Лабораторная работа

Алгоритм CRC (Циклический избыточный код)

**Цель работы -** закрепить теоретические знания об алгоритме циклического избыточного контроля (CRC), применяемом для проверки целостности данных, и научиться реализовывать алгоритм CRC на практике.

**Задачи работы**

* Изучить принципы построения циклических избыточных кодов.
* Реализовать алгоритм вычисления CRC-кода для заданной битовой последовательности.
* Провести тестирование алгоритма CRC на примерах, чтобы понять его способность обнаруживать ошибки в данных.

**Основные понятия и принципы**

Циклический избыточный код (CRC) — это метод обнаружения ошибок, основанный на математических свойствах циклических кодов и реализующийся через деление многочленов. CRC позволяет обнаружить случайные ошибки при передаче данных, проверяя контрольную сумму на стороне получателя.

**Порождающий многочлен** — это фиксированный полином, который используется для создания контрольной суммы. Примеры популярных полиномов:

* CRC-16: *G*(*x*)=*x*16+*x*15+*x*2+1
* CRC-CCITT: *G*(*x*)=*x*16+*x*12+*x*5+1

**Контрольная сумма** — остаток от деления многочлена данных на порождающий многочлен, представляющий собой CRC-код.

**Принцип работы CRC** — при передаче данных контрольная сумма добавляется к данным. При приеме данных их снова делят на порождающий многочлен. Если остаток равен нулю, данные переданы корректно; если нет — обнаружена ошибка.

**Задания для выполнения**

**1. Введение в работу с полиномами и битовыми строками:**

* Определите, что такое порождающий многочлен. Пример: для порождающего многочлена G(*x*)=*x*3+*x*+1 битовая строка представляется как 1011.
* Определите битовую последовательность данных. Например, последовательность данных 11010011.

**2. Реализация алгоритма CRC:**

* Введите данные и порождающий многочлен в виде битовой строки.
* Добавьте к битовой строке данных *r* нулей, где *r* — степень порождающего многочлена.
* Реализуйте побитовый алгоритм деления по модулю 2 (XOR) для вычисления остатка от деления.

**3. Подробный расчет CRC на примере:**

* Данные: 11010011.
* Порождающий многочлен: 10011.
* Присоедините к данным четыре нуля, получив 110100110000.
* Начните побитовое деление с использованием XOR, пока степень остатка не станет меньше степени порождающего многочлена.

**Шаги выполнения:**

* Первый шаг: 11010 XOR 10011 = 01001.
* Сдвиньте остаток и повторите, пока не получите окончательный остаток.
* Запишите итоговый остаток — это и есть CRC-код.

**4. Проверка данных с CRC:**

* Присоедините к исходным данным CRC-код и выполните деление полученной последовательности на порождающий многочлен.
* Если остаток **равен нулю**, то данные приняты корректно, в противном случае — обнаружена ошибка.

**5. Проверьте, как CRC реагирует на изменение битов:**

* Измените один или несколько битов в данных и проверьте, выявляет ли CRC изменение.

**Контрольные вопросы**

1. Какой принцип лежит в основе алгоритма CRC?
2. В чем заключается отличие CRC от других методов обнаружения ошибок, таких как коды Хэмминга или коды Рида-Соломона?
3. Какие типы ошибок CRC способен обнаруживать?
4. Почему алгоритм CRC на практике чаще всего реализуется аппаратно?

**Задания для самостоятельной работы (дополнительно\*)**

1. Реализуйте алгоритм CRC для 8-разрядного кода с порождающим полиномом CRC-8 (*G*(*x*)=*x*8+*x*2+*x*+1.
2. Напишите программу для вычисления CRC-16, применяемого в сетевых протоколах.
3. Проведите эксперимент с несколькими полиномами (например, CRC-12, CRC-CCITT) и сравните их эффективность в обнаружении ошибок.

**Алгоритм расчета CRC**

**Входные данные**:

* data — битовая строка данных (например, 11010011).
* generator — битовая строка порождающего полинома (например, 10011).

**Выходные данные**:

* CRC-код — остаток от деления, используемый как контрольная сумма.

**Шаги алгоритма:**

1. **Преобразование данных**:
   * Преобразуйте data и generator из строк в списки целых чисел, чтобы каждый бит представлялся как отдельный элемент списка (например, 11010011 → [1, 1, 0, 1, 0, 0, 1, 1]).
2. **Добавление нулей к данным**:
   * Добавьте (длина generator - 1) нулей в конец списка data, чтобы создать пространство для контрольной суммы.
3. **Инициализация длины порождающего многочлена**:
   * Задайте generator\_len как длину списка generator.
4. **Побитовое деление**:
   * Выполните следующие шаги для каждого бита data с индексом i, от 0 до (длина data - generator\_len + 1):
     + Если текущий бит data[i] равен 1:
       - Для каждого бита generator[j], от 0 до generator\_len - 1:
         * Выполните побитовое сложение по модулю 2 (операция XOR) между data[i + j] и generator[j].
         * Запишите результат обратно в data[i + j].
5. **Извлечение остатка**:
   * После выполнения побитового деления остаток (контрольная сумма) будет находиться в последних (generator\_len - 1) битах списка data.
6. **Возврат контрольной суммы**:
   * Верните последние (generator\_len - 1) битов списка data как итоговый CRC-код.