Лабораторная работа № 6
Циклические коды

Цель работы: изучение и реализация метода *CRC*, применяемого для контроля целостности данных при передаче и хранении.

Краткая теория

В основе работы данных алгоритмов используется метод проверки целостности массива бит, основанный на свойствах операции взятия остатка в полиномиальной арифметике по модулю 2 с основными операциями двоичной арифметики:

0+0=0, 0+1=1, 1+0=1, 1+1=0, 0*0=0, 0*1=0, 1*0=0, 1*1=1.

Алгоритм вычисления контрольной суммы (*CRC*, англ. *cyclic redundancy check*, проверка избыточности циклической суммы) — способ цифровой идентификации некоторой последовательности данных, который заключается в вычислении контрольного значения её циклического избыточного кода.

С точки зрения математики контрольная сумма (КС, *CRC*) является типом хэш-функции, используемой для вычисления контрольного кода — небольшого количества бит внутри большого блока данных, например, сетевого пакета или блока компьютерного файла, применяемого для обнаружения ошибок при передаче или хранении информации. Результат вычисления *CRC* добавляется в конец блока данных непосредственно перед началом передачи или сохранения данных на каком-либо носителе информации. Впоследствии он проверяется для подтверждения её целостности. Популярность *CRC* обусловлена тем, что подобная проверка просто реализуема в двоичном цифровом оборудовании, легко анализируется, и хорошо подходит для обнаружения общих ошибок, вызванных наличием шума в каналах передачи данных.

CRC Cyclic Redundancy Check (Циклический избыточный контрольный код) - результат операции взятия остатка от деления проверяемого битового массива на некоторое число-делитель. Это число-делитель, называемое образующим полиномом, выбирается так, чтобы само являлось полиномиально простым – не делилось полиномиально нацело на любые числа от 2 до самого себя. Кроме того, есть и другие критерии выбора полинома, направленные на уменьшение вероятности пропуска типичных ошибок в каналах передачи данных.

Полином может быть записан как в виде суммы степеней с ненулевыми (а значит – единичными) коэффициентами, так и маской этих единичек. Порядок записи единиц в маске однозначно связан с порядком обработки бит в проверяемом массиве, потому что в процессе расчета *CRC* промежуточный результат необходимо циклически сдвигать в ту же сторону, что и биты проверяемого массива, причем сдвигать так, чтобы вытеснялись старшие

степени полинома. Самая старшая степень в маске не учитывается, она определяет только число бит маски. Ниже старшие степени отделены пропусками.

Чтобы реализовать проверку с применением *CRC*, помимо маски полинома и порядка следования бит в массиве (определяющего направление циклического сдвига), необходимо знать начальное значение *CRC* и метод завершающей модификации результата вычисления *CRC*.

Типичные методы, применяемые для контроля целостности данных при передаче и хранении:

- *CCITT-CRC-32* [Все распространенные архиваторы и протоколы с *CRC-32*] - биты массива обрабатываются, начиная с младшего бита в байте – *LSB*. Образующий полином:

  $$X^0+X^1+X^2+X^4+X^5+X^7+X^8+X^{10}+X^{11}+X^{12}+X^{16}+X^{22}+X^{23}+X^{26}+X^{32}$$

  Маска = EDB88320h, в которой правые цифры соответствуют старшим степеням, сдвиг выполняется вправо. Начальное значение – 0xFFFFFFFF. Конечная модификация – поразрядная инверсия всех битов результата.

- *CCITT-DOS-16* [архиватор *LHA* и, вероятно, некоторые другие с *CRC*-16] - биты массива обрабатываются, начиная с младшего бита в байте – *LSB*. Образующий полином:

  $$X^0+X^2+X^{15}+X^{16}$$

  Маска = A001h, в которой правые цифры соответствуют старшим степеням, сдвиг выполняется вправо. Начальное значение – 0000. Конечная модификация – отсутствует.

- *CCITT-CRC-16* [протоколы передачи данных с *CRC*-16, Контроль *EMSI*] - биты массива обрабатываются, начиная со старшего бита в байте – *MSB*. Образующий полином:

  $$X^{16}+X^{12}+5^5+X^0$$

  Маска = 0x1021, в которой левые цифры соответствуют старшим степеням, сдвиг выполняется влево. Начальное значение – 0x0000. Конечная модификация – отсутствует.

Алгоритм вычисления: рабочая переменная *W* соответствующей разрядности, в которой будет накапливаться результат, инициализируется начальным значением. Затем для каждого бита *m* входного массива выполняются следующие действия: *W* сдвигается на 1 бит (о направлении сдвига см. выше). В освободившийся бит *W* помещается нуль. Бит, только что вытолкнутый из *W*, сравнивается с битом *m*. Если они не совпали, выполняется операция исключающего ИЛИ над *W* и маской полинома, результат заносится в *W*. И так далее, пока не будут обработаны все биты массива. После чего над *W* производится конечная модификация.

Можно сказать, что обычно так *CRC* считают только в схемных реализациях, потому что это очень медленно – ведь число циклов равно числу бит массива. При реализации на программном уровне обработка ведется

восьмерками бит – байтами. Заводится таблица из 256 элементов. Каждое значение – результат расчета *CRC* над восьмеркой бит индекса элемента:

for i := 0 to 255 do tab[i] := count_crc(i).

После этого расчет *CRC* для массива можно вести байтами. Начало и конец расчета, как и раньше. А цикл идет для каждого байта *Q*:

W := W XOR Q;

W := сдвиг(W, 8) XOR mab[W].

При *LSB*-порядке *Q* операция *XOR* выполняется над младшими битами *W*, а при *MSB*-порядке – над старшими. Индексом в таблице служат именно эти биты.

Байтовый табличный метод требует ощутимых затрат памяти под таблицу. Для *CRC-32* требуется таблица размером в килобайт. Можно предложить компромиссный вариант – считать *CRC*, не восьмерками, а четверками бит. *CRC-32*-таблица из 16 значений займет 64 байта, но скорость будет несколько ниже, чем при большой таблице, хотя существенно выше, чем без нее вообще.

Операция вычисления *CRC* обратима. Не в том смысле, конечно, что по *CRC* можно восстановить весь массив, а в том, что если дано *CRC* разрядности *N* и дан некоторый массив, в котором где-нибудь можно поменять подряд *N* бит, то подогнать этот массив под заданную *CRC* не сложнее, чем посчитать *CRC*. *CRC* не является криптографически устойчивой хеш-функцией.

Задание на лабораторную работу 6

Составьте алгоритмическое и программное обеспечение, реализующее алгоритм *CRC*. В качестве исходных данные – файл. Для созданного программного обеспечения проведите тестирование не менее чем на 10 различных наборах данных.