МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

Пояснювальна записка

до дипломної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему «Дослідження та розробка математичних та комп'ютерних методів стиснення інформації»

Виконав: студент 2 курсу, групи ІСТ-20дм
спеціальності 126 „Інформаційні системи та технології”
_____ Безменов С.Ю.
(підпис)

Керівник,
доцент, д.т.н._____ Лифар В.О.
(підпис)

Рецензент,
доцент, к.т.н._____ Митрохін С.О.
(підпис)

СЄВЄРОДОНЕЦЬК
2021 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки
Кафедра програмування та математики
Освітньо-кваліфікаційний рівень магістр
Спеціальність 126 „Інформаційні системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ,
д.т.н., доцент
_____ Лифар В.О.
«___» _____ 2021 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Безменов С.Ю.

1. Тема роботи Дослідження та розробка математичних та комп'ютерних методів шифрування та стиснення інформації.
керівник роботи Лифар В.О.
затверджені наказом вищого навчального закладу від 30.11.21 №182/15.16
2. Строк подання студентом роботи 20 грудня 2021 р.
3. Вихідні дані до роботи

Об'єктом даної роботи є методи стиснення даних без втрат, а також вплив різних алгоритмів на ступінь стиснення.

3.1 Літературні джерела:

1. Алгоритм Лемпеля - Зіва [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_36.shtml

2. Алгоритм Хаффмана [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_35.shtml

3. Ахо Альфред В. Структури даних і алгоритми/Альфред В. Ахо, Джон Е. Хопкрофта, Джеффрі Д. Ульман.-М.: Видавничий дім "Вільямс", 2000.- 384 с.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ

4.2 Аналіз предметної галузі (огляд літератури), з висвітленням наступних питань:

Огляд методів стиснення даних.

Циклічний надлишковий код. Коефіцієнт компресії.

Основи алгоритмів стиснення даних.

4.3 Основна частина, в якої висвітлити:

Інформаційну модель стиснення з мінімальною надлишковістю.

Програмну реалізацію програмного комплексу з можливістю стиснення файлу і створення шифрованого архіву.

4.4 Висновки

4.4 Перелік використаних джерел

5. Перелік графічного матеріалу немає

6. Дата видачі завдання 10 вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Одержання завдання на виконання роботи	10.09.21	
2	Укладання і погодження з керівником плану і етапів виконання роботи	20.09.21	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	25.09.21	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	30.09.21	
3	Проектування інформаційної моделі задачі що реалізується.	15.10.21	
5	Укладання та тестування програмного продукту	30.10.21	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	15.11.21	
7	Здача готової пояснювальної записки на кафедру	15.12.21	
8	Укладання доповіді і презентації	20.12.21	

Студент _____
(підпис)

Керівник роботи _____
(підпис)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІСТ-20дм Безменова С.Ю.

Науковий керівник

Доцент, д.т.н. _____ Лифар В.О.

Оцінка наукового керівника: _____

Рецензент _____
ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:
_____

Голова ЕК

_____ Лифар В.О.
підпис

РЕФЕРАТ

Текст – 89, рис. – 19, табл. – 9, додатків – 2, літературних джерел – 56

У ході виконання даної дипломної роботи були проведені дослідження предметної області стиснення інформації без втрат та з втратами, й розглянуті алгоритми стиснення даних з мінімальною надлишковістю: кодування Шеннона-Фано, кодування Хаффмана та стисненням із застосуванням словника: кодування Лемпеля-Зіва.

Метою роботи є розробка теоретичних основ стиснення даних, дослідження різних способів стиснення інформації, виявлення найкращих способів архівації з шифруванням та зберігання різного роду даних. Результатом проведених досліджень є створення архіватору з кодовим захистом інформації.

В інтегрованім середовищі розробки Embarcadero RAD Studio XE8 розроблений програмний комплекс архіватору з кодовим захистом інформації. Механізм роботи архіватору заснований на створенні і обробці потокових даних. Ядром архіватору є функції стиснення і розпаковування файлів методом Лемпеля-Зіва. В якості методу і засобу захисту інформації в архіві була задіяна полі алфавітна підстановка (шифр Віжінера).

Результати роботи, зокрема, розроблене програмне забезпечення може бути практично використане при архівному збереження захищеної інформації; механізм архівування та шифрування даних може бути використаний в системах передачі інформації, з метою зменшення трафіку в мережі та забезпечення захищеності даних.

Ключові слова: СТИСНЕННЯ ДАНИХ, КОЕФІЦІЄНТ КОМПРЕСІЇ, ЦИКЛІЧНИЙ НАДЛИШКОВИЙ КОД, АРХІВАТОР.

ЗМІСТ

ВСТУП

Актуальність дослідження**.** Сучасний інформаційно-технологічний світ не можна уявити без архівування даних. Архівація - це стиснення, ущільнення, упаковка інформації з метою її більш раціонального розміщення на зовнішніх носіях [4]. Необхідність архівації пов'язана з резервним копіюванням інформації на диски та дискети з метою збереження програмного забезпечення комп'ютера і захисту його від псування і знищення (умисного, випадкового або під дією комп'ютерного вірусу). Щоб зменшити втрати інформації, слід мати резервні копії всіх програм і файлів.

Архіватори - це програми, які реалізують процес архівації та дозволяють створювати і розпаковувати архіви. Архів в основному використовують для об'єднання великої кількості файлів і полегшення їх розміщення і передачі в інтернеті, але це не єдина можливість архівації даних. Архівація також потрібна для збереження будь-яких важливих файлів. Архів не може бути заражений вірусом, завдяки своєму внутрішньому захисту від доступу, також він чудово зменшує розмір поміщених у нього файлів, що робить його зручним для резервного копіювання на інформаційні носії. Тому це хороший захисту для електронних даних.

В даний час розвитку програм, які архівують, приділено особливу увагу. Різного роду розробки стали з'являтися не лише у державних служб, але також і у приватних осіб. Саме сьогодні активно ведуться дослідження різних способів архівації, модернізуються старі методи, з'являються нові. Інтерес до даної сфери не закінчиться, поки йде розвиток інформаційної інфраструктури глобальних мереж Інтернет [14, 27]. Світ переходить до електронного документообігу і архіви знадобляться для кращого збереження та захисту цих даних.

Сьогодні доступні як носії інформації великого обсягу, так і високошвидкісні канали передачі даних. Проте одночасно з цим ростуть і обсяги інформації, що передається. Якщо кілька років тому ми дивилися 700-мегабайтні фільми, які вміщуються на один CD-диск, то сьогодні фільми в FullHD-якості можуть займати десятки гігабайт. Існують ситуації, в яких стиснення не тільки корисне, але і необхідне. Такими, наприклад, є такі випадки [13]:

- пересилання документів по електронній пошті (особливо великих обсягів документів з використанням мобільних пристроїв) [39, 41];
- потреба в економії трафіку при публікації документів на сайтах;
- економія дискового простору в тих випадках, коли заміна або додавання засобів зберігання важко.

Об'єкт дослідження – методи стиснення даних без втрат, а також вплив різних алгоритмів на ступінь стиснення.

Предмет дослідження – програмний архіватор на основі комбінованих методів стиснення даних з шифруванням інформації.

Мета і завдання дослідження. Метою даної роботи є дослідження різних способів стиснення інформації, розробка теоретичних основ комп'ютерних методів стиснення даних та створення програмного архіватору з кодовим захистом інформації.

Для досягнення поставленої мети вирішувалися наступні завдання:

a) аналіз вибору типу стиснення даних;
b) дослідження алгоритмів кодування з мінімальною надлишковістю;
c) порівняльні дослідження кодування Шеннона-Фано, Хаффмана і Лемпеля - Зіва;
d) програмний процес розміщення файлів в архів;
e) проблема оборотності стиснення інформації.

Таким чином, дана робота, головним змістом якої є дослідження різних алгоритмів стиснення даних і розробка на основі цих методів програмного архіватору зі спеціальним захистом інформації, на сьогоднішній день є актуальною.

РОЗДІЛ 1 МЕТОДИ СТИСНЕННЯ ДАНИХ

1.1. Загальна ідея стиснення даних

Проблема зберігання інформації з'явилася в той час, коли були створені перші ЕОМ, розміри яких поза всякої критики, а обсяги жорстких дисків були менші, ніж ПЗУ в перших мобільних телефонах [28]. Весь прогресивний світ задумався про те, як помістити в маленький об'єм пам'яті як можна більше корисних документів. Вчені стали пропонувати свої напрацювання, але більшість з цих теорем лише доводили можливість стиснення тих або інших даних. Ідей про стиснення і, тим більше, про подальше розтиснення було небагато [42]. Поступово народився ентропійний аналіз даних, що дозволяє оцінити компактність зберігання інформації і можливість її стиснення - завдяки цій події ідеї почали втілюватися в реальність. Була запропонована ідея стиснення в результаті підрахунку частоти появи тих чи інших байт у тексті: текст спочатку оцінюється пакувальником, підраховується частота появи в тексті кожної букви, присутньої в ньому, частота повторення ділянок тексту і т. ін.; складається таблиця цих самих частот, з якої вже другим проходом відбувається упаковка та розпакування.

В основі всіх методів стиснення лежить наступна ідея: якщо представляти елементи, що часто використовуються, короткими кодами, а ті, що рідко використовуються, - довгими кодами, то для зберігання блоку даних потрібно менший об'єм пам'яті, ніж якщо б всі елементи представлялися кодами однакової довжини. Даний факт відомий давно: згадаймо, наприклад, азбуку Морзе, в якій символам, що часто використовуються, поставлено у відповідність короткі послідовності точок і тире, а тим, що рідко зустрічаються, - довгі.

Точний зв'язок між ймовірностями і кодами встановлений в теоремі Шеннона про кодування джерела [28], яка свідчить, що елемент s_i, імовірність появи якого дорівнює $p(s_i)$, найвигідніше представляти $-\log_2 p(s_i)$ бітами. Якщо при кодуванні розмір кодів завжди в точності виходить рівним - $\log_2 p(s_i)$ бітів, то в цьому випадку довжина закодованої послідовності буде мінімальною для всіх можливих

способів кодування. Якщо розподіл ймовірностей $F = \{p(s_i)\}$ незмінний, і вірогідність появи елементів незалежні, то можна знайти середню довжину кодів як зважене середнє

$$H = -\sum_i p(s_i) \cdot \log_2 p(s_i). \quad (1.1)$$

Це значення також називається ентропією розподілу ймовірностей *F* або ентропією джерела в заданий момент часу.

Зазвичай ймовірність появи елемента є умовною, тобто залежить від якоїсь події. У цьому випадку при кодуванні чергового елемента s_i розподіл ймовірностей *F* приймає одне з можливих значень F_k,тобто $F = F_k$ і відповідно $H = H_k$. Можна сказати, що джерело знаходиться в стані k, якому відповідає набір ймовірностей *p_k (s_i)*, створення всіх можливих елементів *s_i*. Тому середню довжину кодів можна розрахувати за формулою [28]

$$H = -\sum_k P_k \cdot Hp = -\sum_{k,i} P_k \cdot p_k(s_i), \quad (1.2)$$

де P_k- імовірність того, що F візьме k - є значення, або, інакше, ймовірність знаходження джерела в стані *k*.

Отже, якщо нам відомо розподіл ймовірностей елементів, що генеруються джерелом, то ми можемо представити дані найбільш компактним чином, при цьому середня довжина кодів може бути обчислена за формулою (1.2).

Але в переважній більшості випадків справжня структура джерела нам невідома, тому необхідно будувати модель джерела, яка дозволила б нам у кожній позиції вхідної послідовності оцінити ймовірність $p(s_i)$ появи кожного елемента s_i, алфавіту вхідної послідовності. У цьому випадку ми оперуємо оцінкою $q(s_i)$ймовірності елемента s_i.

Методи стиснення можуть будувати модель джерела адоптивно у міру обробки потоку даних або використовувати фіксовану модель, створену на основі апріорних уявлень про природу типових даних, що вимагають стиснення.

Процес моделювання може бути або явним, або прихованим. Ймовірності елементів можуть використовуватися в методі як явним, так і неявним чином. Але

стиснення завжди досягається за рахунок усунення статистичної надлишковості у поданні інформації.

Жоден компресор не може стиснути будь-який файл. Після обробки будь-яким компресором розмір частини файлів зменшиться, а решти-збільшиться чи залишиться незмінним. Даний факт виходить виходячи з нерівномірності кодування, тобто різної довжини використовуваних кодів, але найбільш простий для розуміння наступний комбінаторний аргумент.

Існує 2^n різних файлів довжини n біт, де $n=0,1,2,...$ Якщо розмір кожного файлу в результаті обробки зменшується хоча б на 1 біт, то 2^n вихідних файлів буде відповідати найбільше 2^{n-1} стислих файлів, що розрізняються. Тоді, принаймні одному архівному файлу буде відповідати кілька відмінних вихідних, і, отже, його декодування без втрат інформації неможливо в принципі.

Тому неможливий "вічний" архіватор, який здатний нескінченну кількість разів стискати свої архіви. "Найкращим" архіватором є програма копіювання, оскільки в цьому випадку ми можемо бути завжди впевнені в тому, що обсяг даних в результаті обробки не збільшиться.

Регулярна поява заяв про створення алгоритмів стиснення, "що забезпечують стиснення в десятки разів краще, ніж у звичайних архіваторів", або є помилковими чутками, породженими неуцтвом і погонею за сенсацією, або рекламою аферистів. В області стиснення без втрат, тобто власне стиснення, такі революції неможливі. Безумовно, ступінь стиснення компресорами типових даних буде неухильно зростати, але поліпшення складуть у середньому десятки або навіть одиниці відсотків, при цьому кожен наступний етап еволюції коштуватиме значно дорожче попереднього. З іншого боку, у сфері стиснення з втратами, в першу чергу компресії відео, все ще можливе багаторазове поліпшення стиснення при збереженні суб'єктивної повноти одержуваної інформації.

1.2. Подання даних

Розглянемо подвійність природи даних: з одного боку, вміст інформації, а з іншого - її фізична подання. У 1950 році Клод Шеннон (Claude Shannon) заклав основи теорії інформації, у тому числі ідею про те, що дані можуть бути представлені певною мінімальною кількістю бітів. Ця величина отримала назву ентропії даних (термін був запозичений з термодинаміки). Шеннон встановив також, що зазвичай кількість біт у фізичному поданні даних перевищує значення, визначене їх ентропією.

В якості простого прикладу розглянемо дослідження поняття ймовірності з допомогою монети. Можна було б підкинути монету безліч разів, побудувати велику таблицю результатів, а потім виконати певний статистичний аналіз цього великого набору даних з метою формулювання або докази якоїсь теореми. Для побудови набору даних, результати підкидання монети можна було б записувати кількома різними способами: можна було б записувати слова "орел або решка"; можна було б записувати букви "О" або "Р"; або ж можна було б записувати єдиний біт (наприклад "так" або "ні", залежно від того, на яку сторону падає монета). Згідно теорії інформації, результат кожного підкидання монети можна закодувати єдиним бітом, тому останній наведений варіант був би найбільш ефективним з точки зору обсягу пам'яті, необхідного для кодування результатів. З цієї точки зору перший варіант є найбільш марнотратним, оскільки для запису результату єдиного підкидання монети потрібно було б чотири або п'ять символів.

Однак подивимося на це під іншим кутом: у всіх наведених прикладах запису даних ми зберігаємо одні і ті ж результати - одну і ту ж інформацію, використовуючи все менший і менший об'єм пам'яті. Іншими словами, ми виконуємо стиснення даних.

1.3. Типи стиснення інформації

В основі будь-якого способу стиснення лежить модель джерела даних, або, точніше, модель надмірності [22]. Іншими словами, для стиснення даних використовуються деякі апріорні відомості про те, якого роду дані стискаються. Не володіючи такими відомостями про джерело, неможливо зробити ніяких припущень

про перетворення, яке дозволило б зменшити обсяг повідомлення. Модель надмірності може бути статичною, незмінною для всього стиснення повідомлення, або будуватися або параметризуватися на етапі стиснення (і відновлення). Методи, що дозволяють на основі вхідних даних змінювати модель надмірності інформації, називаються адаптивними. Неадаптивними є зазвичай вузькоспеціалізовані алгоритми, які використовуються для роботи з даними, які володіють певними добре окресленими і незмінними характеристиками. Переважна частина досить універсальних алгоритмів є тією чи іншою мірою адаптивними.

Всі методи стиснення даних діляться на два основних класу [6]:

– стиснення без втрат (lossless);
– стиснення з втратами (lossy).

1.3.1. Стиснення без втрат

Стиснення без втрат простіше в реалізації [28]. Це метод стиснення даних, коли при відновленні даних повертається точна копія вихідних даних. При використанні стиснення без втрат можливо повне відновлення початкових даних, стиснення з втратами дозволяє відновити дані з спотвореннями, зазвичай несуттєвими з точки зору подальшого використання відновлених даних. Стиснення без втрат зазвичай використовується для передачі і зберігання текстових даних, комп'ютерних програм, рідше - для скорочення обсягу аудіо та відео, цифрових фотографій і т. п., у випадках, коли спотворення неприпустимі або небажані.

Розглянемо наступну теорему.

Теорема. Для будь-якого файлу розміром N > 0 немає такого алгоритму стиснення без втрат, який:

a) залишає файл тієї ж довжини;
b) створює файл завдовжки не більш N, розміром меншим хоча б на один байт.

Доказ. Не обмежуючи спільності, можна припустити, що файл A зменшився рівно N байт. Розглянемо множину:

$$\left(\sum[\square) \cup \left(\sum \square\right)^{1} \cup \left(\sum \square\right)^{2} \cup \cdots \cup \left(\sum \square\right)^{N-1} \cup \{A\}\right]$$

У цій множині $256^0 + 256^1 + \cdots 256^{N-1} + 1$ вихідних файлів, у той час як стислих не більш ніж$256^0 + 256^1 + \cdots 256^{N-1}$. Тому функція декомпресії неоднозначна, протиріччя.

Теорема доведена.

Втім, ця теорема аніскільки не кидає тінь на стиснення без втрат. Будь-який алгоритм стиснення можна модифікувати так, щоб він збільшував розмір не більше ніж на 1 біт (якщо алгоритм зменшив файл, пишемо «1», потім стислу послідовність, якщо збільшив - пишемо «0», потім вихідну) - так що нестискувані фрагменти не приведуть до безконтрольного «роздування» архіву. «Реальних» же файлів довжини N набагато менше, ніж 256^N (кажуть, що дані мають низьку інформаційну ентропію) - наприклад, малоймовірно, щоб буквосполучення «щи» зустрілося в осмисленому тексті, а в зашифрованому звуці рівень не може за один семпл стрибнути від 0 до 100%. До того ж за рахунок спеціалізації алгоритмів на деякий тип даних (текст, графіку, звук і т. д.) вдається домогтися високого ступеня стиснення: так, універсальні алгоритми, що застосовуються в архіваторах, стискають звук приблизно на третину (в 1,5 рази), в той час як FLAC - в 2,5 рази. Більшість спеціалізованих алгоритмів малопридатні для файлів «чужих» типів: наприклад, звукові дані погано стискаються алгоритмом, розрахованим на тексти.

Розглянемо загальний сенс стиснення без втрат. У вихідних даних знаходять якусь закономірність і з урахуванням цієї закономірності генерують другу послідовність, яка повністю описує вихідну. Наприклад, для кодування двійкових послідовностей [7], в яких багато нулів і мало одиниць, ми можемо використовувати таку заміну:

$$00 \rightarrow 0$$

$$01 \rightarrow 10$$

$$10 \rightarrow 110$$

$$11 \rightarrow 111$$

У такому випадку шістнадцять бітів

00 01 00 00 11 10 00 00

будуть перетворені в тринадцять бітів

0 10 0 0 111 110 0 0.

Така заміна є префіксним кодом, тобто має таку особливість: якщо ми запишемо стислий рядок без пробілів, ми все одно зможемо поставити в ньому прогалини - а значить, відновити початкову послідовність. Найбільш відомим префіксним кодом є код Хаффмана.

Більшість алгоритмів стиснення без втрат працюють у дві стадії: на першій генерується статистична модель для вхідних даних, друга відображає вхідні дані в бітовому поданні, використовуючи модель для отримання «ймовірнісних» (тобто тих, що часто зустрічаються) даних, які використовуються частіше, ніж «не імовірнісні».

1.3.2. Стиснення даних з втратами

Стиснення з втратами, що має значно більшу, ніж стиснення без втрат, ефективність, зазвичай застосовується для скорочення обсягу аудіо та відео та цифрових фотографій в тих випадках, коли таке скорочення є пріоритетним, а повна відповідність вихідних і відновлених даних не потрібна. В свою чергу стиснення з втратами не дозволяє при відновленні отримати ті ж вихідні дані. Це здається недоліком, але для певних типів даних, таких як дані зображення і звуку, різниця між відновленими і вихідними даними не має особливого значення: наші зір і слух не в змозі розрізнити що утворилися відмінності. У загальному випадку алгоритми стиснення з втратами забезпечують більш ефективне стиснення, ніж алгоритми стиснення без втрат (інакше їх не варто було б використовувати взагалі). Для прикладу можна порівняти призначений для зберігання зображень формат з втратами JPEG форматом без втрат GIF. Безліч форматів потокового аудіо та відео, що використовуються в Internet для завантаження мультимедійних матеріалів, є алгоритмами стиснення з втратами.

Існують дві основні схеми стиснення з втратами [28]:

– В трансформуючих кодеках кадри зображень або звуку трансформуються в новий базисний простір і проводиться квантування. Трансформація може здійснюватися або для всього кадру цілком (як, наприклад, у схемах на основі wavelet-перетворення), або по блоках (характерний приклад - JPEG). Результат потім стискається ентропійними методами.
– В прогнозуючих кодеках попередні і/або подальші відліки даних використовуються для того, щоб передбачити поточний відлік зображення або звуку. Помилка між передбаченими даними і реальними разом з додатковою інформацією, необхідною для виробництва прогнозу, потім квантується і кодується.

У деяких системах ці дві техніки комбінуються шляхом використання трансформуючих кодеків для стиснення помилкових сигналів, генерованих на стадії пророкування.

Перевага методів стиснення з втратами над методами стиснення без втрат полягає в тому, що перші істотно перевищують за ступенем стиснення, продовжуючи відповідати поставленим вимогам, а саме - трансформація повинна бути в допустимих межах чутливості людських органів.

Методи стиснення з втратами часто використовуються для стиснення аналогових даних - найчастіше звуку або зображень.

У таких випадках розпакований файл може дуже сильно відрізнятися від оригіналу на рівні порівняння «бітів у біт», але практично не відрізняється для людського вуха або очей в більшості практичних застосувань.

Багато методів фокусуються на особливості будови органів почуттів людини. Психоакустична модель визначає те, як сильно звук може бути стиснутий без погіршення сприйняття якості звуку. Недоліки, завдані стисненням з втратами, які помітні для людського вуха або очей, відомі як артефакти стиснення.

1.4 Коефіцієнт компресії

Коефіцієнт стиснення - основна характеристика алгоритмів стиснення [28]. Він визначається як відношення обсягу вихідних нестиснутих даних до обсягу стислих, тобто:

$$k = \left(1 - \frac{S_c}{S_0}\right) \cdot 100\%, \qquad (1.3)$$

де k - коефіцієнт стиснення, S_0 - обсяг вихідних даних, а S_c - обсяг стиснутих. Таким чином, чим вище коефіцієнт стиснення, тим алгоритм ефективніше. Слід зазначити, якщо $k = 0$, то алгоритм не проводить стиснення, тобто вихідне повідомлення виявляється за обсягом рівним вхідному.

Принципово неможливо отримати алгоритм стиснення без втрат, що при будь-яких даних утворював би на виході дані меншою або рівної довжини. Обґрунтування цього факту полягає в тому, що оскільки число різних повідомлень довжиною n біт становить рівно 2n, число різних повідомлень з довжиною менше або дорівнює n (за наявності хоча б одного повідомлення меншої довжини) буде менше 2n. Це означає, що неможливо однозначно зіставити всі вихідні повідомлення стисненим: або деякі вихідні повідомлення не будуть мати стисненого подання, або кільком вихідним повідомляється буде відповідати одне і те ж стисле, а отже, їх не можна відрізнити.

Коефіцієнт стиснення може бути як постійним (деякі алгоритми стиснення звуку, зображення і т. п., наприклад A-закон, μ-закон, ADPCM, обрізане блочне кодування), так і змінним. У другому випадку він може бути визначений або для кожного конкретного повідомлення, або оцінений за деякими критеріями:

- середній (зазвичай по деякому тестового набору даних);
- максимальний (випадок найкращого стиснення);
- мінімальний (випадок найгіршого стиснення);
- будь-яким іншим.

Коефіцієнт стиснення з втратами при цьому сильно залежить від допустимої похибки стиснення або якості, що зазвичай виступає як параметр алгоритму. У загальному випадку постійний коефіцієнт стиснення здатні забезпечити тільки методи стиснення даних з втратами.

1.5. Допустимість втрат

Основним критерієм відмінності між алгоритмами стиснення є описане вище наявність або відсутність втрат. У загальному випадку алгоритми стиснення без втрат універсальні в тому сенсі, що їх застосування безумовно можливе для даних будь-якого типу, у той час як можливість застосування стиснення з втратами повинна бути обґрунтована. Для деяких типів даних спотворення не припустимі в принципі. У їх числі:

- символьні дані, зміна яких неминуче призводить до зміни їх семантики: програми та їх вихідні тексти, двійкові масиви тощо;
- життєво важливі дані, зміни до яких можуть призвести до критичних помилок: наприклад, одержувані з медичної вимірювальної апаратури або контрольних приладів, космічних літальних апаратів і т. п.;
- проміжні дані, що багаторазово піддаються стиску й відновленню при багатоетапній обробці графічних, звукових та відео даних.

1.6. Системні вимоги алгоритмів

Різні алгоритми можуть вимагати різної кількості ресурсів обчислювальної системи, на яких вони реалізовані:

- оперативної пам'яті (під проміжні дані);
- постійної пам'яті (під код програми і константи);
- процесорного часу.

В цілому, ці вимоги залежать від складності і «інтелектуальності» алгоритму. Загальна тенденція така: чим ефективніше і досконаліше алгоритм, тим більші вимоги до обчислювальних ресурсів він висуває. Тим не менш, у специфічних випадках прості і компактні алгоритми можуть працювати не гірше складних і універсальних. Системні вимоги визначають їх споживчі якості: чим менш вимогливий алгоритм, тим на більш простій, а отже, компактній, надійній і дешевій системі він може бути реалізований.

Так як алгоритми стиснення і відновлення працюють у парі, має значення співвідношення системних вимог до них. Нерідко можна ускладнивши один алгоритм значно спростити інший. Таким чином, можливі три варіанти [6, 28]:

1. Алгоритм стиснення вимагає великих обчислювальних ресурсів, ніж алгоритм відновлення. Це найбільш поширене співвідношення, характерне для випадків, коли одноразово стислі дані будуть використовуватися багаторазово. Як приклад можна привести цифрові аудіо - і відео програвачі.

2. Алгоритми стиснення і відновлення вимагають приблизно рівних обчислювальних ресурсів. Найбільш прийнятний варіант для ліній зв'язку, коли стиснення і відновлення відбувається раз на двох її кінцях (наприклад, в цифрової телефонії).

3. Алгоритм стиснення істотно менш вимогливий, ніж алгоритм відновлення. Така ситуація характерна для випадків, коли процедура стиснення реалізується простим, часто портативним пристроєм, для якого обсяг доступних ресурсів дуже критичний, наприклад, космічний апарат або велика розподілена мережа датчиків. Це можуть бути також дані, розпакування яких потрібно в дуже малому відсотку випадків, наприклад записи камер відеоспостереження.

1.7. Циклічний надлишковий код

Алгоритм обчислення контрольної суми (англ. Cyclic redundancy code, CRC циклічний надлишковий код) - спосіб цифрової ідентифікації деякої послідовності даних, який полягає в обчисленні контрольного значення її циклічного надлишкового коду [19].

Алгоритм CRC базується на властивостях ділення із залишком двійкових многочленів, тобто многочленів над кінцевим полем GF (2). Значення CRC є по суті залишком від ділення многочлена, відповідного вхідним даними, на якийсь фіксований многочлен, який породжує.

Кожній кінцевої послідовності бітів взаємо однозначно зіставляється двійковий многочлен, послідовність коефіцієнтів якого є вихідну послідовність. Наприклад, послідовність бітів 1011010 відповідає многочлену:

Кількість різних многочленів ступеня меншою N дорівнює , що збігається з числом всіх двійкових послідовностей довжини N.

Значення CRC з породжує многочленом $G(x)$ ступеня N визначається як бітова послідовність довжини N, що представляє многочлен $R(x)$, що вийшов в залишку при діленні многочлена $P(x)$, що представляє вхідний потік біт, на многочлен $G(x)$:

де $R(x)$ - многочлен, що представляє значення CRC.

$P(x)$ - многочлен, коефіцієнти якого представляють вхідні дані.

$G(x)$ - породжує многочлен.

Множення x^N здійснюється приписуванням N нульових бітів до вхідної послідовності, що покращує якість хешування для коротких вхідних послідовностей.

При поділі із залишком ступінь многочлена-залишку строго менше ступеня многочлена-дільника, тобто при діленні на многочлен $G(x)$ ступеня N можна отримати різних залишків від ділення. При «правильному» виборі породжує многочлена $G(x)$, залишки від ділення на нього будуть володіти потрібними властивостями хешування - гарним перемішуванням і швидким алгоритмом

обчислення. Друге забезпечується тим, що ступінь породжує многочлена зазвичай пропорційна довжині байта або машинного слова (наприклад 8, 16 або 32).

Операція поділу на примітивний поліном також еквівалентна наступною схемою:

– Нехай обраний примітивний поліном, що задає цикл де Брейна 0010111001011100 ... і блок даних 0111110, побудована таблиця, верхній рядок заповнена блоком даних, а нижні рядки - зміщення на 0,1,2 біт циклу де Брейна.
– Тоді контрольна сума буде дорівнює операції XOR тих стовпців, над якими у верхній частині розташована 1. В цьому випадку, 010 xor 101 xor 011 xor 111 xor 110 = 101 (CRC).

РОЗДІЛ 2 ОСНОВИ АЛГОРИТМІВ СТИСНЕННЯ ДАНИХ

У разі експериментів з підкиданням монети було дуже легко визначити найкращий спосіб зберігання набору даних. Але для інших даних це завдання стає більш складним. При цьому можна застосувати кілька алгоритмічних підходів. Два класи стиснення, які будуть розглянуті в цьому розділі, представляють собою алгоритми стиснення без втрат і називаються кодуванням з мінімальною надлишковістю (minimum redundancy coding), також стисненням із застосуванням словника (dictionary compression).

Кодування з мінімальною надлишковістю - це метод кодування байтів (або, більш строго, символів), при якому байти, які частіше зустрічаються, кодуються меншою кількістю бітів, ніж ті, які зустрічаються рідше [8,11]. Наприклад, в тексті англійською мовою букви E, T и A зустрічаються частіше, ніж букви Q, X і Z. Тому, якщо б вдалося закодувати букви E, T и A меншою кількістю бітів, ніж 8 (як має бути у відповідності зі стандартом ASCII), а букви Q, X і Z - більшим, текст англійською мовою вдалося б зберегти з використанням меншої кількості бітів, ніж при дотриманні стандарту ASCII.

При використанні стиснення із застосуванням словника дані розбиваються на великі фрагменти (звані лексемами), ніж символи. Потім застосовується алгоритм кодування лексем з певною мінімальністю кількості бітів. Наприклад, слова "the", "and" і "to" будуть зустрічатися частіше, ніж такі слова, як "electric", "ambiguous" і "irresistible", тому їх потрібно закодувати меншою кількістю бітів, ніж потрібно було б при кодуванні у відповідності зі стандартом ASCII.

2.1. Стиснення з мінімальною надлишковістю

Розглянемо безпосередньо алгоритми стиснення і відновлення даних. Проведемо дослідження алгоритмів кодування з мінімальною надлишковістю, а потім розглянемо більш складне стиснення із застосуванням словника [43, 47].

Розглянемо докладний опис 2 алгоритмів кодування з мінімальною надлишковістю: кодування Шеннона-Фано (Shannon-Fano), кодування Хаффмана (Haffman). При використанні кожного з цих алгоритмів вхідні дані аналізуються як потік байтів, і різним значенням байтів тим або іншим способом присвоюються різні послідовності бітів.

2.1.1. Кодування Шеннона-Фано

Алгоритм стиснення - кодування Шеннона-Фано [33,49], названий так по імені двох дослідників, які одночасно і незалежно один від одного розробили цей алгоритм: Клода Шеннона (Claude Shannon) і Р. М. Фано (R. M. Fano). Алгоритм аналізує вхідні дані і на їх основі будує бінарне дерево мінімального кодування. Використовуючи це дерево, потім можна виконати повторне зчитування вхідних даних і закодувати їх.

Щоб проілюструвати роботу алгоритму, виконаємо стиснення пропозиції "How much wood could a woodchuck chuck?" ("Скільки дров міг би заготовити дровосік?") Насамперед, речення необхідно проаналізувати. Переглянемо дані і обчислимо, скільки разів у реченні зустрічається кожен символ. Занесемо результати в таблицю 2.1.

Частота появи символів у прикладі

Таблиця 2.1

Символ	**Частота появи**

Пробіл	6
c	6
o	6
u	4
d	3
h	3
w	3
k	2
H	1
a	1
i	1
m	1
?	1

Тепер розділимо таблицю на дві частини, щоб загальне число появ символів у верхній половині таблиці приблизно дорівнювало загальному числу появ у нижній половині. Пропозиція містить 38 символів, отже, верхня половина таблиці повинна відображати приблизно 19 появ символів. Це просто: достатньо помістити розділову лінію між рядком o і рядком u. В результаті цього верхня половина таблиці буде відображати появу 18 символів, а нижня - 20. Таким чином, ми отримуємо таблицю 2.2.

Початок побудови дерева Шеннона-Фано

Таблиця 2.2

Символ	Частота появи
Пробіл	6

Продовження табл.2.2

c	6
o	6
----------відокремлююча лінія 1	

u	4
d	3
h	3
w	3
k	2
H	1
a	1
i	1
m	1
?	1

Тепер зробимо те ж з кожною з частин таблиці: вставимо лінію між рядками так, щоб розділити кожну з частин. Продовжимо цей процес, поки всі букви не виявляться відокремлені одна від іншої. Результуюче дерево Шеннона-Фано представлено в таблиці 2.3.

Завершене дерево Шеннона-Фано

Таблиця 2.3

Символ	Частота появи
Пробіл	6
----------відокремлююча лінія 2	
c	6
----------відокремлююча лінія 3	
o	6
----------відокремлююча лінія 1	
u	4
----------відокремлююча лінія 3	
d	3

Продовження табл.2.3

----------відокремлююча лінія 4	
h	3
----------відокремлююча лінія 2	
w	3
----------відокремлююча лінія 4	

k	2
----------відокремлююча лінія 3	
H	1
----------відокремлююча лінія 5	
a	1
----------відокремлююча лінія 4	
i	1
----------відокремлююча лінія 5	
m	1
----------відокремлююча лінія 6	
?	1

Розділові лінії зображені різними по довжині, щоб розділова лінія 1 була найдовшою, розділова лінія 2 трохи коротше і так далі, аж до найкоротшою розділової лінії. Цей підхід зумовлений тим, що розділові лінії утворюють повернуте на 90° бінарне дерево (щоб переконатися в цьому, можна повернути таблицю на 90° проти годинникової стрілки). Розділова лінія 1 є кореневим вузлом дерева, розділові лінії 2 - двома його дочірніми вузлами і т. ін. Символи утворюють листя дерева. Результуюче дерево у звичайній орієнтації показано на рис.2.1.

Все це дуже добре, але як воно допомагає вирішити завдання кодування кожного символу і виконання стиснення? Що ж, щоб дістатися до символу «пробіл», ми починаємо з кореневого вузла, переміщаємося вліво, а потім знову ліворуч. Щоб дістатися до символу «с», ми зміщуємося вліво з кореневого вузла, потім вправо, а потім ліворуч. Для переміщення до символу «о» потрібно перейти ліворуч, а потім два рази вправо.

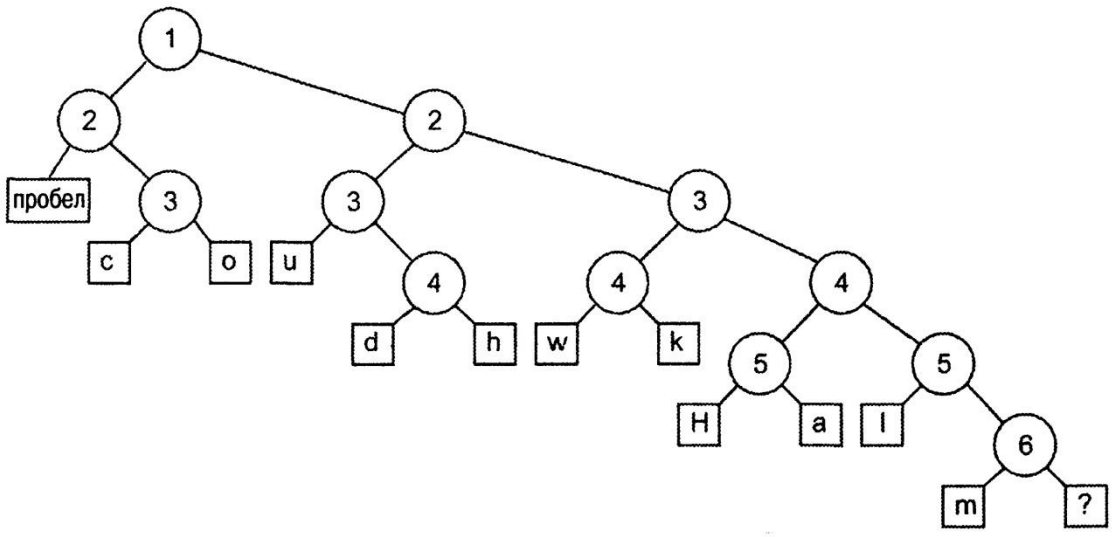

Рис. 2.1 - Дерево Шеннона-Фано

Якщо прийняти, що переміщення вліво еквівалентно нульовому биту, а праворуч - одиничному, можна створити таблицю кодування, наведену в таблиці 2.4.

Коди Шеннона-Фано для прикладу речення

Таблиця 2.4

Символ	Частота появи
Пробіл	00
c	010
o	010
u	100
d	1010
h	1011
w	1100
k	1101
H	11100
a	11101

Продовження табл.2.4

i	11110

m	111110
?	111111

Зараз ми можемо обчислити код для всієї фрази. Він починається з і містить всього 131 біт. Якщо ми припускаємо, що вихідна фраза закодована кодом ASCII, тобто один байт на символ, то оригінальна фраза зайняла б 256 байт, тобто ми отримуємо коефіцієнт стиснення 54%.

Для декодування стисненого потоку бітів ми будуємо те ж дерево, яке було побудовано на етапі стиснення. Ми починаємо з кореневого вузла і вибираємо з стисненого потоку бітів по одному биту. Якщо біт є нульовим, ми пересуваємося вліво, якщо одиничним - вправо. Ми продовжуємо цей процес до тих пір, поки не досягнемо листа, тобто символу, після чого виводимо символ в потік відновлених даних. Потім ми знову починаємо процес з кореневого вузла дерева з метою отримання наступного біта. Зверніть увагу, що оскільки символи розташовані тільки в листі дерева, код одного символу не утворює першу частину коду іншого символу. Завдяки цьому, неправильне декодування стислих даних неможливо. (Бінарне дерево, в якому дані розміщені тільки в листі, називається префіксним деревом (prefix tree).)

Однак при цьому виникає невелика проблема: як розпізнати кінець потоку бітів. Врешті-решт, всередині класу ми будемо об'єднувати вісім бітів у байт, після чого виконувати запис байта. Малоймовірно, щоб потік бітів містив кількість бітів строго кратне 8. Існує два способи вирішення цієї дилеми. Перше - закодувати спеціальний символ, відсутній у вихідних даних, і назвати його символом кінця файлу. Друге - записати в стислий потік довжину нестиснутих даних перед тим, як приступити до стиснення самих даних. Перше рішення змушує нас знайти відсутній у вихідних даних символ і використовувати його (це передбачає передачу цього символу у складі стислих даних програмі відновлення, щоб вона знала, що слід шукати). Або ж можна було б прийняти, що хоча символи даних мають розмір, який дорівнює розміру одного байта, символ кінця файлу має довжину, рівну довжині слова (і задане значення, наприклад 256). Однак ми будемо використовувати друге рішення. Перед стислими

даними ми будемо зберігати довжину нестиснутих даних, і таким чином під час відновлення буде точно відомо, скільки символів потрібно декодувати.

Ще одна проблема застосування кодування Шеннона-Фано [33], на яку досі ми не звертали уваги, пов'язана з деревом. Зазвичай стиснення даних виконується в цілях економії обсягу пам'яті або зменшення часу передачі даних. Як правило, стиснення і відновлення даних рознесене в часі і просторі. Однак алгоритм відновлення вимагає використання дерева. В іншому випадку неможливо декодувати закодований потік. Нам доступні дві можливості. Перша - зробити дерево статичним. Інакше кажучи, одне і те ж дерево буде використовуватися для стиснення всіх даних. Для деяких даних результуюче стиснення буде досить оптимальним, для інших дуже далеким від прийнятного. Друга можливість полягає в тому, щоб тим або іншим способом приєднати саме дерево до стиснутого потоку бітів. Звичайно, приєднання дерева до стиснених даних веде до зниження коефіцієнта стиснення, але з цим нічого не можна поробити.

Кодування Шеннона - Фано є досить старим методом стиснення, і на сьогоднішній день він не представляє особливого практичного інтересу. У більшості випадків, довжина послідовності, стислій за даним методом, дорівнює довжині стислій послідовності з використанням кодування Хаффмана. Але на деяких послідовностях можуть сформуватися неоптимальні коди Шеннона - Фано, тому більш ефективним вважається стиснення методом Хаффмана.

Переваги:

- простота реалізації;
- висока швидкість кодування і декодування.

Основні недоліки цього методу:

- доводиться носити з собою дерево;
- доводиться сканувати текст два рази (перший - при підрахунку частот появи символів, другий - при архівації);
- мізерна ступінь стиснення файлів, що містять майже всі символи;
- важка робота з Ascii-таблицями, наприклад з перейменувати exe'шниками, obj'никами і т.д.

2.1.2. Кодування Хаффмана

Алгоритм кодування Хаффмана дуже схожий на алгоритм стиснення Шеннона-Фано [2]. Цей алгоритм був винайдений Девідом Хаффманом (David Huffman) в 1952 році ("A method for the Construction of Minimum-Redundancy Codes" ("Метод створення кодів з мінімальною надлишковістю")), і виявився ще більш вдалим, ніж алгоритм Шеннона-Фано. Це обумовлено тим, що алгоритм Хаффмана математично гарантовано створює найменший за розміром код для кожного з символів вихідних даних [31, 32, 33]. Аналогічно застосування алгоритму Шеннона-Фано, потрібно побудувати бінарне дерево, яке також буде префіксним деревом, де всі дані зберігаються в листі. Але на відміну від алгоритму Шеннона-Фано, який є спадним, на цей раз побудова буде виконуватися знизу вгору. Спочатку ми виконуємо перегляд вхідних даних, підраховуючи кількість появ значень кожного байти, як це робилося і при використанні алгоритму Шеннона-Фано. Як тільки ця таблиця частоти появи символів буде створена, можна приступити до побудови дерева.

Будемо вважати ці пари символ-кількість "пулом" вузлів майбутнього дерева Хаффмана. Видалимо з цього пулу два вузла з найменшими значеннями кількості появ. Приєднаємо їх до нового батьківського вузла і встановимо значення лічильника батьківського вузла дорівним сумі лічильників його двох дочірніх вузлів. Помістимо батьківський вузол назад в пул. Продовжимо цей процес видалення двох вузлів і додавання замість них одного батьківського вузла до тих пір, поки в пулі не залишиться тільки один вузол. На цьому етапі можна видалити з пулу один вузол. Він є кореневим вузлом дерева Хаффмана.

Описаний процес не дуже наочний, тому створимо дерево Хаффмана для речення "How much wood could a woodchuck chuck?" Ми вже вирахували кількість появ символів цієї пропозиції і представили їх у вигляді таблиці 2.1, тому тепер до неї потрібно застосувати описаний алгоритм з метою побудови повного дерева Хаффмана. Виберемо два вузла з найменшими значеннями (рис.2.2). Існує кілька вузлів, з яких можна вибрати, але ми виберемо вузли "m" і"?". Для обох цих вузлів число появ символів дорівнює 1. Створимо батьківський вузол, значення лічильника якого дорівнює 2, і приєднаємо до нього два обраних вузла як дочірніх. Помістимо

батьківський вузол назад в пул. Повторимо цикл з самого початку. На цей раз ми вибираємо вузли "a" і "1", поєднуючи їх в міні-дерево і поміщаємо батьківський вузол (значення лічильника якого знову рівно 2) назад в пул.

Знову повторимо цикл. На цей раз в нашому розпорядженні є єдиний вузол, значення лічильника якого дорівнює 1 (вузол "H") і три вузла зі значеннями лічильників, рівними 2 (вузол "до" і два батьківських вузла, які були додані перед цим).

Виберемо вузол "до", приєднаємо його до вузла "H" і знову додамо в пул батьківський вузол, значення лічильника якого дорівнює 3. Потім виберемо два батьківських вузла зі значеннями лічильників, рівними 2, приєднаємо їх до нового батьківського вузла зі значенням лічильника, рівним 4, і додамо цей батьківський вузол в пул.

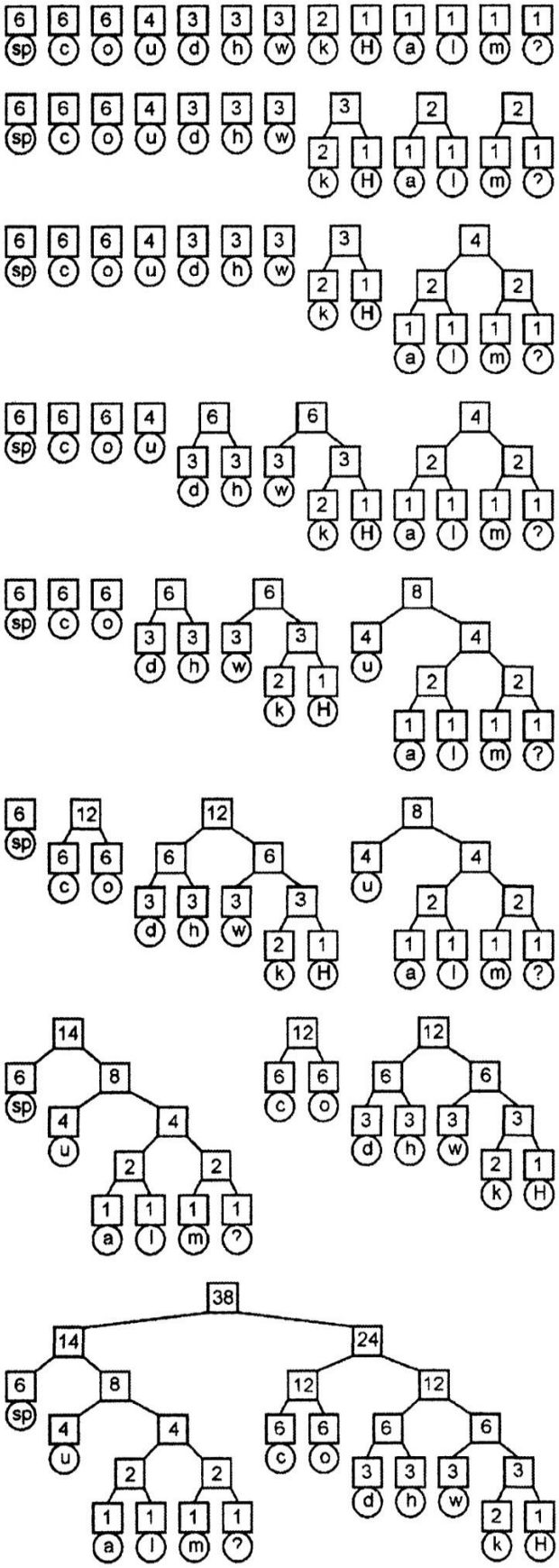

Рис. 2.2 - Побудова дерева Хаффмана

Використовуючи це дерево точно так само, як і дерево, створене для кодування Шенона-Фано, можна обчислити код для кожного з символів у вихідному реченні і побудувати таблицю 2.5.

Коди Хаффмана символів для прикладу речення

Таблиця 2.5

Символ	Частота появи
Пробіл	00
c	100
o	101
u	110
d	1100
h	1101
w	1110
k	11110
H	111111
a	01100
i	01101
m	01110
?	01111

Слід звернути увагу на те, що таблиця кодів - не єдина можлива. Кожен раз, коли є три або більше вузлів, з числа яких потрібно вибрати два, існують альтернативні варіанти результуючого дерева і, отже, вихідних кодів. Але на практиці всі ці можливі варіанти дерев і кодів будуть забезпечувати максимальне стиснення. Всі вони еквівалентні.

Тепер можна обчислити код для всього речення. Він починається з бітів:

11111101111000011100101 00…

і містить всього 131 біт. Якщо б вихідне речення було закодоване ASCII кодами, по одному байту на символ, воно містило б 286 бітів. Таким чином, в даному випадку коефіцієнт стиснення становить приблизно 54%.

Повторимо знову, що, як і при застосуванні алгоритму Шеннона-Фано, необхідно якимось чином стиснути дерево і включити його до складу стислих даних.

Відновлення виконується абсолютно так само, як при використанні кодування Шеннона-Фано: необхідно відновити дерево з даних, що зберігаються в стислому потоці, і потім скористатися ним для зчитування стисненого потоку бітів.

Алгоритм Хаффмана використовує частоту появи однакових байт у вхідному блоці даних, і ставить у відповідність часто зустрічаються блокам ланцюжка біт меншої довжини, і навпаки. Цей код є мінімально - надмірним кодом. Розглянемо випадок, коли, не залежно від вхідного потоку, алфавіт вихідного потоку складається всього з 2 символів - нуля і одиниці.

Тепер складемо загальний алгоритм кодування. В першу чергу при кодуванні алгоритмом Хаффмана, нам потрібно побудувати схему сум. Робиться це наступним чином:

1. Всі букви вхідного алфавіту впорядковуються в порядку убування ймовірностей. Всі слова з алфавіту вихідного потоку (тобто те, чим ми будемо кодувати) спочатку вважаються порожніми (нагадаю, що алфавіт вихідного потоку складається лише з символів {0,1}).
2. Два символи a_{j-1} і a_{j-1} вхідного потоку, що мають найменші ймовірності появи, об'єднуються в один «псевдосимвол» з вірогідністю p рівною сумі ймовірностей символів, що входять в нього. Потім ми доповнюємо 0 до початку слова B_{j-1}, і 1 у початок слова B_j, які будуть згодом бути кодами символів a_{j-1} і a_{j-1} відповідно.
3. Видаляємо ці символи алфавіту вихідного повідомлення, але додаємо в цей алфавіт сформований псевдосимвол (природно, він повинен бути вставлений в алфавіт на потрібне місце, з урахуванням його ймовірності).

Кроки 2 і 3 повторюються до тих пір, поки в алфавіті не залишиться тільки 1 псевдосимвол, що містить всі початкові символи алфавіту. При цьому, оскільки на кожному кроці і для кожного символу відбувається зміна відповідного йому слова B_i (шляхом додавання одиниці або нуля), то після завершення цієї процедури кожному початковому символу алфавіту a_i буде відповідати якийсь код B_i..

Для кращої ілюстрації, розглянемо невеликий приклад.

Нехай у нас є алфавіт, що складається з чотирьох символів - $\{a_1, a_2, a_3, a_4\}$. Припустимо також, що вірогідність появи цих символів рівні відповідно $p_1 = 0.5;\ p_2 = 0.24;\ p_3 = 0.15;\ p_4 = 0.11;$ (сума всіх ймовірностей, очевидно, дорівнює одиниці).

Отже, побудуємо схему для даного алфавіту.

1. Об'єднуємо два символи з найменшими вірогідністю (0.11 і 0.15) в псевдосимвол p'.
2. Видаляємо об'єднані символи, і вставляємо псевдо символ, який вийшов в алфавіт.
3. Об'єднуємо два символи з найменшою ймовірністю (0.24 і 0.26) в псевдосимвол p".
4. Видаляємо об'єднані символи, і вставляємо псевдо символ, який вийшов в алфавіт.
5. Нарешті, об'єднуємо два символи, що залишилися, і отримуємо вершину дерева.

Ілюстрація цього процесу зображена на рис. 2.3.

Основні недоліки цього методу:

- доводиться носити з собою дерево Хаффмана;
- доводиться сканувати текст два рази (перший - при підрахунку частот появи символів, другий - при архівації);
- мізерна ступінь стиснення файлів, що містять майже всі символи;
- важка робота з ASCII-таблицями, наприклад з exe'шниками, obj'никами і т.д.

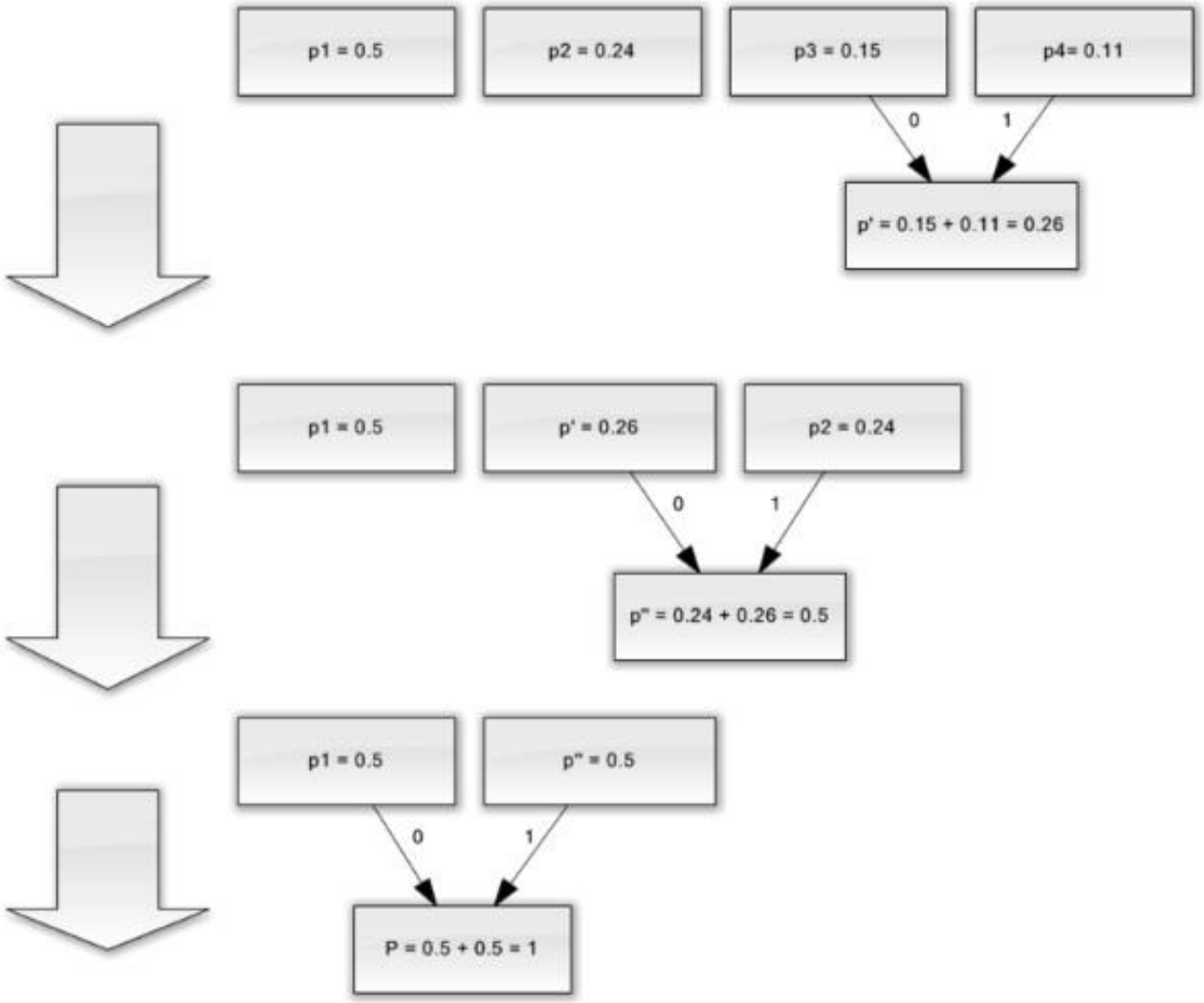

Рис. 2.3 - Схематична реалізація загального алгоритму кодування

Переваги:

- простота реалізації;
- малі обсяги займаної пам'яті при архівації.

2.2. Метод Лемпеля-Зіва

Цей метод використовує зовсім інший підхід, заснований не на частотності появи байт в тексті, а на повторенні слів або частини слів в пакованому тексті [1]. Якщо слово або його фрагмент тексту повторюється, то воно замінюється посиланням на попереднє слово. Архіватор працює в один прохід і становить лише словник, заснований на повторюваних ділянках тексту. Ефективність стиснення тут залежить

лише від розмірів текстового файлу і числа повторень. Безглуздо стискати цим алгоритмом маленькі файли, а тим більше окремі рядки, тому він навряд чи підійде для стиснення кожного рядка масиву логів. На його основі створювалося безліч архіваторів. Сучасні архіватори в більшості своїй також застосовують цей алгоритм спільно з методом Хаффмана для стиснення текстових файлів. LZW використовується в деяких форматах графічних файлів (наприклад, всім відомий і широко використовуваний GIF стискається саме алгоритмом LZW). Якщо говорити про походження цієї абревіатури, нескладно здогадатися, що LZ - похідне від імен головних розробників (Jakob Ziv і Abraham Lempel). Звідки ж буква W? Творці алгоритму не поспішали патентувати його, тому їм користувався хто завгодно, у тому числі розробники Unix. Один з таких хлопців, Террі Велч, вів розробку пакувальника compress. Відповідно, по мірі зміни пакера він змінював і LZW-движок, всіляко оптимізуючи його. Так що Велч назавжди залишився в історії і вніс частину свого імені в абревіатура алгоритму. Пізніше алгоритм модифікувався і не раз, але так як модифікації носили в основному незначний характер, автори програм не стали розширювати абревіатуру.

Алгоритм передбачає кодування послідовності біт шляхом розбиття їх на фрази з подальшим кодуванням цих фраз [22].

Суть алгоритму полягає в наступному: якщо в тексті зустрінеться повторення рядків символів, то повторні рядки замінюються посиланнями (покажчиками) на вихідний рядок. Посилання має формат <префікс, відстань, довжина>. Префікс в цьому випадку дорівнює 1. Поле відстань ідентифікує слово в словнику рядків. Якщо рядка в словнику немає, генерується код символ виду <префікс, символ>, де поле префікс = 0, а поле символ відповідає поточному символу вихідного тексту. Звідси видно, що префікс служить для розділення кодів покажчика від кодів символів. Введення кодів символів, дозволяє оптимізувати словник і підняти ефективність стиснення. Головна алгоритмічна проблема тут полягає у оптимальному виборі рядків, так як це передбачає значний обсяг переборів.

Даний алгоритм при стисканні (кодуванні) динамічно створює таблицю перетворення рядків: певним послідовностей символів (словами) ставляться у відповідність групи біт фіксованої довжини (зазвичай 12-бітові). Таблиця

ініціалізується усіма 1-символьними рядками (у разі 8-бітних символів - це 256 записів). У міру кодування, алгоритм переглядає текст символ за символом, і зберігає кожну нову, унікальну 2-символьний рядок в таблицю у вигляді пари код / символ, де код посилається на відповідний перший символ. Після того як новий 2-символьний рядок збережений в таблиці, на вихід передається код першого символу. Коли на вході читається черговий символ, для нього по таблиці знаходиться рядок, який вже зустрічався, максимальної довжини, після чого в таблиці зберігається код цього рядка з наступним символом на вході; на вихід видається код цього рядка, а наступний символ використовується в якості початку наступного рядка.

Алгоритму декодування на вході потрібно тільки закодований текст, оскільки він може відтворити відповідну таблицю перетворення безпосередньо по закодованого тексту.

Алгоритм методу Лемпела-Зіва має наступний вигляд:

1. Ініціалізація словника всіма можливими односимвольними фразами. Ініціалізація вхідної фрази *w* першим символом повідомлення.
2. Вважати черговий символ *K*з кодованого повідомлення.
3. Якщо КОНЕЦ_СООБЩЕНІЯ, то видати код для *w*, інакше
4. Якщо фраза *wK* вже є в словнику, привласнити вхідний фразі значення *wK* і перейти до кроку 2, інакше видати код w, додати *wK* в словник, присвоїти вхідний фразі значення K та перейти до кроку 2.

На момент своєї появи алгоритм LZW давав кращий коефіцієнт стиснення, для більшості додатків, ніж будь-який інший добре відомий метод того часу. Він став першим широко використовуваним на комп'ютерах методом стиснення даних. Алгоритм був реалізований у програмі compress, яка стала більш-менш стандартною утилітою Unix-систем приблизно в 1986 році. Кілька інших популярних утиліт-архіваторів також використовують цей метод або близькі до нього. У 1987 році алгоритм став частиною стандарту для формату зображень GIF. Він також може (опціонально) використовуватися в форматі TIFF. В даний час, алгоритм міститься в стандарті PDF.

Скільки двійкових розрядів потрібно виділяти для кодування? Відповідь може бути наступним: число розрядів R на кожному кроці кодування дорівнює кількості

розрядів у найбільш довгому з використаних кодів (тобто числу розрядів в останньому використаному порядковому номері). Тому якщо останній використаний код (порядковий номер) дорівнює 13 = 1101, то коди А всіх комбінацій повинні бути чотирирозрядний при кодуванні аж до появи номера 16, після чого всі коди символів починають розглядатися як пятіразрядние (R = 5).

Розглянемо приклад використання даного алгоритму. Нехай вихідний текст являє собою двійковий код (перший рядок таблиці 2.6), тобто символами алфавіту є 0 і 1. Коди цих символів відповідно також 0 і 1. Утворений за методом Лемпеля-Зіва код (LZ-код) показаний у другому рядку таблиці 2.6.

У третьому рядку відзначені кроки кодування, після яких відбувається перехід на подання кодів А збільшеним числом розрядів R. Так, на першому кроці вводиться код 10 для комбінації 00 і тому на наступних двох кроках $R = 2$, після третього кроку R = 3, після сьомого кроку $R = 4$, тобто в загальному випадку $R = K$ після кроку .

Таблиця 2.6

Ісходний текст	0.00.000.01.11.1111.110.0000.00000.1101.1110
LZ-код	0.00.100.001.0011.1011.1101.1010.00110.100010.10001.10110.
R	2 3 4
Введені коди	-10 11 100 101 110 111 1000 1001 1010 1011 1100

У наведеному прикладі LZ-код виявився навіть довше вихідного коду, тому що звичайно короткі тексти не дають ефекту стиснення. Ефект стиснення проявляється у досить довгих текстах і особливо помітний в графічних файлах.

Переваги:

- Не вимагає обчислення ймовірностей зустрічальності символів або кодів.
- Для декомпресії не треба зберігати таблицю рядків у файл для розпакування. Алгоритм побудований таким чином, що ми в змозі відновити таблицю рядків, користуючись тільки потоком кодів.
- Даний тип компресії не вносить спотворень в вихідний графічний файл, і підходить для стиснення растрових даних будь-якого типу.

Недоліки:

– Алгоритм не проводить аналіз вхідних даних тому не оптимальний.

Таким чином, я дослідив і порівняв основні методи стиснення даних і тепер можу перейти безпосередньо до застосування їх на практиці.

РОЗДІЛ 3 РОЗРОБКА АРХІВАТОРУ ТА ЙОГО ФУНКЦІОНАЛУ

3.1. Опис інтерфейсу

При запуску програми відкривається головне вікно програми (рис. 3.1). Завдяки зручному інтерфейсу, програма є легкою у використанні.

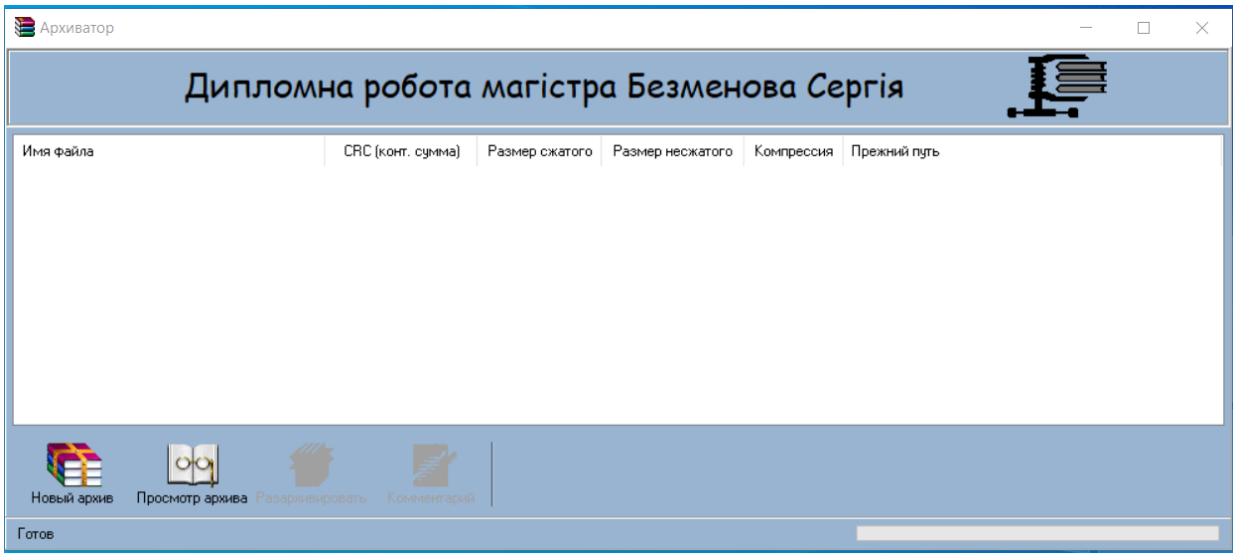

Рис. 3.1 - Головне вікно програми

Архіватор надає наступні можливості:

– Приміщення файлу в архів;
– Витяг файлу з архіву;
– Перегляд змісту архіву;
– Видалення файлу з архіву;
– Додавання коментаря;
– Шифрування даних.

Для того щоб створити архів, необхідно клікнути по кнопці «Новий архів» і у вікні вказати ім'я та місце розташування архіву (рис. 3.2).

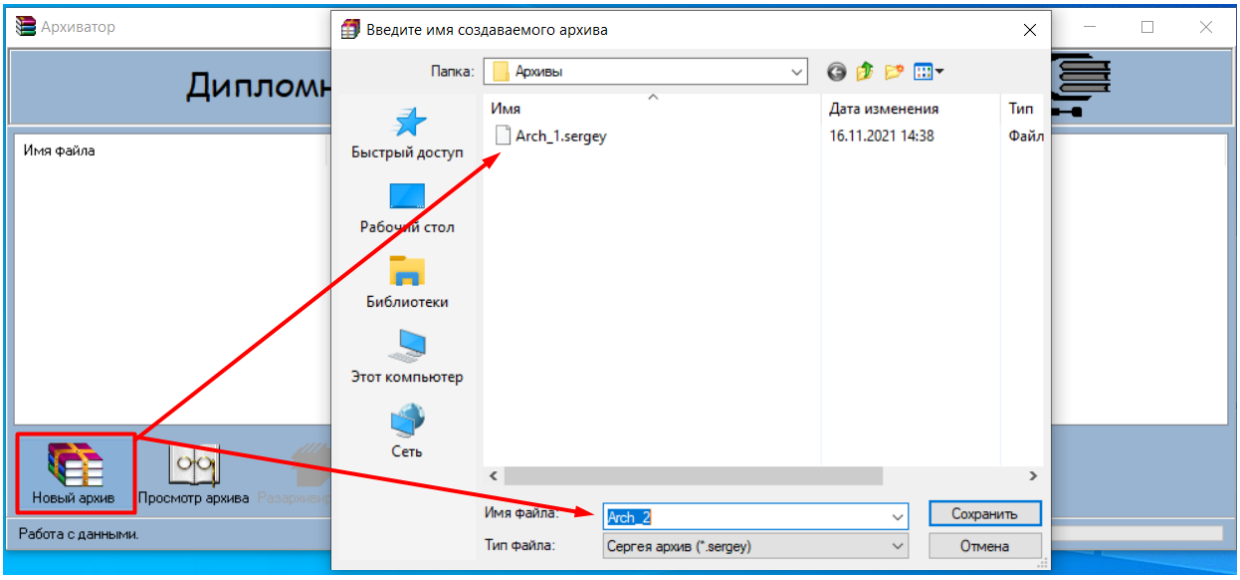

Рис. 3.2 - Створення архіву

Після того, як користувач ввів ім'я та місце розташування архіву необхідно натиснути кнопку «Зберегти». Далі програма виведе діалогове вікно «Додати файл до архіву» (рис. 3.3).

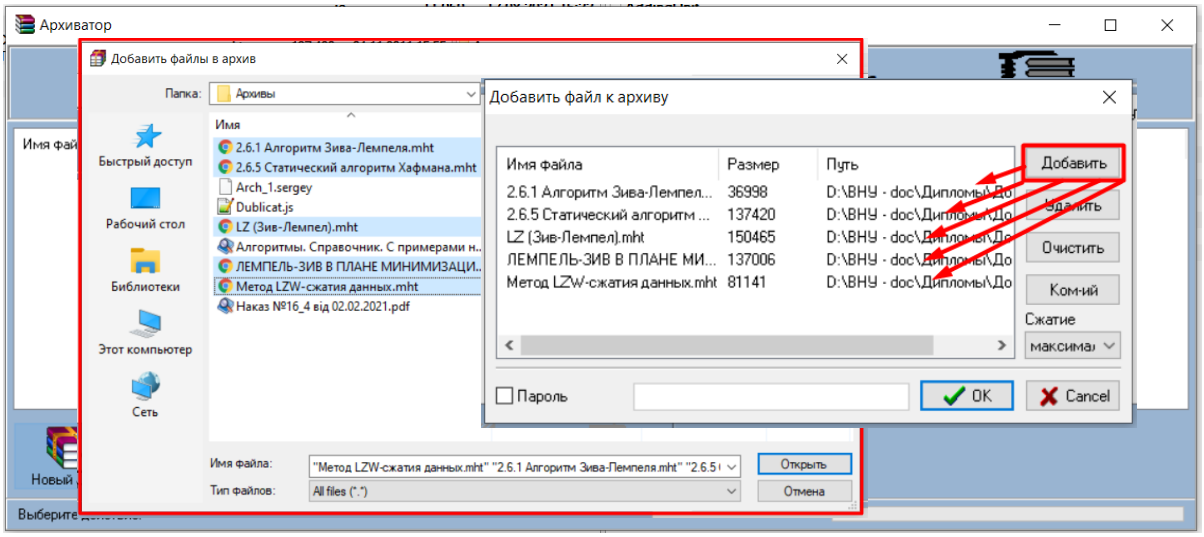

Рис. 3.3 - Додавання файлу до архіву

Дане вікно дає можливість користувачеві додати файл до архіву, натиснувши кнопку «Додати», видалити вибраний файл з переліку архіву, натиснувши кнопку «Видалити», або видалити весь перелік файлів з архіву, скориставшись командою «Очистити». При необхідності можна додати коментар. Для цього необхідно натиснути кнопку «Коментар» і у вікні ввести необхідну інформацію по файлу (рис.3.4).

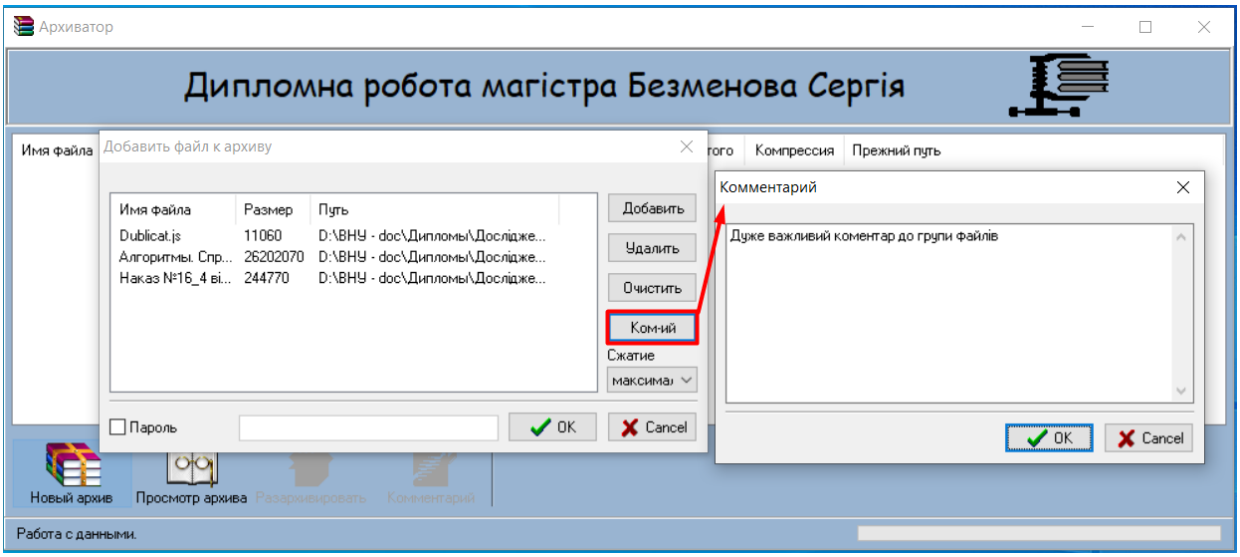

Рис. 3.4 - Додавання коментаря до архіву

Далі користувач вибирає на свій розсуд ступінь стиснення даних (максимальна, нормальна, низька і немає) (рис. 3.5).

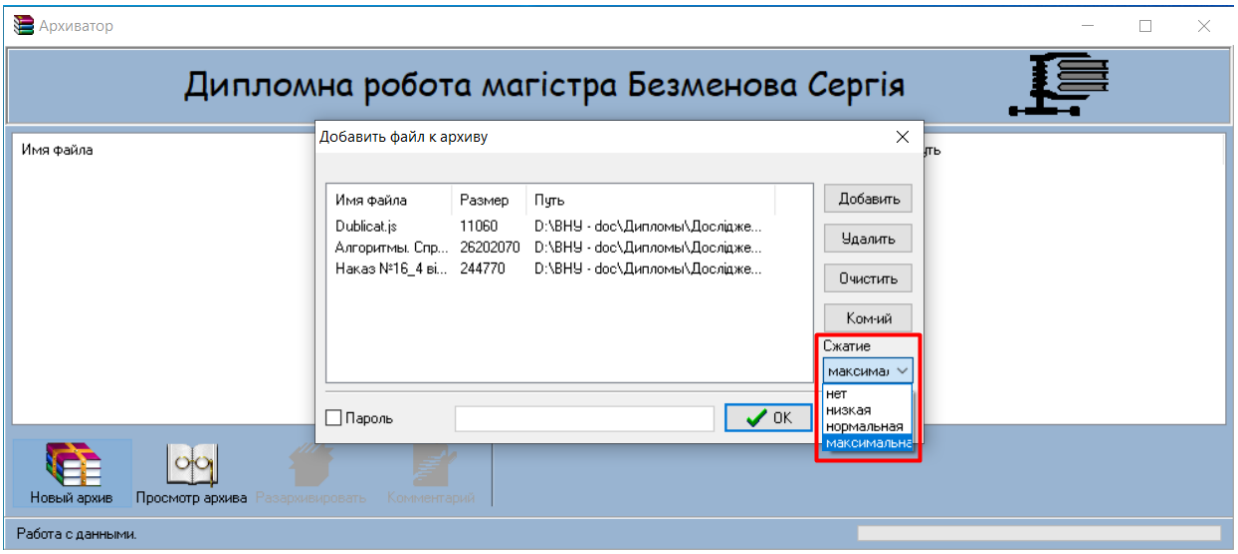

Рис. 3.5 - Вибір ступеня стиснення файлів

Якщо архів необхідно зашифрувати ставимо прапорець «Пароль» і вводимо пароль в поле для введення пароля (рис.3.6).

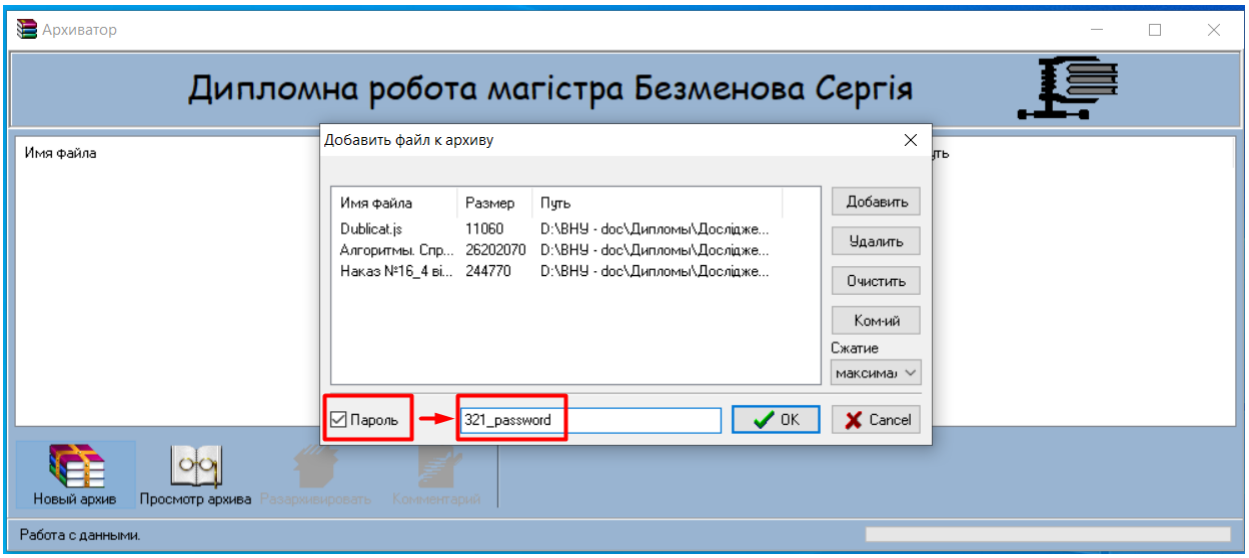

Рис. 3.6 - Введення пароля

Варто звернути увагу на те, що при перегляді архіву, якщо архів з паролем, то на верхній піктограмі видно замок, інакше з'являється картинка «Ключі», тобто це говорить про те, що архів без пароля (рис. 3.7 - 3.8).

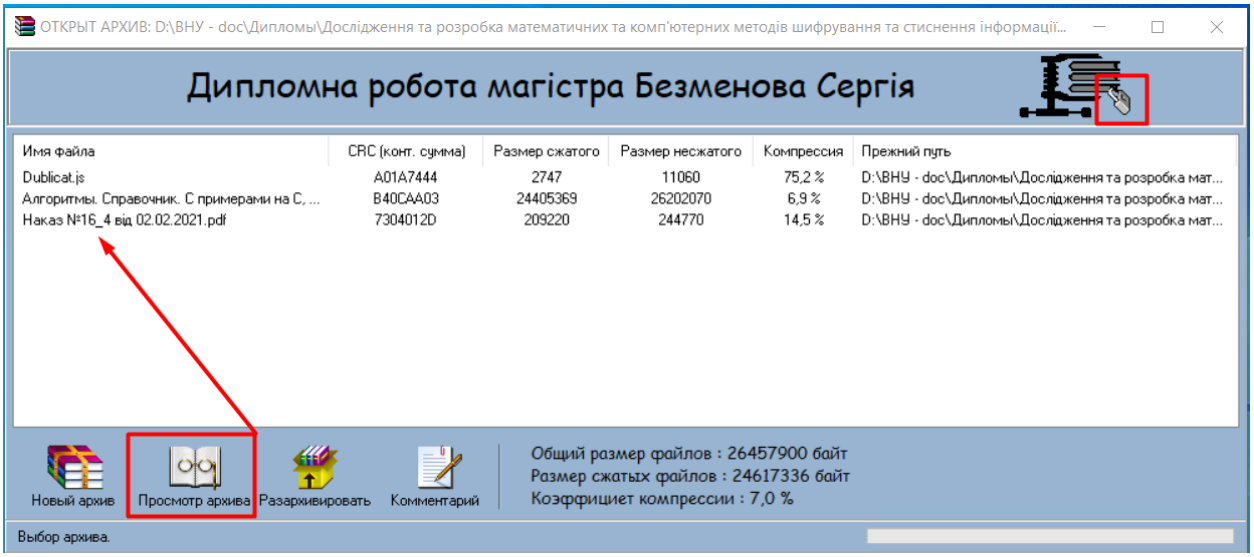

Рис. 3.7 - Архів з паролем

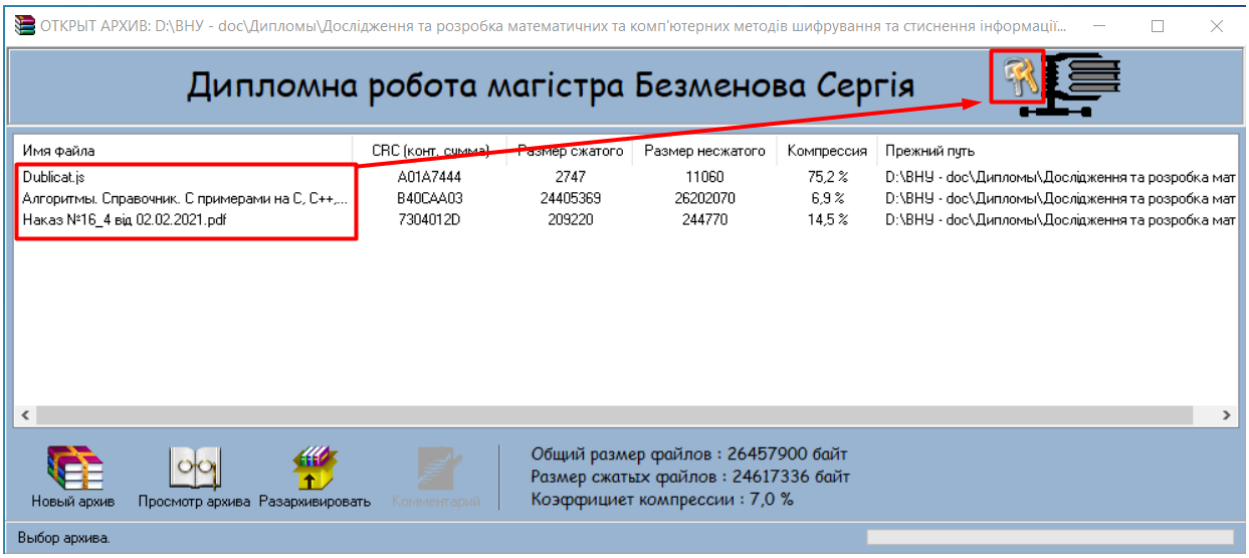

Рис. 3.8 - Архів без пароля

Для завершення створення архіву натискаємо «ОК» (рис. 3.9).

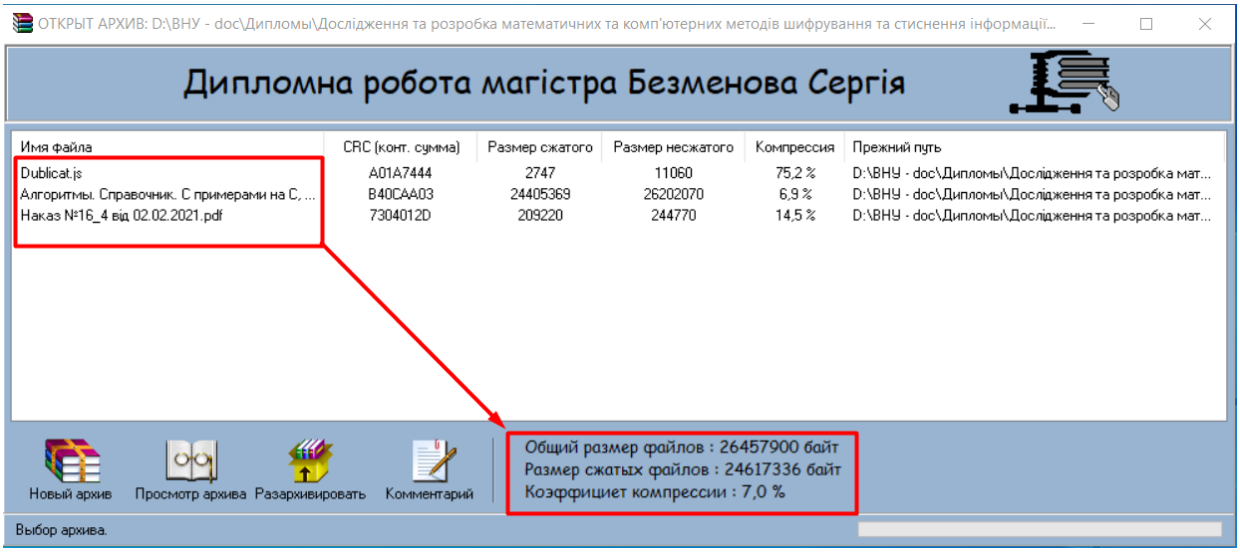

Рис. 3.9 - Готовий архів

Після закінчення процесу стиснення програма виводить перелік архівних файлів, розмір кожного файлу до стиснення і після, ступінь стиснення і шлях до файлу.

Для декодування необхідно клікнути по кнопці «Розархівувати». З'явиться вікно розпакування файлу. Необхідно вказати шлях розархівації вибраного архіву та режим заміни (на випадок якщо будуть конфлікти з існуючими файлами) (рис. 3.10).

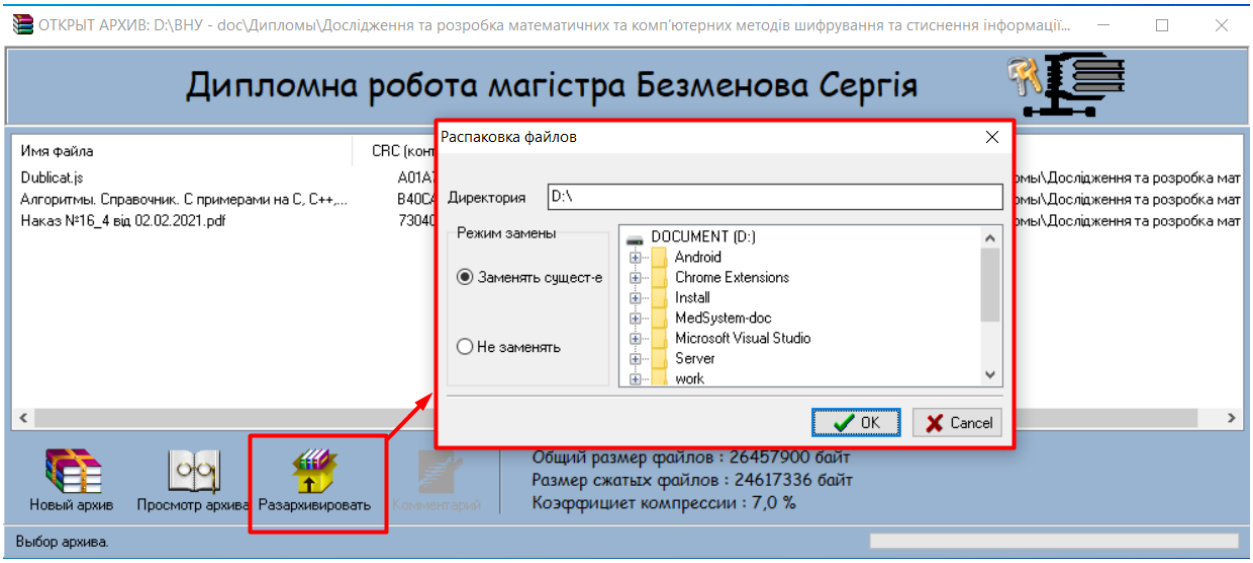

Рис. 3.10 - Процес декодування

У випадку, якщо архів з паролем з'явиться вікно введення пароля. Необхідно ввести пароль. Якщо пароль не вірний з'явиться відповідне повідомлення (рис. 3.11).

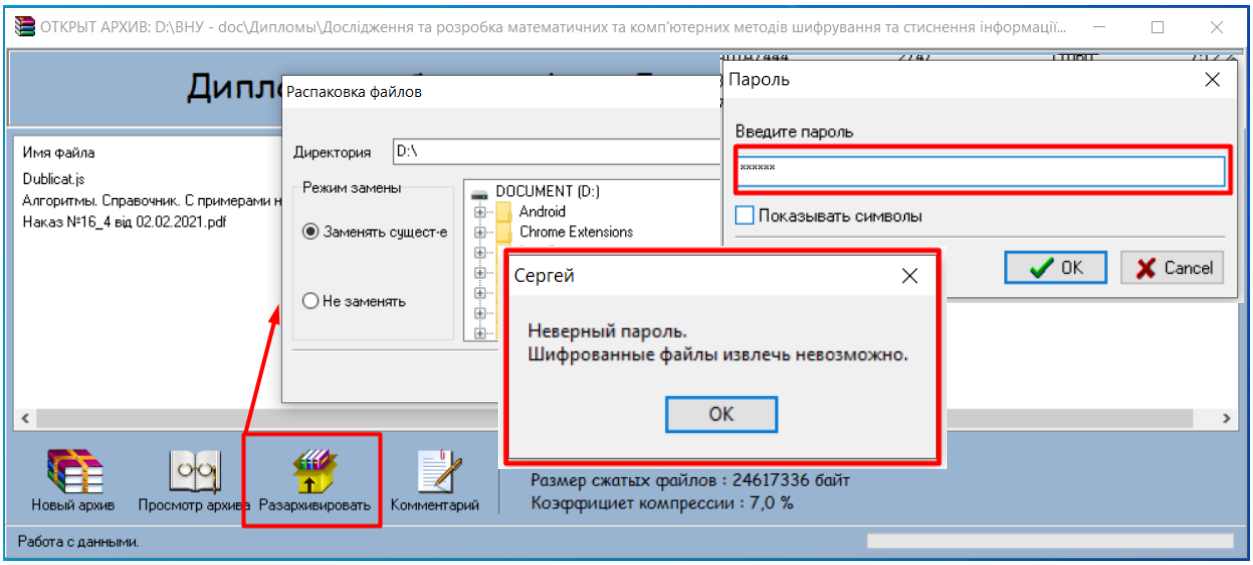

Рис. 3.11 - Випадок введення неправильного пароля

Якщо ж пароль введений вірний, то з'явиться відповідне повідомлення про те, що декодування відбулося і можливо використовувати разархівовані файли (рис. 3.12).

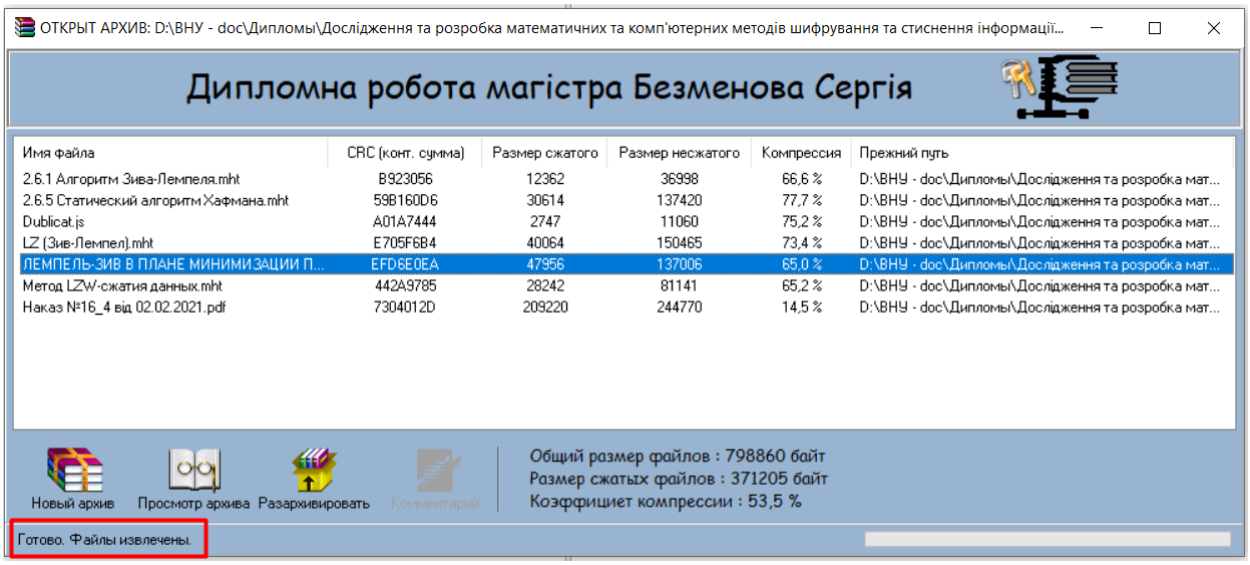

Рис. 3.12 - Закінчення декодування

3.2. Алгоритм стиснення файлу

Ідея алгоритму LZW полягає в тому, що в процесі обробки вихідного файлу формується словник, слова в якому є шматочками коду вихідного файлу [1]. При формуванні стисненого файлу в нього записується не довга послідовність байт, а тільки ідентифікатор відповідного слова зі словника (його номер), або сам символ, якщо потрібного слова немає в словнику. При цьому отримуємо зиск за рахунок того, що дана ланцюжок (слово зі словника) може зустрічатися у файлі декілька разів, а довжина ідентифікатора слова значно коротше самого слова. При збереженні стисненого файлу не потрібно зберігати словник, тому що даний алгоритм дозволяє створити словник в процесі розпаковування стисненого файлу. Блок-схема алгоритму стиснення файлу наведена нижче (рис. 3.13).

Передбачається, що ми маємо словник, в який можемо додавати слова, і здійснювати їх пошук (в словнику). При додаванні нового слова воно отримує свій порядковий номер. Кожне слово унікально, тобто в словнику не може бути двох однакових слів.

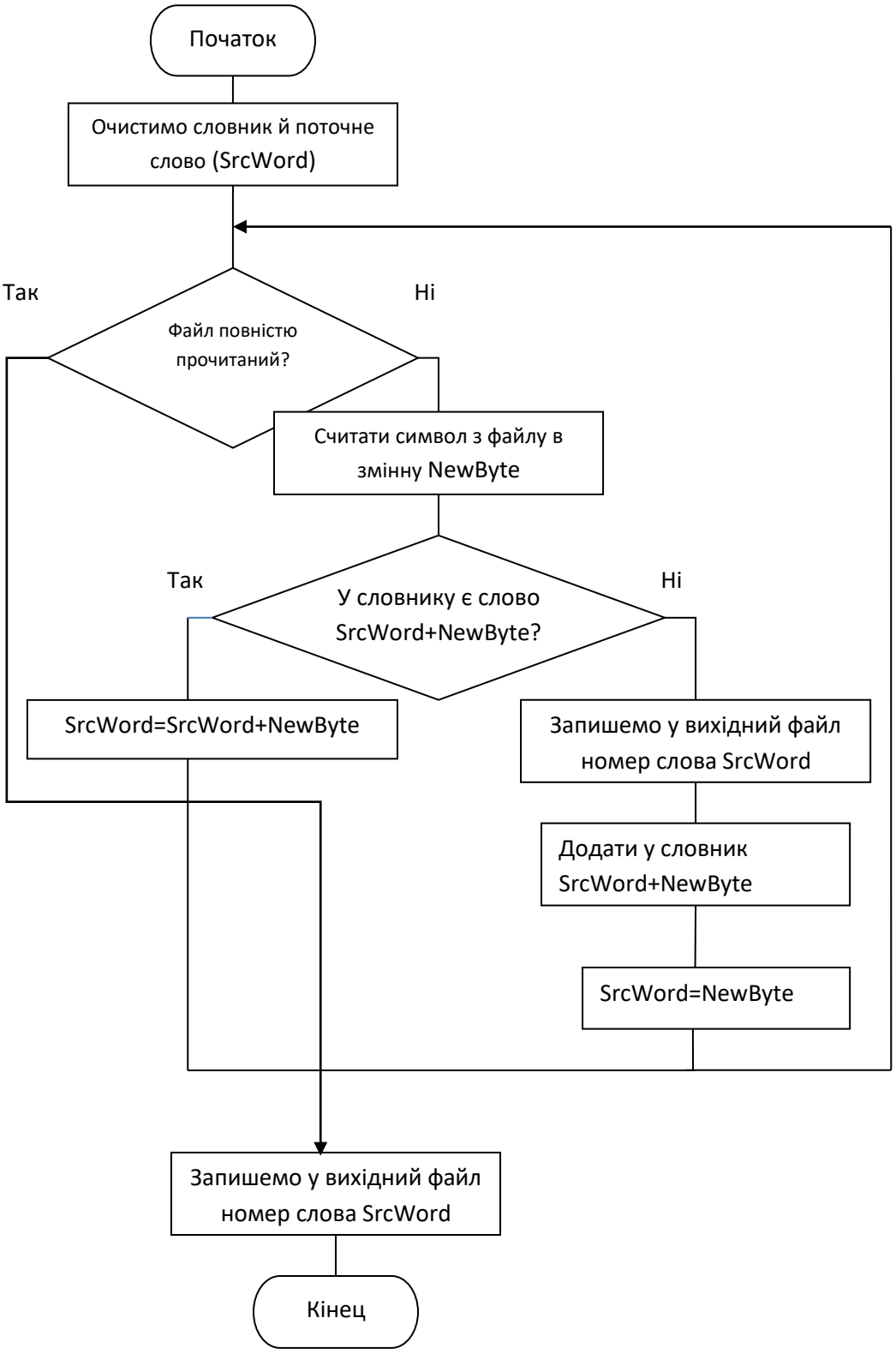

Рис. 3.13 - Блок-схема стиснення файлу

Змінна*NewByte* є символом (байтом), а змінна *SrcWord* - словом (в нашому випадку це *ANSIString*)

Алгоритм стиснення файлу.

1. Очистити словник; очистити *SrcWord*.

2. Якщо вихідний файл повністю лічений, то до п. 6.

3. Считати в *NewByte* черговий символ з вихідного файлу.

4. Якщо в словнику є слово *SrcWord + NewByte*, тоді присвоїти*SrcWord = SrcWord + NewByte*, інакше записати у вихідний файл номер слова SrcWord, додати в словник слово *SrcWord + NewByte*, присвоїти*SrcWord = NewByte.*

5. Перейти до пункту 2.

6. Знайти в словнику слово *SrcWord* і зберегти його номер у вихідному файлі

7. Кінець

Алгоритм розпакування файлу:

Примітка: *NewCode* - це не байт (в даному випадку це тип *Word*)

1. Очистити словник; очистити *SrcWord*

2. Якщо вихідний файл повністю лічений, то до п. 16

3. Считати в *NewCode* дані з стисненого файлу

4. Якщо *NewCode* - це просто символ, тоді перейти до пункту 5, інакше перейти до пункту 8.

5. Записати *NewCode* у вихідний файл.

6. Якщо в словнику є слово *SrcWord + NewByte*, тоді присвоїти*SrcWord = SrcWord + NewByte*, інакше додати в словник слово *SrcWord + NewByte*, присвоїти *SrcWord = NewByte*.

7. Перейти до пункту 2.

8. / / *NewCode* - це номер слова в словнику

Якщо *NewCode* більше, ніж слів у словнику, тобто слова з таким номером в словнику взагалі немає, тоді присвоїти N = 2; присвоїти *NewByte = SrcWord [1]*.

Якщо в словнику є слово *SrcWord + NewByte*, тоді присвоїти*SrcWord = SrcWord + NewByte*, інакше додати в словник слово *SrcWord + NewByte*; привласнити *SrcWord = NewByte*, інакше привласнити *N = 1*.

9. Присвоїти *SaveWord* = слово_із_словаря_с_номером (NewCode).

10. Зберегти *SaveWord* у вихідний файл.

11. Запустити цикл для і від N до довжини слова *SaveWord.*

12. Присвоїти *NewByte = SaveWord [i]*.

13. Якщо в словнику є слово *SrcWord + NewByte*, тоді присвоїти *SrcWord = SrcWord + NewByte*, інакше додати в словник слово *SrcWord + NewByte*; присвоїти*SrcWord = NewByte.*

14. Кінець циклу.

15. Перейти до пункту 2.

16. Кінець.

3.3. Програмна реалізація

Архіватор, що розробляється в дипломній роботі повинен вміти:

- Здійснювати компресію файлів і каталогів без втрат;
- Збирати архів без стиснення (або стискати з різним рівнем компресії);
- Шифрувати дані (зі стиском і без);
- Дописувати коментар до архіву;
- Розраховувати коефіцієнт стиснення файлів;
- Розпаковувати стиснений архів.

Розглянемо структуру створюваного архівного файлу. На початку архіву помістимо заголовок (таб. 3.1), в якому зібрані всі метадані по об'єктах архіву. Після архівного заголовка послідовно йдуть файлові блоки, що зберігають файлові заголовки (таб. 3.2) і тіло файлу після обробки. Визначення меж блоку випливає з заголовка, в якому зберігається розмір блоку.

Формат заголовка архіву

Таблиця 3.1

Назва атрибуту	Розмір
Сигнатура архіву	7 байт
Версія	5 байт
Автор	5 байт
Коментар	100 байт
Кількість файлів	4 байта
Ознака наявності пароля	1 байт

Формат заголовка у файловому блоці

Таблиця 3.2

Назва атрибуту	Розмір
Ім'я файлу	255 байт
Шлях	255 байт
Розмір файлу до стиснення	4 байта
Розмір файлу після стиснення	4 байта
Циклічний надлишковий код	4 байта

Програма Archivator написана в середі швидкої розробки додатків Embarcadero RAD Studio XE8 на мові високого рівня Delphi і складається з шести модулів.

Модуль MainUnit.pas описує головне вікно програми, її інтерфейс і зв'язку з іншими модулями. Основна робота зі стиснення і розпакуванні файлів реалізована в модулі BasicZip.pas, де і здійснюється основна робота.

Модулі PasswordUnit.pas, AddingUnit.pas, EditCommentUnit.pas, ExtractingUnit.pas відповідають відповідно за внесення пароля, читання файлів з диска і їх запис, редагування коментаря, вибір архіву для розархівації.

3.3.1. Стиснення файлу і створення архіву

Механізм роботи архіватору заснований на створенні і обробці даних в потоках [54, 56]. Ядром архіватору є функції стиснення і розпаковування файлів. У програмі експортується клас TCustomZLibStream - нащадок класу TStream. Клас TCustomZLibStream в свою чергу є батьком для TCompressionStream і TdecompressionStream.

Перед викликом функції стиснення створюємо список файлів, що підлягають обробці (екземпляр класу TStringList):

```
FileList: = TStringList.Create;
for index: = 0 to DlgAddFiles.FileList.Items.Count-1 do begin
```

```
        FileForCompress:=DlgAddFiles.FileList.Items.Item[Index]. ubItems.String
        s [1] + '\' + DlgAddFiles.FileList.Items.Item [Index]. Caption;
        FileList.Add (FileForCompress);
 end
```

Вибираємо ступінь компресії:

```
case DlgAddFiles.cbCompression.ItemIndex of
        0: Level: = clNone;
        1: Level: = clFastest;
        2: Level: = clDefault;
        3: Level: = clMax;
 end
```

Виконуємо стиснення і шифрування всього списку файлів викликом функції CreateArchive:

```
if CreateArchive (FileList, DominantDir, Level, Comment, PassWord,
     PassProtect, ArchName) then InfoLBL.Caption: = 'Архів створений.'
else InfoLBL.Caption: = 'Архів не створено.'
```

Функція створення архіву CreateArchive приймає параметри Files_List - файловий список, Level - рівень стиснення, Comment: String - коментар, PassWord: String - пароль (ключ), PassProtect: Boolean - ознака захисту паролем (ключем), ArchiveName - ім'я архіву.

Послідовність роботи функції CreateArchive:

1. Запис даних у файловий потік File_Stream.

2. Виконання стиснення даних з поточного файлового потоку File_Stream.

3. Збереження результату в компресійному потоці (Compressed_Stream), викликом функції CompressStream.

4. Шифрування потоку Compressed_Stream викликом функції CryptStream (результат поміщаємо в Crypt_Stream).

5. Формування заголовка файлового блоку (по формату таб. 3.2).

6. Запис файлового заголовка і шифрованого файлу в архівний потік.

```
{---------------------- функція створення архіву -----------------------------}
```

```
function CreateArchive (Files_List: TStringList; / / файловий список
                    Level: TCompLevel; / / рівень стиснення
                    Comment: String; / / коментар
                    PassWord: String; / / пароль (ключ)
                    PassProtect: Boolean; / / ознака захисту паролем (ключем)
                    ArchiveName: String): Boolean;
var ArcHead: TArchiveHead; / / заголовок архіву
        FileHead: TFileHeader; / / заголовок файлу
        Archive: TMemoryStream; / / потік архівних даних
        File_Stream: TFileStream; / / потік файлових даних
        Crypt_Stream: TMemoryStream; / / потік шифрованих даних
        Compressed_Stream: TMemoryStream; / / потік стиснених даних
         Index_file: longint;
begin
  / / - формуємо заголовок архіву
  ArcHead.Signature: = signature;
  ArcHead.Author: = author;
  ArcHead.Version: = Version;
  ArcHead.PasswordProtect: = PassProtect;
  ArcHead.CountFiles: = Files_List.Count;
  ArcHead.Comment: = Comment;
  ArcHead.ArchPassWord: = PassWord;
  Archive: = TMemoryStream.Create; / / створюємо архівний потік
  Archive.Write (ArcHead, SizeOf (ArcHead)); / / запис заголовка
  / / Записуємо вибрані файли в архівний потік
  for Index_file: = 0 to ArcHead.CountFiles-1 do begin
     File_Stream: = TFileStream.Create (Files_List.Strings [Index_file],
fmOpenRead);
     Crypt_Stream: = TMemoryStream.Create;
     Compressed_Stream: = TMemoryStream.Create;
     Compressed_Stream.Seek (0, soFromBeginning);
```

```
// - виконуємо стиснення даних з поточного файлового потоку (File_Stream), і зберігаємо результат в компресійному потоці (Compressed_Stream)
    if not CompressStream (File_Stream, Compressed_Stream, Level) then begin
      Result: = False;
      Archive.Free;
      File_Stream.Free;
      Compressed_Stream.Free;
      Crypt_Stream.Free;
      Exit;
    end;
// - виконуємо шифрування потоку (Compressed_Stream) результат поміщаємо в Crypt_Stream
    CryptStream (Compressed_Stream, Crypt_Stream, PassWord, ArcHead.PasswordProtect, true);
    / / - формуємо заголовок файлу
    FileHead.FileName: = ExtractFileName (Files_List.Strings [Index_file]);
    FileHead.Pach: = ExtractFileDir (Files_List.Strings [Index_file]);
    FileHead.ReadOnly: = FileIsReadOnly (FileHead.FileName);
    FileHead.PackedSize: = Crypt_Stream.Size;
    FileHead.FileSize: = File_Stream.Size;
    Archive.Write (FileHead, SizeOf (TFileHeader)); / / запис файлового заголовка в архівний потік
    Archive.CopyFrom (Crypt_Stream, Crypt_Stream.Size); / / запис шифрованого потоку в архівний потік
    File_Stream.Free;
    Compressed_Stream.Free;
    Crypt_Stream.Free;
    ShowProgress (ArchiveProgress, Index_file, 0, ArcHead.CountFiles-1);
  end;
  Archive.SaveToFile (ArchiveName); / / вивантажуємо архівний потік в файл
  Result: = True;
```

```
    Archive.Free; / / очищаємо архівний потік
    ShowProgress (ArchiveProgress, 0,0,0);
  end;
  {---------------------- функція компресії потоку даних ---------------------- -}
  function CompressStream (InStream, OutStream: TStream; const Level: TCompLevel
= clDefault): boolean;
  var CompressionStream: TCompressionStream;
  begin
    InStream.Seek (0, soFromBeginning);
    OutStream.Seek (0, soFromBeginning);
    CompressionStream: = TCompressionStream.Create (TcompressionLevel
  (Level), OutStream);
    try
        CompressionStream.CopyFrom (InStream, InStream.Size);
        CompressionStream.Free;
        result: = true;
    except result: = false;
    end;
  end;
```

3.3.2. Шифрування й дешифрування файлу

В якості методу і засобу захисту інформації в архіві була задіяна поліалфавітная підстановка (шифр Вижинера) [33,34]. Для підвищення стійкості шифру, ключ (він же використовується як пароль до архіву) застосовується в заміні відповідних байт стисненого файлу не зберігається в тілі файлу архіву, а зберігається лише ознака наявності такого ключа в заголовку архіву (таб. 3.1).

```
{------------ функція шифровки / дешифровки потоку даних ------------------}
  Function CryptStream (InStream, OutStream: TStream;
                Password: string;
```

```
                    isCrypt, crypt: boolean): Boolean;
var Index: LongInt;
    PassLength, PassIndex, Buffer_Data: Byte;
    koef: shortInt;
begin
  InStream.Seek (0, soFromBeginning);
  OutStream.Seek (0, soFromBeginning);
  index: = 0;
  passindex: = 1;
  PassLength: = Length (Password);
  if (PassLength = 0) then begin / / якщо немає пароля-копіюємо потік InStream в
      OutStream
    OutStream.CopyFrom (InStream, InStream.Size);
    OutStream.Seek (0, soFromBeginning);
    Result: = false;
    Exit;
  end;
  if not isCrypt then begin
    OutStream.CopyFrom (InStream, InStream.Size);
    OutStream.Seek (0, soFromBeginning);
    Result: = true;
    Exit;
  end;
 if crypt then koef: = 1
  else koef: = -1;
  while index <> InStream.Size do begin
    InStream.Seek (Index, soFromBeginning);
    InStream.Read (Buffer_Data, 1);
    / / Шифруємо байт
   Buffer_Data: = Buffer_Data + koef * Byte (Password [Passindex]);
    if PassIndex <PassLength then Inc (PassIndex)
```

```
      else PassIndex: = 1;
      OutStream.Write (Buffer_Data, 1);
      Inc (Index);
    end;
    Result: = True;
    OutStream.Seek (0, soFromBeginning);
  end;
```

3.3.3. Розпакування архіву

Ініціалізували дії перед розпакуванням файлу аналогічні діям при архівуванні [54,56].

Перед викликом функції декомпресії створюємо рядкові списки імен і атрибутів файлів, упакованих в архіві:

```
Files: = TStringList.Create;
Sizes: = TStringList.Create;
PackSizes: = TStringList.Create;
ReadOnlys: = TStringList.Create;
Pachs: = TStringList.Create;
```

Потім отримуємо в рядкових списках імена й атрибути файлів, упакованих в архіві (якщо вибраний файл архів):

```
if (not ShowListFiles (DlgOpenArch.FileName, Author, Comment, Files,
Sizes, PackSizes, ReadOnlys, Pachs)) then begin
      Files.Free;
      Sizes.Free;
      PackSizes.Free;
     ReadOnlys.Free;
  Pachs.Free;
        Exit;
    end
    else if Comment <>'' then BtnEditComment.Enabled: = True
```

```
else BtnEditComment.Enabled: = False.
```

І заполненяется компоненти форми значеннями списків:

```
for Count: = 0 to Files.Count-1 do begin
FileList.Items.Add.Caption: = Files.Strings [Count];
FileList.Items.Item [Count]. SubItems.Add (PackSizes.Strings [Count]);
FileList.Items.Item [Count]. SubItems.Add (Sizes.Strings [Count]);
FileList.Items.Item [Count]. SubItems.Add (Format ('% 2.1f%%', [100 * (1 -
StrToInt (PackSizes.Strings [Count]) /
  StrToInt (Sizes.Strings [Count]))]));
  FileList.Items.Item [Count]. SubItems.Add (Pachs.Strings [Count]);
end.
```

При розархівування в разі наявності пароля проводиться перевірка пароля введеного користувачем. У разі невірного пароля - програма завершує свою роботу достроково (без розпакування).

```
if IsPassword (DlgOpenArch.FileName, ArchPassWord) then begin
   if not GetPassword (Self, PassWord, ArchPassWord) then begin
    OutList.Free;
    Exit;
   end;
  end;
```

При введенні вірного пароля проводиться вилучення файлів з архіву за допомогою виклику функції ExtractFiles:

```
if ExtractFiles (DlgOpenArch.FileName, OutList, PassWord, ExtractDir,
OverWrite) then InfoLBL.Caption: = 'Готово. Файли витягнуті. '
else InfoLBL.Caption: = 'Файли не витягнуто.';
{---------- функція витягання файлів з архіву ----------}
function ExtractFiles (FileName: string; Files: TStringList;
                PassWord: String; ExtractDir: String;
                        OverWrite: Boolean): Boolean;
var ArcHead: TArchiveHead;
    FileHead: TFileHeader;
```

```
            Archive: TFileStream;
            CryptFile, CompressFile, SaveStream: TMemoryStream;
            Count, p, FilePos: LongInt;
            b: byte;
        begin
          Archive: = TFileStream.Create (FileName, fmOpenRead);
          Archive.Seek (0, soFromBeginning);
          Archive.Read (ArcHead, sizeof (TArchiveHead));
          if (ArcHead.Signature <> Signature) then begin
               Archive.Free;
               ShowMsg ('Це не архів!', Wnd);
               Exit;
          end;
          if (ArcHead.Version <> Version) then begin
               Archive.Free;
               ShowMsg ('Невідома версія архіву!', Wnd);
               Exit;
          end;
          for count: = 0 to archead.CountFiles-1 do begin
Archive.Read (FileHead, sizeof (TFileHeader)); / / зчитуємо заголовок файлу
            if IsNameInList (FileHead.FileName, Files) then begin
               SaveStream: = TMemoryStream.Create;
               CompressFile: = TMemorystream.Create;
               CryptFile: = TMemoryStream.Create;
               SaveStream.CopyFrom (Archive, FileHead.PackedSize);
               CryptStream (SaveStream, CryptFile, PassWord, ArcHead.
                                   PasswordProtect, false);
               if not ExpandStream (CryptFile, CompressFile) then begin
ShowMsg ('Помилка декомпресії (помилка у файлі або невірний
           пароль)! ', Wnd);
                    Archive.Free;
```

```
                SaveStream.Free; CompressFile.Free; CryptFile.Free;
               Exit;
            end;
        if not FileExists (ExtractDir + '\' + FileHead.Pach + '\' + filehead.FileName)
            then CompressFile.SaveToFile (ExtractDir + filehead.FileName)
            else
            if (not OverWrite) then CompressFile.SaveToFile (ExtractDir +
filehead.FileName);
        FileSetReadOnly (ExtractDir + filehead.FileName,
                              filehead.ReadOnly);
        ShowProgress (ArchiveProgress, Count, 0, archead.CountFiles-1);
         SaveStream.Free;     CompressFile.Free;              CryptFile.Free;
          end
          else begin
            Archive.Seek (FileHead.PackedSize, soFromCurrent);
            ShowProgress (ArchiveProgress, Count, 0, ArcHead.CountFiles-1);
          end;
       end;
       Archive.Free;
       ShowProgress (ArchiveProgress, 0,0,0);
       Result: = true;
    end;
```

3.3.4 Порівняння результатів роботи програми

Розглянемо ступінь стиснення розробленого архіватору Archivator для різних файлів [46].

Для цього візьмемо файли наступних розширень: *. doc, *.exe, *.wav, *. xls, *.bmp, *.mht. Беремо довільні файли. Ступінь стиснення само собою буде залежати не тільки від типу файлу, а й від вмісту самого файлу. Тобто різних за розміром файлів типу *.doc буде різний коефіцієнт стиснення.

Ступінь стиснення вибраних файлів наведена на рис.3.14.

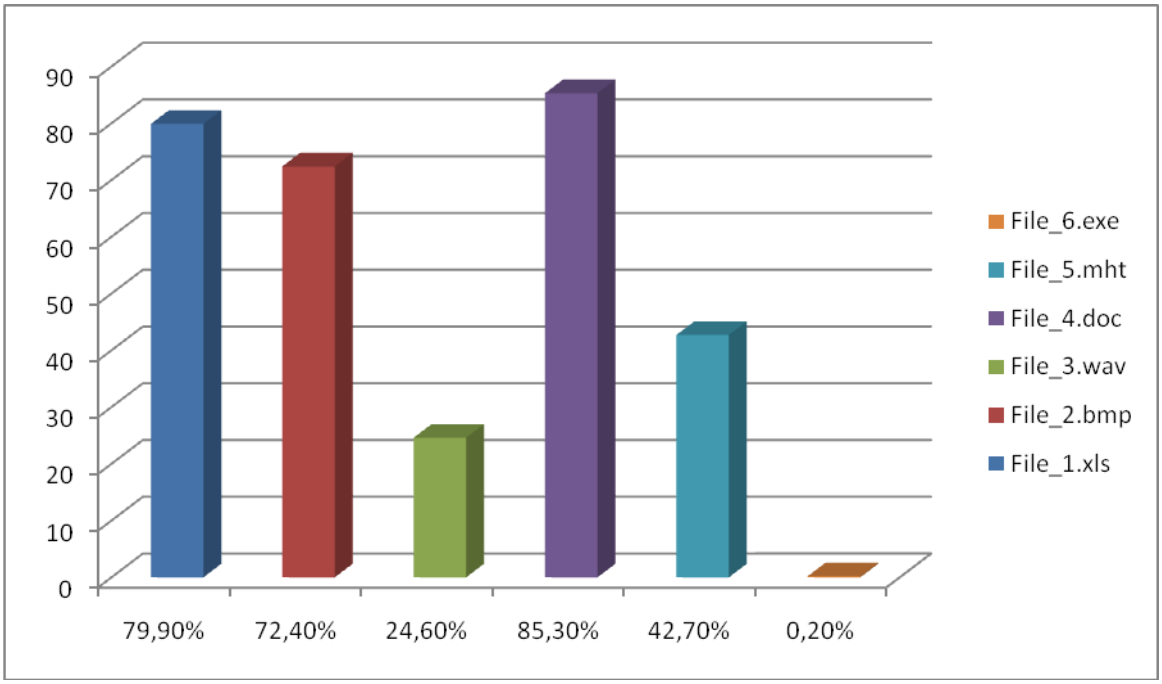

Рис. 3.14 - Результати стиснення за допомогою Archiver

Тепер проведемо порівняння ступенів стиску архівних файлів, створених архіватором Archiver та існуючих архіваторів WinRar і Zip.

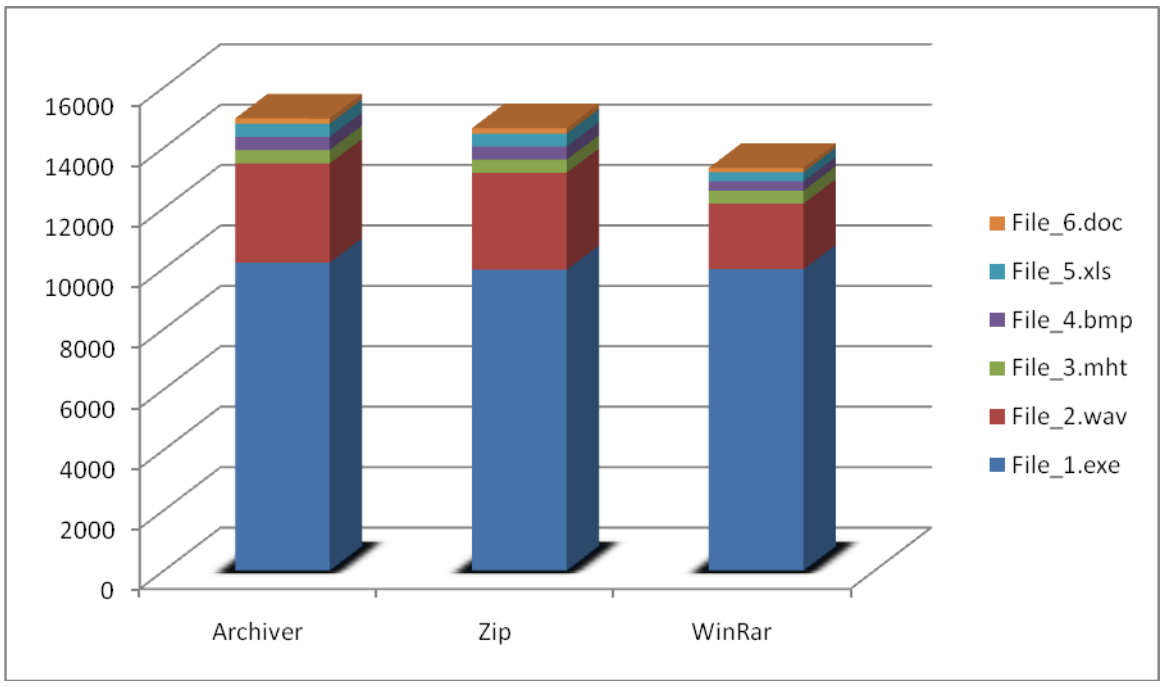

Рис. 3.15 - Стиснення даних різними архіваторами

Нижче зображені загальні розміри архівів (рис. 3.16).

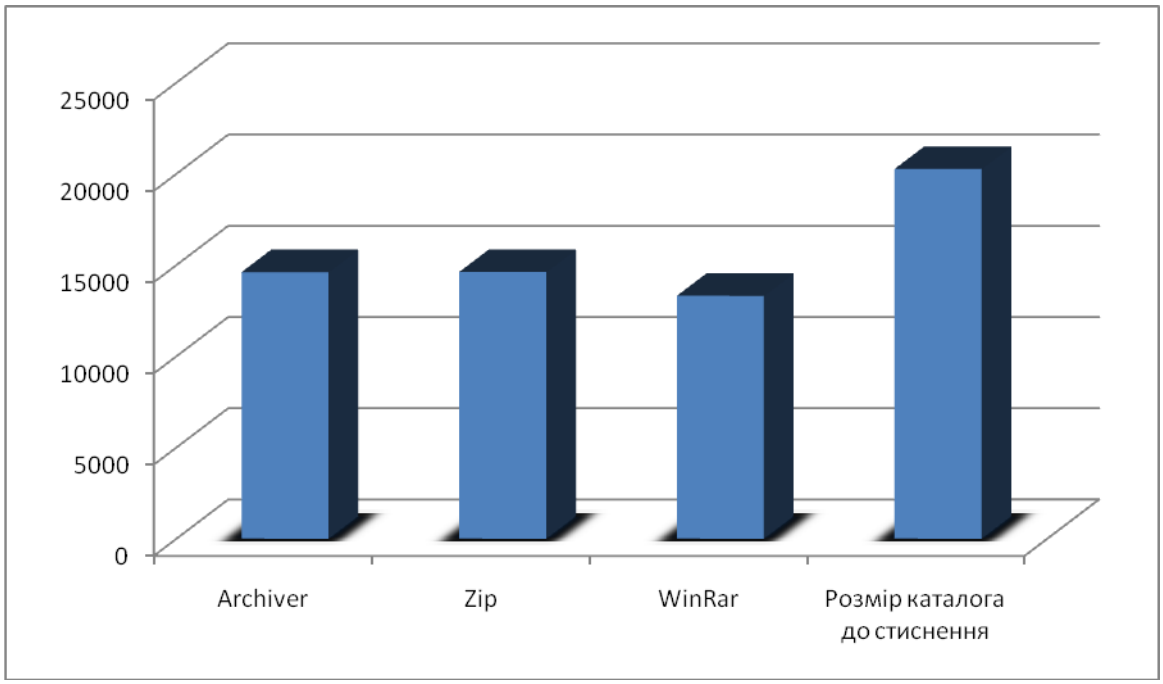

Рис. 3.16 - Загальний розмір архівів

Для rar і zip формату був виставлений параметр звичайного стиснення, який використовується і в програмі. Робота програми тестувалася на різних типах файлів. Використовувалися файли графічного, текстового, мультимедійного та інших форматів. Всі дані про процеси стиснення зображені на рис. 3.15.

Ці ж дані нижче зображені в таблиці 3.3.

Розмір стислих даних

Таблиця 3.3

	Розмір архіву
Початковий розмір каталогу з даними	20 278 Кб
WinRar	13 326 Кб
Zip	14 631 Кб
Archiver	14 622 Кб

З вище наведених даних можна зробити висновок, що коефіцієнт стиснення для WinRar склав близько 34.28%, для Zip - 27.84%, а для Archiver - 27.89%. Тобто архіватор Archiver в даному випадку спрацював краще, ніж Zip і трохи гірше, ніж Rar.

Також для подальшого поліпшення ефективності роботи програми можна реалізувати такі можливості:

1) збереження інформації про фото (дата створення / зміни, права доступу);

2) додавання багатопоточності в процесі архівування;

3) асоціювання архівних файлів з програмою;

4) використання в системах передачі інформації запропонованого механізму архівування та шифрування даних (з метою зменшення трафіку в мережі та забезпечення захищеності даних).

ВИСНОВКИ

У ході виконання дипломної роботи був розроблений метод шифрування даних, що надав можливість використання архівації з паролем. Це ще в більшій мірі дозволило захищати інформацію від ворожих атак та варіантів несанкціонованого використання.

В процесі реалізації були досліджені алгоритми стиснення з мінімальною надлишковістю та алгоритм стиснення із застосуванням словника. В якості вихідних даних були використані файли графічного, текстового, мультимедійного та інших форматів.

Також був розроблений архіватор, який уособлює в собі практичне використання алгоритмів стиснення інформації без втрат даних.

Основними результатами роботи є**:**

a) аналіз вибору типу стиснення даних;
b) дослідження алгоритмів кодування з мінімальною надмірністю;
c) порівняльні дослідження кодування Шеннона-Фано, Хаффмана і Лемпеля - Зіва;
d) можливість компресії файлів і каталогів без втрат;
e) збирання архіву без стиснення (або стискати з різним рівнем компресії);
f) шифрування даних;
g) можливість дописувати коментар до архіву;
h) розрахування коефіцієнта стиснення файлів і циклічного надлишкового коду.

Результати, отримані в роботі, показують перспективність вивчення даної проблематики, безсумнівну корисність використання методів стиснення й шифрування, які збільшують можливість захисту інформації в порівнянні з випадками простого збереження даних. Розроблена програма архівування не лише досягла поставленої мети, а й показала результати конкуруючі з результатами провідних архіваторів, таких як WinRar і Zip.

СПИСОК ЛІТЕРАТУРИ

1. Алгоритм Лемпеля - Зіва [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_36.shtml

2. Алгоритм Хаффмана [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_35.shtml

3. Архангельський А. Я. Прийоми програмування в Delphi на основі VCL / А. Я. Архангельський. - М.: «Біном-Пресс», 2006 р. - 944 с.

4. Архівація [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_33.shtml

5. Ахо Альфред В. Структури даних і алгоритми/Альфред В. Ахо, Джон Е. Хопкрофта, Джеффрі Д. Ульман.-М.: Видавничий дім "Вільямс", 2000.- 384 с.

6. Бердишев В. І., Апроксимація функцій, стиснення чисельної інформації, додатки / В. І. Бердишев, Л. В. Петрак. - Єкатеринбург: УрВ ран, 1999. - 296 с.

7. Березюк Н. Т. Кодування інформації (двійкові коди) / Н. Т. Березюк, А. Г. Андрущенко, С. С. Мощіцкій и др. - Харків: видавнича об'єднання "Вища школа", 1978. - 252 с.

8. Берлекемп Е. Алгебраическая теория кодирования / Е. Берлекемп. - М.: Мир, 1971. - 478 с.

9. Бондаренко В. А. Фрактальное стиснення зображень по Барнслі-Слоану / В. А. Бондаренко, В. Л.Дольніков. / / Автоматика і телемеханіка. - 1994. - № 5. - С.12-20.

10. Ватолін Д. С. Алгоритмы сжатия изображений. Методичний посібник / / Д. С. Ватолін. - М.: Видавничий відділ факультету Обчислювальної Математики і Кібернетики МГУ ім. М.В.Ломоносова, 1999. - 76 с.

11. Вернер М. Основи кодування /М.Вернер. - М.: Техносфера, 2004. - 288 с.

12. Тимчасові санітарні норми і правила для працівників обчислювальних центрів № 4559-88 (ВСНиПРВЦ) / Міністерство охорони здоров'я СРСР.

13. Гайдишев І. Аналіз та обробка даних / І. Гайдишев. - Спб: Питер, 2001. - 750 с.

14. Глушаков С. В. Робота в мережі Internet / С. В. Глушаков, Д. В. Ломотько- [2-е вид., Доп. і перераб]. - Харків: Фоліо, 2003. - 399 с.

15. Голуб Н. Г. Мистецтво програмування на Асемблері: Лекції і вправи / Н. Г. Голуб. - [2-е изд., Испр. і доп]. - СПб. : ДіаСофтЮП, 2002. - 644 с.

16. Гришина Н. В. Організація комплексної системи захисту інформації / Н. В. Гришина. - М.: Геліос АРВ, 2007. - 256 с.

17. ДСанПіН 3.3.2.007-1998. Державні санітарні правила и норми роботи з візуальнімі дисплейними терміналамі електронно-обчислювальних машин.

18. Духін А. А. Теорія інформації /А.А. Духін. - М.: Геліос АРВ, 2007.-248 с.

19. Жоголєв Е. А. Ж.78 Технологія програмування / Євген Андрійович Жоголєв. - М.: Науковий Світ, 2004. - 216 с.

20. Захаров Максим. Короткий опис методу фрактального стиснення [Електронний ресурс] / Максим Захаров. - Режим доступу: http://sochi.org.ru/cgi-bin/ht2/ht2-cgi.cgi?=cinfo $ FRACOMP

21. Зелінський Сергій. У цифрових лещатах. [Електронний ресурс] / Сергій Зелінський. - Режим доступу: http://bessarab.ucoz.ru/publ/fizikam/szhatie_dannykh/2-1-0-6

22. Касілов О. В. Деякі питання стиснення даних [Електронний ресурс] / О.В.Касілов, В.А.Кравец. - Режим доступу: http://users.kpi.kharkov.ua/lre/MicroCAD/sekcia13/Oleg_97.htm

23. Касперски К. Мистецтво дизассемблирования / Кріс Касперски, Єва Рокко. - БХВ-Петербург, 2008. - 780 с.

24. Ключар А. А., Матьяш В. А., Щокін С. В. Структури та алгоритми обробки даних: навч. посібник / А. А. Ключар, В. А. Матьяш, С. В. Щокін. - СПб. : ГУАП. СПб., 2003. - 172 с.

25. Колесник В. Д. Курс теорії інформації / В. Д. Колесник, Г. Ш.Полтирев. - М.: Наука, 1982. - 416 с.

26. Лідовскій В. В. Теорія інформації: Навчальний посібник / В. В. Лідовскій. - М.: Компанія Супутник +, 2004. - 111 с.

27. Мельников В. П. Інформаційна безпека та захист інформації: навчальний посібник для студ. вищ. навч. закладів / В. П. Мельников, С. А. Клейменов, А. М. Петраков; под. ред. С. А. Клейменова. - 3-е изд., Стер. - М.: Видавничий центр «Академія», 2008. - 336 с.

28. Методи стиснення даних. Пристрій архіваторів, стиснення зображень і відео/Д. Ватолін, А. Ратушняк, М. Смирнов, В. Юкін. - М.: Діалог-МИФИ, 2003. - 384 с.

29. Міано Дж. Формати й алгоритми стиску зображень у дії / Дж. Міано. - М.: Изд-во Тріумф, 2003. - 336 с.

30. Морелос-Сарагоса Р. Мистецтво завадостійкого кодування. Методи, алгортми, застосування / Р. Морелос-Сарагоса. - М.: Техносфера, 2005.-320 с.

31. Панасенко С. Алгоритм шифрування SAFER + / С. Панасенко, А. Удовицький / / Банки та технології. - 2004. - № 2. - С. 60-64.

32. Панасенко С. П. Алгоритми шифрування. Спеціальний довідник / С. П. Панасенко. - СПб. : БХВ-Петербург, 2009. - 576 с.

33. Панасенко С. П. Сучасні алгоритми шифрування / С. П. Панасенко / / BYTE. - 2003. - № 8. - С. 18-22.

34. Петров А. А. Комп'ютерна безпека: криптографічні методи захисту / А. А. Петров. - М.: ДМК, 200. - 448 с.

35. Правила охорони праці при експлуатації електронно-обчислювальних машин. № 382/3675 від 17.06.1999.

36. Принципи архівації [Електронний ресурс]. - Режим доступу: http://www.citforum.urbannet.ru/internet/infsecure/its2000_34.shtml

37. Протоколи стиснення даних [Електронний ресурс]. - Режим доступу: http://www.vicgain.ru/modems/8.htm

38. Пишкін Е. В. Структури даних і алгоритми: реалізація на С / С + +. Навчальний посібник / Є. В. Пишкін. - Санкт-Петербург: ФТК СПб ГПУ, 2009р, 200 с.

39. Романець Ю. В. Захист інформації в комп'ютерних системах і мережах / Ю. В. Романець, П. А. Тимофєєв, В. Ф. Шаньгина. - М.: Радіо і зв'язок, 2001. - 376 с.

40. Семенюк В. В. Економне кодування дискретної інформації / В. В. Семенюк. - СПб.: СПбГІТМО (ТУ), 2001. - 115 с.

41. Соколов А. В. Захист інформації в розподілених корпоративних мережах і системах / А. В. Соколов, В. Ф. Шаньгина. - М.: ДМК Пресс, 2002. - 656 с.

42. Селомон Д. Стиснення даних, зображення і звуку / Д. Селомон. - М.: Техносфера, 2004р. - 368 с.

43. Теорія прикладного кодування: Учеб. Посібник / В. К.Конопелько, В. А. Ліпнінскій, В. Д. Дворніков та ін - Мн.: БДУІР, 2004. - 398 с

44. Уелстід С. Фрактали і вейвлети для стиснення зображень в дії / С. Уелстід. - М.: Тріумф, 2003. - 320 с.

45. Фігурне В. Е. IBM PC для користувача. Короткий курс / Віктор Евальдовіч Фігурне. - М.: ИНФРА-М, 1998. - 480 с.

46. Формати стиснення даних [Електронний ресурс]. - Режим доступу: http://www.russianelectronics.ru/leader-r/review/8602/doc/46598/

47. Фундаментальні алгоритми на С + +. Аналіз / Структури даних / Сортування / Пошук / [Пер. з англ.Роберт Седжвік]. - К.: Видавництво «ДиаСофт», 2001. - 688 с.

48. Фундаментальні алгоритми з структури даних в Delphi / [пер. з англ. Джуліан М. Бакнел]. - СПб: ТОВ «ДіаСофтЮП», 2003. - 560 с.

49. Чернега В. С. Стиснення інформації в комп'ютерних мережах. Навчальний посібник для вузів / В. С. Чернега. - Севастополь: Вид-во СевГТУ, 1997. - 214 с.

50. Щупак Ю. Win32 API. Ефективна розробка додатків. / Юрій Щупак. - СПб.: Питер, 2007. - 572 с.

51. Witten I. H. Arithmetic Coding for Data Compression / I. H. Witten, R. M. Neal, J. G. Cleary / / - Commun. ACM 30. - 1987. - № 6. - P. 520-540.

52. Ziv J. A universal algorithm for sequential data compression / J. Ziv, A. Lempel / / IEEE Transactions on Information Theory Vol. IT-23. - May 1977. - N.3. -P. 337-343.

53. Ziv J. Compression of individual sequences via variable rate coding / J. Ziv, A. Lempel / / IEEE Transactions on Information Theory. Vol. IT-24. - September 1978. - N.5. - P. 530-535.

54. Burrows M. A block-sorting Lossless Data Compression Algorithm / M. Burrows, D. J. Wheeler / / Digital Systems Research Center. SRC report 124. - May 10, 1994.

55. Lelewer Debra A. Data Compression / Debra A. Lelewer, Daniel S. Hirschberg [Електронний ресурс]. - Режим доступу: http://www.ics.uci.edu/ ~ dan / pubs / DataCompression.html

56. Bently J. L. A locally adaptive data compression algorithm / J. L. Bently, D. D. Sleator, R. E. Tarjan, V. K. Wei / / Communications of the ACM, Vol. 29. - April 1986. - No. 4. - P. 320-330.

Додаток А

Лістинг програми архіватору Archiver

Файл MainUnit

```
unit MainUnit;


interface

uses  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,  Dialogs, ComCtrls, ToolWin, BasicZip, StdCtrls, ImgList,
ExtCtrls,XpMan,  Menus;

type

  TMainForm = class(TForm)

    DlgOpenArch: TOpenDialog;   DlgSaveArch: TSaveDialog;

    WndPanel: TPanel;    StatusBar: TPanel;

    Bevel2: TBevel;    Bevel4: TBevel;    Bevel5: TBevel;

    Bevel6: TBevel;   Bevel7: TBevel;    InfoLBL: TLabel;

    Bevel8: TBevel;    GeneralProgress: TProgressBar;

    FileList: TListView;    Tool: TToolBar;    BtnNew: TToolButton;

    BtnOpen: TToolButton;    BtnExtract: TToolButton;

    BtnEditComment: TToolButton;   Bevel1: TBevel;    Panel1: TPanel;

    Bevel3: TBevel;    Label1: TLabel;    ActionMenu: TPopupMenu;

    Selectall1: TMenuItem;    N1: TMenuItem;    Extractselected1: TMenuItem;

    Image1: TImage;    Image_Key: TImage;    Image_Lock: TImage;

    Label_size: TLabel;    BigImg: TImageList;

    procedure BtnOpenClick(Sender: TObject);

    procedure BtnNewClick(Sender: TObject);

    procedure BtnExtractClick(Sender: TObject);

    procedure BtnEditCommentClick(Sender: TObject);

    procedure Selectall1Click(Sender: TObject);

    procedure Image1DblClick(Sender: TObject);

  end;


var

  MainForm: TMainForm;  Comment:String;  OverWrite:Boolean;
```

```
implementation
uses AddingUnit, ExtractingUnit, EditCommentUnit,PassWordUnit;
{$R *.dfm}
procedure TMainForm.BtnOpenClick(Sender: TObject);
var Author, ArchPassWord : String;
    Files, Sizes, PackSizes, ReadOnlys, Pachs, CRC : TStringList;
    Count, AllSizes, AllPackSizes : LongInt;
begin
  try
    InfoLBL.Caption:='Выбор архива.';   Application.ProcessMessages;
    if not DlgOpenArch.Execute then begin
      InfoLBL.Caption:='Готов.';  Application.ProcessMessages;
      Exit;
    end;
    FileList.Clear;
// создание строчных списков имен и атрибутов файлов, упакованных в архиве
    Files:=TStringList.Create;   CRC:=TStringList.Create;
    Sizes:=TStringList.Create;   PackSizes:=TStringList.Create;
    ReadOnlys:=TStringList.Create;  Pachs:=TStringList.Create;
    Wnd:=Handle;   Caption:='ОТКРЫТ АРХИВ: '+DlgOpenArch.FileName;
// получаем в строчных списках имена и атрибуты файлов, упакованных в архиве
(если выбранный файл архив)
if (not
ShowListFiles(DlgOpenArch.FileName,Author,Comment,Files,CRC,Sizes,PackSizes,Rea
dOnlys,Pachs)) then begin
       Files.Free; CRC.Free; Sizes.Free; PackSizes.Free;
       ReadOnlys.Free; Pachs.Free;  Exit;
    end
    else if Comment<>'' then BtnEditComment.Enabled:=True
         else BtnEditComment.Enabled:=False;

    // заполнение компонента значениями списков
    for Count:=0 to Files.Count-1 do begin
        FileList.Items.Add.Caption:=Files.Strings[Count];
```

```
        FileList.Items.Item[Count].SubItems.Add(CRC.Strings[Count]);
        FileList.Items.Item[Count].SubItems.Add(PackSizes.Strings[Count]);
        FileList.Items.Item[Count].SubItems.Add(Sizes.Strings[Count]);
        FileList.Items.Item[Count].SubItems.Add(Format('%2.1f %%',[100*(1-
StrToInt(PackSizes.Strings[Count])/

StrToInt(Sizes.Strings[Count]))]));
        FileList.Items.Item[Count].SubItems.Add(Pachs.Strings[Count]);
        Inc(AllSizes,StrToInt(Sizes.Strings[Count]));
        Inc(AllPackSizes,StrToInt(PackSizes.Strings[Count]));
    end;
    if not IsPassword(DlgOpenArch.FileName,ArchPassWord)then begin
       Image_key.Visible:=True; Image_Lock.Visible:=False;
    end
    else  begin
       Image_key.Visible:=False; Image_Lock.Visible:=True;
    end;

Label_size.Caption:='       Общий размер файлов : '+IntToStr(AllSizes)+ '
байт'+ #13+#10+
                         '       Размер сжатых файлов :
'+IntToStr(AllPackSizes)+ ' байт'+ #13+#10+
                         '       Коэффициет компрессии : '+Format('%2.1f
%%',[100*(1-AllPackSizes/AllSizes)]);
    Files.Free;  CRC.Free;  Sizes.Free; PackSizes.Free;
    ReadOnlys.Free; Pachs.Free; BtnExtract.Enabled:=True;
  except
  end
end;

procedure TMainForm.BtnNewClick(Sender: TObject);
var FileList:TStringList;    Index:LongInt;    FileForCompress:String;
    PassWord:String;   PassProtect:Boolean;   Level:TCompLevel;
    ArchName:String;   DominantDir:String;
begin
```

```
  InfoLBL.Caption:='Работа с данными.';
  Application.ProcessMessages;
  Caption:='Архиватор';
  if not DlgSaveArch.Execute then begin
     InfoLBL.Caption:='Выберите действие.';   Exit;
  end;
  if DlgAddFiles.ShowModal=mrOk then begin
      PassWord:=DlgAddFiles.PasswordEdit.Text;
      PassProtect:=DlgAddFiles.cbUsePassword.Checked;
      FileList:=TStringList.Create;
      //--формируем (копируем) список файлов
      for index:=0 to DlgAddFiles.FileList.Items.Count-1 do begin

FileForCompress:=DlgAddFiles.FileList.Items.Item[Index].SubItems.Strings[1]+
'\'+DlgAddFiles.FileList.Items.Item[Index].Caption;
       FileList.Add(FileForCompress);
    end;
      //--выбор степени компрессии
      case DlgAddFiles.cbCompression.ItemIndex of
      0:Level:=clNone;      1:Level:=clFastest;
      2:Level:=clDefault;   3:Level:=clMax;
      end;
//--при необходимости заменяем расширение архива
      if (UpperCase(ExtractFileExt(DlgSaveArch.FileName))<>'.sergey) then
ArchName:=DlgSaveArch.FileName+'.sergey;
      InfoLBL.Caption:='Загрузка архива, подождите.';
      Application.ProcessMessages;
      //--выполняем сжатие и шифрование всего списка файлов
      if
CreateArchive(FileList,DominantDir,Level,Comment,PassWord,PassProtect,ArchName)
then
           InfoLBL.Caption:='Архив создан.'
      else InfoLBL.Caption:='Архив не создан.';
      FileList.Free;
  end
```

```
  else InfoLBL.Caption:='Выберите действие.';  DlgAddFiles.Free;
end;
procedure TMainForm.BtnExtractClick(Sender: TObject);
var OutList:TStringList;
    Index:LongInt;  PassWord, ArchPassWord:String; ExtractDir:String;
begin
  if FileList.SelCount=0 then Exit;
  Application.CreateForm(TDlgExtractFiles, DlgExtractFiles);
  DlgExtractFiles.DirsTree.Root:=PatchDirEXE(Application.ExeName);
  if DlgExtractFiles.ShowModal=mrOk then  begin
    Application.ProcessMessages;
    InfoLBL.Caption:='Работа с данными.';
    Application.ProcessMessages;
    Wnd:=Handle;
    PassWord:='';
    ExtractDir:=DlgExtractFiles.ExtractDirEdit.Text;
  if DlgExtractFiles.OverWriteBox.ItemIndex=0 then OverWrite:=true
  else OverWrite:=False;
if ExtractDir[Length(ExtractDir)]<>'\' then ExtractDir:=ExtractDir+'\';
    OutList:=TStringList.Create;
    for index:=0 to FileList.Items.Count-1 do begin
      if FileList.Items.Item[index].Selected then
OutList.Add(FileList.Items.Item[Index].Caption);
    end;
    // проверка пароля (если он есть)
    if IsPassword(DlgOpenArch.FileName,ArchPassWord) then begin
        if not GetPassword(Self, PassWord, ArchPassWord) then  begin
          OutList.Free;
          Exit;
        end;
    end;
    InfoLbl.Caption:='Извлечение, подождите.';
    Application.ProcessMessages;
```

```
    // извлечение файлов из архива
    if ExtractFiles(DlgOpenArch.FileName,OutList,PassWord,ExtractDir,OverWrite)
then
       InfoLBL.Caption:='Готово. Файлы извлечены.'
    else InfoLBL.Caption:='Файлы не извлечены.';
    OutList.Free;
  end;
  DlgExtractFiles.Free; BtnExtract.Enabled:=True;
end;

procedure TMainForm.BtnEditCommentClick(Sender: TObject);
begin
  Application.CreateForm(TDlgEditComment, DlgEditComment);
  DlgEditComment.Comment.Text:=Comment;
  if DlgEditComment.ShowModal= mrOk then  Comment:=DlgEditComment.Comment.Text;
  DlgEditComment.Free;
end;
end.
```

Додаток Б

Файл BasicZip

```
unit BasicZip;

interface

uses ZLib, Classes, Windows, SysUtils, Controls, ComCtrls, forms, ZipMessages,
Dialogs, U_CRC;

type

  TCompLevel = (clNone, clFastest, clDefault, clMax);

  {----архивный заголовок---}

  TArchiveHead=packed record

    Signature:array[0..6] of char;

    Version:String[4];    Author:array[0..4] of char;

    Comment:string[100];  CountFiles:longint;

    PasswordProtect:Boolean;   ArchPassWord:ShortString;

  end;

  {----файловый заголовок---}

  TFileHeader=packed record

    FileName:ShortString;  CRC:LongWord; Pach:ShortString;

    ReadOnly:Boolean;   FileSize:longint;  PackedSize:LongInt;

  end;

  {---функция создания архива-----------}

  function CreateArchive(Files_List:TStringList;

    DominantDir:String; Level:TCompLevel; Comment:String;

    PassWord:String; PassProtect:Boolean; ArchiveName:String):Boolean;

  {---функция извлечения файлов из архива----}

  function ExtractFiles(FileName:string;

    Files:TStringList; PassWord:String; ExtractDir:String;

                        OverWrite:Boolean):Boolean;

  {--функция получения всех файловых атрибутов  из архива в списке--}

  function ShowListFiles(ArchiveFileName:String; Var Author,Comment:string;
FileList,CRC, SizeList,PSizeList,ReadOnlyList,PatchsList:TStringList):boolean;

  {---функция получения признака наличия у архива пароля-----------}

  function IsPassword(FileName:string;
```

```
                    var ArchPassWord:string):boolean;
  {---функция получения текущей директории---------------}
  function PatchEXE(pach:string):string;
  {---функция получения текущего диска-------------------}
  function PatchDirEXE(pach:string):string;
  {---отображение выполнения операции--------------------}
  procedure ShowProgress(Bar:TProgressBar;
                         Value,Min,Max:LongInt);
  {---проверка вхождения файла в список------------------}
  function IsNameInList(Name:String;
                        List:TStringList):Boolean;
  Procedure Go(Form:TForm);
var
  ArchiveProgress:TProgressBar;  Wnd:hWnd;
implementation
const
  signature='ARCHIVE';
  Author='SERGEY';
  Version='1.1';
{--------функция шифровки/дешифровки потока данных-------------------}
Function CryptStream(InStream,OutStream:TStream;
                     Password:string;
                     isCrypt, crypt : boolean):Boolean;
var Index:LongInt;
    PassLength, PassIndex, Buffer_Data:Byte;
    koef:shortInt;
begin
  InStream.Seek(0,soFromBeginning);
  OutStream.Seek(0,soFromBeginning);
  index:=0;
  passindex:=1;
  PassLength:=Length(Password);
```

```
  if (PassLength=0)then begin //если нет пароля-копируем поток InStream в
OutStream
      OutStream.CopyFrom(InStream,InStream.Size);
      OutStream.Seek(0,soFromBeginning);
      Result:=false;
      Exit;
  end;
  if not isCrypt then begin
      OutStream.CopyFrom(InStream,InStream.Size);
      OutStream.Seek(0,soFromBeginning);
      Result:=true;
      Exit;
  end;
  if crypt then koef:=1
  else koef:=-1;
  while index<>InStream.Size do  begin
      InStream.Seek(Index,soFromBeginning);
      InStream.Read(Buffer_Data,1);
      // шифруем байт
      Buffer_Data:=Buffer_Data + koef*Byte(Password[Passindex]);
      if PassIndex < PassLength then Inc(PassIndex)
      else PassIndex:=1;
      OutStream.Write(Buffer_Data,1);
      Inc(Index);
  end;
  Result:=True;
  OutStream.Seek(0,soFromBeginning);
end;
{-------------функция компрессии потока данных----------------------}
function CompressStream(InStream, OutStream :TStream; const Level : TCompLevel
= clDefault):boolean;
var CompressionStream:TCompressionStream;
begin
  InStream.Seek(0,soFromBeginning);
```

```
  OutStream.Seek(0,soFromBeginning);
  CompressionStream:=TCompressionStream.Create(TCompressionLevel(Level),
OutStream);
  try
    CompressionStream.CopyFrom(InStream, InStream.Size);
    CompressionStream.Free;   result:=true;
  except  result:=false;
  end;
end;
 {-------------функция декомпрессии потока данных--------------------}
function ExpandStream(InStream, OutStream :TStream):boolean;
var Count : integer;
    DecompressionStream: TDecompressionStream;
    Buffer: Byte;
begin
  InStream.Seek(0,soFromBeginning);
  OutStream.Seek(0,soFromBeginning);
  DecompressionStream:=TDecompressionStream.Create(InStream);
  Try
    repeat
      Count:=DecompressionStream.Read(Buffer, sizeof(buffer));
      if Count<>0 then OutStream.Write(Buffer, Count);
    until Count=0;
    Result:=True;
  Except
    Result:=False;  DecompressionStream.Free;
  End;
  DecompressionStream.Free;
end;
{-------------функция создания архива-------------------------------}
function CreateArchive(Files_List:TStringList; // файловый список
                       DominantDir:String;     //
                       Level:TCompLevel;       // уровень сжатия
```

```
                        Comment:String;         // комментарий
                        PassWord:String;        // пароль (ключ)
                        PassProtect:Boolean;    // признак защиты паролем
(ключем)
                        ArchiveName:String):Boolean;
var ArcHead  : TArchiveHead;  // заголовок архива
    FileHead : TFileHeader;   // заголовок файла
    Archive           : TMemoryStream; // поток архивных данных
    File_Stream       : TFileStream;   // поток файловых данных
    Crypt_Stream      : TMemoryStream; // поток шифрованных данных
    Compressed_Stream : TMemoryStream; // поток сжатых данных
    Index_file:longint;
begin
  //--формируем заголовок архива
  ArcHead.Signature       := signature;
  ArcHead.Author          := author;
  ArcHead.Version         := Version;
  ArcHead.PasswordProtect := PassProtect;
  ArcHead.CountFiles      := Files_List.Count;
  ArcHead.Comment         := Comment;
  ArcHead.ArchPassWord    := PassWord;
  Archive := TMemoryStream.Create; // создаем архивный поток
  Archive.Write(ArcHead,SizeOf(ArcHead)); //запись заголовка
  // записываем выбранные файлы в архивный поток
  for Index_file:=0 to ArcHead.CountFiles-1 do  begin
      File_Stream :=
TFileStream.Create(Files_List.Strings[Index_file],fmOpenRead);
      Crypt_Stream := TMemoryStream.Create;
      Compressed_Stream := TMemoryStream.Create;
      Compressed_Stream.Seek(0,soFromBeginning);
//--выполняем сжатие данных из текущего файлового потока (File_Stream),
// и сохраняем результат в компрессионном потоке (Compressed_Stream)
      if not CompressStream(File_Stream,Compressed_Stream,Level) then begin
          Result := False;
```

```
          Archive.Free;
          File_Stream.Free;
          Compressed_Stream.Free;
          Crypt_Stream.Free;
          Exit;
      end;
      //--выполняем шифрование потока (Compressed_Stream)
      //          результат помещаем в Crypt_Stream
      CryptStream(Compressed_Stream, Crypt_Stream, PassWord,
ArcHead.PasswordProtect, true);
      //--формируем заголовок файла
      FileHead.FileName := ExtractFileName(Files_List.Strings[Index_file]);
      FileHead.CRC := CalcFileCRC32(File_Stream);
      FileHead.Pach := ExtractFileDir(Files_List.Strings[Index_file]);
      FileHead.ReadOnly := FileIsReadOnly(FileHead.FileName);
      FileHead.PackedSize := Crypt_Stream.Size;
      FileHead.FileSize   := File_Stream.Size;
      Archive.Write(FileHead,SizeOf(TFileHeader));       // запись файлового
заголовка в архивный поток
      Archive.CopyFrom(Crypt_Stream, Crypt_Stream.Size); // запись шифрованного
потока в архивный поток
      File_Stream.Free;
      Compressed_Stream.Free;
      Crypt_Stream.Free;
  ShowProgress(ArchiveProgress, Index_file, 0, ArcHead.CountFiles-1);
  end;
  Archive.SaveToFile(ArchiveName); //выгружаем архивный поток в файл
  Result := True;
  Archive.Free;  // очищаем архивный поток
  ShowProgress(ArchiveProgress,0,0,0);
end;
{-----------функция извлечения файлов из архива----------------------}
function ExtractFiles(FileName:string;Files:TStringList;
                      PassWord:String;ExtractDir:String;
```

```
                    OverWrite:Boolean):Boolean;
var ArcHead  : TArchiveHead;
    FileHead : TFileHeader;
    Archive  : TFileStream;
    CryptFile, CompressFile, SaveStream : TMemoryStream;
    Count, p, FilePos:LongInt;  b:byte;
begin
  Archive:=TFileStream.Create(FileName,fmOpenRead);
  Archive.Seek(0,soFromBeginning);
  Archive.Read(ArcHead,sizeof(TArchiveHead));
  if (ArcHead.Signature<>Signature) then begin
      Archive.Free; ShowMsg('Это не архив!!!',Wnd);   Exit;
  end;
  if (ArcHead.Version<>Version) then begin
      Archive.Free; ShowMsg('Неизвестная версия архива!!!',Wnd);
      Exit;
  end;
  for count:=0 to archead.CountFiles-1 do begin
      Archive.Read(FileHead, sizeof(TFileHeader)); // считываем заголовок файла
      if IsNameInList(FileHead.FileName,Files) then begin
          SaveStream   := TMemoryStream.Create;
          CompressFile := TMemorystream.Create;
          CryptFile    := TMemoryStream.Create;
          {for p:=0 to FileHead.PackedSize-1 do begin
              Archive.Read(b,1);     SaveStream.Write(b,1);
              ShowProgress(ArchiveProgress,p,0,filehead.PackedSize);
          end;}
          SaveStream.CopyFrom(Archive,FileHead.PackedSize);
          CryptStream(SaveStream, CryptFile, PassWord, ArcHead.PasswordProtect,
false);
          if not ExpandStream(CryptFile,CompressFile) then begin
              ShowMsg('Ошибка декомпрессии (ошибка в файле или неверен
пароль)!!!',Wnd);
```

```
              Archive.Free;  SaveStream.Free;   CompressFile.Free;
CryptFile.Free;

              Exit;

          end;

          if not FileExists(ExtractDir+'\'+FileHead.Pach+'\'+filehead.FileName)

          then CompressFile.SaveToFile(ExtractDir+filehead.FileName)

          else if (not OverWrite)then
CompressFile.SaveToFile(ExtractDir+filehead.FileName);
FileSetReadOnly(ExtractDir+filehead.FileName,filehead.ReadOnly);

          ShowProgress(ArchiveProgress,Count,0,archead.CountFiles-1);

          SaveStream.Free;   CompressFile.Free;   CryptFile.Free;

      end

      else begin

          Archive.Seek(FileHead.PackedSize,soFromCurrent);

          ShowProgress(ArchiveProgress,Count,0,ArcHead.CountFiles-1);

      end;

  end;

  Archive.Free; ShowProgress(ArchiveProgress,0,0,0);

  Result:=true;

end;

{функция получения всех файловых атрибутов  из архива в списки------}

function ShowListFiles(ArchiveFileName:String;

                       Var Author,Comment:string;

                       FileList, CRC, SizeList, PSizeList, ReadOnlyList,
PatchsList : TStringList):boolean;

var ArcHead:TArchiveHead; FileHead:TFileHeader;  Archive:TFileStream;

    Count:longint;

begin

  FileList.Clear;  CRC.Clear;  SizeList.Clear;

  PSizeList.Clear;  ReadOnlyList.Clear;

  Archive:=TFileStream.Create(ArchiveFileName,fmOpenRead);

  Archive.Read(ArcHead,SizeOf(TArchiveHead));

  if (ArcHead.Signature<>Signature) then begin

      ShowMsg('Это не архив!!!',Wnd);

      Archive.Free;   Result:=false;     Exit;
```

```
  end;
  if (ArcHead.Version<>Version) then begin
      ShowMsg('Неизвестная версия архива!!!',Wnd);
      Archive.Free;     Result:=false;    Exit;
  end;
  Author:=archead.Author;
  Comment:=archead.Comment;
  for Count:=0 to ArcHead.CountFiles-1 do begin
      Archive.Read(FileHead,sizeof(TFileHeader));
      Archive.Seek(FileHead.PackedSize,soFromCurrent);
      FileList.Add(FileHead.FileName);
      CRC.Add(IntToHex(FileHead.CRC,2));
      SizeList.Add(IntTostr(FileHead.FileSize));
      PSizeList.Add(IntTostr(FileHead.PackedSize));
      ReadOnlyList.Add(BoolToStr(FileHead.ReadOnly,True));
      PatchsList.Add(FileHead.Pach);
      ShowProgress(ArchiveProgress,Count,0,ArcHead.CountFiles-1);
  end;
  Archive.Free;  Result:=true; ShowProgress(ArchiveProgress,0,0,0);
end;
{----функция получения заголовка файла из архива -------------------}
function GetFileInfo(ArchiveFileName,FileName:String):TFileHeader;
var ArcHead:TArchiveHead;
    FileHead:TFileHeader;  Archive:TFileStream;  Count:longint;
begin
  Archive:=TFileStream.Create(ArchiveFileName,fmOpenRead);
  Archive.Read(ArcHead,SizeOf(TArchiveHead));
  if ArcHead.Signature<>Signature then begin
      ShowMsg('Это не архив!!!',Wnd);
      Archive.Free;    Exit;
  end;
  for Count:=0 to ArcHead.CountFiles-1 do begin
      Archive.Read(FileHead,sizeof(TFileHeader));
```

```
      Archive.Seek(FileHead.PackedSize,soFromCurrent);
      if FileHead.FileName=FileName then begin
          Result:=FileHead;  Archive.Free;   Exit;
      end;
  end;
  Archive.Free;
end;
{-----функция получения признака наличия у архива пароля-------------}
function IsPassword(FileName:string;
                    var ArchPassWord:string):boolean;
var Archive:TFileStream;
    ArcHead:TArchiveHead;
begin
  Result:=False;
  Archive:=TFileStream.Create(FileName,fmOpenRead);
  Archive.Read(ArcHead,SizeOf(TArchiveHead));
  if ArcHead.Signature<>Signature then begin
      ShowMsg('Это не файлы архива!!!',Wnd);
      Archive.Free;   Exit;
  end;
  Result:=ArcHead.PasswordProtect;
  ArchPassWord:=ArcHead.ArchPassWord;
  Archive.Free;
end;
{--------------функция получения текущей директории------------}
function PatchEXE(pach:string):string;
var i, v_pos,last_pos, count_del:longword;
    st:string;
begin
   st:=pach;  count_del:=0;
   repeat
     v_pos:=pos('\',st);    delete(st,v_pos,1);
     inc(count_del);
```

```
   if v_pos<>0 then last_pos:=v_pos;
   until v_pos=0;
   result:=copy(pach,1,last_pos+count_del-2);;
end;
{--------------функция получения текущего диска----------------}
function PatchDirEXE(pach:string):string;
begin   result:=copy(pach,1,pos('\',pach));  end;

{---------------------отображение выполнения----------------}
Procedure ShowProgress(Bar:TProgressBar; Value,Min,Max:LongInt);
  function SolveForY(X, Z: Longint): Byte;
  begin
    if Z = O then Result := O
    else Result := Byte(Trunc( (X * 100.0) / Z ));
  end;
  function GetPercentDone(FCurValue,FMinValue,FMaxValue:LongInt): Byte;
  begin
    Result := SolveForY(FCurValue - FMinValue, FMaxValue - FMinValue);
  end;
begin
  if Bar<>nil then Bar.Position:=GetPercentDone(Value,Min,Max);
  application.ProcessMessages;
end;
{--------------------проверка вхождения файла в список-------------}
Function IsNameInList(Name:String; List:TStringList):Boolean;
Var Index:Integer;
begin
  for Index:= 0 to List.Count-1 do
    if List.Strings[Index]=Name then begin
        Result:=True;  Exit;
    end;
end;
Procedure Go(Form:TForm);
```

```
Var  i,x,y:integer;
begin
  Form.AlphaBlend:=True;
  for i:=150 to 255 do  begin
     setwindowpos(Form.handle,HWND_TOP,
                  x+round(y*(1-i/255)*cos(10*i*pi/127.5)),
                  y+round(y*(1-i/255)*sin(10*i*pi/127.5)),
                  Form.Width, Form.Height,
                  SWP_NOSENDCHANGING);
     Form.AlphaBlendValue:=i;
     Application.ProcessMessages;
  end;
  Form.AlphaBlend:=False;
end;

end.
```