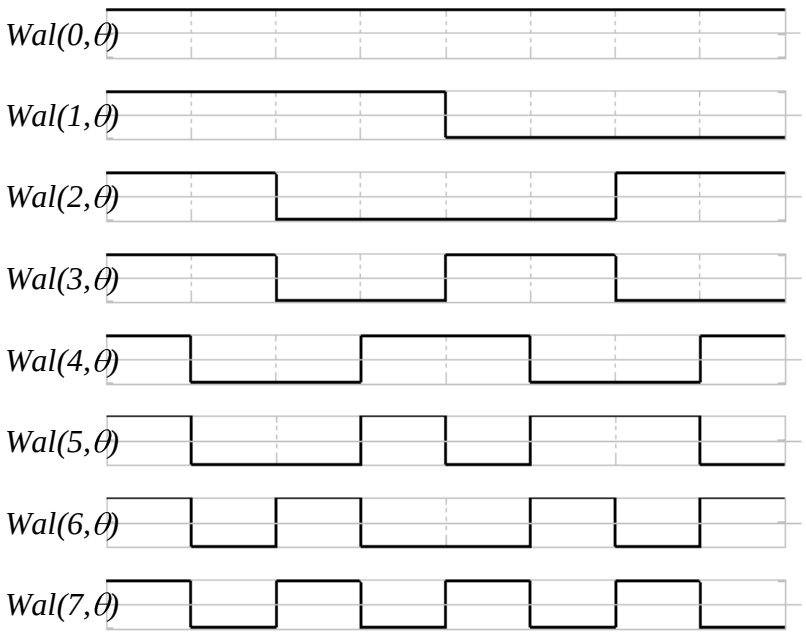


Рисунок 1.7 – Система функцій Уолша.

Відомі схеми генераторів функцій Уолша базуються на методі формування функцій Уолша із системи Радемахера, але використання в ньому двійкового коду зумовлює виникнення помилки неоднозначності. На основі доведеної властивості (1.11) ортогональності добутків функцій Грея розроблено альтернативний метод формування функцій Уолша із системи функцій Грея:

$$Wal(i,\theta) = Gry(0,\theta)^{a_0}\, Gry(1,\theta)^{a_1} \ldots Gry(n-1,\theta)^{a_{n-1}} = \prod_{k=0}^{n-1} (Gry(k,\theta))^{a_k}, \quad (1.15)$$

де $i = a_{n-1}a_{n-2}\ldots a_1 a_0$ – подання в двійковому коді порядкового номера функції Уолша $Wal(i,\theta)$.

У відомих генераторах і перетворювачах функції Уолша формуються згідно (1.13) на основі двійкового коду наступним способом. Функції Радемахера відповідають значенням розрядів двійкового коду:

$$Rad(m,\theta_s) = \overline{r_{n-m}} - r_{n-m} = \begin{cases} 1, & якщо\ r_{n-m} = 0, \\ -1, & якщо\ r_{n-m} = 1, \end{cases} \quad (1.15)$$

де r_m – значення m-го розряду двійкового коду аргумента θ_s, $m=0,1,...,n$.

При підстановці (1.18) у вираз (1.13) функції Уолша визначаються на основі двійкового коду аргументу θ_s:

$$\begin{aligned} Wal(i,\theta_s) &= (\overline{r_{n-1}} - r_{n-1})^{b_0} (\overline{r_{n-2}} - r_{n-2})^{b_{n1}} \ldots (\overline{r_0} - r_0)^{b_{n-1}} = \\ &= (\overline{b_0 r_{n-1} \oplus b_1 r_{n-2} \oplus \ldots \oplus b_{n-1} r_0}) - (b_0 r_{n-1} \oplus b_1 r_{n-2} \oplus \ldots \oplus b_{n-1} r_0) =, \quad (1.16) \\ &= \overline{\varphi_i}(\theta_s) - \varphi_i(\theta_s) \end{aligned}$$

де $\varphi_i(\theta_s) = b_0 r_{n-1} \oplus b_1 r_{n-2} \oplus \ldots \oplus b_{n-1} r_0$.

Із використанням наведеної методики можна визначити спосіб формування функцій Уолша на основі аргумента, поданого в коді Грея.

Функції Грея $Gry(k,\theta_s)$ відповідають значенням n розрядів коду Грея θ_s згідно залежності:

$$Gry(k,\theta_s) = \overline{h_{n-k-1}} - h_{n-k-1} = \begin{cases} 1, & якщо \ \ h_{n-k-1} = 0, \\ -1, & якщо \ \ h_{n-k-1} = 1, \end{cases} \qquad (1.17)$$

де h_k – значення k-го розряду коду Грея аргумента θ_s; $k = 0,1,\ldots,n-1$.

Функції Уолша визначаються при підстановці функцій Грея з (1.17) у (1.14):

$$Wal(i,\theta_s) = (\overline{h_{n-1}} - h_{n-1})^{a_0} (\overline{h_{n-2}} - h_{n-2})^{a_1} \ldots (\overline{h_0} - h_0)^{a_{n-1}},$$

де $\theta_s = h_{n-1}\ldots h_1 h_0$ – код Грея, $i = a_{n-1}a_{n-2}...a_0$ – подання у двійковому коді числа i.

Перетворення добутку в правій частині рівності з використанням основних тотожних співвідношень булевої алгебри дозволяє визначити функції Уолша:

$$Wal(i,\theta_s) = \overline{(a_0 h_{n-1} \oplus a_1 h_{n-2} \oplus \ldots \oplus a_{n-1} h_0)} - (a_0 h_{n-1} \oplus a_1 h_{n-2} \oplus \ldots \oplus a_{n-1} h_0), \qquad (1.18)$$

$$Wal(i,\theta_s) = \overline{\varphi_i}(\theta_s) - \varphi_i(\theta_s), \qquad (1.19)$$

де $\varphi_i(\theta_s) = a_0 h_{n-1} \oplus a_1 h_{n-2} \oplus \ldots \oplus a_{n-1} h_0$.

Перевагою перетворювача на основі (1.18) є використання кодів Грея, які дозволяють зменшити помилки в засобах перетворення та обробки.

Таким чином, повна система Уолша, яка утворюється із систем Радемахера та Грея, є базисом для виконання ортогональних перетворень і формування функцій Ґалуа.

1.3 Базис Крестенсона

Узагальненням системи функцій Уолша на *p*-значний випадок, де *p*— просте число, є система функцій Крестенсона. Функції Крестенсона (рисунок 1.8) $\chi_{\omega}^{(p)}(z)$ є частково-постійними функціями з інтервалом значень аргументів $[0, p^m)$ і визначаються для будь-якого з натуральних ω, m при $0 \le \omega \le p^m - 1$ наступним чином:

$$\chi_{\omega}^{(p)}(p) = \exp(j\frac{2\pi}{p}\sum_{i=0}^{m-1}\omega^{(m-1-i)}z^{(i)}), \qquad (1.20)$$

де $j = \sqrt{-1};\ \omega^{(i)}, z^{(i)} \in [0,1,..., p-1]$ i

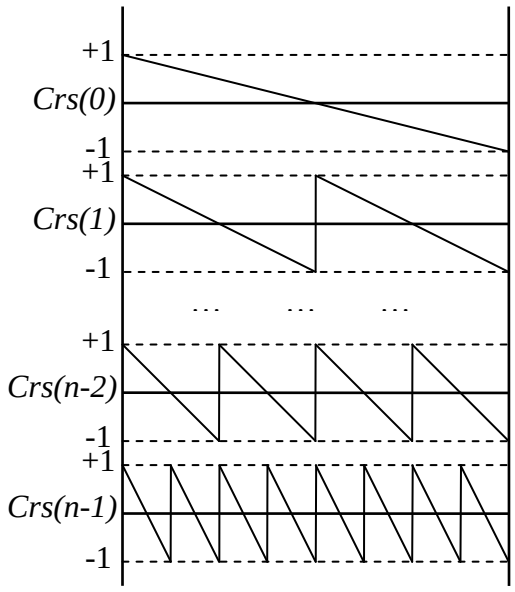


Рисунок 1.8 – Система функцій Крестенсона.

$$\left.\begin{aligned}\omega &= \sum_{i=0}^{m-1}\omega^{(i)}p^{m-1-i,}\\ z &= \sum_{i=0}^{\infty}z^{(i)}p^{m-1-i}\end{aligned}\right\}. \qquad (1.21)$$

Отже, отримується:

$$\chi_{\omega}^{(2)}(z) = W_{\omega}(z), \qquad (1.22)$$

при $p=2$ система функцій Крестенсона співпадає з системою функцій Уолша.

Розглянемо підмножину $K_{r,i}^{(p)}(z)$ функцій Крестенсона $K_{r,i}^{(p)}(z)$, для яких індекс ω містить усього одну p компоненту. Ці функції є узагальненнями функцій Радемахера :

$$K_{r,i}^{(p)}(z) = \chi_{r,p^i}^{(p)}(z) = \exp(j\frac{2\pi}{p} rz^{(i)}) \qquad (1.23)$$

(r=1,2, ... , p – 1, i=0,1, ..., m-1).

Отже,

$$K_{j,i}^{(2)}(z) = R_{i+1}(z), \qquad (1.24)$$

у зв'язку з цим функції $K_{j,i}^{(2)}(z)$ називаються узагальненими функціями Радемахера. Узагальнені функції Радемахера при p=3, m=2 приведені в таблиці 1.1

Можна встановити взаємно-однозначну відповідність h між безліччю значень функцій Крестенсона і безліччю чисел $[0,1,..., p-1]$ наступним чином:

$$h\left(\exp(j\frac{2\pi}{p}q)\right) = q,\ (q = 0,1,..., p-1). \qquad (1.26)$$

Таблиця 1.1 – Узагальнені функції Радемахера.

$\chi_{\omega}^{(3)}(z)$	z								
	0	1	2	3	4	S	6	7	8
$\chi_0^{(3)}(z)$	1	1	1	1	1	1	1	1	1
$\chi_1^{(3)}(z)$	1	1	1	e_1	e_1	e_1	e_2	e_2	e_2
$\chi_2^{(3)}(z)$	1	1	1	e_2	e_2	e_2	e_1	e_1	e_1
$\chi_3^{(3)}(z)$	1	e_1	e_2	1	e_1	e_2	1	e_1	e_2
$\chi_4^{(3)}(z)$	1	e_1	e_2	e_1	e_2	1	e_2	1	e_1

$\chi_5^{(3)}(z)$	1	e_1	e_2	e_2	1	e_1	e_1	e_2	1
$\chi_6^{(3)}(z)$	1	e_2	e_1	1	e_2	e_1	1	e_2	e_1
$\chi_7^{(3)}(z)$	1	e_2	e_1	e_1	1	e_2	e_2	e_1	1
$\chi_8^{(3)}(z)$	1	e_2	e_1	e_2	e_1	1	e_1	1	e_2

Тоді, як видно з (1.25), після перетворення *h* функції $K_{1,i}^{(p)}(z)$ співпадають з відповідним разрядом $z^{(t)}$ *p*-того коду аргументу *p*.

Функції Крестенсона можуть бути виражені через узагальнені функції Радемахера.

З (1.18) і (1.21) отримується:

$$\chi_{\omega}^{(p)}(z) = \prod_{i=0}^{m=1} (K_{1,i}^{(p)}(z))\omega^{(m-1-i)} \qquad (1.26)$$

Формула (1.26) є узагальненням і показує зв'язок функцій Радемахера і Уолша.

$K_{r,i}^{(3)}(z)$	z								
	0	1	2	3	4	S	6	7	8
$K_{0,0}^{(3)}(z)$	1	1	1	1	1	1	1	1	1
$K_{1,0}^{(3)}(z)$	1	1	1	e_1	e_1	e_1	e_2	e_2	e_2
$K_{2,0}^{(3)}(z)$	1	1	1	e_2	e_2	e_2	e_1	e_1	e_1
$K_{1,1}^{(3)}(z)$	1	e_1	e_2	1	e_1	e_2	1	e_1	e_2
$K_{2,1}^{(3)}(z)$	1	e_1	e_1	1	e_2	e_1	1	e_2	e_1

До властивостей функцій Крестенсона можна віднести повноту і ортогональність (таблиця 1.2).

Теорема. Множина p^m функцій Крестенсона утворює повну ортогональну систему функцій в просторі частково-постійних функцій, визначених на $[0,p^m)$, причому:

$$\sum_{z=0}^{p^m-1} x_t^{(p)}(z)\overline{x_q^{(p)}(z)} = \begin{cases} p^m \text{ при } t = q \\ 0 \text{ при } t \neq q \end{cases} \qquad (1.27)$$

і, якщо $\Phi(z)$ — частково-постійна функція і

$\sum_{z=0}^{p^m-1} \Phi(z)\overline{x_q^{(p)}(z)} = 0$ $(\omega = 0,1,..., p^m - 1)$, то $\Phi(z) \equiv 0$

З (1.27) при $t=0$ випливає:

$$\sum_{z=0}^{p^m-1} x_\omega^{(p)}(z) = 0 \text{ при } \omega \neq 0 \qquad (1.28)$$

Теорема. Нехай функція $\Phi(z)$ — частково-постійна, яка представляє деяку систему p-тих функцій від m аргументів. Тоді:

$$\Phi(z) = \sum_{\omega=0}^{p^m-1} S(\omega) x_\omega^{(p)}(z), \qquad (1.29)$$

Таблиця 1.2 – Повнота та ортогональність функцій Крестенсона

z,ω	$z^{(0)}$	$z^{(1)}$	$f^{(0)}(z)$	$f^{(1)}(z)$	$\Phi(z)$	$9S(\omega)$
0	0	0	0	2	2	36
1	0	1	2	2	8	0
2	0	2	0	2	2	0
3	1	0	1	0	3	0
4	1	1	0	1	1	$1,5-1,5\sqrt{3}\,j$
5	1	2	2	2	8	$-10,5-7,5\sqrt{3}\,j$
6	2	0	2	1	7	0
7	2	1	1	0	3	$-10,5+7,5\sqrt{3}\,j$

8	2	2	0	2	2	$1{,}5+1{,}5\sqrt{3}\,\mathrm{j}$

де:

$$S(\omega) = p^{-m} \sum_{z=0}^{p^m-1} \Phi(z)\overline{x_\omega^{(p)}(z)}. \qquad (1.30)$$

Отже, якщо початкова система набуває ненульових значень не більше ніж в p^m точках, то і спектр по Крестенсону S(ω) набуває ненульових значень також не більше ніж в p^m точках. Це дає можливість відновлення початкової системи p-тих функцій по її спектру по Крестенсону.

Теорема (Симетрія індексу і аргументу). Для будь-кого ω,z $\in$ [0,1,…,p^m-1]

$$x_\omega^p(z) = x_z^p(\omega)$$

Теорема (Зрушення аргументу). Для будь-якого:

$$\omega,z,\tau \in [0,1,\ldots,p^m\text{-}1], \qquad (1.31)$$

$$x_\omega^{(p)}(z \oplus \tau) = x_\omega^{(p)}(z)x_\omega^{(p)}(\tau) \pmod p, \qquad (1.32)$$

з попередніх теорем слідує, що безліч функцій Крестенсона замкнута відносно операції множення.

Ізоморфізм між лінійними функціями p- тої логіки і функціями Крестенсона. Функція $f(z^{(0)}, z^{(1)}, \ldots, z^{(m-1)})$ p-значної логіки називається лінійною, якщо її можна представити згідно співвідношення:

$$f(z^{(0)}, z^{(1)}, \ldots, z^{(m-1)}) = \bigoplus_{i=0}^{m-1} c_i z^{(i)} \pmod p \qquad (1.33)$$

де $c_i \in [0,1,\ldots,p\text{-}1]$.

Лінійні функції утворюють комутативну групу з груповою операцією «сума по модулю *p*» *і з* числом елементів p^m. Функції Крестенсона $x_\omega^{(p)}(z)$ створюють групу відносно операції множення.

Теорема. Мультиплікативна група функцій Крестенсона ізоморфна групі лінійних функцій *p*-значної логіки. Ізоморфізм *h* задається співвідношенням:

$$h(x_\omega^{(p)}(z)) = \bigoplus_{i=0}^{m-1} \omega^{(m-1-i)} z^{(i)} \pmod p \tag{1.34}$$

Цей ізоморфізм *h* є продовженням ізоморфізма між узагальненими функціями Радемахера і компонентами *p*-того розкладання аргументу.

Розглянемо тепер методи обчислення спектру по Крестенсону. Один метод, природньо, визначається при $\psi_\omega(z) = x_\omega^{(p)}(z)$. Інший метод полягає в узагальненні методу обчислення спектру по Уолшу через матриці Адамара. Будується матриця $x_\omega^{(p)}$, ω - рядок якої буде $x_\omega^p(0),\ x_z^p(1),...,x_z^p(p^m-1)(\omega = 0,1,...,p^m-1)$

При m = 1 і m = *2* для *p=3* $x_1^{(3)}$ *і* $x_1^{(3)}$ має вигляд:

$$x_1^{(3)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & e_1 & e_2 \\ 1 & e_2 & e_1 \end{pmatrix} \tag{1.35}$$

$$x_2^{(3)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & e_1 & e_1 & e_1 & e_2 & e_2 & e_2 \\ 1 & 1 & 1 & e_2 & e_2 & e_2 & e_1 & e_1 & e_1 \\ 1 & e_1 & e_2 & 1 & e_1 & e_2 & 1 & e_1 & e_2 \\ 1 & e_1 & e_2 & e_1 & e_2 & 1 & e_2 & 1 & e_1 \\ 1 & e_1 & e_2 & e_2 & 1 & e_1 & e_1 & e_2 & 1 \\ 1 & e_2 & e_1 & 1 & e_2 & e_1 & 1 & e_2 & e_1 \\ 1 & e_2 & e_1 & e_1 & 1 & e_2 & e_2 & e_1 & 1 \\ 1 & e_2 & e_1 & e_2 & e_1 & 1 & e_1 & 1 & e_2 \end{bmatrix} \tag{1.36}$$

Усі властивості функцій Крестенсона можуть бути сформульовані з використанням матриць $x_m^{(p)}$, подібно до того як це мало місце для функцій Уолша і матриць Адамара.

Представимо початкову функцію і її спектр по Крестенсону у вигляді векторів $\vec{\Phi} = [\Phi(0), \Phi(1), ..., \Phi(p^m - 1)),\ \vec{S} = (S(0), S(1), ..., S(p^m - 1))]$

Тоді з (1.30) випливає, що:

$$\vec{S} = p^{\overline{-m}\, x_m^{(p)}} \vec{\Phi}. \qquad (1.37)$$

Формула (1.37) дає простий і зручний при невеликих p і m спосіб побудови спектру. Обчислення спектру по Крестенсону по формулі (1.37) вимагає p^{2m} операцій додавання і віднімання (по p^m операцій на коефіцієнт).

Якщо A симетрична матриця розміру $p^m \times p^m$ то будемо позначати $\overleftarrow{A}$ матрицю, ω -й рядок і стовпець яки є ω рядок і стовпець матриці A, де, якщо $\omega = \sum_{i=0}^{m-1} \omega^{(m-1-i)} p^i, \overleftarrow{\omega} = \sum_{i=0}^{m-1} \omega^{(i)} p^{(i)}$.

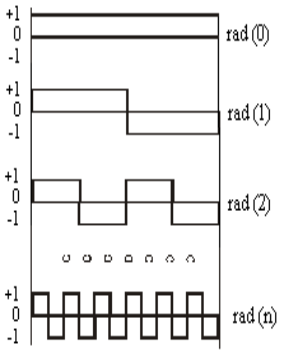
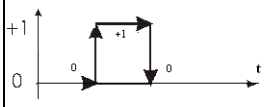
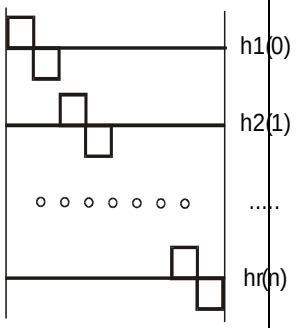
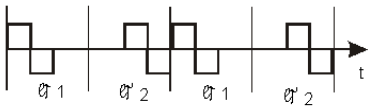
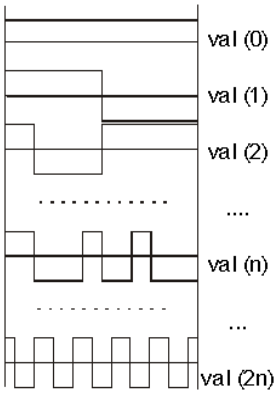
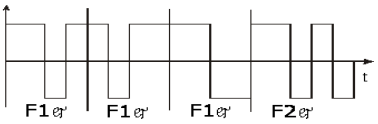
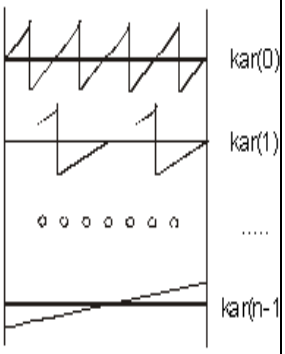
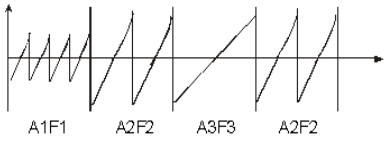
Утворюється матриця:

$$A_m^{(p)} \begin{bmatrix} e_0^0 e_{p-1}^0 ... e_{p-1}^0 & 0 & \\ 0 & e_0^0 e_1^0 ... e_{p-1}^0 & 0 \\ e_0^1 e_1^1 ... e_{p-1}^0 & 0 & e_0^0 e_1^0 ... e_{p-1}^0 \\ 0 & e_0^1 e_1^1 ... e_{p-1}^1 & 0 \\ & 0 & e_0^1 e_1^1 ... e_{p-1}^1 \\ e_0^{p-1} e_{p-1}^{p-1} ... e_{p-1}^{p-1} & 0 & \\ 0 & e_0^{p-1} e_1^{p-1} ... e_{p-1}^{p-1} & 0 \\ & 0 & e_0^{p-1} e_1^{p-1} ... e_{p-1}^{p-1} \end{bmatrix} B^{m-1}$$

$$e_q = \exp(j \frac{2\pi}{p} q) \quad (q = 0,1,..., p-1).$$

Представлення систем ортогональних функцій різних ТЧБ подані у таблиці 1.3.

Таблиця 1.3 – Представлення теоретико-числових базисів

Базис	Представлення базису	Базисна функція та об'єм матриці V	Модуляція сигналу та спектр
1	2	3	4
Радема-хера		$Rad(n,\theta)=sign[2^n\pi\cdot\theta]$ $V=N\cdot\log_2 N$	Базисні функції Радемахера є основою для модуляції Прямокутних сигналів  Базис Радемахера породжує двійкову систему числення
Хаара		$Har(n,\theta,i) =$ $= sign[\sin(i\cdot$ $\cdot 2^n\pi,\theta)]$ $V=N^2$	Використовується при фазовій модуляції сигналу.  В даному базисі використо-вуються розрядно-позиційні коди.
Уолша		$Had\ (h,x) =$ $\prod_{i=1}^{\kappa}[ri\ (x)]hi$	Породжує частотно-фазові методи модуляції сигналу.  Базис Уолша має найбільш широкий спектр сигналу
Крестен-сона		$N_i = res\sum_{i=1}^{n}(b_i\cdot b_i)\bmod P$ $V=\sum_{i=1}^{m}\log_2(P_i)$	1.1 Даний базис породжує амплітудно-частотні методи модуляції.  Базис представлений трикутними функціями.

З метою оцінки ефективності кодування даних на основі різних ТЧБ доцільно провести аналіз кодових матриць, які породжують різні системи числення.

При цьому важливою характеристикою кожного базису є об'єм його кодової матриці M_j та число активних елементів m_j, що визначає характеристики надлишковості представлення інформації на основі аналітичної оцінки $V_i = n_i \cdot N_i$, де n_j – розрядність числа; N_i – число незалежних кодових значень.

Кожен з названих базисів характеризується визначеним об'ємом кодової матриці для представлення даних. При цьому найбільш надлишковим базисом є унітарний, в якого кодова матриця $V = N^2$, а число активних кодових елементів $n = N^2/2$, де N – діапазон кодування даних. Аналогічну надлишковість забезпечує базис Хаара, в два рази меншу надлишковість забезпечує базис Крейга, тобто $V = N^2/4$, а $n = N^2/8$. Максимально широке застосування для кодування даних в сучасних КС отримали базиси Радемахера та Крестенсона, в яких $V = N \log_2 N$. Дані базиси відповідно породжують двійкову систему числення та систему числення залишкових класів.

Базис Уолша максимально широко використовується в сучасних телекомунікаційних КС. Даний базис породжує систему ортогональних шумоподібних сигналів, які використовуються в сотових системах мобільного зв'язку.

Світовий досвід створення процесорів для комп'ютерних систем за останні 50 років, поряд з застосуванням теоретико-числового базису (ТЧБ) Радемахера, який породжує двійкову систему числення, демонструє тенденцію все ширшого застосування інших ТЧБ, в тому числі: унітарного, Хаара, Крестенсона, який оснований на використання модулярної арифметики, тобто теорії залишків. Тому розробка програмної системи пошуку залишків великорозрядних чисел є актуальною задачею, яка широко використовується в кодуванні інформаційних потоків, системах захисту від несанкціонованого доступу.

2 ТЕОРІЯ ТА АЛГОРИТМИ ПОШУКУ ЗАЛИШКІВ ВЕЛИКОРОЗРЯДНИХ ЧИСЕЛ

2.1 Математична основа методу залишків

Принципи кодування методом залишків базуються на теоретико-числовому підході і представляють собою клас векторних багаторівневих кодів поля Галуа. Застосування цього класу багаторівневих кодів поля Галуа дозволяє зменшити надлишковість вимірювальної інформації за рахунок виключення з інформаційного потоку кодів рангів діофантового рівняння.

$$Y_i = b_i (\mathrm{mod}\, P),$$

яке відповідає лінійному рівнянню з розв'язком у цілих числах

$$Y_i = a_i \cdot P + b_i, \tag{2.1}$$

де $a_i = \breve{E}\left[\frac{Y_i}{P}\right]$ - ранг цифрового відліку Y_i;

$0 \le b_i \le P-1$ - найменший невід'ємний залишок Y_i по модулю P, тобто:

$$b_i = rezY_i (\mathrm{mod} P). \tag{2.2}$$

З виразу (2.1) видно, що величина Y_i може бути однозначно відновлена, якщо крім b_i можна обчислити ранг a_i на основі попередньо відомого Y_{i-1}. Тобто існує рішення функціонеру:

$$Y_i = F(Y_{i-1}, b_i). \tag{2.3}$$

Покажемо, що достатньо простим при цьому є рекурентний алгоритм обчислення рангу a_i по текучому значенню b_i і відомому рангу a_{i-1} попереднього відліку Y_{i-1}.

Для доведення існування функціонала Y_i рішимо рівняння (2.1) відносно a_i :

$$a_i = \frac{Y_i - b_i}{P}. \tag{2.4}$$

Враховуючи цілочисельний характер рангу a_i , введемо праву частину останнього виразу під знак цілочисельної функції:

$$a_i = E\left[\frac{Y_i - b_i}{P}\right] \tag{2.5}$$

Виразимо значення відліку Y_i через результат попереднього вимірювання Y_{i-1}.

$$Y_i = Y_{i-1} + \alpha P, \tag{2.6}$$

де α – деяка величина, які визначає відмінність відліків Y_i та Y_{i-1}.

Підставивши (2.6) у a_i і виконавши прості перетворення отримаємо:

$$a_i = E\left[\frac{Y_{-1} - b_i}{P} + \alpha\right], \tag{2.7}$$

де α має зміст помилки обчислення рангу відліку Y_i значення Y_{i-1}.

Ранг відліку Y_i з рівняння (2.7) може бути точно визначений виходячи з властивостей цілочисельної функції $E[\cdot]$, якщо величина помилки його обчислення менша одиниці, тобто $0 \le \alpha < 1$.

Доведемо ствердність формули (2.7) при умові, що зміна рангу a_{i-1} та a_{i-} не перевищує одиниці.

$$(a_i - a_{i-1}) \le 1. \qquad (2.8)$$

На рисунок 2.1 показані варіанти зміни цифрових відліків Y_i та Y_{i-1}. при виконанні умови (2.8).

Нехай відлік Y_{i-1} має ранг a_{i-1} і залишок b_{i-1}.

Згідно рисунку 2.1 можливі три випадки отримання відліків, які відрізняються значенням рангів a_i від рангу a_i відліку Y_{i-1}.:

1) Ранг Y_i збільшується на одиницю $(a_i = a_{i-1} + 1)$.

Очевидно, що в цьому випадку приріст відліку Y_i не може бути менший P- b_{i-1}, а також не перевищує величини αP, тобто:

$$P - b_{i-1} \le P + b_{i-1} - b_i \le \alpha P,$$

або підсилюючи нерівність:

$$P - \alpha P \le b_{i-1} - b_i \le b_{i-1}. \qquad (2.9)$$

Підставивши Y_{i-1}.у рівняння (2.7) і потім у (2.2), отримаємо рішення для шуканого Y_i.

$$Y_i = \breve{E}\left[a_{i-1} + \frac{b_{i-1} - b_i}{P} + \alpha \right] P + b_i. \qquad (2.10)$$

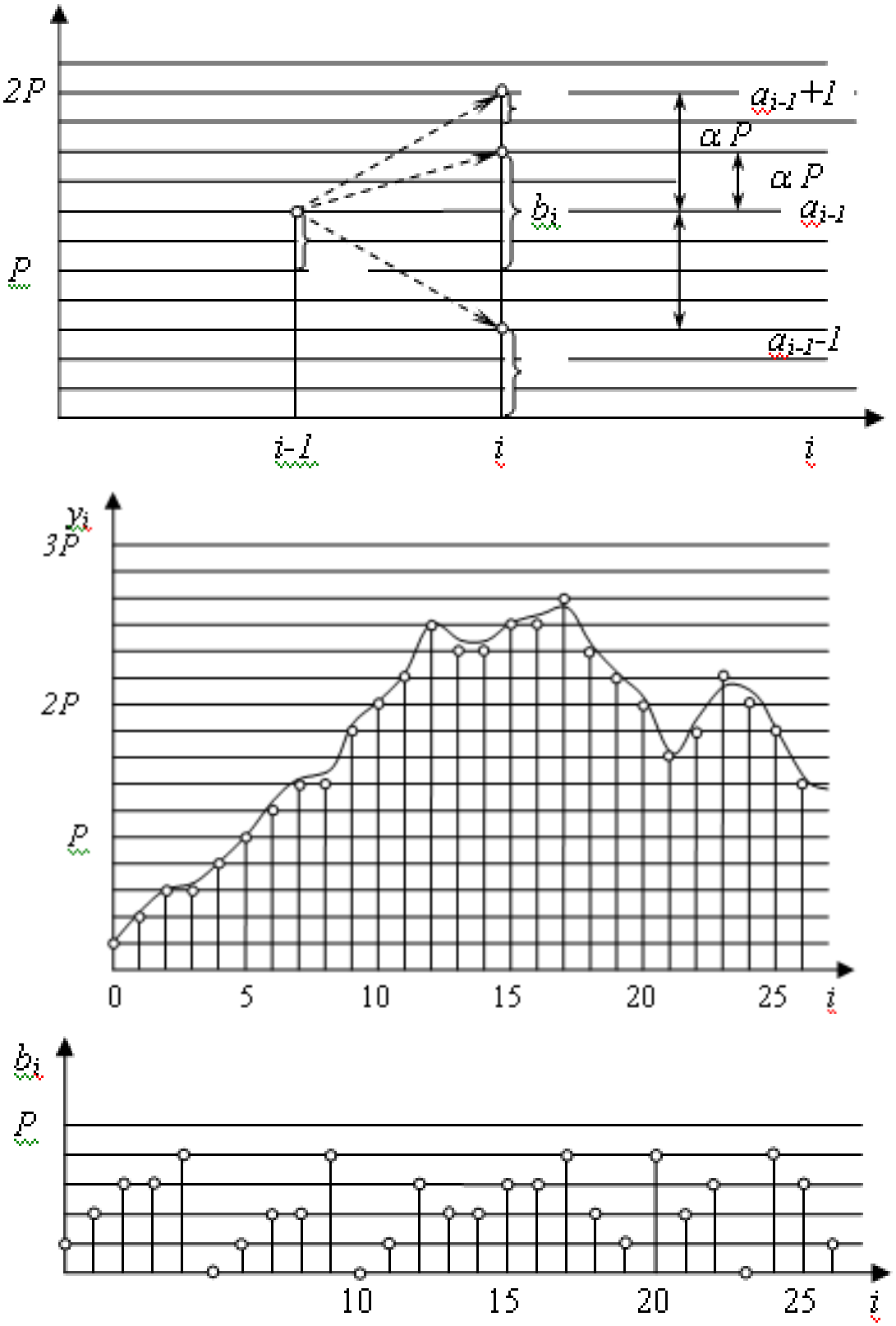


Рисунок 2.1 – Рекурентне кодування інформаційних потоків методом залишків.

З співвідношень (2.9) та (2.10) витікає оцінка для текучого відліку у вигляді нерівності:

$$E\left[\alpha_{i-1}+\frac{P-\alpha P}{P}+\alpha\right]P+b_i \le Y_i \le E\left[a_{i-1}+\frac{b_{i+1}}{P}+\alpha\right]P+b_i, \qquad (2.11)$$

звідси враховуючи, що $b_{i-1} < P$, отримаємо:

$$E\left[a_{i-1}+1\right]P+b_i \le Y_i \le E\left[a_{i-1}+1+\alpha\right]P+b_i \qquad (2.12)$$

Верхня границя у (2.12) співпадає з нижньою, оскільки $\alpha < 1$ *і* $E\left[a_{i-1}+\alpha+1\right] = \left[a_{i-1}+1\right]$.

Тому

$$Y_i = E\left[a_{i-1}+1\right]P+b_i$$

і справедливість формули (2.7) доведена для випадку 1).

2) ранг по відношенню до Y_{i-1}. зберігається $(a_i = a_{i-1})$.

Використовуючи рівняння (2.12) та нерівність:

$$-\alpha P \le b_{i-1}-b_i \le \alpha P,$$

які слідують з рисунок 2.1, оцінимо, як і у випадку п.1) значення Y_i:

$$E\left[a_{i-1}-\frac{\alpha P}{P}+\alpha\right]P+b_i \le Y_i \le E\left[a_{i-1}-\frac{\alpha P}{P}+\alpha\right]P+b_i,$$

Звідки

$$E\left[a_{i-1}\right]P+b_i \le Y_i \le E\left[a_{i-1}+2\alpha\right]P+b_i. \qquad (2.13)$$

Границі в (2.13) співпадають, якщо:

$$2\alpha = \frac{2(b_{i-1} - b_i)}{P} < 1 \quad або \quad \frac{b_{i-1} - b_i}{P} < 0.5 \qquad (2.14)$$

і формула (2.7) доведена для випадку 2).

3) Ранг Y_i зменшується на одиницю $(a_i = a_{i-1} - 1)$.

Приріст відліку Y_i згідно рисунок 2.1 у цьому випадку не менше залишку b_{i-1} і не перевищує αP.

Оцінюючи, як і у попередніх випадках значення Y_i:

$$E\left[a_{i-1} - \frac{b_{i-1} - P}{P} + \alpha\right]P + b_i \le Y_i \le E\left[a_{i-1} - \frac{2P - P}{P} + \alpha\right]$$

і підсилюючи оцінку нерівності знизу $b_{i-1} \ge -\alpha P$, отримаємо:

$$E[a_{i-1} - 1]P + b_i \le Y_i \le E[a_{i-1} - 1 + 2]P + b_i. \qquad (2.15)$$

Підставивши помилку обчислення рангу α в умову (2.14), добиваємося співпадання границь у (2.15). Тим самим доведення формули (2.7) завершено.

З викладеного доведення, видно, що умова існування алгоритму обчислення рангу відліку Y_i на основі Y_{i-1} та b_i по формулі (2.7) виконується, коли для двох послідовних відліків справедлива нерівність:

$$-0.5 < \frac{b_{i-1} - b_i}{P} < 0.5 \qquad (2.16)$$

Остання умова також обмежує допустиму величину першої різниці між відліками Y_{i-1}. та Y_i:

$$|Y_i - Y_{i-1}| < 0.5P, \qquad (2.17)$$

що добре узгоджується другим випадком, коли залишки b_i та b_{i-1} належать одному рангу $a_i = a_{i-1}$.

Таким чином, рівняння (2.10) задовольняє вимоги однозначного декодування для всіх розглянутих випадків, якщо $\alpha = 0.5$. При цьому у відповідності з (2.8) помилка обчислення рангу a_i завжди буде достатньою.

Враховуючи встановлений знак помилки α у виразах (2.7) та (2.10), отримаємо шукану рекурентну формулу для відновлення потоку вимірювальних даних, які кодуються методом залишків багаторівневим векторним кодом поля Галуа:

$$Y_i = \hat{E}\left[\frac{Y_{i-1} + b_i}{P} + 0.5\right]P + b_i \qquad (2.18)$$

де $\hat{E}[\cdot]$ - цілочисельна функція з округленням до більшого цілого.

Розглянемо метод кодування даних методом залишків на прикладі рисунок 2.1.

Кодування ведеться по модулю P=5, тому найбільший перший приріст відліків $|Y_i - Y_{i-1}| \le 2$ згідно умови (2.17).

Нехай i=8, тоді: $Y_{i-1}=7$; $b_i=2$; $b_{i+1}=4$; $b_{i+2}=0$; $b_{i+3}=1$. Згідно (2.18) маємо:

$$Y_i = \hat{E}\left[\frac{7-2}{5} + 0.5\right]5 + 2 = \hat{E}[1.5]5 + 2 = 7;$$

$$Y_{i+1} = \hat{E}\left[\frac{7-4}{5} + 0.5\right]5 + 4 = \hat{E}[1.1]5 + 4 = 9;$$

$$Y_{i+1} = \hat{E}\left[\frac{9-0}{5} + 0.5\right]5 + 0 = \hat{E}[2.3]5 + 0 = 10;$$

$$Y_{i+1} = \hat{E}\left[\frac{10-1}{5} + 0.5\right]5 + 1 = \hat{E}[2.3]5 + 1 = 11.$$

Таким чином, доведені умови та продемонстрована однозначність кодування та декодування інформаційних потоків багаторівневими векторними кодами поля Галуа.

Найширше застосування такі коди отримали у алгоритмах стиснення інформації та спектрального аналізу сигналів на основі нелінійних цифрових фільтрів.

2.2 Кодування інформаційних потоків в системі залишкових класів з довільним порядком реєстрації даних

Суть даного метода полягає у тому, що кожен цифровий відлік Y_{ij}, *i*-го виміру *j*-го каналу множать на базисне число B_j і створюють нове значення $a_{ij} = Y_{ij} \cdot B_j$. Числа a_{ij} сумують по модулю *P* і фіксують у вигляді числа:

$$N_{ik} = res \sum_{j=1}^{k} a_{ij} (\mathrm{mod}\, P),$$

де *k*- число каналів.

Таким чином N_k однозначно виражає набір цифових відліків Y_{i1}, $Y_{i2}, \ldots, Y_{ij}, \ldots . Y_{ik}$. При цьому однозначність забезпечується коли $0 \le Y_{ij} = A_j \le P_j - 1$, де: A_j- діапазон квантування цифрових відліків у *j*-му каналі, P_j- модулі перетворення СЗК.

Однозначне відновлення відліків Y_{ij} після передавання або збереження коду N_{ik} виконується на основі прямого перетворення в СЗК.

$$Y_{ij} = res N_{ik} (\mathrm{mod}\, P_j).$$

Коєфіцієнт стиснення інформації згідно оцінки міри Хартлі визначається згідно відношення ентропії вхідних даних, де код кожного цифрового відліку з довільним порядком реєстрації супроводжується ідентифікаційним кодом номера каналу $k_c = \frac{I_{вх}}{I_{СЗК}}$.

Тобто:

$$I_{вх} = \sum_{j=1}^{k} \hat{E}\left[\log_2 A_j\right] + k\hat{E}\left[\log_2 k\right];$$

$$I_{вих} = \hat{E}\left[\log_2 (P-1)\right].$$

Чисельне моделювання описаного методу пакування даних на основі кодів Галуа цілочисельної СЗК показало, що значення.

$$\sum_{j=1}^{k} \hat{E}\left[\log_2 A_j\right] \approx \hat{E}\left[\log_2 (P-1)\right] + 1.$$

Таким чином ефект стиснення даних визначається згідно виразу:

$$k_c = 1 + k\hat{E}\left[\log_2 k\right] / \hat{E}\left[\log_2 (P-1)\right].$$

При зростанні числа каналів оцінка коефіцієнта стиснення інформації асимптотично наближається до величини 1,8.

Наприклад:

$0 \le Y_{ij} \le 10; j \in 4,$ тоді $P_j = (10,11,12,13)$ і $P - 1 = 17159$

$$k_c = 1 + 4 \cdot \frac{2}{15} = 1 + 0{,}53 \approx 1.5.$$

1.2 На основі аналізу характеристик та функціональних можливостей використання ТЧБ для формування структуризованих даних (СД) на низових рівнях РКС розроблений метод компактного завадозахищеного формування СД на основі ТЧБ Крестенсона, який описується наступним алгоритмом:

1.3

$$\left.\begin{array}{l} x_1 \overset{\cup}{\rightarrow} x_{i1} \rightarrow p_1 \rightarrow b_1 \\ x_2 \overset{\cup}{\rightarrow} x_{i2} \rightarrow p_2 \rightarrow b_2 \\ \ldots \\ x_j \overset{\cup}{\rightarrow} x_{ij} \rightarrow p_j \rightarrow b_j \\ \ldots \\ x_m \overset{\cup}{\rightarrow} x_{im} \rightarrow p_{k-1} \rightarrow b_{k-1} \\ D_i \rightarrow p_k \rightarrow b_k \end{array}\right\} N_k = res\sum_{j=}^{k} b_j B_j (\mathrm{mod}\ P) \qquad (2.19)$$

$$B_j = \frac{P}{p_j} m_j = 1(\mathrm{mod}\, P_j),$$

де $x_j(t)$ - аналогові дані телеметрії;

x_{ij} - цифрові дані телеметрії;

D_i – техніко-економічні дані;

$p_1, p_2 \ldots p_k$ – система взаємно простих модулів;

$b_1, b_2 \ldots\ldots b_j \ldots\ldots b_k$ – набір найменших невід'ємних залишків;

B_j – система ортогональних базисів СЗК.

2.3 Каскадне кодування даних на основі методу залишків та системи залишкових класів

Процеси кодування та декодування інформації методом залишків виконуються наступним чином. Вхідний масив даних на деякому інтервалі часу описується матрицею:

$$Y_{ij} = \begin{vmatrix} Y_{11} & \dots & Y_{i1} & \dots & Y_{k1} & \dots & Y_{n1} \\ Y_{12} & \dots & Y_{i2} & \dots & Y_{k2} & \dots & Y_{n2} \\ . & . & . & . & . & . & . \\ Y_{1k} & \dots & Y_{ik} & \dots & Y_{kk} & \dots & Y_{nk} \end{vmatrix}, \qquad (2.20)$$

де Y_{ij}- цифровий відлік *i*-го вимірювання *j*-го каналу *n*, *k*- число відліків (n>k).

Застосувавши метод залишків, матрицю (2.20) перетворюємо у матрицю залишків $\|b_{ij}\|$ шляхом нелінійної модульної операції в коді поля Галуа.

$$\left\|b_{ij}\right\| = resY_{ij}(\mathrm{mod}\, P_j), i \in \overline{1,n};\ j \in \overline{1,k}, \qquad (2.21)$$

де $P_j - 1 \geq 2\left|Y_{ij} - Y_{i-1\,j}\right|\max$ - є умовою однозначності перетворень.

Вираз (2.21) відповідає лінійній формі:

$$Y_{ij} = a_{ij} \cdot P_j + b_{ij},$$

де $a_{ij} = \overset{\vee}{E}\left[\frac{Y_{ij}}{P_j}\right]$ - ранг відліку Y_{ij} по модулю P_j.

Тому для однозначного відновлення цифрових відліків матриці (2.20) на основі рекурентності формули:

$$Y_{i+1,\,j} = \overset{\vee}{E}\left[\frac{Y_{ij} - b_{i+1,\,j}}{P_j}\right]P_j + b_{ij} \qquad (2.22)$$

необхідно матрицю залишків (2.21) доповнити вектором-строкою опорних рангів $a_{ij}(i \in \overline{1,k})$ і елементом *X*,

$$\begin{vmatrix} b_{11} & \dots & b_{i1} & \dots & b_{k1} & b_{k+1,1} & \dots & b_{n1} \\ b_{12} & \dots & b_{i2} & \dots & b_{k2} & b_{k+1,2} & \dots & b_{n2} \\ . & . & . & . & . & . & . & . \\ b_{1j} & \dots & b_{ij} & \dots & b_{kj} & b_{k+1,j} & \dots & b_{nj} \\ . & . & . & . & . & . & . & . \\ b_{1k} & \dots & b_{ik} & \dots & b_{kk} & b_{k+1,k} & \dots & b_{nk} \\ a_{1k+1} & \dots & a_{ik+1} & \dots & b_{k,k+1} & X_{k+1} & \dots & X \end{vmatrix}, \qquad (2.23)$$

де елементи $0 \le X \le P_{k+1} - 1$ утворюють поле довільних інформаційних повідомлень по mod P_{k+1}.

Таким чином, однозначність перетворень матриць (2.23) та (2.20) забезпечується в розширеній СЗК по модулю P_{k+1}. Тоді, застосувавши зворотнє перетворення СЗК у розширеній системі модулів $P = P_1 \cdot P_2 \cdot ... P_k \cdot P_{k+1}$, отримаємо:

$$N_{i,k+1} == res \sum_{j=1}^{k+1} C_{ij} \cdot B'_j (\mathrm{mod}\, P),$$

де C_{ij} елементи векторів стовбців

$$C_{il} = \begin{pmatrix} b_{i1} \\ \dots \\ b_{ik} \\ a_{i,k+1} \end{pmatrix}, \qquad C_{il} = \begin{pmatrix} b_{i1} \\ \dots \\ b_{ik} \\ X \end{pmatrix}$$

$$1 \le i \le k; \qquad k+1 \le i \le n;$$

B_{ij} – ортогональні базиси розширеної СЗК.

Структура даного перетворення СЗК має наступний вигляд:

$$\left.\begin{array}{l} x_1 \circlearrowright \to x_{i1} \to b_{i1}(\mathrm{mod}\, q_1) \to p_1 \to b_1 \\ x_2 \circlearrowright \to x_{i2} \to b_{i2}(\mathrm{mod}\, q_2) \to p_2 \to b_2 \\ \ldots \\ x_j \circlearrowright \to x_{ij} \to b_{ij}(\mathrm{mod}\, q_j) \to p_j \to b_j \\ \ldots \\ x_{k-1} \circlearrowright \to x_{ik-1} \to b_{ik-1}(\mathrm{mod}\, q_{k-1}) \to p_{k-1} \to b_{k-1} \\ D_k \to p_k \to b_k \end{array}\right\} N_k = res\sum_{j=}^{k} b_j B_j (\mathrm{mod}\, P), \tag{2.24}$$

де D_k - службові алфавітно-цифрові дані

Для однозначності перетворення (2.24) діапазони квантування відліків A_j по каналах розраховуються по формулі:

$$A_j = P_j \cdot P_{k+1} - 1.$$

Для захисту кодових слів $N_{i,k+1}$ від помилок необхідно введення деякої надлишковості шляхом застосування нерозділимих арифметичних AN-кодів, що дозволяє ефективно виявляти та виправляти помилки при відносно короткій їх двійковій довжині 16 - 64 біт.

Тому, розширивши набір модулів:

$$\rho' = P_0 \cdot P_1 \cdot P_2 \cdot \ldots \cdot P_{k+1},$$

та визначивши новий набір базисних чисел B_i'' і доповнивши матрицю (2.24) нульовим стовбцем і строкою, отримаємо:

$$\begin{vmatrix} b_{0,0} & b_{1,0} & \dots & b_{k,0} & b_{k+1,0} & \dots & b_{n,0} \\ b_{1,0} & b_{1,1} & \dots & b_{k,1} & b_{k+1,1} & \dots & b_{n,1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{0,j} & b_{0,j} & \dots & b_{0,j} & b_{k+1,j} & \dots & b_{n,j} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{0,k} & b_{1,k} & \dots & b_{k,k} & b_{k+1,k} & \dots & b_{n,k} \\ X_0 & a_1 & \dots & a_k & X_{k+1} & \dots & X_n \end{vmatrix} \quad (2.25)$$

В названих системах вектор стовбець $b_{0,0},...b_{0,k}$ - використовується для ідентифікації блоку даних матриці залишків $b_{i,j}$ та стартової інформації. Символу X_0 присвоюється код стану об'єкта управління $X_0 = S_i$, параметри якого описуються блоком даних (2.25). Вектор-строка є нульовим, тобто всі значення $b_{i,0} = 0$, що забезпечує захист даних від помилок. Символи $X_{k+1},...X_n$ використані для кодування даних, які вводяться оператором з пульта і кодуються півбайтами (4 біти) по модулю $P_{k+1} = 16$.

Застосування модуля $P_{k+1} = 16$ при кодуванні $N_{i,k+1}$ у двійковій системі числення дозволяє виділяти службову інформацію по $\mathrm{mod}\, P_0$ без декодування чисел $N_{i,k+1}$ шляхом вилучення 4-х молодших бітів. Тобто:

$$b_{i,k} = (a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0)_N;\ a_i \in \overline{0,1}.$$

Позитивними якостями викладених методів пакування даних на основі сум полів Галуа та СЗК є наступні:

- строга зворотність та однозначність перетворення інформаційних повідомлень;
- неадаптивність процедур перетворення;
- рівномірність швидкості інформаційних потоків на виході кодера;
- формування байторієнтованих інформаційних даних;
- структуризованість блоків даних та їх захист від помилок;
- зменшення надлишковості інформації.

До недоліків слід віднести деяку громіздкість зворотного перетворення СЗК у цілочисельній формі, що включає операції множення, ділення та потребує розрахунку нормуючих вагових коефіцієнтів m_i.

2.4 Алгоритм пошук залишків великорозрядних чисел в розмежованій системі числення Радемахера-Крестенсона

Для знаходження $b_i = b \bmod p_i$ слід скористатися розмежованою системою числення Радемахера-Крестенсона, яка дозволяє зменшити обчислювальну складність з $O1((n+1)^2 + n)$ до $O(\log_2 n/2)$ реалізації даної операції. Представимо b у двійковій формі: $b=b_{n-1}2^{n-1} + b_i \cdot 2^i + \ldots + b_1 \cdot 2^1 + + b_0 \cdot 2^0$, де b_i=0,1, і сформуємо таблицю 2.1.

Таблиця 2.1 – Знаходження залишків степенів двійки

2^{n-1}	2^{n-2}	…	2^i	…	2	1
b_{n-1}	b_{n-2}	…	b_i	…	b_1	b_0
$p_{1\,n-1}$	$p_{1\,n-2}$	…	$p_{1\,i}$	…	$p_{1\,1}$	$p_{1\,0}$

Щоб знайти елемент p_{1i} необхідно попередній елемент p_{1i-1} домножити на 2 (дописати в кінці 0 у двійковому записі) і порівняти з модулем p_i. Остаточна формула для p_{1i} матиме такий вигляд:

$$p_{1i} = \begin{cases} 2 \cdot p_{1\,i-1}, & 2 \cdot p_{1\,i-1} < p_i \\ 2 \cdot p_{1\,i-1} - p_1, & 2 \cdot p_{1\,i-1} \geq p_1. \end{cases}$$

На рисунку 2.2 показано графіки залежності обчислювальних складностей від розрядності чисел *n*. З рисунка видно, що використання запропонованого алгоритму, який грунтується на використанні розмежованої системи числення Радемахера–Крестенсона, дозволяє істотно зменшити

обчислювальну складність пошуку залишків великорозрядних чисел відносно класичного.

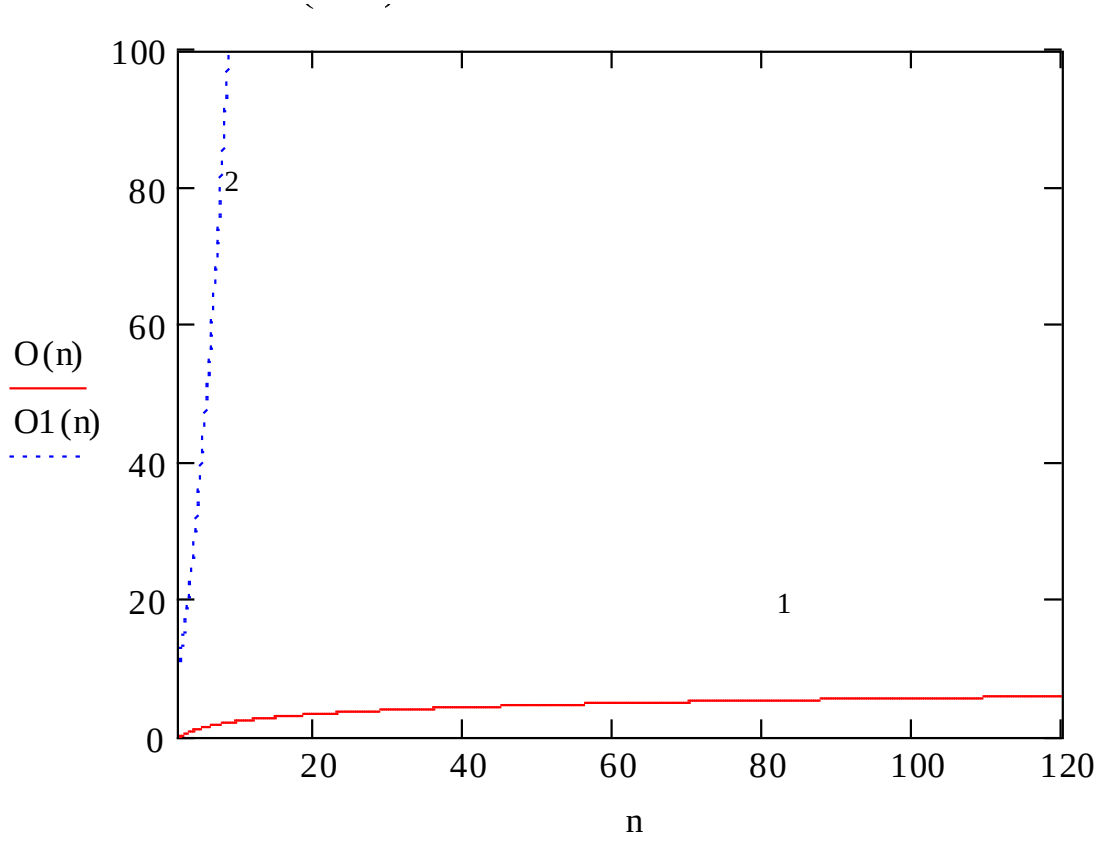


Рисунок 2.2 – Графіки залежності обчислювальних складностей від розрядності чисел n запропонованим методом (1) та класичним (2).

Основними перевагами запропонованих теоретичних положень та алгоритмів є зменшення обчислювальної складності, що відкриває нові перспективи високопродуктивного опрацювання великорозрядних чисел в задачах захисту інформаційних потоків в комп'ютерних системах та знаходження мультистепеневої функції за заданим модулем.

2.5 Оцінка швидкодії суматора по модулю *Pj*

Згідно архітектури пірамідального та лінійного міжбазисних перетворень по модулю P_j, показаних на рисунках 2.3 та 2.4, порівняння їх характеристик

приведено на рисунку 2.3, звідки видно, що лінійна архітектура потребує в два рази менше обладнання по відношенню до пірамідальної.

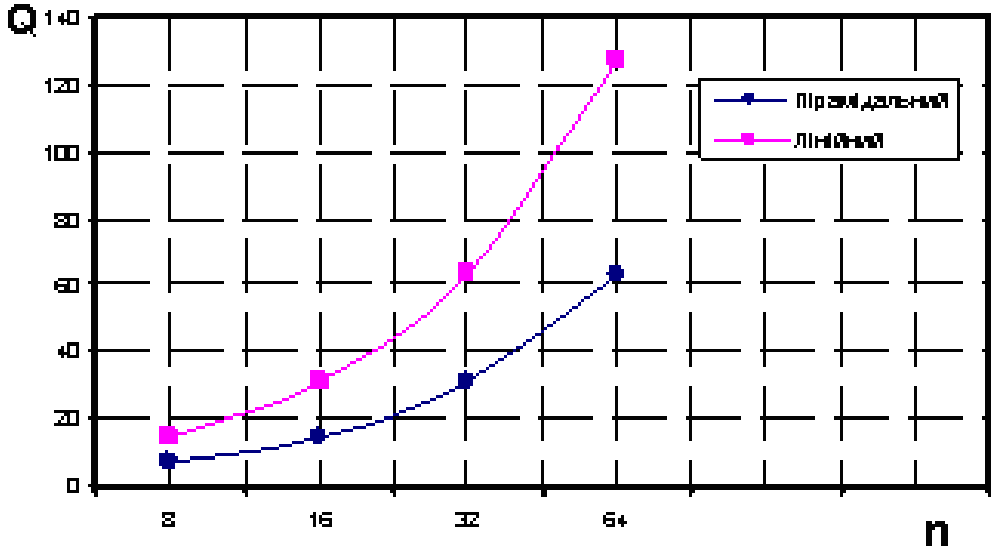


Рисунок 2.3 – Об’єм мікроелектронного обладнання міжбазисного перетворення Радемахера–Крестенсона.

Результати аналізу швидкодії двох досліджуваних архітектур міжбазисного перетворення при унітарному кодуванні залишків (див. рисунок 2.4) розраховується згідно виразів:

$$V_p = \frac{1}{2+\log_2 n};\ V_l = \frac{1}{n};$$

де n – число розрядів; V_p – швидкодія пірамідального МБП;

V_l – швидкодія лінійного МБП.

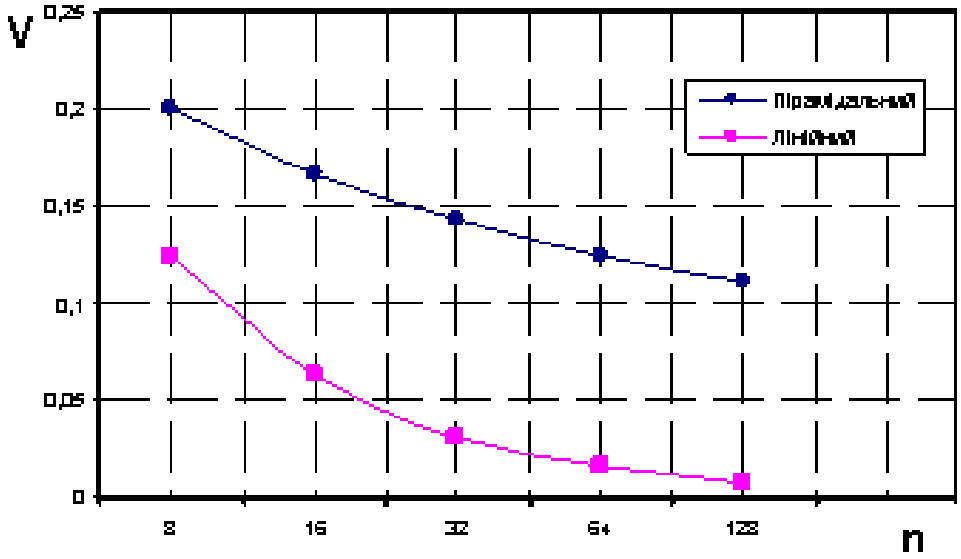


Рисунок 2.4 – Швидкодія міжбазисного перетворення Радемахера–Крестенсона.

3 ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО, ПРОГРАМНОГО, ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору основних проектних рішень

3.1.1 Короткий опис C ++ Builder

C++ (Сі-плюс-плюс або Сі-плас-плас) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом (англ. Bjarne Stroustrup) в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) у 1979 році та названа «Сі з класами». Страуструп перейменував мову у C++ у 1983 р. Базується на мові C. Визначена стандартом ISO/IEC 14882:2003 [1].

У 1990-х роках C++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відео ігри. C++ суттєво вплинула на інші, популярні сьогодні, мови програмування: C# та Java.

При створенні C++ прагнули зберегти сумісність з мовою C. Більшість програм на C справно працюватимуть і з компілятором C++. C++ має синтаксис, заснований на синтаксисі C.

Нововведеннями C++ порівняно з C є:

- підтримка об'єктно-орієнтованого програмування через класи;
- підтримка узагальненого програмування через шаблони;
- доповнення до стандартної бібліотеки;
- додаткові типи даних;
- обробка винятків;
- простори імен;
- вбудовані функції;
- перевантаження операторів;
- перевантаження імен функцій;

- посилання і оператори управління вільно розподіленою пам'яттю.

У 1998 році ратифіковано міжнародний стандарт мови C++: ISO/IEC 14882 «Standard for the C++ Programming Language». Поточна версія цього стандарту — ISO/IEC 14882:2003.

Стандарт Сі++ на 1998 рік складається з двох основних частин: ядра мови і стандартної бібліотеки. Стандартна бібліотека Сі++ увібрала в себе бібліотеку шаблонів STL, що розроблялася одночасно із стандартом. Зараз назва STL офіційно не вживається, проте в кругах програмістів на Сі++ ця назва використовується для позначення частини стандартної бібліотеки, що містить визначення шаблонів контейнерів, ітераторів, алгоритмів і функторів.

Стандарт Сі++ містить нормативне посилання на стандарт Сі від 1990 року і не визначає самостійно ті функції стандартної бібліотеки, які запозичуються із стандартної бібліотеки Сі.

Поза тим, існує величезна кількість бібліотек Сі++, котрі не входять в стандарт. У програмах на Сі++ можна використовувати багато бібліотек Сі.

Стандартизація визначила мову програмування Сі++, проте за цією назвою можуть ховатися також неповні, обмежені достандартні варіанти мови. Спочатку мова розвивалася поза формальними рамками, спонтанно, у міру завдань, що ставилися перед ним. Розвиток мови супроводив розвиток кросс-компілятора Cfront. Нововведення в мові відбивалися в зміні номера версії кросс-компілятора. Ці номери версій кросс-компілятора розповсюджувалися і на саму мову, але стосовно теперішнього часу мову про версії мови Сі++ не ведуть.

Borland C ++ Builder, сьогодні є одним з найбільш досконалим візуальним середовищем швидкої розробки на Сі ++ для Windows. До її складу входить близько 200 різних компонентів, а створення закінченої програми вимагає мінімуму зусиль. Найближчий конкурент Borland C ++ Builder - це не система Microsoft Visual C ++, яка побудована за іншою схемою і Microsoft Visual Basic. Проте ефективність програм, що створюються за допомогою C ++ Builder, в десятки разів перевершує швидкодію програм, написаних на MS Visual Basic.

Та й за кількістю вільних доступних компонентів рівних середовищу C ++ Builder сьогодні не знайти.

У цієї системи є рідний брат – середовище Borland Delphi, технологія роботи з якої повністю збігається з технологією, прийнятою в C ++ Builder. Тільки в Delphi програмний код пишеться не на мові C ++, а на мові програмування Паскаль, точніше на його об'єктно-орієнтованої версії ObjectPascal. Але найцікавіше, що Borland C ++ Builder дозволяє писати програму при бажанні одночасно і на Сі ++, і на Паскалі!

Замість окремого інструментарію, що оперує візуальними елементами управління, в C ++ Builder інтегрована так звана Палітра компонент, розділена картотечними вкладками на декілька функціональних груп. Функціональні можливості поставляються компонент можна досить просто модифікувати, а також розробляти компоненти, що володіють абсолютно новим оригінальним поведінкою.

Система містить Бібліотеку з понад 100 повторно використовуваних візуальних компонент, які перетягуються мишею на форму і відразу стають елементами управління прототипу вашої програми. Крім відомих елементів управління Windows (кнопки, лінійки прокрутки, поля редагування, прості і комбіновані списки і т.д.) Бібліотека містить нові компоненти підтримки діалогів, обслуговування баз даних та багато інших.

Після розміщення компонент на формі, Інспектор об'єктів допоможе встановлювати їх властивості і наказувати дії кодам обробки. Проект будуватиметься поступово, на тлі вироблених вами змін у властивостях, подіях і функціях використовуваних елементів. Добре продумано розділення і редагування програмного модуля по двох його частинах: інтерфейсною і власне кодовою.

C ++ Builder підтримує основні принципи об'єктно-орієнтованого програмування - інкапсуляцію, поліморфізм і множинне спадкування, а також нововведені специфікації і ключові слова в стандарті мови C ++.

C ++ Builder забезпечує високу швидкодію при компіляції і збірці 32-розрядних додатків для сучасних операційних систем Windows 95 і Windows

NT, включаючи OLE взаємодія клієнт-сервер. Система навіть відображає час, витрачений на основні етапи побудови програм. Результуючі програми добре оптимізовані за швидкістю виконання і витрат пам'яті. Хоча налагоджувальний режим низького рівня повністю інтегрований в середу C ++ Builder, до налагодження також довелося звикати. Дизайнер форм. Інспектор об'єктів і інші засоби залишаються доступними під час роботи програми, тому вносити зміни можна в процесі налагодження.

C ++ Builder підтримує зв'язок з різними базами даних 3-х видів: dBASE і Paradox: Sybase, Oracle, InterBase і Informix; Excel, Access, FoxPro і Btrieve. Механізм BDE (Borland Database Engine) додає обслуговуванню зв'язків з базами даних дивовижну простоту і прозорість. Провідник Database Explorer дозволяє зображати зв'язки і об'єкти баз даних графічно.

Завдяки засобам управління проектами, двосторонньої інтеграції додатка й синхронізації між засобами візуального і текстового редагування, а також вбудованому відладнику (з асемблерним вікном прокрутки, покроковим виконанням, точками зупинки, трасуванням і т.п.) - C ++ Builder корпорації Borland представляє собою вражаюче середовище розробки .

3.1.2 Переваги мови C++

Швидкодія. Швидкість роботи програм на C++ практично не поступається програмам на C, хоча програмісти отримали в свої руки нові можливості і нові засоби.

Масштабованість. На мові C++ розробляють програми для самих різних платформ і систем.

Можливість роботи на низькому рівні з пам'яттю, адресами, портами. (Що, при необережному використанні, може легко перетворитися на недолік.)

Можливість створення узагальнених алгоритмів для різних типів даних, їх спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів.

Підтримуються різні стилі та технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси).

3.1.3 Недоліки мови C++

Наявність безлічі можливостей, що порушують принципи типобезпеки приводить до того, що в C++ програми може легко закрастися важковловима помилка. Замість контролю з боку компілятора розробники вимушені дотримуватися вельми нетривіальних правил кодування. По суті ці правила обмежують C++ рамками якогось безпечнішої підмови. Більшість проблем типобезпеки C++ успадкована від C, але важливу роль в цьому питанні грає і відмова автора мови від ідеї використовувати автоматичне управління пам'яттю (наприклад, збірку сміття). Так візитною карткою C++ стали вразливості типу «переповнювання буфера».

Погана підтримка модульності. Підключення інтерфейсу зовнішнього модуля через препроцесорну вставку заголовного файлу (#include) серйозно уповільнює компіляцію, при підключенні великої кількості модулів. Для усунення цього недоліку, багато компіляторів реалізують механізм прекомпіляциі заголовних файлів (англ. Precompiled Headers).

Недостача інформації про типи даних під час компіляції (CTTI).

Мова C++ є складною для вивчення і для компіляції.

Деякі перетворення типів неінтуїтивні. Зокрема, операція над беззнаковим і знаковим числами видає беззнаковий результат.

Препроцесор C++ (успадкований від C) дуже примітивний. Це приводить з одного боку до того, що з його допомогою не можна (або важко) здійснювати деякі завдання метапрограмування, а з іншою, в наслідку своєї примітивності, він часто приводить до помилок і вимагає багато дій з обходу потенційних проблем. Деякі мови програмування (наприклад, Scheme і Nemerle) мають набагато могутніші і безпечніші системи метапрограмування (також звані макросами, але макроси C/C++ вони мало нагадують).

З кінця 1990-х в співтоваристві C++ набуло поширення так зване метапрограмування на базі шаблонів. По суті, воно використовує особливості шаблонів C++ в цілях реалізації на їхній базі інтерпретатора примітивної функціональної мови програмування, що виконується під час компіляції. Сама

по собі ця можливість вельми приваблива, але, внаслідкок вище згаданого, такий код вельми важко сприймати і відладжувати. Мови Lisp/Scheme, Nemerle і деякі інші мають могутніші і водночас простіші для сприйняття підсистеми метапрограмування. Крім того, в мові D реалізована порівнянна по потужності, але значно простіша в застосуванні підсистема шаблонного метапрограмування.

Хоча декларується, що C++ мультипарадигмена мова, реально в мові відсутня підтримка функціонального програмування. Частково, даний пропуск усувається різними бібліотеками (Loki, Boost) що використовують засоби метапрограмування для розширення мови функціональними конструкціями (наприклад, підтримкою лямбд/анонімних методів), але якість подібних рішень значно поступається якості вбудованих у функціональні мови рішень. Такі можливості функціональних мов, як зіставлення зі зразком взагалі украй складно емулювати засобами метапрограмування.

C++ успадкував багато проблем мови C:

Операція присвоювання позначається як =, а операція порівняння як == . Їх легко сплутати, і така конструкція буде синтаксично правильною, але приведе до важковломимого багу. Особливо часто це відбувається в операторах if і while, наприклад, програміст може написати if (i=0) замість if (i==0) (Разом з тим, основна маса компіляторів видає в таких випадках попередження.) Уникнути помилку такого типу можна, якщо писати всі операції порівняння у такому вигляді: if (0==i). До того ж багато мов (Бейсик, Паскаль) використовують символ «=» саме в операціях порівняння.

Операції присвоювання (=), інкрементації (++), декрементації (--) та інші повертають значення. У поєднанні з великою кількістю операцій це дозволяє, але не зобов'язує, програміста створювати код, що важко читається. З іншого боку, один з основних принципів мов C і C++ — дозволяти програмістові писати в будь-якому стилі, а не нав'язувати «хороший» стиль. До того ж це іноді дозволяє компілятору створювати оптимальніший код.

Макроси (#define) є могутнім, але небезпечним засобом. У мові C++, на відміну від C, необхідність в небезпечних макросах з'являється значно рідше

завдяки шаблонам і вбудованим функціям. Але в успадкованих стандартних С-бібліотеках багато потенційно небезпечних макросів.

Дехто вважає недоліком мови С++ відсутність системи збірки сміття. З іншого боку, в С++ є достатньо засобів (класи з конструкторами і деструкторами, стандартні шаблони, передача параметрів за посиланням), що дозволяють майже виключити використання небезпечних вказівників. Проте, відсутність вбудованої збірки сміття дозволяє користувачеві самому вибрати стратегію управління ресурсами.

Крім того, збірка сміття серйозно уповільнює роботу програми, і це недолік там, де продуктивність є критично важливою.

3.2 Розробка інтерфейсу користувача

При розробці програмного забезпечення було використано наступні класи компонент:

TSpeedButton – кнопки.

TPanel - цей компонент - панель - є несучою конструкцією для розміщення інших елементів управління. На відміну від простої рамки (TBevel) панель сама є віконним елементом управління і батьком для всіх розміщених на ній компонентів. Часто її використовують для створення панелей інструментів, рядків стану і т. п.

TStaticText - цей клас доповнює стандартний клас TLabel тим, що є похідним TWinControl, тобто містить віконний дескриптор. Треба відзначити, що, будучи віконним компонентом, цей варіант мітки може розташовуватися на (або над) іншими віконними компонентами у відповідності з z-order.

TTimer - компонент інкапсулює функції таймера Windows API.

TPopupMenu - цей компонент описує спливаюче меню. На відміну від головного, власне меню такого типу може бути майже у кожного віконного елемента управління на формі (крім перемикачів), а також у самої форми.

VCL компонент TMemo є обгорткою стандартного багаторядкового елемента редагування (multiline edit control) Windows. TMemo реалізує стандартну поведінку класу TCustomMemo, публікуючи багато його властивості, але не додаючи своєї власної поведінки.

У компоненті TEdit (Поле введення) зберігається текст, який можна поміщати в даний компонент, як під час проектування, так і під час виконання програми. Компонент класу TEdit являє собою однорядкове редаговане текстове поле. З його допомогою можна вводити і / або відображати досить довгі текстові рядки.

На рисунку 3.1 зображено розміщення об’єктів вищенаведених класів на головній формі додатку.

Також було створено два модуля, що створюють відповідно два обчислювальні потоки з однаковими пріоритетами, що дозволяє порівняти обчислювальні можливості двох методів.

Для створення потоків використовуються можливості класу TThread. З точки зору операційної системи потік - це її об'єкт. При створенні він отримує дескриптор і відстежується ОС.

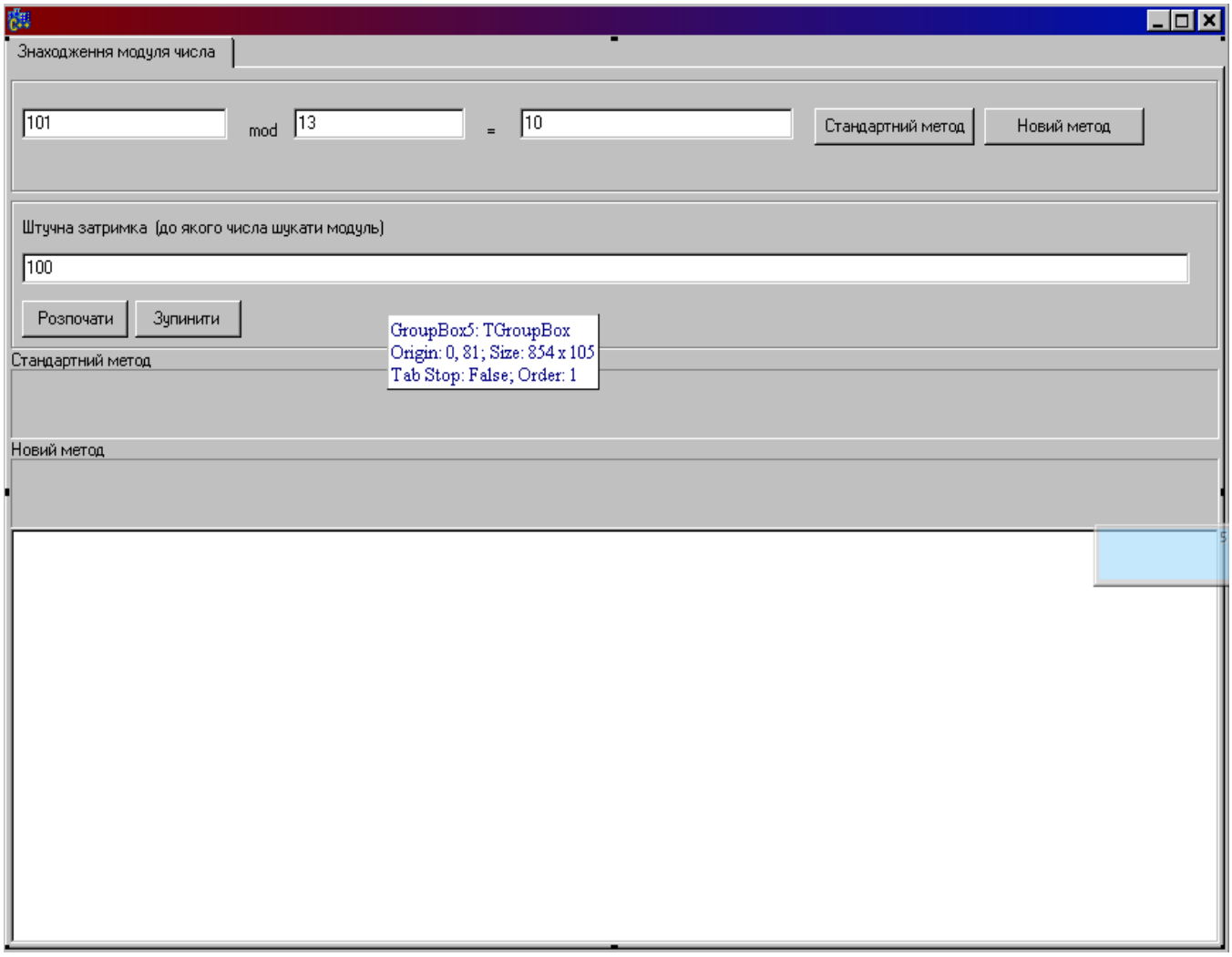


Рисунок 3.1 – Головна форма додатку. Етап розробки.

Об'єкт класу TThread - це конструкція C++ Builder, відповідна потоку ОС. Цей об'єкт VCL створюється до реального виникнення потоку в системі і знищується після його зникнення.

Хоча формальний опис Execute - метод abstract, але майстер створення нового об'єкта TThread створює для вас порожній шаблон цього методу.

Перевизначаючи метод Execute, ми можемо тим самим закладати в новий потоковий клас те, що буде виконуватися при його запуску. Якщо потік був створений з аргументом CreateSuspended, рівним False, то метод Execute виконується негайно, інакше Execute виконується після виклику методу Resume.

Якщо потік розрахований на однократне виконання будь-яких дій, то ніякого спеціального коду завершення всередині Execute писати не треба.

Конструктор об'єкта:

constructor Create (CreateSuspended: Boolean);

отримує параметр CreateSuspended. Якщо його значення дорівнює True, новостворений потік не починає виконуватися до тих пір, поки не буде зроблений виклик методу Resume. У випадку, якщо параметр CreateSuspended має значення False, конструктор завершується і тільки потім потік починає виконання.

TProgressBar призначений для відображення процесів, які займають помітне час, наприклад, копіювання великих файлів, налаштування програми, установку програми на комп'ютері і т.п. Інший компонент, який виконує аналогічні функції - TGauge.

Компонент TProgressBar являє собою горизонтальну або вертикальну смугу, яку заповнюють у міру розвитку відображуваного процесу. Основна властивість компонента - Position. Це позиція, яку можна задавати по мірі протікання процесу, починаючи зі значення Min на початку процесу, і закінчуючи значенням Max в кінці. Властивість Orientation визначає горизонтальну або вертикальну орієнтацію шкали компонента. Властивість Smooth визначає безперервне або дискретне відображення процесу. Відображення ходу процесу можна здійснювати, задаючи значення позиції - Position. Наприклад, якщо повна тривалість процесу характеризується значенням цілої змінної Count (обсяг всіх копійованих файлів, число налаштувань, кількість циклів якогось процесу), а виконана частина - цілої змінної Current, то задавати позицію діаграми у випадку, якщо використовуються значення мінімальної і максимальної позиції за замовчуванням (тобто 0 і 10), можна оператором

3.3 Розробка основних функціональних можливостей

Для роботи з великорозрядними числами була розроблена бібліотека на мові C++. Ця бібліотека оперує з вказівниками на динамічні масиви. Нижче наведена нотація структури класу:

```
class bitarray
{
 public:
 bool *mainarray;
 long array_size;
 long number_size;
 long count_of_smaller_bit0;// покишо не використовується
 bool is_one;
 long cikle;
 int mul2();
 int minusbit(long);
 int minus(bitarray &vidyemnyk);
 bool isbigger(bitarray &another);
 AnsiString ToString();
 void ToVector(vector<bool>& temp);
 int nullarray();
 long to_long();
 long addbit(long bit);
 long addbitarray(bitarray &number);
bitarray();
 bitarray(long);
 bitarray(AnsiString);
 bitarray(long,long);
 bitarray(AnsiString,long);
 bitarray(vector<bool>& temp,long,long);
 ~bitarray(void){delete[] mainarray;};
 void random_(long size);
 static void calculate_part(bitarray &number);
```

```
static long mod(bitarray &number,bitarray &modul);
 static long modpresent(bitarray &number,bitarray &modul,TMemo *memo);
 static double bitarray::mod_time (bitarray &number,bitarray &modul);
void newsize(long numbersize);
 static void randomnumber(bitarray &number,long countof_bit);
static void from_bitarray_to_verylong(bitarray &number,verylong *a);
 static AnsiString from_verylong_string_binary(verylong a);
static AnsiString from_bitarray_to_verylong_(bitarray &number,verylong a);
static bool save_to_file(AnsiString filename,verylong a);
 static bool load_from_file(AnsiString filename,verylong *a);
};
```

Пам'ять виділяється динамічно з використання оператора new і delete.

New ключове слово виділяє пам'ять для об'єкта або масив об'єктів типу name з вільної доступної памяті і повертає відповідний типізований, ненульовий покажчик на об'єкт. У разі невдачі, new повертає нуль або генерує виняток. Коли new використовується для виділення пам'яті для C + + об'єкт класу, конструктор об'єкта викликається після виділення пам'яті.

За допомогою оператора delete відбувається звільнення пам'яті, виділеної з оператором new.

Для знаходження залишку великорозрядних чисел по високорозрядному модулю розроблена спеціальна функція.

```
long bitarray::mod(bitarray &number,bitarray &modul){
bitarray res(1,number.array_size);
for (int i=number.number_size-2;i>=0;i--){
res.mul2();
if (number.mainarray[i])res.mainarray[0]=true;
if (res.isbigger(modul))res.minus(modul);
};
return res.to_long();
};
```

Функція повертає залишок представлений в форматі long.

Для динамічного виділення і перерозподілу оперативної памяті створено відповідну функцію newsize:

```
void bitarray::newsize(long numbersize){
delete[ ] mainarray;
 cikle=0;
 array_size=numbersize;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
 nullarray();
 int i;
 mainarray[0]=false;
 number_size=1;
};
```

Нижче наведена функція addbit призначена для додавання одного біта до велико-розрядного числа.

```
long bitarray::addbit(long bit){
 if (bit<array_size){
 int i;
 for (i=bit;i<number_size&&mainarray[i]==true;i++)mainarray[i]=false;
 if (i<number_size)mainarray[i]=true ;
  if (i==number_size){mainarray[i]=true;number_size++;};
  if (i>number_size){
   for (i=number_size;i<bit;i++){mainarray[i]=false;};
   mainarray[bit]=true;
   number_size=bit+1;
  };
  };
  return 1;
};
```

Використовуючи функцію addbit було створено функцію для додавання двох велико-розрядних чисел.

```
long bitarray::addbitarray(bitarray &number){
 for (int i=0;i<number.number_size;i++){
  if (number.mainarray[i])addbit(i);
 };
return 1;
};

bitarray::bitarray(){
randomize();

};
```

Для множенння великорозрядного числа на два, а також збільшення його розміру на один нижній біт створена функція mul2().

```
int bitarray::mul2(){ //перевірено
 if (bitarray::to_long()!=0){
 if (number_size<array_size){
  for (int i=number_size;i>=0;i--){
   if (mainarray[i]){
    mainarray[i]=false;
    mainarray[i+1]=true;
   };
  };
  number_size++;
 }; };
 //count_of_smaller_bit0++;
 return 1;
};
```

Для віднімання одного біту від числа створена функція minusbit, яка в дечому аналогічна функції додавання біту.

```
int bitarray::minusbit(long bitnumber){ //перевірено
 //if (bitnumber=>0&&bitnumber<=number_size+count_of_smaller_bit0){
  if
((bitnumber>=0&&bitnumber<number_size)&&count_of_smaller_bit0==0){
  int i,a=0;
  if (mainarray[bitnumber]==true){
   mainarray[bitnumber]=false;
  }else {
  for
(i=bitnumber;i<number_size&&mainarray[i]==false;i++)mainarray[i]=true;
  mainarray[i]=false;
  };

  for (i=0;i<number_size;i++)
  if (mainarray[i])a=i;
  number_size=a+1;
 };
 if (number_size==1&&mainarray[0]==true)is_one=true;
//};
 return 1;
};
```

Функція minus забезпечує віднімання двох великорозрядних чисел і побудована на основі функції minusbit.

```
int bitarray::minus(bitarray &vidyemnyk){//перевірено
 if (isbigger(vidyemnyk)){

  for (int i=vidyemnyk.number_size-1;i>=0;i--){
   if (vidyemnyk.mainarray[i]==true)bitarray::minusbit(i);
    };
 };
return 1;
```

```
};
```

Також важливою функцією в бібліотеці для роботи з велико-розрядними числами є функція isbigger.

```
bool bitarray::isbigger(bitarray &another){//перевірено
 bool is=false;
 if (another.number_size<number_size+count_of_smaller_bit0) is=true;
 if (another.number_size>number_size+count_of_smaller_bit0) is=false;
 if (another.number_size==number_size+count_of_smaller_bit0){
  int i;
  for (i=another.number_size-1;i>0&&mainarray[i]==another.mainarray[i];i--);
  if (mainarray[i]){is=true;} else {is=false;};
 };
 return is;

// return true;
};
```

Вижче наведені функції забезпечують основні функціональні можливості на основі, яких можна будувати складніші алгоритми та розвивати біліотеку.

3.4 Тестування та відлагодження програмного продукту

Тестування програмного забезпечення – техніка контролю якості, що перевіряє відповідність між реальною та очікуваною поведінкою програми завдяки кінцевому набору тестів, що обираються певним чином.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування як процес, своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Зазвичай, поняття якості обмежується такими поняттями, як коректність, надійність, практичність, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Склад і зміст супутньої документації процесу тестування, визначається стандартом IEEE 829-1998 Standard for Software Test Documentation. Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів — це по суті дослідницький та творчий процес, а не тільки створення і виконання рутинної процедури.

Отже для проведення тестування необхідно провести перевірку функціональних можливостей додатку.

Після запуску додатку на екрані буде відображена форма що на рисунку 3.2.

На головній формі додатку зображено елементи що дозволяють знайти залишок числа по вказаному модулю двома методами, кнопки :

- «Стандартний метод »
- «Новий метод»

Також додаток дозволяє порівняти швидкість виконання опреацій знаходження залишку двома методами над впорядкованою числовою послідовністю з випадково згенерованими модулями.

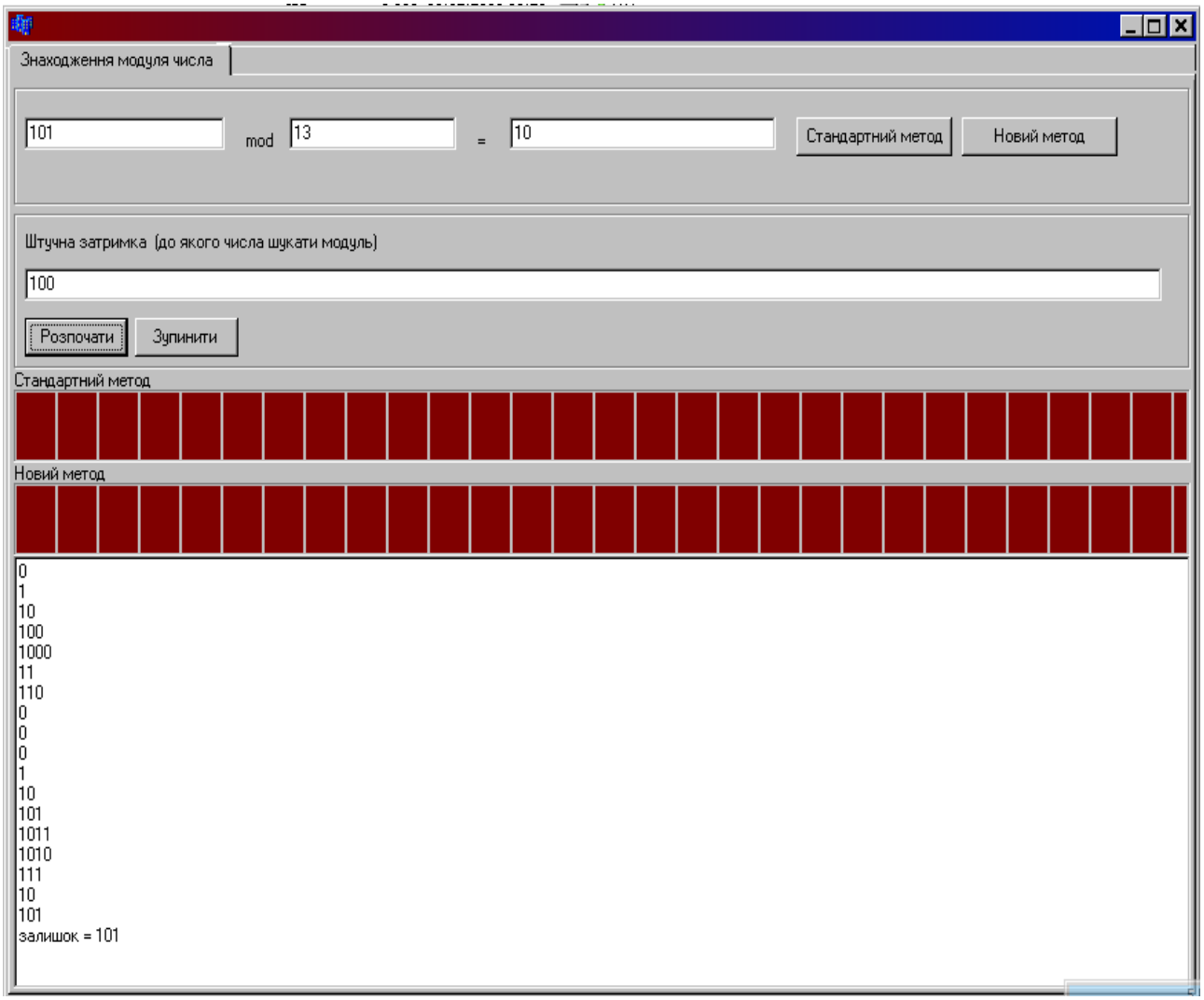


Рисунок 3.2 – Головна форма додатку.

Верхню межу чисел можна вказати у відповідному полі головної форми.

3.5 Застосування та дослідження методу залишків в теорії передачі даних

Теоретико-числовий базис Крестенсона породжує систему числення залишкових класів (СЗК).

Метод зменшення надлишковості технологічних сигналів на основі СЗК базується на основі теорії діофантових рівнянь і залишків:

$$x_i = a_i \cdot p + b_i,$$

де a_i – ранг;

p – модуль;

b_i – найменший невід'ємний залишок.

Діофантове рівняння:

$$x_i \equiv b_i (\text{mod}\ p),$$

або операція прямого кодування по залишках:

$$b_i = res\ x_i (\text{mod}\ \ p),$$

де *res* – символ операції отримання залишку.

Зворотня операція:

$$x_i = \overset{\vee}{E}\left[\frac{x_{i-1} - b_i}{p} + 0{,}5\right] \cdot p + b_i, \qquad (3.1)$$

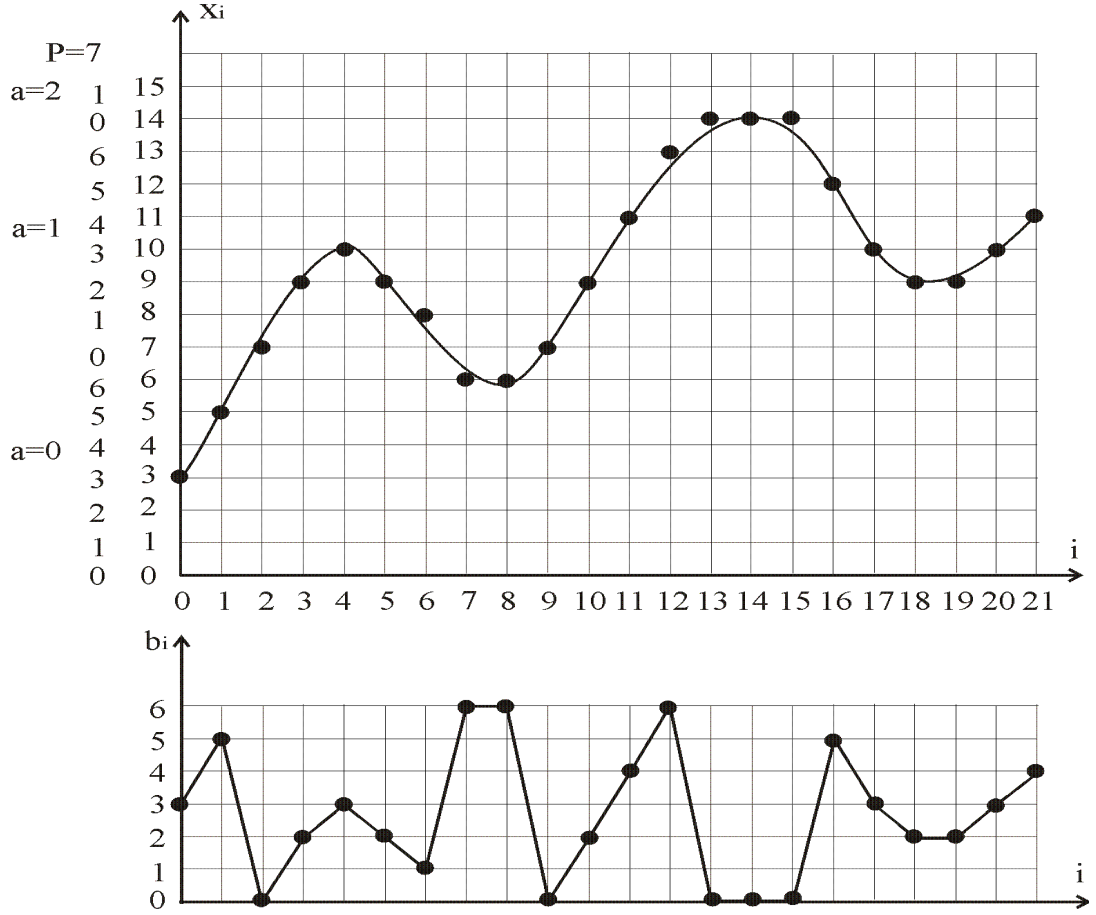
де $\overset{\vee}{E}[$ – цілочисельна функція з округленням до меншого цілого.

Умова однозначності кодування методом залишків виконується, якщо

$$\Delta x_{\max} \le \frac{p-1}{2}, \qquad (3.2)$$

для $p = 5$, $\Delta x_{i\max} \le 2$, $p = 7$, $\Delta x_{i\max} \le 3$.

Інформація на рисунку 3.3 закодована у вигляді залишків. При виконанні умови (3.2) процес x_i можна однозначно представити послідовністю залишків b_i.



1.4 Рисунок 3.3 – Кодування аналогового сигналу методом залишків.

1.5

Ефект стиснення даних досягається за рахунок представлення інформаційних відліків відповідними залишками меншої розрядності.

Розрядність коду x_i визначається за формулою Хартлі:

$$n = \hat{E}\left[\log_2 A\right],$$

де n – розрядність двійкового коду для представлення величини x_i;

$\hat{E}[$ – цілочисельна функція з округленням до більшого цілого;

А – діапазон квантування сигналу $0 \le x_i \le A$.

Розрядність коду залишків b_i визначаємо за формулою:

$$n_z = \hat{E}\left[\log_2 p\right].$$

Отже, формула для коефіцієнту стиснення буде мати наступний вигляд:

$$k = \frac{n}{\hat{E}\left[\log_2 p\right]}.$$

1.6 При кодуванні аналогового сигналу, об’єм масиву становить:

1.7

$$V = n \cdot m = 4 \cdot 22 = 88 \text{ біт.}$$

Після кодування методом залишків об’єм масиву становить:

$$V = n_z \cdot m = 3 \cdot 22 = 66 \text{ біт.}$$

Декодування даних відбувається за формулою, при $x_0 = 3$ визначаємо x_1:

$$x_1 = \check{E}\left[\frac{x_0 - b_1}{p} + 0.5\right] \cdot p + b_1 = \check{E}\left[\frac{3-5}{7} + 0.5\right] \cdot 7 + 5 = 5;$$

$$x_2 = \check{E}\left[\frac{5-0}{7} + 0.5\right] \cdot 7 + 0 = 7;\ x_3 = \check{E}\left[\frac{7-2}{7} + 0.5\right] \cdot 7 + 2 = 9;$$

$$x_4 = \check{E}\left[\frac{9-3}{7} + 0.5\right] \cdot 7 + 3 = 10;$$ і. т.д.

З приведених розрахунків значень сигналу $x_1 \div x_4$ по відповідних залишках видно, що отримані значення відповідають значенням інформаційних відліків до виконання операції кодування.

Отже, для представленого на рисунку 3.3 сигналу при використанні методу кодування та зменшення надлишковості даних на основі залишків коефіцієнт стиснення дорівнює $k = 1{,}33$ рази.

4 ОХОРОНА ПРАЦІ

Основною метою розділу охорони праці є уникнення можливості виробничого травматизму, професійних отруєнь і захворювань, пожеж і вибухів, аварій, забруднення довкілля при будівництві та використанні об'єкта проектування [18].

Даний дипломний проект передбачає розробку програмної системи пошуку залишків чисел великої розрядності.

В розділі охорона праці проводиться розрахунок безпечних умов праці для приміщення з комп'ютерами.

Обслуговування апаратури виконується в кімнаті розміщення обладнання контролера базових станцій. Обслуговуючий персонал займається контролюванням роботи апаратури, виявленням аварій та їх усуненням.

Контроль за роботою здійснюється за допомогою комп'ютерного обладнання, тому ця робота відноситься до категорії легких, які виконуються в сидячому, стоячому положенні або пов'язані з незначним рухом, але вона не відноситься до систематичної фізнчної роботи або до перенесення важких предметів.

Виходячи зі СН 245–71 і ГОСТ 12.1.005–88, а також, беручи до уваги характер робіт, відповідно до яких, площа приміщення на одного працівника в прbиміщенні дорівнює (6 м^2), приймаємо:

$$S_n = n\, S_0, \qquad (4.1)$$

де S_0 – площа приміщення, що виводиться на одного працівника;

n – кількість працівників.

Оскільки в приміщенні працює 6 чоловік, тоді необхідна площа для роботи повинна становити: $S_n = 6 \cdot 6 = 36$ м^2.

Реальна площа приміщення становить 48 м^2, тобто відповідає вимогам санітарних норм.

Згідно ГОСТ 12.1.005-88, в приміщенні повинні підтримуватися певні метеорологічні умови, що визначаються температурою відносною вологістю повітря, тиском та швидкістю руху повітря. Ці фактори впливають на терморегуляцію, тобто спроможністю організму людини підтримувати нормальну температуру тіла (в межах 36 – 37 °С).

Тепловіддача від організму може здійснюватись шляхом тепловипромінення, конвекції і випаровування. При підвищеній температурі навколишнього середовища тепловіддача здійснюється лише за рахунок випаровування поту. Перегрівання тіла до 40 – 41 °С приводить до порушення водно – сольового обміну, виникнення судорожної хвороби і теплового удару з втратою свідомості.

Робота в умовах пониженої температури повітря, особливо при підвищеній вологості і швидкості руху, призводить до переохолодження тіла, що супроводжується виникненням простудних захворювань. Мінусова температура повітря призводить до обморожування, що розглядається як виробнича травма.

Для робочої зони нашого приміщення оптимальні і допустимі значення температури, відносної вологості і швидкості руху повітря встановлюються з врахуванням трудоємності і складності роботи, яка виконується. Користувачі персональних комп'ютерів належать до групи 1а – легкі роботи.

Відповідно з цим і ГОСТ 12.1.005–88 вибираємо необхідні метеорологічні умови (таблиця 4.1).

Для підтримання відповідних метеорологічних умов в приміщенні встановлено обладнання системи центрального опалення, але в зимовий період його тепловіддача є недостатньою. Доцільним є проведення ущільнення конструктивів вікон і дверей, щоб припинити втрати тепла.

Решту метеорологічних умов забезпечує обладнання повного кондиціонування повітря. Воно забезпечуе постійність температури, вологості, руху і чистоти повітря.

Таблиця 4.1 – Оптимальні та допустимі метеоумови

Період	Кате-	Температура, °С	Відносна	Швидкість

року	горія робіт			вологість повітря		повітря	
		опти-мальна	допус-тима	опти-мальна	допус-тима	опти-мальна	допу-стима
Холодний	Легка 1а	22–24	21–25	40–60	35–75	0,1	0,05–0,2
Теплий	Легка 1а	23–25	22–28	40–60	35–75	0,1	0,05–0,2

Сприятливі умови роботи забезпечують як високу продуктивність праці, так і позитивно впливають на психологічний стан людини, на її працездатність і здоров'я. Особливо важливе біологічне і гігієнічне значення для людини має природне освітлення, тому при проектуванні виробничих приміщень важливо передбачити наявність природного освітлення СНиП II–4–79.

Проведемо розрахунок природного освітлення згідно зі СНиП II–4–79 «Природне і штучне освітлення. Норми проектування», а при необхідності розрахуємо додаткове штучне освітлення приміщення.

Розрізняють три системи природного освітлення: бокове, верхнє, комбіноване. Для кількісної оцінки виробничого освітлення важливою технічною характеристикою є освітленість робочої поверхні. Густина світлової енергії на площі E (лк) визначається за формулою:

$$E = dF/dS, \qquad (4.2)$$

де dF – світловий потік, який характеризує потужність світлового випромінювача (лм), рівномірно розподілений по площі dS ($м^2$).

Коефіцієнт природного освітлення, який являє собою відношення освітленості в даній точці середини приміщення $E_в$ до зовнішнього горизонтального освітлення $E_з$ визначаємо за формулою:

$$I = E_в/E_з. \qquad (4.3)$$

Заміри натурного освітлення проводяться люксметром 10116.

Розміри приміщення становлять:

$En \cdot B = 6 \times 8$ м^2; висота приміщення $h = 3$ м, S – світловий отвір вікон 1 – 1,9 м^2. Віконне скло подвійне. Характеристика зорової роботи відноситься до високої точності. Це відповідає нормі природного освітлення КПО $I_н$= 2 % при боковому освітленні.

При боковому освітленні використовується формула:

$$100\frac{S_0}{S_n} = \frac{I_н \cdot K_з \cdot \eta_{10}}{\tau_0 \cdot VI} K_б; \qquad (4.4)$$

де S_0 – площа світлових отворів, м^2;

S_n – площа підлоги, м ;

$K_з$ – коефіцієнт світлопроникнення;

η_{10} – світлова характеристика вікон;

τ_0 – загальний коефіцієнт світлопроникності;

VI – коефіцієнт, який враховує відбивання світла від поверхні;

$K_б$ – коефіцієнт, який враховує затемнення будинками, що стоять навпроти.

Для приміщення розмірами 6 · 8 · 3 площа $S = 48$ м^2; для Ln/B = 8/6 =1,33; B/H = 6/3 = 2; η_{10}= 16.

Для середньозваженого коефіцієнта відображення стелі, стін і підлоги, який дорівнює 0,4, коефіцієнт VI становить 2,4, $K_б$ приймаємо – 1,4.

Для приміщень з повітряним середовищем, в якому концентрація пилу менше 1 мг/м^3 $K_з$ =1,4; оскільки $I_н$= 2 % , коефіцієнт τ_0 визначаємо за формулою:

$$\tau_{0=} \tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4 \cdot \tau_5; \qquad (4.5)$$

де τ_1, τ_2, τ_3 – коефіцієнти світлопропускання матеріалу вікна, виду вікна і його конструкції: для віконного, листового, подвійного скла τ_1=0,8; для дерев'яних подвійних роздільних оправ до вікон τ_2=0,6; для залізобетонних конструкцій τ_3=0,8;

τ_4 – коефіцієнт, який враховує витрати світла в сонцезахисних конструкціях: для жалюзі і штор, що регулюються, дорівнює 1;

τ_5 – коефіцієнт, який враховує втрати світла в захисній сітці, що встановлюється під світильником — дорівнює 0,9.

Отже: $\tau_0 = 0{,}8 \cdot 0{,}6 \cdot 0{,}8 \cdot 1 \cdot 0{,}9 = 0{,}35$.

Визначаємо площу світлових отворів S_0:

$$S_0 = \frac{I_н \cdot K_з \cdot \eta_{10} \cdot S_п}{100 \cdot \tau_0} = K_б; \quad (4.6)$$

де $S_п$ — стандартна площа вікна.

Кількість вікон визначаємо за формулою:

$$S_0 = \frac{2 \cdot 1{,}4 \cdot 16 \cdot 1{,}4 \cdot 48}{100 \cdot 0{,}35 \cdot 24} = 3{,}47(м^2) \quad (4.7)$$

Відповідно: n =3,47/1,9 = 1,83 = 2 вікна. Таким чином, для забезпечення КПО $I_н$= 2 % у приміщенні повинно бути два вікна площею 1 ,9 м^2.

Для освітлення приміщення, коли природного освітлення недостатньо або взагалі немає, використовуеться штучне освітлення.

Світловий потік Φ – це потужність світлової енергії, що оцінюється за світловим відчуттям, яке воно справляє на органи зору людини:

$$\Phi = dQ/dt. \quad (4.8)$$

Сила світла I – це відношення світлового потоку до величини тілесного кута, в якому рівномірно розподілено випромінювання:

$$I = dF/d\omega. \quad (4.9)$$

Освітленість E – густина світлового потоку на освітлюваній поверхні:

$$E = d\Phi/dS. \quad (4.10)$$

Яскравість L – поверхнева густина сили світла у заданому напрямку:

$$L = dl/dS \cdot cos(\alpha). \quad (4.11)$$

Коефіцієнт відбиття β – відношення відбитого світлового потоку до падаючого: $\beta = Ф_{відб}/Ф_{пад}$.

Фон – поверхня, що прилягає безпосередньо до об'єкта розпізнавання, на який цей об'єкт сприймається. Фон характеризує коефіцієнт відбиття (залежить від кольору поверхні та від її фактури). Фон світлий $Ф$>0,4; середній – $Ф$ = =0,2 – 0,4; темний $Ф$<0,2.

Контраст – ступінь розпізнавання яскравості об'єкта і фону:

$$K=(L_0-L_ф) / L_0. \quad (4.12)$$

Контраст великий, то K>0,5; середній – K = 0,2 – 0,5; маленький – K<0,2.

Коефіцієнт пульсацій $K_п$ – критерій оцінки відносної глибини коливань освтленості в результаті зміни в часі світлового потоку газорозрядних ламп при живленні їх змінним струмом:

$$K_п = (E_{макс} - E_{мін}) \cdot 100\% / (2 \cdot E_{сер}) \quad (4.13)$$

де $E_{сер}$ – значення освітленості за період.

Розміри приміщення: A = 8 м, B = 6 м, H = 3 м. Нормована освітленість 300 лк. Показник приміщення: $i = A \cdot B/(H \cdot (A+B)) = 8 \cdot 6/(3 \cdot (8+6)) = 1,14$.

Вибираємо світильник НОДЛ з коефіцієнтом використання світлового потоку η = 49%. Сумарний світловий потік:

$$Ф = ((E_н \cdot S \cdot k \cdot Z)/ \eta) \cdot 100\%, \quad (4.14)$$

де $E_н$ – нормована освітленість, лк;

S – площа приміщення, м^2;

k – коефіцієнт запасу;

Z – коефіцієнт мінімальної освітленості;

η – коефіцієнт використання світлового потоку.

Φ = ((300·48· 1,75·1,1)/49)·100% = 56 572 лм.

Вибираємо лампи ЛТБ-80 р ,$\Phi_л$— 4300 лм, тоді кількість ламп дорівнює: $N=\Phi/\Phi_л$ = 56572/4300 = 14 шт. Кількість світильників: $N_c=N/2$=7 шт.

Перерахуємо значення E:

$$E=\frac{N\cdot\Phi_Л\cdot\eta}{S\cdot k\cdot Z\cdot 100\%}=\frac{14\cdot 4300\cdot 49}{48\cdot 1{,}75\cdot 1{,}1\cdot 100\%}=319{,}3. \qquad (4.15)$$

Отже, штучне освітлення забезпечує освітленість E = 319 лк, що є більшим за $E_н/E_н$= 300 лк, тобто розрахунок проведений правильно.

Рівень шуму дорівнює 75 дБ, що відповідає вимогам ГОСТу, тому захисних заходів не передбачається.

Електричний струм при дії на людину може викликати як місцеві, так і загальні пошкодження. Місцеві електротравми – це опіки, нагрівання внутрішніх органів, механічні пошкодження (розрив тканин м'язів), порушення біоелектричних процесів у організмі, електроліз органічних рідин. Зовнішніми проявами електротравм можуть бути термічні опіки, електричні ознаки на шкірі, металізація поверхні шкіри, електроофтальмія (ураження зору під дією ультрафіолетових променів при іскровому розряді). Загальне ураження струмом відбувається при проходженні струму через нервові центри, центри дихання і роботи серця (електричний удар).

Небезпека ураження тим більша, чим більший струм проходить через людину, але крім цього, впливають: тривалість і шлях проходження струму, його вид, частота і виробничі умови.

Умови ураження людини електричним струмом такі:

– двофазне дотикання (двофазне включения людини в мережу);

– однофазне дотикання, наближення на небезпечну віддаль до неізольованих дротів з напругою більше 1000 В;

– дотик до корпуса обладнання, що не проводить струм, але опинилося під напругою;

– перебування в зоні дії атмосферної електрнки;

– вхід у зону дії електромагнітного поля.

Згідно класифікації приміщень за ступенем небезпеки ураження електричним струмом (ПУЕ 1.1.6) приміщення роботи системи відноситься до першого (без підвищеної небезпеки).

Електричні установки, до яких відноситься переважна більшість обладнання системи, вимагають дотримання правил електробезпеки, оскільки в процесі експлуатації або проведения профілактичних робіт людина може доторкнутись до частин, що знаходяться під напругою 220 В, тому виникає необхідність у захисті персоналу від ураження електричним струмом. Дуже велике значения для запобігання електротравматизму має правильна організація експлуатації, обслуговування системи. Під цим розуміється точне виконання ряду організаційних та технічних заходів, які встановлені діючими «Правилами технічної експлуатації електроустановок споживачів і правил техніки безпеки при експлуатації електроустановок споживачів» (ППЕ і ПТБ споживачів) і «Правилами побудови електропристроїв» (ППЕ). Основними технічними засобами. які забезпечують безпеку робіт в електроустановках, є: захисне заземления, занулення, вирівнювання потенціалів, захисне включення, електричний розподіл мереж, мала напрута, подвійна ізоляція. Використання цих засобів у різноманітних поєднаннях дозволяє захистити людину від ураження струмом.

Захисне заземлення – це навмисне електричне з'єднання з землею або її еквівалентом металевих неструмопровідних частин, які можуть бути під напругою. У приміщенні розміщення контролера базових станцій заземлено всі шафи з обладнанням, а також вся комп’ютерна техніка. Приміщення, де знаходиться система, обладнується контуром-шиною захисного заземлення, яка з'єднується із заземлювачем. Контур-шина виготовляється з мідного дроту

діаметром 6 мм у перерізі і вкладається по периметру приміщення. Місця перетину дротів пропаюються з застосуванням бікислотного флюсу. Для під'єнання заземлювальних провівників на шину наварюються гвинти М8. У дипломній роботі проведу розрахунок захисного заземления згідно порядку, встановленого ПУЕ.

Згідно вимог ПУЕ 1.7.65 в електроустановках з напругами до 1 кВ при потужності трансформатора менше 100 кВт опір заземлювача повинен бути не більше 10 Ом.

1. Визначаемо розрахунковий опір землі: $ro_{р.з.} = Ф\ ro_з$, де $Ф$ – коефіцієнт сезонності, який враховує коливання питомого опору при зміні вологості грунту протягом року; використовується стержневий заземлювач (рисунок 4.1) довжиною l = 2 м при глибині закладання від вершини h = 0,5 м, $Ф$ = 1,1 для четвертої кліматичної зони. Питомий опір грунту: $ro_З$ = 300 Ом·м - для піску; $ro_{р.з.}$ = 1,1 ·300 = 330 Ом·м.

2. Визначаємо опір R, розтікання струму в землі від одного вертикального заземлювача:

$$R_B = \frac{ro_{п.з.}}{2 \cdot 3{,}14 \cdot l}\left(ln\frac{2 \cdot 1}{d} + \frac{1}{2} ln\frac{4 \cdot t + l}{4 \cdot t - l}\right), \qquad (4.16)$$

де l – довжина заземлювача (l = 2 м);

d=0,05 м – діаметр заземлювача за таблицею при U<1 кВ та при S<100 кВА;

t –відстань від поверхні землі до середини заземлювача,

$t=h+l/2=0{,}5+2/2=$ 1,5 м; $R_B = \frac{330}{2 \cdot 3{,}14 \cdot 2}\left(ln\frac{2 \cdot 2}{0{,}05} + \frac{1}{2} ln\frac{4 \cdot 1{,}5 + 2}{4 \cdot 1{,}5 - 2}\right) = 133{,}3\ Ом.$

3. Приблизна кількість заземлювачів: $n = \frac{R_в}{R_{в.нтмМ}} = \frac{133{,}3}{10} = 13{,}3 \approx 14.$

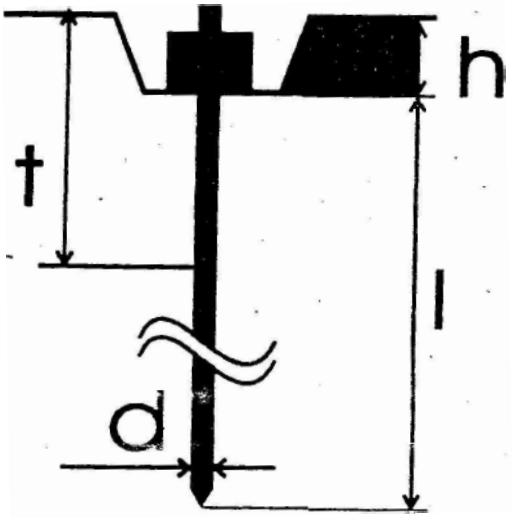


Рисунок 4.1 – Схема розташування одиничного заземлювача в грунті

4. Знаходимо із таблиць коефіцієнт використання вертикальних заземлювачів, який враховує ефект екранування при вибраному значенні $k=a/l$, де a — віддаль між заземлювачами, м; k = 1,2 при а = 2,4 м; отже коефіцієнт використання вертикального заземлювача за таблицями дорівнює η_e= 0,56.
5. Кількість вертикальних заземлювачів з урахуванням η_e обчислюємо за формулою $n=\frac{R_в}{R_{в.норм}\cdot\eta_в}=\frac{133,277}{10\cdot0,56}=23,799=24.$

6. Довжина горизонтального заземлювача для розміщення по контуру $L=a\cdot n$ = 2,4 м · 24 = 57,6 м.

7. Опір горизонтального заземлювача $R_г$ (Ом), прокладеного на глибині h = 0,5 м від поверхні землі:

$$R_г=\frac{ro_{р.к.}}{2\cdot3,14\cdot L}ln\frac{2\cdot L}{b\cdot h}=\frac{330}{2\cdot3,14\cdot57,6}ln\frac{2\cdot57,6}{0,04\cdot0,5}=7,3 \qquad (4.17)$$

де b = 0,04 м — ширина штабової сталі, з якої виготовлений заземлювач.

8. Обчислюємо загальний опір:

$$R_к=\frac{R_в\cdot R_о}{n\cdot R_д\eta_в+R_в\eta_д}=\frac{133,3\cdot7,3}{24\cdot7,3\cdot0,56+133,3\cdot0,27}=7,5 \text{ Ом}. \qquad (4.18)$$

Результат є менше 10 Ом, тобто виконується нормуюча умова $R_{з.}<R_{з.норм.}$.

Велика увага приділяється дотриманню обслуговуючим персоналом правил роботи в приміщенні, яке призначене для експлуатації системи. У приміщенні не повинно бути сторонніх людей. Працівники повинні використовувати спецодяг. Безпека роботи обслуговуючого персоналу в приміщенні забезпечується:

– наявністю нормальних проходів між обладнанням;

– використанням спеціальних технічних меблів;

– використанням електрозахисних засобів (діелектричних килимків, гумових рукавиць);

– наявністю аварійного освітлення (E=2 лк);

– обладнанням розеток з напругою 220 В;

– заземленням корпусів обладнання і апаратури освітлювальних пристроїв.

Одне з основних місць в охороні праці займає пожежна безпека.

Першочергове завдання пожежної профілактики — це запобігання пожеж. Під пожежною профілактикою розуміють комплекс організаційних і технічних заходів, спрямованих на забезпечення безпеки людей, на запобігання пожеж, обмеження їх розповсюдження, а також на створення умов для успішного гасіння пожеж. Пожежно-профілактичні заходи розробляються та виконуються разом, в тісному взаємозв'язку з усіма проектними, будівельними та експлуатаційними роботами.

Приміщення чергування технічного персоналу забезпечуеться протипожежним івентарем (вуглекислотними вогнегасниками типу ВВ–2). Проходи між рядами і вихід не повинні загромаджуватись. У випадку виникнення пожежі перш за все потрібно виключити джерело живлення, сповістити про пожежу в пожежну частину. Евакуювати сторонніх людей, які могли опинитися в небезпечшй зоні і лише після цього приступити до гасіння пожежі і рятування цінного обладнання.

Один вуглекислотний вогнегасник ВВ–2 розрахований на 40–50 $м^2$ примі-щення. Для ліквідації невеликих пожеж можна використовувати деякі

порошкові матеріали (хлориди лужних металів, соду, пісок і т. д.), що подаються в зону горіння порошковими вогнегасниками.

Будівля, в якій знаходиться наше приміщення, обов'язково має резервний вихід на випадок екстреної евакуації працівників і неможливості використання основного виходу.

За вибухопожежною і пожежною небезпекою приміщення і будівлі згідно ОНТП–24–86 і СНТП 2.09, СНТП 02–85 діляться на категорії А, Б, В, Г, Д.

Для приміщення чергування персоналу встановлена категорія пожежної небезпеки Д (СНІП 2.09.02–85) при ступені вогнестійкості (СНІП Н–90–81), що означає наявність у приміщенні негорючих речовин та матеріалів у холодному стані.

Для швидкого сповіщення пожежної схорони при виникненні пожежі приміщенні використовується електрична пожежна сигналізащя. Система електричної пожежної сигналізації виявляє пожежу на початковій стадії і сповіщае про місце її виникнення, а також автоматично включає стаціонарні установки гасіння пожеж.

Автоматичні сповіщувачі при ознаках пожежі здійснюють посилання сигналу. Сповіщувачі типу АТИП–1, АТИП–3 і АТИП–3М спрацьовують внаслідок теплової деформації (при 80–100 °С) біметалічних пластинок і мають розраховану площу обслуговування в приміщеннях до 15 м^2. Комбіновані теплові і димові сповіщувачі типу КИ–1 мають чутливий елемент у вигляді іонізуючої камери (реагування на дим) і терморезистори (реагування на тепло). Температура спрацювання цих сповіщувачів 50–80°С, площа обслуговування 100 м^2.

Передбачені нами заходи з охорони праці в першу чергу призначені для уникнення нещасних випадків, що можуть виникнути на підприємстві.

В іншому передбачені заходи з охорони праці відповідають вимогам нормативних документів та актів та забезпечують нормальну, ефективну і безпечну для здоров'я людини виробничу діяльність.

ВИСНОВКИ

В дипломному проекті розглянуто важливий клас фундаментальної задачі теорії чисел, а саме, знаходження залишків чисел великої розрядності (ЧВР), яка широко використовується в прикладних задачах захисту інформації, обробці та кодуванні даних та ін.

В зв’язку з цим актуальною задачею, яка розглядається в дипломному проекті, є розробка теоретичних основ пошуку залишків ЧВР з використанням теоретико-числових базисів Радемахера та Крестенсона, застосування яких дозволяє зменшити часову складність.

Проведено аналіз відомих методів знаходження залишків, які ґрунтується на використанні десяткової системи числення, згідно якого для знаходження залишків ЧВР необхідно виконати ділення, виділити цілу частину від ділення, знайти добуток цілої частини на модуль та різницю ЧВР та знайденого добутку. Також можна декілька разів від ЧВР відняти модуль, поки різниця не стане меншою від’ємника. Дані алгоритми легко реалізовуються програмним шляхом, але їх часова складність залишається великою, оскільки операція ділення є досить трудомісткою.

Встановлено основні недоліки алгоритмів знаходження залишків ЧВР в десятковій системі числення та запропоновано, реалізовано метод на основі розмежованої системи числення залишкових класів, який характеризується меншою обчислювальною складністю в порівнянні з відомими, також можливість розпаралелення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Анісімов А.В. Алгоритмічна теорія великих чисел. Модулярна арифметика великих чисел.-К.: Видавничий дім «Академперіодика», 2001.-152с.
2. Боюн В. П. Теоретико-информационные основы преобразования и обработки информации в системах реального времени // Проектирование и применение средств микропроцессорной техники: Сб. научн. Трудов.– Киев: ИК им. В.М.Глушкова АН УССР, 1986. – С. 9–17.
3. Бухштаб А.А. Теория чисел. - М.: Просвещение, 1966. – 384 с.
4. Вербицький О.В. Вступ до криптології.- Львів: Вид-во наук.тех. л-ри, 1998.-248с.
5. Дадаев Ю.Г. Теория арифметических кодов.-М.:Радио и связь, 1981.-270с.
6. Данильченко Л.С. О некоторых эффективных алгоритмах вычисления остатка и возведения в степень многорозрядных чисел//Кибернетика и системный анализ, №3, 1996.-С.145-151.
7. Задірака В.К. Комп’ютерна арифметика багаторозрядних чисел: Наукове видання.// Задірака В.К., Олексюк О.С. /-Київ, 2003.-264с.
8. Задірака В.К. Методи захисту фінансової інформації: // Задірака В.К. / Навчальний посібник. -Тернопіль:Збруч, 2000.-460с.
9. Задірака В.К. Компютерна криптологія: // Задірака В.К., Олексюк О.С. / Підручник. -Київ:2002.-504с.
10. Залманзон Л. А. Преобразования Фурье, Уолша, Хаара и их применение в управлении связи и других областях // Залманзон Л. А./ – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 496 с.
11. Ибрагимов В.А., Крикун З.Н., Николайчук Я.Н. Кодирование информации методом вычетов. - // Автоматизация и телемеханизация нефтяной промышленности.–1974.– № 1.– С. 26-28.
12. Николайчук Я.М.Теоретичні основи побудови та структура спецпроцесорів в базисі Крестенсона.// Николайчук Я.М., Волинський О.І.,

Кулина С.В./ Вісник Хмельницького національного університету.- Хмельницький.- 2007.- №3.- Т1.- С.85-90.

13. Николайчук Я.М. Выбор оптимальных параметров кодирования методом вычетов.// Николайчук Я.М., Зевелев С.Я., Крикун З.Н./ Респ. сб. «Автоматизация и телемеханизация нефтяной промышленности». – 1975. – №2. – С.22-24.
14. Николайчук Я.М. Напрямки розвитку процесорів комп'ютерних систем на основі дискретних теоретико-числових базисів. Поступ в науку. Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2008. - №4. Т1. – С.8-11.
15. Романец Ю.В. Защита информации в компьютерных системах и сетях.// Романец Ю.В., Тимофеев П.А., Шангин В.Ф./-М.:Радио и связь, 1999.-328с.
16. Саломаа А. Криптография с открытым ключом/Пер.с англ.-М.:Мир, 1996.-318с.
17. Торгашев В.А. Система остаточных классов и надежность ЦВМ.-М.:Советское радио, 1970.-118с.
18. Методичні вказівки до написання розділу «Охорона праці» в дипломних проектах з освітньо–кваліфікаційного рівня «Спеціаліст» для спеціальності 7.091501 «Комп'ютерні системи та мережі» / Г.В.Сапожник, Н.М.Васильків. – Тернопіль: ТАНГ, 2004. – 24 с.
19. Методичні рекомендації до виконання дипломного проекту з освітньо–кваліфікаційного рівня „Спеціаліст". Спеціальність „Комп'ютерні системи та мережі" / О.М.Березький, Н.М.Васильків, І.В.Васильцов, Р.Б.Трембач / Під ред. М.П.Карпінського – Тернопіль : ТНЕУ, 2008. – 38 с.

Додаток А

Лістинг головної форми

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "test_.h"
#include "timetest.h"
#include "Chart.h"
#include "bitarray.h"
#include "array_metod_thread.h"

#include "test_standatr_mod.h"
#include "test_new_mod.h"
//#include <vector.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;
array_metod *m;
vector_metod_thread *v;
vector_cikle *vc;
array_cikle_threat *ac;
test_standatr_mod *tsm;
test_new_mod *tnm;
timetest *tt;
mersen_part_finder_threat *mpft;
//---------------------------------------------------------------------------
```

```
__fastcall TForm3::TForm3(TComponent* Owner)
 : TForm(Owner)
{
}
void __fastcall TForm3::Button13Click(TObject *Sender)
{
Edit8->Text=IntToStr(StrToInt(Edit6->Text)%StrToInt(Edit7->Text));
}
//---------------------------------------------------------------------------

void __fastcall TForm3::Button14Click(TObject *Sender)
{
bitarray number(long(StrToInt(Form3->Edit6->Text)));
bitarray modul(long(StrToInt(Form3->Edit7->Text)));
Form3->Edit8->Text=IntToStr(bitarray::mod(number,modul));
//bitarray res(1,number.array_size);

//Memo3->Lines->Add(res.ToString());

}
//---------------------------------------------------------------------------

void __fastcall TForm3::Button15Click(TObject *Sender)
{
tsm= new test_standatr_mod(false);
tnm= new test_new_mod (false);
//if (Form3->CheckBox2->Checked) Form4->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::Button16Click(TObject *Sender)
```

```
{
tsm->Suspend();
tnm->Suspend();
}
//---------------------------------------------------------------------------
```

Додаток Б

Реалізація потоку виконання операції модуля згідно запропонованого алгоритму

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "test_new_mod.h"
#include "bitarray.h"
#include "test_.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

// Important: Methods and properties of objects in VCL can only be
// used in a method called using Synchronize, for example:
//
// Synchronize(UpdateCaption);
//
// where UpdateCaption could look like:
//
// void __fastcall test_new_mod::UpdateCaption()
// {
// Form1->Caption = "Updated in a thread";
// }
//---------------------------------------------------------------------------

__fastcall test_new_mod::test_new_mod(bool CreateSuspended)
 : TThread(CreateSuspended)
{
}
```

```
//---------------------------------------------------------------------------
void __fastcall test_new_mod::Execute()
{

 long c;
 Form3->ProgressBar3->Max=StrToInt(Form3->Edit9->Text);
 bitarray modul(long(StrToInt(Form3->Edit7->Text)));
 bitarray number(0);
 for(int i=0;i<=StrToInt(Form3->Edit9->Text);i++){
c=bitarray::modpresent(number,modul,Form3->Memo3);
 number.random_(i);
 Form3->ProgressBar3->Position=i;



// c=i%StrToInt(Form3->Edit7->Text);
 };

 //---- Place thread code here ----
}
//---------------------------------------------------------------------------
```

Додаток В

Реалізація потоку виконання операції модуля згідно відомого алгоритму

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "test_standatr_mod.h"
#include "test_.h"
#include "chart.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

// Important: Methods and properties of objects in VCL can only be
// used in a method called using Synchronize, for example:
//
// Synchronize(UpdateCaption);
//
// where UpdateCaption could look like:
//
// void __fastcall test_standatr_mod::UpdateCaption()
// {
// Form1->Caption = "Updated in a thread";
// }
//---------------------------------------------------------------------------

__fastcall test_standatr_mod::test_standatr_mod(bool CreateSuspended)
 : TThread(CreateSuspended)
{
}
```

```
//---------------------------------------------------------------------------
void __fastcall test_standatr_mod::Execute()
{
TDateTime DateTime = Time();
double d,e;
//Form4->series1 = new TLineSeries(Form4->Chart1);
Form3->ProgressBar5->Max=StrToInt(Form3->Edit9->Text);
 long c,a=0;
 for(int i=0;i<=StrToInt(Form3->Edit9->Text);i++){
 a++;
Form3->ProgressBar5->Position=i;
//DateTime = Time();
//d=DateTime.Val;
 c= i%StrToInt(Form3->Edit7->Text);

//DateTime = Time();
//d=DateTime.Val-d;
//if (d>0) Form4->series1->Add(d*100000000000,"",clBlack);

 }

// Form4->series1->Marks->Visible=true;
 // Form4->Chart1->AddSeries(Form4->series1);
 //---- Place thread code here ----
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

Додаток Г

Бібліотека опрацювання велико-розрядних чисел

```
#pragma hdrstop
#include "bitarray.h"
```

```
//#include "unit3.h"
//#include "stdlib.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
long bitarray::addbit(long bit){
 if (bit<array_size){
 int i;
 for (i=bit;i<number_size&&mainarray[i]==true;i++)mainarray[i]=false;
 if (i<number_size)mainarray[i]=true ;

 if (i==number_size){mainarray[i]=true;number_size++;};


 if (i>number_size){
 for (i=number_size;i<bit;i++){mainarray[i]=false;};
 mainarray[bit]=true;
 number_size=bit+1;
 };
 };
 return 1;
};
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
long bitarray::addbitarray(bitarray &number){
 for (int i=0;i<number.number_size;i++){
```

```
 if (number.mainarray[i])addbit(i);
 };
return 1;
};

bitarray::bitarray(){
randomize();

};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
long bitarray::to_long(){//перевірено
long sum=0;
 if (number_size<=32){
 for (int i=0;i<number_size;i++)if (mainarray[i]) sum=sum+pow(2,i);
 };
return sum;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int bitarray::nullarray(){
for (int i=0;i<array_size;i++)mainarray[i]=false;
return 1;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int bitarray::minusbit(long bitnumber){ //перевірено
 //if (bitnumber=>0&&bitnumber<=number_size+count_of_smaller_bit0){
 if
((bitnumber>=0&&bitnumber<number_size)&&count_of_smaller_bit0==0){
 int i,a=0;
```

```
        if (mainarray[bitnumber]==true){
        mainarray[bitnumber]=false;
        }else {
        for
(i=bitnumber;i<number_size&&mainarray[i]==false;i++)mainarray[i]=true;
        mainarray[i]=false;
        };

        for (i=0;i<number_size;i++)
        if (mainarray[i])a=i;
        number_size=a+1;
        };
        if (number_size==1&&mainarray[0]==true)is_one=true;
        //};
        return 1;
       };
       //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
       int bitarray::mul2(){ //перевірено
        if (bitarray::to_long()!=0){
        if (number_size<array_size){
        for (int i=number_size;i>=0;i--){
        if (mainarray[i]){
        mainarray[i]=false;
        mainarray[i+1]=true;
        };
        };
        number_size++;
        }; };
        //count_of_smaller_bit0++;
        return 1;
```

```
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bitarray::bitarray(long num){//перевірено
randomize();
 array_size=32;
 cikle=0;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
 nullarray();
 int i;
 if (num==1){is_one=true;}else{is_one=false;};
 for (i=0; num; num>>=1, i++){
 mainarray[i]=(bool((num&1)?(1):(0)));
 };
 number_size=i;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bitarray::bitarray(AnsiString s){
randomize();
// в процесі розробки !!!!!!!!!!!!
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bitarray::bitarray(long num,long reserve){
randomize();
 cikle=0;
 array_size=reserve;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
```

```
        nullarray();
        int i;
        if (num==1){is_one=true;}else{is_one=false;};
        for (i=0; num; num>>=1, i++){
        mainarray[i]=(bool((num&1)?(1):(0)));
        };
        number_size=i;
       };
       //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
       bitarray::bitarray(AnsiString s,long reserve){
       randomize();
       // в процесі розробки !!!!!!!!!!!!!
       };
       //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
       bitarray::bitarray(vector<bool>& temp,long size,long reserve){
       randomize();
        int i;
        cikle=0;
        count_of_smaller_bit0=0;
        array_size=reserve;
        mainarray=new bool [array_size];
        nullarray();
       // if(size<=temp.size()&&size<=reserve){
        for(i=0;i<size;i++) mainarray[i]=temp[i];
        //}
        // else {
        /// в процесі розробки !!!!!!!!!!!!!
        // };
        if (number_size==1&&mainarray[0]==true)is_one=true;
```

```
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
AnsiString bitarray::ToString(){
//перевірено
 AnsiString s;
 count_of_smaller_bit0=0;
 for(int i=number_size-1;i>=0;i--){
 if (mainarray[i]){s=s+"1";} else {s=s+"0";};
 };
return s;
};

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int bitarray::minus(bitarray &vidyemnyk){//перевірено
 if (isbigger(vidyemnyk)){

 for (int i=vidyemnyk.number_size-1;i>=0;i--){
 if (vidyemnyk.mainarray[i]==true)bitarray::minusbit(i);
 };
 };
return 1;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



bool bitarray::isbigger(bitarray &another){//перевірено
 bool is=false;
 if (another.number_size<number_size+count_of_smaller_bit0) is=true;
```

```
 if (another.number_size>number_size+count_of_smaller_bit0) is=false;
 if (another.number_size==number_size+count_of_smaller_bit0){
 int i;
 for (i=another.number_size-1;i>0&&mainarray[i]==another.mainarray[i];i--);
 if (mainarray[i]){is=true;} else {is=false;};
 };
 return is;

// return true;
};

long bitarray::mod(bitarray &number,bitarray &modul){

bitarray res(1,number.array_size);

for (int i=number.number_size-2;i>=0;i--){
res.mul2();
if (number.mainarray[i])res.mainarray[0]=true;
if (res.isbigger(modul))res.minus(modul);

};
return res.to_long();
return 0;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
long bitarray::modpresent(bitarray &number,bitarray &modul,TMemo *memo){
memo->Lines->Add(number.ToString()+" mod "+modul.ToString()+" - ?");
bitarray res(1,number.array_size);
memo->Lines->Add("1");
```

```
        for (int i=number.number_size-2;i>=0;i--){
        res.mul2();
        if (number.mainarray[i])res.mainarray[0]=true;
        if (res.isbigger(modul))res.minus(modul);
        memo->Lines->Add(res.ToString());
        };

        memo->Lines->Add("залишок = "+res.ToString());
        return res.to_long();

        };
        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        double bitarray::mod_time (bitarray &number,bitarray &modul){//повертає залишок в число і час за який йоло було знайдено
        TDateTime DateTime = Time();
        bitarray res(1,number.array_size);
        for (int i=number.number_size-2;i>=0;i--){
        res.mul2();
        if (number.mainarray[i])res.mainarray[0]=true;
        if (res.isbigger(modul))res.minus(modul);
        };
        //number.mainarray=res.mainarray;
        //number.array_size=res.array_size;
        //number.count_of_smaller_bit0=res.count_of_smaller_bit0;
        //number.number_size=res.number_size;
        //res.mainarray=0;
        return Time().Val-DateTime.Val;
        };
        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
void bitarray::random_(long size){
//delete[] mainarray;
array_size=size+1;
number_size=size;
mainarray=new bool [array_size];
cikle=0;

for (int i=1;i<number_size-1;i++){
//randomize();
if (random(2)==1){mainarray[i]=true;}else {mainarray[i]=false;};};
mainarray[number_size-1]=true;
mainarray[0]=true;
mainarray[size-1]=true;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
void bitarray::randomnumber(bitarray &number,long countof_bit){

 number.newsize(countof_bit+1);
 number.number_size=countof_bit;

 for (int i=0;i<number.number_size-1;i++){
 if
(random(2)==0){number.mainarray[i]=false;}else{number.mainarray[i]=true;};
 };
 number.mainarray[countof_bit-1]=true;
 number.mainarray[0]=true;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
void bitarray::newsize(long numbersize){
```

```
        delete[] mainarray;
         cikle=0;
         array_size=numbersize;
         count_of_smaller_bit0=0;
         mainarray=new bool [array_size];
         nullarray();
         int i;
         mainarray[0]=false;
         number_size=1;
        };
        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
        void bitarray::from_bitarray_to_verylong(bitarray &number,verylong *a){
         char *str;
         zsetlength(a,number.number_size ,str);
         for(long i=0;i<number.number_size;i++)
         {
         if (number.mainarray[i]==true){
         zsetbit(a, i);
         }else{
         //if (zbit(a,i))zsetbit(&a, i);
         };
         // zadd( b, c, &c);
        // z2mul(b,&b);
         // zcopy(d, &c);

         };
        };

        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
```

```
AnsiString bitarray::from_verylong_string_binary(verylong a){
 // z2mul(a,&a); z2mul(a,&a); // не працює правильно
 long number_weight=z2log(a);
 AnsiString s="";
 for (int i=number_weight-1;i>=0;i--)
 if (zbit(a, i)==1){s=s+'1';}
 else{s=s+'0';};
 //if (zbit(a, number_weight-1)==1)s=s+'1';
 return s;
/* char *str;
str= new char [32];
for (int i=0;i<32;i++)str[i]='0';
zswrite(str,a);
s=str;
 return s; */
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
AnsiString  bitarray::from_bitarray_to_verylong_(bitarray  &number,verylong
a){
AnsiString s;
//verylong c=0;
//verylong b=0;
//zsetbit(&b, 0);
 //zcopy();
 char *str;
 zsetlength(&a,number.number_size ,str);
 for(long i=0;i<number.number_size;i++)
 {
 if (number.mainarray[i]==true){
```

```
 zsetbit(&a, i);
 }else{
 // if (zbit(a,i))zsetbit(&a, i);
 };
 // zadd( b, c, &c);
// z2mul(b,&b);
 // zcopy(d, &c);

 };

 // zcopy(c, a);
 //a=&c;



//str= new char [32];
//for (int i=0;i<32;i++)str[i]='0';
//zswrite(str,a);
//s=str;
//return s;
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bool bitarray::have_that_cikle(verylong a,long cikle){
verylong res=0;
zsetbit(&res, 0);
long i;
bool cikle_smaller=false;
for (i=0;i<cikle&&!cikle_smaller;i++)
 {
 z2mul(res,&res);
 zmod(res,a,&res);
```

```
        if (i<cikle-1) cikle_smaller=(zscompare(res, 1)==0);
        };
       if (zscompare(res, 1)==0&&!cikle_smaller) {return true;} else {return false;};
       zfree(&res);
       };
       //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
        bool bitarray::save_to_file(AnsiString filename,verylong a){
        long number_weight=z2log(a);
        char s;
        FILE *filetosave;
        filetosave=fopen(filename.c_str(),"w");
        for (int i=0;i<=number_weight-1;i++){
        if (zbit(a, i)==1){s='1'; }
        else{s='0';};
        fputc((int)s,filetosave); };
        fclose(filetosave);
        return true;

        };

       //**********************************************************************
```

```
        bool bitarray::load_from_file(AnsiString filename, verylong *a){
        char *str;

        char s;
        long i=0;
        FILE *filetosave;
        filetosave=fopen(filename.c_str(),"r");
```

```
 while(!feof(filetosave)){
 s=char(fgetc(filetosave));

if (s=='1') {
 zsetlength(a,i+1,str);
 zsetbit(a, i);
};
 i++;
};
 fclose(filetosave);
};
```