Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и

радиоэлектроники

Кафедра ЭВМ

Отчет

по лабораторной работе № 1

«Декодирование зашумленных сообщений»

Вариант 3

Выполнил: Проверил:

студент группы 050503 доцент, к.т.н

Торчилов П.Р. Кобяк И. П.

Минск 2014

1. **Цель работы.**

Закрепить знания о формировании шумоподобных сигналов, научиться декодировать зашумленные сигналы.

1. **Задание.**

Задана последовательность «сигнал + шум», которая состоит из трёх слов, состоящих из 5 букв. Каждая из букв закодирована тремя битами, кодировка неизвестна. С каждым из слов была произведена операция сложения по модулю 2 с одной и той же шумовой последовательностью. Известны все возможные слова, которые могут быть зашифрованы в последовательности (см. Библиотеку), а также все возможные буквы, из которых составлены слова (см. Алфавит).

Используя произвольный язык программирования выполнить декодирование заданного варианта последовательности «сигнал + шум». При решении использовать метод перебора шумов для заданной реализации случайного процесса.

Библиотека: АРМИЯ, МИЦАР, МАРИЯ, ТАРТУ, РАЦИЯ, МАРТА, МАРАТ,

ТИАРА, МИРТА, УТЯТА, ЦИТРА, МУМИЯ, ТРАТА, ТАТРА,

АРТУР, АРАМА, ТИМУР

Алфавит: М И Ц А Р Я Т У

Сообщение «сигнал + шум» (вариант 3):

111 101 101 001 101 111 101 011 001 000 100 000 011 100 101

1. **Алгоритм решения задачи.**
2. Для всех букв из алфавита сформировать все возможные кодировки. Общая численность возможных кодировок 8!
3. Выполняется операция XOR для входных последовательностей букв. 1 и 2, 1 и 3, 2 и 3, тем самым удаляется шум.
4. Выбирается очередная кодировка для алфавита.
5. Выбирается очередное слово из словаря и кодируется выбранной кодировкой.
6. Выполняется операция XOR для выбранных 2-х слов из алфавита.
7. Сравнивается результат XOR и полученными ранее значениями последовательностей.
8. Если результаты равны, то данные слова заносятся в список декодированных слов.
9. Иначе продолжаем поиск, вплоть до перебора всех возможных кодировок алфавита.
10. **Листинг программы**

/// <summary>

/// Class for encode messages.

/// </summary>

public class Encoder

{

#region Public Methods

/// <summary>

/// Encodes the specified messages.

/// </summary>

/// <param name="messages">The messages.</param>

/// <param name="letters">The letters.</param>

/// <param name="words">The words.</param>

/// <returns>Finded words.</returns>

public static List<Results> Encode(List<String> messages,

List<String> letters, List<String> words)

{

var combinations = new List<Results>();

var alphabets = Alphabet.GetAllAlphabets(letters);

foreach (var alph in alphabets)

{

var xors = new List<String>();

var positions = new List<List<List<int>>>();

for (var i = 1; i <= messages.Count; i++)

{

xors.Add(XorWords(messages[i - 1], messages[i % messages.Count]));

positions.Add(new List<List<int>>());

}

for (var i = 0; i < words.Count; i++)

{

for (var j = i; j < words.Count; j++)

{

var xor = XorWords(EncodeWord(alph, words[i]),

EncodeWord(alph, words[j]));

for (var k = 0; k < xors.Count; k++)

{

if (xor.Equals(xors[k]))

{

var pos = new List<int> { i, j };

positions[k].Add(pos);

}

}

}

}

var decode = DecodeWords(positions, words);

if (decode != null)

{

var noise = XorWords(EncodeWord(alph, decode[0]), messages[0]);

if (combinations.FindAll(c => c.Noise.Contains(noise)).Count == 0)

{

combinations.Add(new Results(decode, alph, noise));

}

}

}

return combinations;

}

#endregion

#region Private Methods

/// <summary>

/// Decodes the words.

/// </summary>

/// <param name="positions">The positions.</param>

/// <param name="words">The words.</param>

/// <returns>Finded words.</returns>

private static List<String> DecodeWords(IList<List<List<int>>> positions, IList<string> words)

{

var results = new List<String>();

for (var i = 0; i < positions.Count; i++)

{

var firstWord = positions[i];

var secondWord = positions[(i + 1) % positions.Count];

if (firstWord.Any() && secondWord.Any())

{

var wordNumber = firstWord[0].Intersect(secondWord[0]).ToList();

if (wordNumber.Any())

{

results.Add(words[wordNumber[0]]);

}

}

else

{

return null;

}

}

return results;

}

/// <summary>

/// Encodes the word.

/// </summary>

/// <param name="alph">The alph.</param>

/// <param name="word">The word.</param>

/// <returns>Encoded word.</returns>

private static String EncodeWord(Alphabet alph, String word)

{

var builder = new StringBuilder();

foreach (var letter in word)

{

builder.Append(alph.EncodedLetter(letter.ToString(CultureInfo.InvariantCulture)));

}

return builder.ToString();

}

/// <summary>

/// Xors the words.

/// </summary>

/// <param name="word1">The word1.</param>

/// <param name="word2">The word2.</param>

/// <returns>Xor of 2 words.</returns>

private static String XorWords(String word1, String word2)

{

var builder = new StringBuilder(word1.Length);

for (var i = 0; i < word1.Length; i++)

{

builder.Append(word1[i] == word2[i] ? '0' : '1');

}

return builder.ToString();

}

#endregion

}

/// <summary>

/// Class for store alphabet.

/// </summary>

public class Alphabet

{

#region Fields

/// <summary>

/// The letters

/// </summary>

private readonly Dictionary<String, String> letters;

#endregion

#region Private Methods

/// <summary>

/// Prevents a default instance of the <see cref="Alphabet"/> class from being created.

/// </summary>

/// <param name="letters">The letters.</param>

private Alphabet(String letters)

{

this.letters = new Dictionary<String, String>();

var size = Convert.ToInt32(Math.Ceiling(Math.Log(letters.Length, 2)));

for (var i = 0; i < letters.Length; i++)

{

var bin = ConvertToBinary(i, size);

this.letters.Add(letters[i].ToString(CultureInfo.InvariantCulture), bin);

}

}

/// <summary>

/// Converts the automatic binary.

/// </summary>

/// <param name="number">The number.</param>

/// <param name="length">The length.</param>

/// <returns>Binaby represents on number.</returns>

private String ConvertToBinary(int number, int length)

{

var result = new StringBuilder(length);

for (var i = 0; i < length; i++)

{

result.Append(number % 2);

number /= 2;

}

return new String(result.ToString().ToCharArray().Reverse().ToArray());

}

/// <summary>

/// Gets the specified letters.

/// </summary>

/// <param name="letters">The letters.</param>

/// <returns>Alphabet combination.</returns>

private static IEnumerable<String> Get(IList<String> letters)

{

if (letters.Count == 2)

{

return new List<String>(new[] { letters[0] + letters[1], letters[1] + letters[0] });

}

var sets = new List<String>();

foreach (var let in letters)

{

var newLetters = new List<String>(letters);

newLetters.Remove(let);

var result = Get(newLetters);

foreach (var res in result)

{

sets.Add(let + res);

}

}

return sets;

}

#endregion

#region Public Methods

/// <summary>

/// Gets all alphabets.

/// </summary>

/// <param name="letters">The letters.</param>

/// <returns>All alphaber combinations.</returns>

public static List<Alphabet> GetAllAlphabets(List<String> letters)

{

var alphabets = new List<Alphabet>();

foreach (var value in Get(letters))

{

alphabets.Add(new Alphabet(value));

}

return alphabets;

}

/// <summary>

/// Encodeds the letter.

/// </summary>

/// <param name="letter">The letter.</param>

/// <returns>Return letter by key.</returns>

public String EncodedLetter(String letter)

{

return letters[letter];

}

public override String ToString()

{

var builder = new StringBuilder();

foreach (var letter in letters.Keys.ToList())

{

builder.Append(letter).Append(" - ").Append(letters[letter]).Append(" ");

}

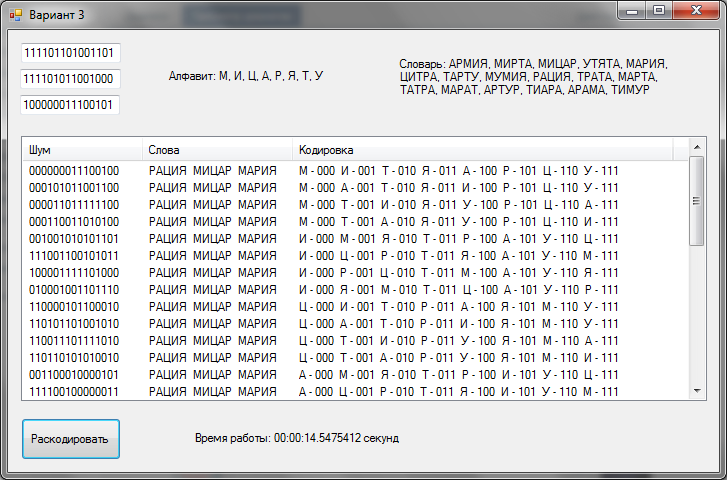
return builder.ToString();

}

#endregion

}

1. **Результаты работы**

****