## §2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго отборочного этапа — 3 месяца с учетом того, что задачи открываются участникам в онлайн-системе последовательно, все задачи закрывались для решения одновременно. Задачи для 9-11 классов носят междисциплинарный характер и в более простой форме воссоздают инженерную задачу заключительного этапа. Решение каждой задачи дает определенное количество баллов. Баллы зачисляются в полном объеме за правильное решение задачи в зависимости от значимости задачи для финального этапа. В данном этапе можно получить суммарно от 0 до 10 баллов.

**Задача 2.1.1 (1 балл)**

*Условие:*

Для передачи сообщений в сетях очень часто приходится их разбивать на части – пакеты, которые собираются на принимающей стороне. Это приходится делать, так как каналы связи имеют ограниченную пропускную способность – способность передать количество информации за единицу времени. К тому же, пакетная передача данных становится более устойчива к обрывам связи и ошибкам во время передачи – достаточно заново передать недошедшие или испорченные пакеты, а не отправлять всё сообщение заново. Однако в сети бывает такое, что порядок прихода пакетов не совпадает с порядком их отправки. Для этого пакеты нумеруют по порядку их отправления. Принимающая сторона дожидается принятия последнего пакета, а затем (или по мере прихода пакетов) упорядочивает их по мере возрастания порядковых номеров пакетов – таким образом восстанавливается принятое сообщение.

В этой задаче вам предлагается реализовать очень простую имплементацию алгоритма разбиения сообщения на пакеты и их сборку на принимающей стороне. Допустим, что в нашем канале не происходят ошибки и все пакеты приходят так, как их отправили с точностью до порядка отправки. Каждый пакет занумерован, первый пакет содержит количество пакетов.

Требуется реализовать функцию, которая восстанавливает исходное сообщение по полученной серии пакетов.

**Формат данных**

На вход подаётся строка вида:

**«{пакет $i_1$},{пакет $i_2$},…,{пакет $i_n$}».**

**Длина входной строки** $l$: $1 \leq l \leq 4 * 10^6$

Для всех пакетов, у которых порядковый номер не равен 0, пакет ij имеет формат:

**«{порядковый номер пакета n}.{данные}»**, где {данные} – строка из цифр и букв латинского алфавита. Размер строки данных не ограничен ни снизу, ни сверху.

Для пакета с порядковым номером 0 формат следующий:

**«0.{кол-во пакетов}.{данные}»**

Например: «2.d,0.3.Hello,1.Worl» - обозначает возможный приход пакетов при разбиении сообщения «HelloWorld»

На выходе функция должна вернуть исходное сообщение в виде строки, состоящей из цифр и букв латинского алфавита.

**На вход может прийти пустая строка. В этом случае ответом задачи является строка «error».**

Примеры:

**Sample Input 1:**

```
4.l,1.llo,3.r,2.Wo,5.d,0.6.He
```

**Sample Output 1:**

```
HelloWorld
```

**Sample Input 2:**

```
3.4,1.2,2.3,0.4.1
```

**Sample Output 2:**

```
1234
```

**Sample Input 3:**

**Sample Output 3:**

```
error
```

*Решение:*

```
import sys

def solve(data):
    if data == "":
        return "error"
    packets = map(lambda x: x.split('.', 2), data.split(','))
    packets = sorted(packets, key = lambda a: int(a[0]))
    for i in range(len(packets)):
        packets[i] = packets[i][1+int(i==0)]
    return "".join(packets)

if __name__ == '__main__':
    data = sys.stdin.read().split(",", 1)
    print(solve(data))
```

*Проверка:*

```
import random
import base64
import os
import re

MIN_STR_LEN = 0
MAX_STR_LEN = 4*10**5

def rand_weighted(minv, maxv, weight_coef):
    v = int(random.triangular(minv, maxv, minv+(maxv-
minv)*weight_coef))
    return max(v, minv)

def generate_str(min_len, max_len):
```

```
    NORM_COEF = 3
    l = rand_weighted(min_len, max_len/NORM_COEF, 0.1)

    s = base64.b64encode(os.urandom(l)).decode('ascii')
    pattern = re.compile('[/+=]+')
    return pattern.sub('', s)

sample_tests = [
                "HelloWorld",
                "1234",
                ""
]

def generate_test(s):
    l = len(s)
    if l == 0:
        return ("","")

    packets = []
    fslc = rand_weighted(1, l, -0.3)
    slc = fslc
    l -= slc
    pos = 0
    i = 1
    while l!=0:
        pos += slc
        slc = rand_weighted(1, l, 0.1)
        l -= slc
        packets.append("%d.%s" % (i, s[pos:pos+slc]))
        i+=1

    packets.append("0.%d.%s" %(len(packets)+1, s[:fslc]))
    random.shuffle(packets)
    return (",".join(packets), s)

def generate():
    num_tests = 30
    tests = []
    for sample in sample_tests:
        tests.append(generate_test(sample))
    for i in range(num_tests):
        tests.append(generate_test(generate_str(MIN_STR_LEN,
MAX_STR_LEN)))
    return tests


def solve(dataset):
    if dataset == "":
        return "error"
    packets = map(lambda x: x.split('.', 2), dataset.split(','))
    packets = sorted(packets, key = lambda a: int(a[0]))
    for i in range(len(packets)):
        packets[i] = packets[i][1+int(i==0)]
    return "".join(packets)

def check(reply, clue):
    return (not len(clue) and reply == "error") or reply == clue
```

**Задача 2.1.2 (2 балла)**

*Условие:*

Задача аналогична предыдущей. Вам нужно реализовать свой алгоритм разбиения и сборки пакетов. Но теперь в канале есть ошибки: пакеты могут теряться. Пакет либо доходит до получателя в неизменном виде, либо не приходит вовсе. Формат пакетов не ограничен. От вас не требуется восстанавливать сообщение, в случае, если один и более пакетов были потеряны при передаче. Эта ситуация - одна из первых проблем, с которой сталкиваются при передаче данных по реальным каналам.

Реализовать функцию, разбивающую сообщение в режиме "encode" на пакеты заданной длины и собирающую исходное сообщение из пакетов в режиме "decode". В случае, если нельзя закодировать или декодировать сообщение, вывести "error".

Замечание: все пакеты должны иметь строго ту длину, которая была задана при вызове функции.

**Формат данных**

На вход подаются три параметра: данные (**data**), режим работы (**mode**), размер пакета (**unit size**). Режим работы **(mode)** принимает два значения: "encode" и "decode".

Данные (data), в случае mode == "encode": строка, состоящая из цифр и букв латинского алфавита. Строка имеет размер $l: 1 \leq l \leq 4 * 10^6$ символов.

**Данные(data)**, в случае mode == "decode": строка, которая была возвращена после одного из вызовов функции в режиме encode.

**Unit size** – размер каждого пакета. Произвольное число от 1 до размера строки data $l$.

**Примеры**

(в данных примерах алгоритм выбран таким образом, что в первом закодированном пакете содержится количество пакетов. В остальном алгоритм кодирования и декодирования аналогичен Задаче 2.1.1.)

**Sample Input 1:**
«Hello», «encode», «3»
**Sample Output 1:**
«0.6,1.H,2.e,3.l,4.l,5.o»
**Sample Input 2:**
«0.6,1.H,5.o,4.l,2.e,3.l», «decode», «3»
**Sample Output 2:**
«Hello»
**Sample Input 3:**
«0.6,2.e,3.l,4.l,1.H», «decode», «3»
**Sample Output 3:**
**«error»**

*Решение*

```
import sys
import math

SPACE_CHAR = "+"

# Padding used to make uniform sized packets
def pad(s, l):
    return s + SPACE_CHAR*(l-len(s))

# Strip padding from the data
```

```
def strip(s):
    return s.replace(SPACE_CHAR, "")

def sort(packets):
    return sorted(packets, key = lambda p: int(p[0], 36))

def decode(data):
    if data == "":
        return "error"
    packets_raw = data.split(',') # split packets
    packets = map(lambda x: x.split('.', 1), packets_raw) # split data
inside packets
    packets = sort(packets) # sort packets, find header
    header = packets[0]
    if header[0] != '0' or len(packets) != int(strip(header[1]), 36):
       # some packets are lost - fewer than stated in header
       return "error"
    for i in range(1, len(packets)):
        packets[i] = strip(packets[i][1])
    return "".join(packets[1:])

def digit_to_char(digit):
    if digit < 10:
        return str(digit)
    return chr(ord('a') + digit - 10)

def str_base(number,base):
    if number < 0:
        return '-' + str_base(-number, base)
    (d, m) = divmod(number, base)
    if d > 0:
        return str_base(d, base) + digit_to_char(m)
    return digit_to_char(m)

def calculate_size(data, packet_size):
    num_packets = math.ceil(len(data) / packet_size)
    index_pad_max = math.ceil(math.log(num_packets)/math.log(36)+1)
    data_size = packet_size - 1 - index_pad_max
    return num_packets, data_size

def encode(data, packet_size):
    if data == "":
        return "error"
    num_packets, data_size = calculate_size(data, packet_size)
    cur_len = 0
    index = 1
    packets = []
    while cur_len < len(data):
        index_str = str_base(index, 36)
        packet = "%s.%s" % (index_str,
data[cur_len:cur_len+packet_size - 1 - len(index_str)])
        packets.append(pad(packet,packet_size))
        cur_len += packet_size - 1 - len(index_str)
        index += 1
    packets.append(pad("0.%s" % str_base(index, 36), packet_size))
    return ",".join(packets)
```

```
def handle_message(data, mode, packet_size):
    handled_message = encode(data, packet_size) if mode == "encode"
else decode(data)
    return handled_message
```

**Проверка**

```
import random
import base64
import os
import re
import math
from multiprocessing import Process

SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178=="
SECRET_CHEATING_STR =
"1943a345ecef300c1ee39edc43270bf18620360f2eb57d49a575337e4adba15e=="
SECRET_ENCODE_FAILED_STR =
"5d28a90f4498a81461efbaf6f628a19d9778390bb5c81a393dd936181cc3d826=="

MIN_STR_LEN = 0
MAX_STR_LEN = 1*10**6
STR_LEN_TRI_SCALE = 0

NUM_TESTS = 50

MIN_DROPS = 1
MAX_DROPS = max(1, int(NUM_TESTS*0.3))
MAX_DROP_RATE = 0.3

used_ids = set()

TEST_TEMPLATES = [
    "HelloWorld",
    "1234"
]

def generate_str(min_len, max_len):
    NORM_COEF = 3
    l = int(random.triangular(min_len, max_len, min_len+(max_len-
min_len)*STR_LEN_TRI_SCALE)/NORM_COEF)
    l = max(min_len, l)
    s = base64.b64encode(os.urandom(l)).decode('ascii')
    pattern = re.compile('[/+=]+')
    return pattern.sub('', s)

def generate_dataset(s = None, drop = False):
    dataset_id = 0
    while dataset_id in used_ids:
        dataset_id = random.randint(0, 0xFFFFFFFF)
    used_ids.add(dataset_id)
    if s is None:
        s = generate_str(MIN_STR_LEN, MAX_STR_LEN)
    unit_size = random.randint(1, len(s))
    num_packets = max(1, int(math.ceil(len(s)/unit_size)))
    unit_size += len(hex(num_packets))
    drops = 0
```

```
    if drop:
        drops = random.randint( 1, max(1,
int(num_packets*MAX_DROP_RATE)) )

    return ( "%s %d %d" % (s, unit_size, drops), "%s %i" % (s, drops))

tests_ = []
def generate():
    # include templates
    drops_remain = random.randint(MIN_DROPS, MAX_DROPS)

    for template in TEST_TEMPLATES:
        tests_.append(generate_dataset(template))
    for i in range(NUM_TESTS):
        drop = False
        if drops_remain:
            drop = bool(drops_remain/random.randint(1, NUM_TESTS-i))
        drops_remain -= drop
        tests_.append(generate_dataset(None, drop))
    return tests_

def solve(dataset):
    return SECRET_CHEATING_STR

def check(reply, clue):
    if reply == SECRET_CHEATING_STR: # cheating
        return True
    if reply == SECRET_ENCODE_FAILED_STR:
        return False, "Packet size in 'encode' mode is wrong."
    elif reply.startswith(SECRET_FAILED_WITH_ERROR_STR):
        return False, "Code error: %s" % reply.split(" ", 1)[1]
    else:
        answer, dropped = clue.split(" ", 1)
        dropped = int(dropped)
        return check_decode(reply, answer if not dropped else "error",
dropped)


def check_decode(reply, answer, dropped):
    if reply != answer:
        return False, "Decoder's answer does not match the actual
message"
    return True
```

*Файловые шаблоны*

```
::python3
::header
import random
import sys

SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178=="
SECRET_ENCODE_FAILED_STR =
"5d28a90f4498a81461efbaf6f628a19d9778390bb5c81a393dd936181cc3d826=="

encoded, unit_size, num_drop = sys.stdin.read().split(" ", 2)
```

```
unit_size = int(unit_size)
num_drop = int(num_drop)

def check_and_transform_encode(reply, unit_size):
    try:
        packets = reply.split(',')

        for i in range(len(packets)):
            packet = packets[i]
            if len(packet) != unit_size:
                return None
        to_remove = sorted(random.sample(range(len(packets)),
num_drop))
        for i in range(len(to_remove)):
            del packets[to_remove[len(to_remove) - i - 1]]
        random.shuffle(packets) # shuffle remaining packets
        return ",".join(packets)
    except Exception as e:
        return None

::footer
try:
    encoded = handle_message(encoded, "encode", unit_size)

    encoded = check_and_transform_encode(encoded, unit_size)
    if encoded is None:
        print(SECRET_ENCODE_FAILED_STR)
    else:
        print(handle_message(encoded, "decode", unit_size))
except Exception as e:
    print("%s %s" % (SECRET_FAILED_WITH_ERROR_STR, "Exception during
runtime"))

::code
def handle_message(data, mode, packet_size):
    handled_message = ""
    # your code goes here
    return handled_message

::c++11
::code

#include <string>
#include <vector>

std::string handle_message(const std::string &data, const std::string
&mode, size_t packet_size) {
    std::string handled_message;
    //your code goes here
    return handled_message;
}

::header

#include <random>
#include <iostream>
#include <sstream>
#include <algorithm>
```

```
#include <assert.h>

const std::string SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";
const std::string SECRET_ENCODE_FAILED_STR =
"5d28a90f4498a81461efbaf6f628a19d9778390bb5c81a393dd936181cc3d826==";

std::vector<std::string> split(const std::string &s, char delim) {
    std::stringstream ss(s);
    std::string item;
    std::vector<std::string> elems;
    while (std::getline(ss, item, delim)) {
        elems.push_back(std::move(item));
    }
    return elems;
}

std::vector<int> FisherYatesShuffle(std::size_t size, std::size_t
max_size, std::mt19937& gen)
{
    assert((max_size == 0 && size == 0) || size < max_size);
    std::vector<int> b(size);

    for(std::size_t i = 0; i != max_size; ++i) {
        std::uniform_int_distribution<> dis(0, i);
        int j = dis(gen);
        if(j < b.size()) {
            if(i < j) {
                b[i] = b[j];
            }
            b[j] = i;
        }
    }
    return b;
}

bool check_and_transform_encode(std::string &message, size_t
unit_size, size_t num_drop) {
    try {
        std::random_device rd;
        std::mt19937 gen(rd());

        auto packets = split(message, ',');
        auto s = packets.size();
        for( size_t i=0; i<s; ++i) {
            if(packets[i].size() != unit_size)
                return false;
        }
        auto to_remove = FisherYatesShuffle(num_drop, packets.size(),
gen);
        std::sort(to_remove.begin(), to_remove.end());
        for(size_t i = to_remove.size()-1; i!=-1; --i)
            packets.erase(packets.begin()+to_remove[i]);

        std::shuffle( packets.begin(), packets.end(), gen ); //
shuffle remaining packets
        std::stringstream ss;
        for (size_t i=0; i<s; ++i)
```

```
            ss << packets[i] << (i==s-1?"":",");
        message = ss.str();
        return true;
    }
    catch(...)
    {
        return false;
    }
}

::footer
int main() {
    std::string message;
    size_t unit_size, num_drop;

    std::cin >> message >> unit_size >> num_drop;
    try {
        message = handle_message(message, "encode", unit_size);
        if (!check_and_transform_encode(message, unit_size, num_drop))
            std::cout << SECRET_ENCODE_FAILED_STR;
        else
            std::cout << handle_message(message, "decode", unit_size);
    }
    catch(...) {
        std::cout << SECRET_FAILED_WITH_ERROR_STR << " " << "Exception
during runtime.";
    }
    return 0;
}

::mono c#

::header

using System;
using System.Collections.Generic;
using System.Text;



    class Test
    {
        private static Random rng = new Random();

        public static void Shuffle<T>(ref List<T> list)
        {
            int n = list.Count;
            while (n > 1) {
                n--;
                int k = rng.Next(n + 1);
                T value = list[k];
                list[k] = list[n];
                list[n] = value;
            }
        }

        static List<int> FisherYatesShuffle(int size, int maxSize) {
            List<int> res = new List<int>();
```

```
            for(int i = 0; i < maxSize; ++i) {
                var j = rng.Next(0, maxSize);
                if(j < res.Count) {
                    if(i < j) {
                        res[i] = res[j];
                    }
                    res[j] = i;
                }
            }
            return res;
        }


        static bool CheckAndTransformEncode(ref string encoded, int
unitSize, int numDrop) {
            try {
                List<string> packets = new
List<string>(encoded.Split(','));
                var s = packets.Count;
                for( int i=0; i<s; ++i) {
                    if(packets[i].Length != unitSize)
                        return false;
                }
                var toRemove = FisherYatesShuffle(numDrop,
packets.Count);

                toRemove.Sort(
                    new Comparison<int>(
                            (i1, i2) => i2.CompareTo(i1)
                    ));

                for(int i=0; i< toRemove.Count; ++i)
                    packets.RemoveAt(toRemove[i]);

                Shuffle<string>(ref packets);
                encoded = String.Join(",",packets);
                return true;
            }
            catch {
                return false;
            }
        }

::code
static string HandleMessage(string data, string mode, int unitSize) {
    //your code goes here
    return "";
}
::footer

        const string SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";
        const string SECRET_ENCODE_FAILED_STR =
"5d28a90f4498a81461efbaf6f628a19d9778390bb5c81a393dd936181cc3d826==";

        static void Main(string[] args)
        {
           string input = Console.ReadLine();
```

```
            string[] array = input.Split(' ');
            int unitSize = Int32.Parse(array[1]);
            int numDrop = Int32.Parse(array[2]);
            try {

                var encode = HandleMessage(array[0], "encode",
unitSize);
                if (!CheckAndTransformEncode(ref encode, unitSize,
numDrop))
                    Console.Write(SECRET_ENCODE_FAILED_STR);
                else
                    Console.Write(HandleMessage(encode, "decode",
unitSize));
            }
            catch {
                Console.Write(SECRET_FAILED_WITH_ERROR_STR);
            }
        }
    }

::java8
::header
import java.io.BufferedReader;
import java.io.Console;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

class Test
{
    private static Random rng = new Random();

    public static <T> void Shuffle(List<T> list)
    {
        int n = list.size();
        while (n > 1) {
            n--;
            int k = rng.nextInt(n + 1);
            T value = list.get(k);
            list.set(k, list.get(n));
            list.set(n, value);
        }
    }

    static List<Integer> FisherYatesShuffle(int size, int maxSize) {
        List<Integer> res = new ArrayList<Integer>();
        for(int i = 0; i < maxSize; ++i) {
            int j = rng.nextInt(maxSize);
            if(j < res.size()) {
                if(i < j)
                    res.set(i, res.get(j));
                res.set(j, i);
            }
        }
        return res;
    }
```

```
    static Boolean CheckAndTransformEncode(String encoded, int
unitSize, int numDrop) {
        try {
            List<String> packets = Arrays.asList(encoded.split(","));
            Integer s = packets.size();
            for( int i=0; i<s; ++i) {
                if(packets.get(i).length() != unitSize)
                    return false;
            }
            List<Integer> toRemove = FisherYatesShuffle(numDrop,
packets.size());
            toRemove.sort(
                    (o1, o2) -> o2.compareTo(o1)
            );

            for(int i=0; i< toRemove.size(); ++i)
                packets.remove(toRemove.get(i));

            Shuffle(packets);
            encoded = String.join(",",packets);
            return true;
        }
        catch (Exception e) {
            return false;
        }
    }
::code
static String handleMessage(String data, String mode, int unitSize) {
    //your code goes here
    return "";
}
::footer
    private static String SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";
    private static String SECRET_ENCODE_FAILED_STR =
"5d28a90f4498a81461efbaf6f628a19d9778390bb5c81a393dd936181cc3d826==";

    public static void main(String[] args) throws IOException {
        BufferedReader buffer=new BufferedReader(new
InputStreamReader(System.in));

        String input = buffer.readLine();
        String[] array = input.split(" ");
        int unitSize = Integer.parseInt(array[1]);
        int numDrop = Integer.parseInt(array[2]);
        try {

            String encode = handleMessage(array[0], "encode",
unitSize);
            if (!CheckAndTransformEncode(encode, unitSize, numDrop))
                System.out.print(SECRET_ENCODE_FAILED_STR);
            else
                System.out.print(handleMessage(encode, "decode",
unitSize));
        }
        catch (Exception e){
            System.out.print(SECRET_FAILED_WITH_ERROR_STR);
        }
```

```
    }
}
```

**Задача 2.1.3 (3 балла)**

В реальных системах связи передаваемый сигнал искажается.

Искажение сигнала воспринимается принимающей стороной как ошибки внутри пакетов данных.

Ошибки классифицируются на несколько типов:

- Замена - передаваемый символ a (в двоичном канале - "0" или "1") заменяется символ b на с вероятностью Pab.
- Стирание - передаваемый символ a заменяется на некоторый символ x , который не принадлежит алфавиту передаваемых символов с вероятностью p ax. Декодеру известна позиция искажения, но неизвестно искажённое значение. Например: "101101" может быть принято декодером как "x01x01" после стирания в первой и четвёртой позиции.

Выброс - передаваемый символ a i исключается из передачи данных с вероятностью p a i ,∗ . Например, выброс в третьей позиции преобразует сообщение "00101101" в "0001101". Такой тип ошибок - самый сложный для обнаружения и исправления, так как при декодировании неизвестна даже позиция такого искажения.

Все вероятности искажений зависят от множества факторов и могут меняться со временем. Для конкретного канала они вычисляются экспериментальным путём.

В этой и следующих задачах вероятности искажений будут известными и постоянными. Во всех задачах здесь и далее ошибками будут считаться ошибки замены, если не сказано иное.

Задачу исправления или обнаружения ошибок в передаваемых данных можно решить путем добавления дополнительной информации к сообщению - контрольной суммы.

Очевидно, что алгоритмы из первых задач не решают задачу исправления или обнаружения ошибок в передаваемых данных. Одна из идей, как можно решить проблему передачи данных по каналам – добавлять некоторую дополнительную информацию к сообщению. Хотя алгоритмы из первых задач не решают проблемы этого этапа, в них используется контрольная сумма для проверки целостности разбиения пакетов – нумерация пакетов и их общее количество. Для проверки данных уже внутри пакетов, необходимо опираться на некоторые факты о данных и передавать эту информацию вместе с пакетом.

Одним из самых старых алгоритмов вычисления контрольных сумм является алгоритм вычисления чётности данных - проверке на чётность количества "1" или "0" в двоичном канале.

Например, в переданном пакете "11110" последний бит отражает чётность количества единиц в данных. Если произойдёт одно или другое нечётное число ошибок внутри пакета в любом месте, этот пакет можно восстановить или определить наличие ошибки. Однако, если произойдёт чётное число ошибок, данный алгоритм не сможет гарантированно определять наличие ошибок и восстанавливать данные.

Этот алгоритм применялся при проверке правильности ввода машинного кода на ленту и используется до сих пор в некоторых протоколах передачи данных.

Наиболее часто использующийся алгоритм – циклический избыточный код, CRC (Cyclic Redundancy Check).

CRC код определяется "полиномом" - некоторым числом, и "длиной кода" - количеством

в двоичной записи этого числа.
Например, для CRC с полиномом $101_2$(с записью в полиномиальном виде: $1 + 0 * x + 1 * x^2$) длина кода r равна 3 (длину кода можно считать как позицию справа последнего ненулевого символа).

Сообщение, для которого должна быть вычислена контрольная сумма, представляется в двоичном виде как большое число. Например, английская буква "a" при переводе в двоичное число из кода ASCII представляется как $01100001_2$ . Это число дополняется нулями слева столько раз, какова длина кода ( $01100001000_2$ ) и получившееся число делится в двоичной системе счисления на "полином". После двоичного деления, его остатком и будет вычисленная контрольная сумма ($001_2$).

Алгоритм проверки простой - вычисляется эта же контрольная сумма для принятых данных и сравнивается с полученной контрольной суммой.
Данный алгоритм позволяет определять до $2^r - 1$ ошибок, где r – длина кода. Такой код, однако, не может указать позиции ошибок или помочь в их исправлении.

Существуют и используются более сложные алгоритмы вычисления и проверки контрольных сумм. Некоторые из них будут использоваться в следующих задачах.

*Условие:*

В этой задаче вам предлагается проверить переданные данные на наличие ошибок с использованием метода контрольных сумм.

Необходимо реализовать функцию, которая по входной строке и заданному алгоритму вычисления контрольной суммы возвращает:

- исходное сообщение в случае передачи без ошибок,
- строку, содержащую номера пакетов, переданных с ошибками.

Гарантируется, что число ошибок в пакетах не превышает возможности по обнаружению ошибок у выбранного алгоритма вычисления контрольной суммы.

Ошибки генерируются случайным образом и могут появиться в любом месте кроме точек-разделителей блоков внутри пакета.

**Формат данных**

На вход подаются два параметра: данные (data), обозначение алгоритма (mode).
Обозначение алгоритма принимает следующие значения: "p" - выбран алгоритм проверки на чётность, "c" - выбран алгоритм CRC-32.
Данные (data) имеют следующий формат:
**«{пакет i1},{пакет i2},…,{пакет in}».**
**Длина входной строки** $l$: $1 \leq l \leq 4 * 10^6$
Для всех пакетов, у которых порядковый номер не равен 0, пакет ij имеет формат:
**«{контрольная сумма}.{контрольная сумма номера пакета}.{порядковый номер пакета}.{данные}»**
*, где {данные} – строка из **любых** 8-битных символов в случае наличия ошибки, или цифр и букв латинского алфавита в противном случае. Размер строки данных ограничен сверху 32 символами.*
*, {контрольная сумма} - число в десятичном формате, не превышающее $ 2^{64}-1 $. Для алгоритма проверки на чётность строки длины N это число в двоичном формате представляется следующим образом:*
***"0000...0{контрольная сумма символа 1}{контрольная сумма символа 2}....{контрольная сумма символа N}"***
**,** *где {контрольная сумма символа i} - бит со значением 1 или 0, а количество нулей слева таково, что длина такой двоичной строки кратна 8.*
*, {контрольная сумма номера пакета} - контрольная сумма, вычисленная от*

*порядкового номера пакета. Примечание: контрольная сумма номера пакета вычисляется от **строкового представления** номера, а не от самого числа.*
Для пакета с порядковым номером 0 формат следующий: **«{контрольная сумма}.{контрольная сумма номера пакета checksum(0)}.0.{кол-во пакетов}.{данные}»**
***Примечание:** общая контрольная сумма вычисляется от **всей** строки после **первой точки.** Например, для пакета некоторого сообщения x="20.1.1.l". Общая контрольная сумма - 20 - вычисляется от строки "1.1.l".*

На выходе функция должна вернуть либо исходное сообщение, состоящие из букв и цифр латинского алфавита, либо строку вида: **"{номер пакета},{номер пакета},...{номер пакета}",** содержащую номера пакетов, которые были переданы с ошибками. Если ошибка произошла в номере пакета, вместо номера данного пакета указать "-".
**Замечание по алгоритму проверки:**
Ошибки могут распределиться так, что будут повреждены только контрольные суммы или одна из контрольных сумм внутри пакета. Пакет, у которого повреждены только контрольные суммы, считается **неповреждённым**. Пакет, у которого была повреждена контрольная сумма порядкового номера пакета (но сам порядковый номер не был повреждён) считается пакетом с **неповреждённым индексом**. Это учитывается при проверке решения.

**Sample Input 1:**
```
p 20.1.1.l,0.0.0.3.Hel,657.1.2.oWorld
```
**Sample Output 1:**
```
HelloWorld
```
**Sample Input 2:**
```
p 6.0.0.2.Hel,1397.1.1.loWorld
```
**Sample Output 2:**
```
0,1
```
**Sample Input 3:**
```
c 3301236689.2212294583.1.loWorld,2898817664.4108050209.0.2.Hel
```
**Sample Output 3:**
```
HelloWorld
```
**Sample Input 4:**
```
c 1187106770.2212294583.5.oWorld,2064812460.4108050209.4.2.Hell
```
**Sample Output 4:**
```
-,-
```

**Алгоритм вычисления контрольной суммы методом проверки на чётность**
Давайте уточним алгоритм вычисления контрольной сумма для метода проверок на чётность (режим "p" задачи)
В описании задачи уже было показано, что алгоритм проверки на чётность вычисляет чётность числа единиц или нулей во входном слове: для слова $x = 111_2$контрольной суммой будет $ck = 1_2$, а для слова $x' = 00010001_2 \rightarrow ck = 0_2$ – если проверяется чётность числа единиц в слове.
(можно также заметить, что если длина слова $x$ - чётное число (например, $\boldsymbol{8}$), то контрольные суммы при подсчёте единиц и нулей совпадают. Для восьмибитных символов удобнее всего вычислять чётность числа единиц)
Давайте представим входное слово $x$ как вектор значений: $x = (x_1, x_2, \ldots, x_n)$ . Введём понятие веса вектора $x$ – количества ненулевых значений вектора $x$: $|| x || = \sum_{i:x_i \neq 0_2}^{L} 1$, где$|| x ||$– вес вектора $x$. Тогда результатом вычисления контрольной суммы является вес вектора $x$ по модулю 2: $ck(x) = || x || \ mod\ 2$. Для двоичных векторов функция

контрольной суммы может быть определена как:
$ck(x) = \bigoplus x_i, i \in [1, n]$, где $\bigoplus$ – поэлементное "исключающее ИЛИ" для вектора $x$
Например, для слова $x = 1101_2$: $ck(x) = 1 \bigoplus 1 \bigoplus 0 \bigoplus 1 = 1_2$.
Отправляемое сообщение - входное слово $I$, которое, как уже было сказано выше, можно представить как вектор (бинарных) элементов. Таким образом для отправляемого сообщения $I$ мы можем вычислить его контрольную сумму. Эта сумма будет состоять из одного бита $ck$ : "0" или "1" – в зависимости от веса всего сообщения.
Уже было сказано, что с помощью такой контрольной суммы возможно гарантированно обнаружить одну ошибку или любое ***нечётное*** число ошибок. Этого может быть вполне достаточно для проверки целостности передачи ***одного символа*** в сообщении. Так, например, в нескольких режимах протокола передачи данных UART вместе с каждым 8-битным символом передаётся девятый бит, который зарезервирован для контрольной суммы переданного символа.
Однако этого явно недостаточно, если нам нужно гарантировать правильность принятых данных: если в сообщении всегда появляются хотя бы две ошибки, мы можем гарантировать правильность проверок только в половине случаев, если вероятность ошибки неизвестна, что, по сути, аналогично подбрасыванию монетки. Более того, не получится определить даже примерное расположение ошибок или их (примерное) количество, если контрольная сумма укажет на их наличие.
Очевидно, если не менять сам метод вычисления контрольной суммы, то необходимо увеличить ***количество контрольных сумм*** для сообщения. Например, вычислять её для каждого символа в сообщении, а не для всего сообщения целиком. Тогда, *контрольной суммой сообщения* $I$ является слово$ckm(I) = (ck(I1), ck(I2), \dots ck(In))$, состоящее из контрольных сумм каждого символа в сообщении. Для сообщения из *n* символов количество элементов в слове контрольной суммы также равно n.
Например, для сообщения:
$I = (H, i)_{ascii} = (72{,}105)_{10} = (0100.1000{,}0110.1001)_2$, $ckm(I) = (ck(H), ck(i)) = (ck(0100.1000), ck(0110.1001)) = (0{,}0)$
Опишем пошаговый алгоритм вычисления:

```
checksum_char(ch of char):
0. bit ck := 0
1. for each bit in ch:
2. ck := ck XOR bit
3. return ck

checksum(s of array[n] of char):
0. integer ckm = 0
1. for i in [0, n-1]:
2. ckm[i] := checksum_char(s[i])
3. return ckm
```

Рассмотрим алгоритм на примере того, как вычислялись общая контрольная сумма и контрольная сумма порядкового номера для пакета: "657.1.2.oWorld" (общая контрольная сумма равна $657_{10} = 10.1001.0001_2$, контрольная сумма порядкового номера равна $1_{10} = 1_2$):
1. Посчитаем контрольную сумму порядкового номера:$index = ckm(2_{ascii}) = ckm(50_{10}) = ckm(0011.0010_2)$
1.1. $index = 1_2 = 1_{10}$

2. Посчитаем общую контрольную сумму:$total = ckm(1.2.oWorld_{ascii})$:
2.1.$ckm(1.2.oWorld) = (ck(1), ck(.), ck(2), ck(.), ck(o), ck(W), ck(o), ck(r), ck(l), ck(d))_{ascii}$

2.2. Вычислим ck(...) отдельно для каждого символа:
2.2.1. $ck(1)_{ascii} = ck(49)_{10} = ck(0011.0001)_2 = 1$
2.2.2.$ck(.)_{ascii} = ck(46)_{10} = ck(0010.1110)_2 = 0$
2.2.3.$ck(2)_{ascii} = ck(50)_{10} = ck(0011.0010)_2 = 1$
2.2.4.$ck(o)_{ascii} = ck(111)_{10} = ck(0110.1111)_2 = 1$
2.2.5.$ck(W)_{ascii} = ck(87)_{10} = ck(0101.0111)_2 = 1$
2.2.6.$ck(r)_{ascii} = ck(114)_{10} = ck(0111.0010)_2 = 0$
2.2.7.$ck(l)_{ascii} = ck(108)_{10} = ck(0110.1100)_2 = 0$
2.2.8. $ck(d)_{ascii} = ck(100)_{10} = ck(0110.0100)_2 = 1$
2.3. $total = (ck(1), ck(.), ck(2), ck(.), ck(o), ck(W), ck(o), ck(r), ck(l), ck(d))ascii = (1,0,1,0,0,1,0,0,0,1)$
2.4 $total = 10.1001.0001_2 = 675_{10}$

*Решение*

```
import base64
import os
import zlib
def parity_byte_sum(b):
    parity = 0
    while b != 0:
        parity ^= b & 1
        b >>= 1
    return parity


def parity_sum(s):
    parity_num = 0
    for i in range(len(s)):
        parity_num <<= 1
        parity_num |= parity_byte_sum(ord(s[i]))
    return parity_num

def crc32_sum(s):
    return zlib.crc32(s.encode())


ALGORITHMS = {
    'p': parity_sum,
    'c': crc32_sum
}

def solve(data, algo):
    packets = data.split(',') # obtain packets
    errors = []
    packet_data = []

    for p in packets:
        d_checksum, d_content = p.split('.', maxsplit=1)
        checksum = ALGORITHMS[algo](d_content) # calculate full
checksum for data content with index
        i_checksum, index, d_content = d_content.split('.',
maxsplit=2)
        index_check = ALGORITHMS[algo](index) # calculate checksum for
index
        index_correct = str(index_check) == i_checksum # check if
```

```
index is correct
        correct = str(checksum) == d_checksum and index_correct #
check if data is correct
        if not index_correct:
           # maybe it is only checksum that is damaged?
            # Let's substitute probably damaged checksum with ours
            # and calculate full checksum again
            s = "%d.%s.%s" % (index_check, index, d_content)
            checksum = ALGORITHMS[algo](s)
            if str(checksum) == d_checksum:
                errors.append(str(index)) # so index checksum is
really damaged
            else:
                errors.append('-') # nope, index is damaged
        elif not correct:
            errors.append(str(index)) # nope, data is not correct
        # No errors:
        d_content = d_content.split('.')
        if len(errors) == 0:
            packet_data.append((index, d_content[-1]))
    if not len(errors):
        packet_data.sort(key=lambda x: x[0])
        return "".join(map(lambda x: x[1], packet_data)) # assembly
packets
    else:
        return ",".join(errors)

def handle_message(algorithm, data):
    answer = solve(data, algorithm)
    return answer
```

*Проверка*

```
import random
import base64
import os
import re
import zlib

def nth_index(n, value, iterable):
    i = -1
    for j in range(n):
        i = iterable.index(value, i + 1)
    return i

MIN_STR_LEN = 1
MAX_STR_LEN = 1 * (10 ** 3)

NUM_TESTS_GENERAL = 20
NUM_TESTS_INDEXES = 20

MAX_PACKET_LEN = 32


def parity_byte_sum(b):
    parity = 0
    while b != 0:
        parity ^= b & 1
```

```
        b >>= 1
    return parity


def parity_sum(s):
    parity_num = 0
    for i in range(len(s)):
        parity_num <<= 1
        parity_num |= parity_byte_sum(ord(s[i]))
    return parity_num


def crc32_sum(s):
    return zlib.crc32(s.encode())


def rand_weighted(minv, maxv, weight_coef):
    v = int(random.triangular(minv, maxv, minv + (maxv - minv) *
weight_coef))
    return max(v, minv)


def generate_str(min_len, max_len):
    NORM_COEF = 3
    l = rand_weighted(min_len, max_len / NORM_COEF, 0.1)

    s = base64.b64encode(os.urandom(l)).decode('ascii')
    pattern = re.compile('[/+=]+')
    return pattern.sub('', s)


SAMPLE_TESTS = [
    ("HelloWorld", 'p', 0, False),
    ("HelloWorld", 'p', 1, False),
    ("HelloWorld", 'c', 0, False),
    ("HelloWorld", 'c', 1, True),
]

ALGORITHMS = {
    'p': {'min_errors': 0, 'max_errors': 1, 'check': parity_sum},
    'c': {'min_errors': 0, 'max_errors': 5, 'check': crc32_sum}
}


def noise(char, num_errors):
    char = ord(char)
    i_actual_errors = min(8, num_errors)
    positions = [i for i in range(8)]
    actual_errors = i_actual_errors
    while actual_errors:
        index = random.randint(0, len(positions)-1)
        actual_errors -= 1
        char ^= 1 << positions[index]
        del positions[index]
    char = chr(char)
    return char if char != '\0' and char != ',' and char != '.' else
'a', num_errors - i_actual_errors
```

```
def make_packet(string_slice, packet_index, algorithm,
none_or_number_of_packets, fixed_errors, break_index):
    if none_or_number_of_packets is None:
        spacket = "%d.%s" % (packet_index, string_slice)
    else:
        spacket = "%d.%d.%s" % (packet_index,
none_or_number_of_packets, string_slice)
    spacket = "%d.%s" %
(ALGORITHMS[algorithm]['check'](str(packet_index)), spacket)
    packet = "%d.%s" % (ALGORITHMS[algorithm]['check'](spacket),
spacket)
    min_errors = min(len(string_slice),
ALGORITHMS[algorithm]['min_errors'])
    max_errors = min(len(string_slice),
ALGORITHMS[algorithm]['max_errors'])
    num_errors = random.randint(min_errors, max_errors) if
fixed_errors is None else fixed_errors
    c_num_errors = num_errors

    new_chars = []
    packet_dot_index = nth_index(2, '.', packet)
    packet_data_dot_index = nth_index(3, '.', packet)
    while c_num_errors != 0:
        to_break = None
        while to_break is None or packet[to_break] == '.':
            if break_index:
                to_break = random.randint(packet_dot_index + 1,
packet_data_dot_index - 1)
            else:
                #if random.randint(0, 1):
                    to_break = random.randint(0, packet_dot_index - 1)
                #else:
                 #   to_break = random.randint(packet_data_dot_index +
1, len(packet)-1)
        new_char, c_num_errors = noise(packet[to_break], c_num_errors)
        new_chars.append((new_char, to_break))
    packet = list(packet)
    for nc in new_chars:
        packet[nc[1]] = nc[0]

    return "".join(packet), num_errors > 0


def generate_test(test_set):
    s = test_set[0]
    algo = test_set[1]
    l = len(s)
    if l == 0:
        raise ValueError("Length must be not less than 1")
    packets = []
    unordered_errors = 0
    ordered_errors = []
    first_slice = random.randint(1, min(MAX_PACKET_LEN,l))
    slc = first_slice
    l -= slc
    pos = 0
    i = 1
```

```
    while l != 0:
        pos += slc
        slc = random.randint(1, min(MAX_PACKET_LEN,l))
        l -= slc
        str_slice = s[pos:pos + slc]
        packet, has_errors = make_packet(str_slice, i, algo, None,
test_set[2], test_set[3])
        packets.append(packet)
        if has_errors:
            if not test_set[3]:
                ordered_errors.append(i)
            else:
                unordered_errors += 1
        i += 1

    str_slice = s[:first_slice]
    packet, has_errors = make_packet(str_slice, 0, algo,
len(packets)+1, test_set[2], test_set[3])
    packets.append(packet)
    if has_errors:
        if not test_set[3]:
            ordered_errors.append(0)
        else:
            unordered_errors += 1
    random.shuffle(packets)
    if unordered_errors == 0 and len(ordered_errors) == 0:
        answer = "0%s" % s
    else:
        answer = "1%d.%s" % (unordered_errors, ",".join(map(lambda x:
str(x), ordered_errors)))

    return ("%s %s" % (algo, ",".join(packets)), answer)


def generate_test_set(break_index):
    algo = random.choice(['p', 'c'])
    return (generate_str(MIN_STR_LEN, MAX_STR_LEN), algo, None,
break_index)


def generate():
    tests = []
    for sample in SAMPLE_TESTS:
        tests.append(generate_test(sample))
    for i in range(NUM_TESTS_GENERAL):
        tests.append(generate_test(generate_test_set(True)))
    for i in range(NUM_TESTS_INDEXES):
        tests.append(generate_test(generate_test_set(False)))

    return tests

def solve(dataset):
    algo, packets = dataset.split(" ", maxsplit=1)
    packets = packets.split(',')
    errors = []
    packet_data = []

    for p in packets:
```

```
        d_checksum, d_content = p.split('.', maxsplit=1)
        checksum = ALGORITHMS[algo]["check"](d_content)
        i_checksum, index, d_content = d_content.split('.',
maxsplit=2)
        index_check = ALGORITHMS[algo]["check"](index)
        index_correct = str(index_check) == i_checksum
        correct = str(checksum) == d_checksum and index_correct
        if not index_correct:
            # check checksum with our checksum
            s = "%d.%s.%s" % (index_check, index, d_content)
            checksum = ALGORITHMS[algo]["check"](s)
            if str(checksum) == d_checksum:
                errors.append(str(index))
            else:
                errors.append('-')
        elif not correct:
            errors.append(str(index))
        # head
        d_content = d_content.split('.')
        if len(errors) == 0:
            packet_data.append((index, d_content[len(d_content)-1]))
    if not len(errors):
        packet_data.sort(key=lambda x: x[0])
        return "".join(map(lambda x: x[1], packet_data))
    else:
        return ",".join(errors)

def check(reply, clue):
    if clue[0] == '0':
        return True if reply == clue[1:] else False, "Исходное
сообщение не совпадает с ответом."
    else:
        # check number of errors
        non_ordered, ordered = clue[1:].split('.')
        non_ordered = int(non_ordered)
        ordered = ordered.split(',')
        if len(ordered) == 1 and ordered[0] == '':
            ordered = []
        else:
            ordered = list(map(lambda x: int(x), ordered))
        reply_non_ordered = 0
        reply_errors = reply.split(',')
        if len(reply_errors) != (non_ordered + len(ordered)):
            return False, "Число найденных пакетов с ошибками не
совпадает с ответом."
        for reply_error in reply_errors:
            if reply_error == '-':
                reply_non_ordered += 1
            else:
                try:
                    idx = ordered.index(int(reply_error))
                    del ordered[idx]
                except ValueError:
                    return False, "Индекс пакета с ошибкой не
присутствует в ответе."
        if len(ordered) != 0:
            return False, "Один или несколько индексов пакетов с
ошибками не присутствуют в ответе."
```

```
        return True if reply_non_ordered == non_ordered \
                    else False, "Найденное число пакетов с ошибками в
индексе не равно числу в ответе."
```

*Файловые шаблоны:*

```
::python3
::code
def handle_message(algorithm, data):
    answer = ""
    # your code here
    return answer

::footer
import sys

def wrapper():
    algorithm, data = sys.stdin.read().split(" ", 1)
    print(handle_message(algorithm, data))

wrapper()

::java8

::header
import java.io.BufferedReader;
import java.io.Console;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

class Test
{

::code
static String handleMessage(String algorithm, String data) {
    //your code goes here
    return "";
}
::footer

    public static void main(String[] args) throws IOException {
        BufferedReader buffer=new BufferedReader(new
InputStreamReader(System.in));

        String input = buffer.readLine();
        String[] array = input.split(" ", 2);
        System.out.print(handleMessage(array[0], array[1]));
    }
}


::c++11
::code

#include <string>
#include <vector>
```

```
std::string handle_message(const std::string &algorithm, const
std::string &data) {
    std::string handled_message;
    //your code goes here
    return handled_message;
}

::header

#include <random>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <assert.h>

::footer
int main() {
    std::string algorithm, data;

    std::cin >> algorithm >> data;
    std::cout << handle_message(algorithm, data);

    return 0;
}

::mono c#
::header

using System;
using System.Collections.Generic;
using System.Text;

    class Test
    {

::code
static string HandleMessage(string data, string algorithm) {
    //your code goes here
    return "";
}
::footer
        static void Main(string[] args)
        {
           string input = Console.ReadLine();
           string[] array = input.Split(' ');
           Console.Write(HandleMessage(array[0], array[1]));
        }
    }
```

**Задача 2.1.4 (4 балла)**

Передача данных по каналу с шумами.

После проверки данных принимающая сторона может запросить повторную передачу данных.

Повторная передача данных с ошибками является эффективным протоколом обеспечения доставки данных в случаях, когда пропускная способность канала достаточно высока, стоимость передачи данных низка и помехи в канале считаются малыми. Например, в оптоволоконных каналах связи.

Однако повторная передача неприменима для каналов с нестабильной связью, так как вероятность получения ошибочного пакета с каждой повторной новой передачей растет экспоненциально и зависит от объёма шума в канале по отношению к полезному сигналу:

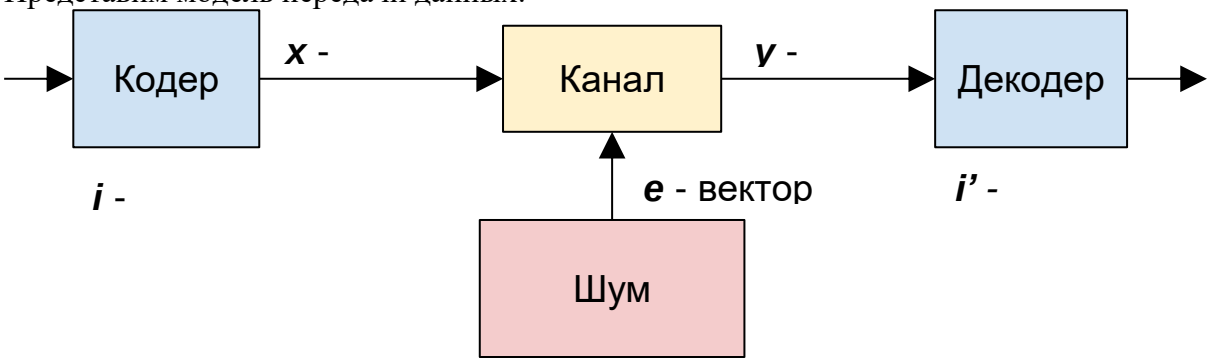
$$p_{\ received\ error}\ (n) = 1\ - {p_{no\ errors}}^{n}, n\ - \text{номер повторной передачи.}$$

$p_{no\ errors} < 1 -$ вероятность получения пакета без потерь и ошибок.

В таких каналах необходимо использовать помехоустойчивые алгоритмы кодирования и декодирования.

//

Представим модель передачи данных:




Пару в виде кодера и декодера называют **кодом.**

Кодер можно представить в виде функции $f(i): i \to x$, которая ставит в соответствие отправляемому через кодер слову ***i***, называемому **информационным**, некоторое другое слово ***x***, называемое **кодовым**.

Канал передачи с шумом представляется в виде функции $e(x): x \to y$, которая сопоставляет кодовому слову x другое слово y, которое можно описать как поэлементную сумму вектора ошибок и кодового слова:

$$\underline{y} = \underline{x} + \underline{e} = \{ x_0 + e_0, x_1 + e_1, \dots, x_i + e_i, \dots \}$$

Для двоичного канала сумму можно заменить на “Исключающее ИЛИ”

$$\underline{y} = \underline{x} \oplus \underline{e} = \{ x_0 \oplus e_0, x_1 \oplus e_1, \dots, x_i \oplus e_i, \dots \}$$

Декодер представим в виде композиции функций $dc(y) = f^{-1}(c(y)): y \to i'$ , которая переводит принятое слово x в слово i’, также называемое кодовым.

Функция $f^{-1}(x'): x' \to i'$ называется функцией декодирования, а функция $c(y): y \to x'$ называется функцией коррекции. Функция коррекции “пытается” узнать значение e’ и, в общем случае, её можно представить таким образом: $c(y) = y - e' = x'$. Если значение $e'$ равно значению e, то $c(y) = y - e = x$ - то кодовое слово восстанавливается успешно, и $f^{-1}(x) = i$ - успешно восстанавливается отправленное слово. Значение вектора ошибок ***e*** неизвестно принимающей стороне.

Задача декодера в общем случае заключается в том, чтобы i и i’ совпадали как можно чаще при любых значениях вектора ошибок.

Стоит сказать, что декодер может работать в двух режимах: корректирующем и обнаруживающем.

Во время работы в обнаруживающем режиме декодер обнаруживает ошибки, но, в отличие от корректирующего режима, не может декодировать данные, если они были повреждены.

Существует правило для способности декодеров обнаруживать и корректировать ошибки: если кодер способен гарантированно исправлять не менее k ошибок, то он

может обнаруживать не менее 2k ошибок, верно и обратное.
Коды разделяются по виду передачи данных.
Коды можно разделить на поточные и блочные.
Блочный код - код фиксированной длины, то есть такой код, для которого кодеру и декодеру точно известен размер кодового слова. В случаях, когда длина информационного слова больше, чем длина кодового слова, оно разбивается на несколько блоков. А когда меньше - дополняется “нулевым” блоком.
Для поточных кодов декодеру неизвестен объём принимаемых данных, что может привести к большему количеству дополнительных данных, необходимых для декодирования.

Задача декодирования применима не только к передаче данных, но и к её хранению. Например, для оптических дисков задача декодирования была одной самых из главных при их разработке и использовании. Источником шума являлись царапины и оптические аберрации, мешающие считывать поверхность диска. В алгоритмах исправления ошибок во время хранения данных в оперативной памяти помехами может считаться электромагнитное излучение источников питания и соседних электрических схем.
Эта задача до сих пор является ограничивающим фактором для развития каналов связи.

Попытаемся решить задачу декодирования.
Самым простым её решением является многократное повторение символов в информационном слове. Декодирование такого слова при этом происходит мажоритарным алгоритмом: каких символов в повторениях больше.
Например: информационное слово $i = 1011_2, d = 5$- количество повторений на каждый символ.
Кодовым словом тогда является слово $x = 11111.00000.11111.11111_2$.
Алгоритм декодирования очевиден: для каждых 5 символов подсчитывается количество “1” и “0”. В качестве декодированного символа выбирается тот, который яв. Например:
Принятое слово $y = 10111.00110.11111.11101_2$, для первого символа:
$y_1 = 10111, n_1 = 4, n_0 = 1 \Rightarrow i_1 = 1$

Для остальных символов аналогично.

Такой код гарантированно обнаруживает до 4 ошибок и гарантированно восстанавливает не менее двух ошибок в любом месте.

В общем случае можно сказать, что данный код восстанавливает $k = [\frac{d-1}{2}]$ ошибок в любом месте и обнаруживает до d-1 ошибок в любом месте, что соответствует правилу для способности декодеров обнаруживать и корректировать ошибки.
Введем понятие *скорости кода.*
Скорость кода для слова i - отношение числа информационных символов к числу символов соответствующего кодового слова.
Для кода для всех слов определяется по-разному в зависимости от условий. Определим скорость кода как среднее для всех возможных информационных слов длины n:

$$c = \frac{1}{q^n} \sum_{i \epsilon P_q^n} \frac{|i|}{|f(i)|},$$

где n - длина слов, q - размер алфавита (для двоичного канала q=2), $P_q^n$- множество всех слов длины n, $|i|$- длина слова i, |f(i)| - длина кодового слова.

Очевидно, что для алгоритма кодирования повторениями скоростью кода является $c = \frac{1}{d}$, и для исправления k ошибок оно будет не более $c = \frac{1}{2k+1}$.
Для примера выше $c = \frac{1}{5}$. Для исправления 10 ошибок доля полезных данных в закодированном слове уже не превышает 5%.
Для применения в реальных системах такой алгоритм не подходит ещё и по другой

причине, связанной с тем, что большинство ошибок появляются сериями - ошибками, идущими подряд, количество которых может превышать теоретическую способность гарантированно восстанавливать данные.

Для алгоритм кодирования повторениями в этом случае невозможно восстановить данные без потерь, так как один или несколько символов будут полностью повреждены. Однако существуют и применяются алгоритмы кодирования, для которых вероятность восстановить данные даже в этом случае не равна нулю.

Одним из самокорректирующихся алгоритмов является код Хэмминга. Несмотря на его характеристики: возможность исправлять только одну ошибку и обнаруживать только две, он обладает теоретически возможной минимальной избыточностью: $l = log_2 n + 1$ - дополнительных символов требуется для кодирования слова длины n.

Сам алгоритм является измененным алгоритмом проверки на чётность. Информационное слово разбивается на блоки длины n. Каждый блок кодируется и декодируется отдельно от всех остальных.

Алгоритм кодирования для двоичного алфавита состоит в следующем:

1) К слову добавляются $log_2 n + 1$дополнительных символов $p_k$, предварительно равных нулю, в позиции $2^k, k \in [0, log_2 n]$.
2) Затем составляется матрица A размера $[log_2 n + 1, n + log_2 n + 1]$, каждый столбец которой - двоичное представление текущего текущего номера столбца.
3) Каждый символ $p_k$равен количеству единиц в информационном слове, которое умножено поэлементно с k-ой строкой матрицы A по модулю 2: $p_k = |A_k \cdot i| \, mod \, 2$.

Алгоритм декодирования аналогичен алгоритму кодирования:

1) Вычисляются новые символы $p_k'$аналогично шагу 3 из алгоритма кодирования.
2) Если все значения новых символов равны нулю, ошибки при передаче не было и перейти к шагу 4, иначе перейти к шагу 3.
3) Запись символов

   $p_{log_2 n} \oplus p'_{log_2 n}, p_{log_2 n - 1} \oplus p'_{log_2 n - 1}, p_k \oplus p'_k, \ldots, p_1 \oplus p'_1$ является двоичным представлением позиции ошибки в кодовом слове - при инвертировании символа получится истинное значение этого бита.
4) Из кодового слова удаляются дополнительные символы.

Например:

Пусть дано слово из 8 символов: $i = 01101111_2$

1) Добавим к нему три дополнительных символа, равных 0, в позиции 1, 2, 4, 8:
   a) $i = 0001.1001.1110_2$
2) Составим матрицу A размера [3, 12], составленную из номеров стоблцов. (номера начинаются с 1):

   1 0 1 0 1 0 1 0 1 0 1 0 p1
   0 1 1 0 0 1 1 0 0 1 1 0 p2
   0 0 0 1 1 1 1 0 0 0 0 1 p3
   0 0 0 0 0 0 0 1 1 1 1 1 p4
3) Вычислим все значения символов $p_1 \ldots p_4$:
   a) (0*1)+(0*0)+(0*1)+(1*0)+(1*1)+(0*0)+(0*1)+(1*0)+(1*0)+(1*1)+(1*0)+(0*1) = p1 = 0
   b) (0*0)+(0*1)+(0*1)+(1*0)+(1*0)+(0*1)+(0*1)+(1*0)+(1*0)+(1*1)+(1*1)+(0*0) = p2 = 0
   c) (0*0)+(0*0)+(0*0)+(1*1)+(1*1)+(0*1)+(0*1)+(1*0)+(1*0)+(1*0)+(1*0)+(0*1) = p3 = 0
   d) (0*0)+(0*0)+(0*0)+(1*0)+(1*0)+(0*0)+(0*0)+(1*1)+(1*1)+(1*1)+(1*1)+(0*1)

= p4 = 0

e) Закодированное слово: $x = 0001.1001.1110_2$

Пусть в позиции 7 произошла ошибка: $y = 0001.1011.1110_2$. Декодируем это слово:

1) Составим матрицу A размера [3, 12], составленную из номеров стоблцов. (номера начинаются с 1):

1 0 1 0 1 0 1 0 1 0 1 0 p1
0 1 1 0 0 1 1 0 0 1 1 0 p2
0 0 0 1 1 1 1 0 0 0 0 1 p3
0 0 0 0 0 0 0 1 1 1 1 1 p4

2) Вычислим все значения символов $p_1 \dots p_4$:
   a) (0*1)+(0*0)+(0*1)+(1*0)+(1*1)+(0*0)+(1*1)+(1*0)+(1*0)+(1*1)+(1*0)+(0*1) = p1' = 1
   b) (0*0)+(0*1)+(0*1)+(1*0)+(1*0)+(0*1)+(1*1)+(1*0)+(1*0)+(1*1)+(1*1)+(0*0) = p2' = 1
   c) (0*0)+(0*0)+(0*0)+(1*1)+(1*1)+(0*1)+(1*1)+(1*0)+(1*0)+(1*0)+(1*0)+(0*1) = p3' = 1
   d) (0*0)+(0*0)+(0*0)+(1*0)+(1*0)+(0*0)+(1*0)+(1*1)+(1*1)+(1*1)+(1*1)+(0*1) = p4' = 0
   e) И номер позиции ошибки: $e = (0 \oplus 0)(0 \oplus 1)(0 \oplus 1)(0 \oplus 1) = 0111_2 = 7$

Такой алгоритм используется в тех случаях, когда возникающие ошибки либо редки и равновероятны, либо не являются критичными для работы системы, например в радиосвязи и в системе исправления ошибок оперативной памяти.

Существуют и другие коды: коды Рида-Маллера (Reed-Muller codes), коды Рида-Соломона (Reed-Solomon codes), коды Боуза-Чоудхури-Хоквингема (BCH codes), коды Гоппы (Goppa codes).

Каждый из этих кодов обладает некоторыми свойствами, которые позволяют находить им более практическое применение в разных областях. Например, БЧХ используются в сетях GSM, так как позволяют подстраиваться динамически под условия канала передачи. А коды Рида-Маллера использовались для кодирования оптических дисков, так как устойчивы к сериям ошибок, например, при длинных царапинах дисков.

На практике часто применяются реализации *каскадных кодов* - коды, состоящие из нескольких других кодов.

Они строятся каскадно: каждый код, кроме первого и последнего, принимает на вход кодовое слово предыдущего кода, а выход текущего кода принимается на вход следующего кода. Выход последнего кода - выход каскадного кода, а вход первого кода - вход каскадного кода.

Такие коды позволяют комбинировать полезные свойства разных кодов.

*Условие:*

В этой задаче вам необходимо разработать свой или применить существующий код для стабильной передачи данных в канале с шумом. Шум генерируется случайным образом с некоторой вероятностью, которая заранее известна. Вам нужно предоставить функцию, которая в зависимости от режима, кодирует или декодирует принятые данные. Гарантируется, что уровень шума постоянен для одних и тех же данных в режиме кодирования и декодирования.

Ограничение по времени: 2 сек.

Ограничение по памяти: 256 МБ.

**Формат данных**

На вход подаются три параметра в следующем порядке: обозначение алгоритма (mode),

уровень шума (noise_level), данные (data).
Режим работы (mode): строка, принимает значения “decode” и “encode”.
Уровень шума (noise_level): число с плавающей точкой, вероятность инвертирования бита, $|noise\_level - 0.5| \leq 0.3$
Входные данные (data):

- В режиме работы “encode”: строка, состоящая из цифр и букв латинского алфавита. Длина строки $l$: $1 \leq l \leq 4 * 10^6$
- В режиме работы “decode”: строка, обработанная ранее вызовом функции в режиме encode, которая может содержать ошибки при передаче данных. Может содержать любые символы, кроме нулевого.

Примечание: канал связи не изменяет пробельные символы, однако уменьшает количество идущих подряд символов до 1. Пробельные символы также участвуют в оценке скорости кода.

**Целевые параметры**

Задание считается пройденным тогда и только тогда, когда:

- Скорость кодирования $c \geq 0.2$
- Отношение числа пройденных тестов к общему количеству тестов $k \geq 0.8$
- Ни в одном тесте не было допущено больше 2 ошибок декодирования.

**Тестирование**

Для тестирования собственных кодов вы можете использовать написанную для этого программу:
https://github.com/FakeEmperor/NTIChannelTester/raw/master/bin/ChannelTester.exe . Если вы не используете операционную систему семейства Microsoft, вы можете скомпилировать программу из исходного кода, написанного на C++:
https://github.com/FakeEmperor/NTIChannelTester .
Цель программы: облегчить тестирование алгоритма и нахождение его параметров. Программа использует файловый интерфейс для тестирования работы алгоритма - результаты работы вашего кода вам необходимо записывать в файл. Программа может генерировать входные данные для работы вашего алгоритма в режиме кодера и декодера.

**Пример использования программы для генерации входных данных для режима кодирования:**
*channel_tester.exe* ***-g -io_source*** *source.txt* ***-noise_levels*** *0,0.2,0.9* ***-num_sources*** *3* ***-max_source_size*** *100*

Эти параметры указывают, что необходимо сгенерировать 3 тестовых входных данных на каждый уровень шума, уровни шума: 0, 0.2 и 0.9, каждое входное слово имеет длину не больше 100. Результат будет записан в файл source.txt. Каждый набор входных данных разделен новой строкой.

**Пример использования программы для генерации входных данных для режима декодирования:**
*channel_tester.exe* ***-s -io_source*** *source.txt* ***-in_encoded*** *encoded.txt* ***-io_noised*** *noised.txt*

Эти параметры указывают, что необходимо сгенерировать 3 тестовых входных данных для режима декодирования, используя закодированные данные из файла encoded.txt входные данные из файла source.txt. Результат будет записан в файл noised.txt. Каждый набор входных данных разделен новой строкой.

**Пример использования программы для тестирования работы алгоритма:**
*channel_tester.exe* ***-d -io_source*** *source.txt* ***-in_decoded*** *decoded.txt* ***-io_noised*** *noised.txt* ***-io_encoded*** *encoded.txt* ***-out_report*** *report.txt*

Эти параметры указывают, что необходимо протестировать декодирования алгоритма и собрать статистику его работы, используя результаты из файла decoded.txt, исходные, зашумлённые и закодированные данные из файлов source.txt, noised.txt, encoded.txt, соответственно. Результат тестирования будет записан в файл report.txt.

Примечание: каждая строка разделяется строго символами “\r\n”. Пожалуйста, учтите это при чтении входных файлов и записи результатов работы своей программы.

*Решение:*

```
# REED-SOLOMON CODE
# IMPORTED FROM @link
http://web.mit.edu/~emin/www.old/source_code/py_ecc

# Copyright Emin Martinian 2002.  See below for license terms.
# Version Control Info: $Id: ffield.py,v 1.5 2007/03/15 02:49:44 emin
Exp $

import string, random, os, os.path, cPickle

# The following list of primitive polynomials are the Conway
Polynomials
# from the list at
# http://www.math.rwth-aachen.de/~Frank.Luebeck/ConwayPol/cp2.html

gPrimitivePolys = {}
gPrimitivePolysCondensed = {
    1  : (1,0),
    2  : (2,1,0),
    3  : (3,1,0),
    4  : (4,1,0),
    5  : (5,2,0),
    6  : (6,4,3,1,0),
    7  : (7,1,0),
    8  : (8,4,3,2,0),
    9  : (9,4,0),
    10 : (10,6,5,3,2,1,0),
    11 : (11,2,0),
    12 : (12,7,6,5,3,1,0),
    13 : (13,4,3,1,0),
    14 : (14,7,5,3,0),
    15 : (15,5,4,2,0),
    16 : (16,5,3,2,0),
    17 : (17,3,0),
    18 : (18,12,10,1,0),
    19 : (19,5,2,1,0),
    20 : (20,10,9,7,6,5,4,1,0),
    21 : (21,6,5,2,0),
    22 : (22,12,11,10,9,8,6,5,0),
    23 : (23,5,0),
    24 : (24,16,15,14,13,10,9,7,5,3,0),
    25 : (25,8,6,2,0),
    26 : (26,14,10,8,7,6,4,1,0),
```

```
    27 : (27,12,10,9,7,5,3,2,0),
    28 : (28,13,7,6,5,2,0),
    29 : (29,2,0),
    30 : (30,17,16,13,11,7,5,3,2,1,0),
    31 : (31,3,0),
    32 : (32,15,9,7,4,3,0),
    33 : (33,13,12,11,10,8,6,3,0),
    34 : (34,16,15,12,11,8,7,6,5,4,2,1,0),
    35 : (35, 11, 10, 7, 5, 2, 0),
    36 : (36, 23, 22, 20, 19, 17, 14, 13, 8, 6, 5, 1, 0),
    37 : (37, 5, 4, 3, 2, 1, 0),
    38 : (38, 14, 10, 9, 8, 5, 2, 1, 0),
    39 : (39, 15, 12, 11, 10, 9, 7, 6, 5, 2 , 0),
    40 : (40, 23, 21, 18, 16, 15, 13, 12, 8, 5, 3, 1, 0),
    97 : (97,6,0),
    100 : (100,15,0)
    }

for n in gPrimitivePolysCondensed.keys():
    gPrimitivePolys[n] = [0]*(n+1)
    if (n < 16):
        unity = 1
    else:
        unity = long(1)
    for index in gPrimitivePolysCondensed[n]:
        gPrimitivePolys[n][index] = unity
    gPrimitivePolys[n].reverse()


class FField:
    def __init__(self,n,gen=0,useLUT=-1):


        self.n = n
        if (gen):
            self.generator = gen
        else:
            self.generator =
self.ConvertListToElement(gPrimitivePolys[n])


        if (useLUT == 1 or (useLUT == -1 and self.n < 10)): # use
lookup table
            self.unity = 1
            self.Inverse = self.DoInverseForSmallField
            self.PrepareLUT()
            self.Multiply = self.LUTMultiply
            self.Divide = self.LUTDivide
            self.Inverse = lambda x: self.LUTDivide(1,x)
        elif (self.n < 15):
            self.unity = 1
            self.Inverse = self.DoInverseForSmallField
            self.Multiply = self.DoMultiply
            self.Divide = self.DoDivide
        else: # Need to use longs for larger fields
            self.unity = long(1)
            self.Inverse = self.DoInverseForBigField
            self.Multiply = lambda a,b:
```

```
self.DoMultiply(long(a),long(b))
            self.Divide = lambda a,b: self.DoDivide(long(a),long(b))



    def PrepareLUT(self):
        fieldSize = 1 << self.n
        lutName = 'ffield.lut.' + `self.n`
        if (os.path.exists(lutName)):
            fd = open(lutName,'r')
            self.lut = cPickle.load(fd)
            fd.close()
        else:
            self.lut = LUT()
            self.lut.mulLUT = range(fieldSize)
            self.lut.divLUT = range(fieldSize)
            self.lut.mulLUT[0] = [0]*fieldSize
            self.lut.divLUT[0] = ['NaN']*fieldSize
            for i in range(1,fieldSize):
                self.lut.mulLUT[i] = map(lambda x:
self.DoMultiply(i,x),
                                         range(fieldSize))
                self.lut.divLUT[i] = map(lambda x: self.DoDivide(i,x),
                                         range(fieldSize))
            fd = open(lutName,'w')
            cPickle.dump(self.lut,fd)
            fd.close()


    def LUTMultiply(self,i,j):
        return self.lut.mulLUT[i][j]

    def LUTDivide(self,i,j):
        return self.lut.divLUT[i][j]

    def Add(self,x,y):
        """
        Adds two field elements and returns the result.
        """

        return x ^ y

    def Subtract(self,x,y):
        """
        Subtracts the second argument from the first and returns
        the result.  In fields of characteristic two this is the same
        as the Add method.
        """
        return self.Add(x,y)

    def DoMultiply(self,f,v):
        """
        Multiplies two field elements (modulo the generator
        self.generator) and returns the result.

        See MultiplyWithoutReducing if you don't want multiplication
        modulo self.generator.
        """
```

```
        m = self.MultiplyWithoutReducing(f,v)
        return self.FullDivision(m,self.generator,
                                 self.FindDegree(m),self.n)[1]

    def DoInverseForSmallField(self,f):
        """
        Computes the multiplicative inverse of its argument and
        returns the result.
        """
        return self.ExtendedEuclid(1,f,self.generator,
                                   self.FindDegree(f),self.n)[1]

    def DoInverseForBigField(self,f):
        """
        Computes the multiplicative inverse of its argument and
        returns the result.
        """
        return self.ExtendedEuclid(self.unity,long(f),self.generator,
                                   self.FindDegree(long(f)),self.n)[1]

    def DoDivide(self,f,v):
        """
        Divide(f,v) returns f * v^-1.
        """
        return self.DoMultiply(f,self.Inverse(v))

    def FindDegree(self,v):
        """
        Find the degree of the polynomial representing the input field
        element v.  This takes O(degree(v)) operations.

        A faster version requiring only O(log(degree(v)))
        could be written using binary search...
        """

        if (v):
            result = -1
            while(v):
                v = v >> 1
                result = result + 1
            return result
        else:
            return 0

    def MultiplyWithoutReducing(self,f,v):
        """
        Multiplies two field elements and does not take the result
        modulo self.generator.  You probably should not use this
        unless you know what you are doing; look at Multiply instead.

        NOTE: If you are using fields larger than GF(2^15), you should
        make sure that f and v are longs not integers.
        """

        result = 0
        mask = self.unity
        i = 0
        while (i <= self.n):
```

```
            if (mask & v):
                result = result ^ f
            f = f << 1
            mask = mask << 1
            i = i + 1
        return result


    def ExtendedEuclid(self,d,a,b,aDegree,bDegree):
        """
        Takes arguments (d,a,b,aDegree,bDegree) where d = gcd(a,b)
        and returns the result of the extended Euclid algorithm
        on (d,a,b).
        """
        if (b == 0):
            return (a,self.unity,0)
        else:
            (floorADivB, aModB) =
self.FullDivision(a,b,aDegree,bDegree)
            (d,x,y) = self.ExtendedEuclid(d, b, aModB, bDegree,
                                          self.FindDegree(aModB))
            return
(d,y,self.Subtract(x,self.DoMultiply(floorADivB,y)))

    def FullDivision(self,f,v,fDegree,vDegree):
        """
        Takes four arguments, f, v, fDegree, and vDegree where
        fDegree and vDegree are the degrees of the field elements
        f and v represented as a polynomials.
        This method returns the field elements a and b such that

            f(x) = a(x) * v(x) + b(x).

        That is, a is the divisor and b is the remainder, or in
        other words a is like floor(f/v) and b is like f modulo v.
        """

        result = 0
        i = fDegree
        mask = self.unity << i
        while (i >= vDegree):
            if (mask & f):
                result = result ^ (self.unity << (i - vDegree))
                f = self.Subtract(f, v << (i - vDegree))
            i = i - 1
            mask = mask >> self.unity
        return (result,f)


    def ShowCoefficients(self,f):
        """
        Show coefficients of input field element represented as a
        polynomial in decreasing order.
        """

        fDegree = self.n

        result = []
```

```
        for i in range(fDegree,-1,-1):
            if ((self.unity << i) & f):
                result.append(1)
            else:
                result.append(0)

        return result

    def ShowPolynomial(self,f):
        """
        Show input field element represented as a polynomial.
        """

        fDegree = self.FindDegree(f)
        result = ''

        if (f == 0):
            return '0'

        for i in range(fDegree,0,-1):
            if ((1 << i) & f):
                result = result + (' x^' + `i`)
        if (1 & f):
            result = result + ' ' + `1`
        return string.replace(string.strip(result),' ',' + ')

    def GetRandomElement(self,nonZero=0,maxDegree=None):
        """
        Return an element from the field chosen uniformly at random
        or, if the optional argument nonZero is true, chosen uniformly
        at random from the non-zero elements, or, if the optional
argument
        maxDegree is provided, ensure that the result has degree less
        than maxDegree.
        """

        if (None == maxDegree):
            maxDegree = self.n
        if (maxDegree <= 1 and nonZero):
            return 1
        if (maxDegree < 31):
            return random.randint(nonZero != 0,(1<<maxDegree)-1)
        else:
            result = 0L
            for i in range(0,maxDegree):
                result = result ^ (random.randint(0,1) << long(i))
            if (nonZero and result == 0):
                return self.GetRandomElement(1)
            else:
                return result



    def ConvertListToElement(self,l):
        """
        This method takes as input a binary list (e.g. [1, 0, 1, 1])
        and converts it to a decimal representation of a field
element.
```

```
        For example, [1, 0, 1, 1] is mapped to 8 | 2 | 1 = 11.

        Note if the input list is of degree >= to the degree of the
        generator for the field, then you will have to call take the
        result modulo the generator to get a proper element in the
        field.
        """

        temp = map(lambda a, b: a << b, l, range(len(l)-1,-1,-1))
        return reduce(lambda a, b: a | b, temp)

    def TestFullDivision(self):
        """
        Test the FullDivision function by generating random
polynomials
        a(x) and b(x) and checking whether (c,d) == FullDivision(a,b)
        satsifies b*c + d == a
        """
        f = 0

        a = self.GetRandomElement(nonZero=1)
        b = self.GetRandomElement(nonZero=1)
        aDegree = self.FindDegree(a)
        bDegree = self.FindDegree(b)

        (c,d) = self.FullDivision(a,b,aDegree,bDegree)
        recon = self.Add(d, self.Multiply(c,b))
        assert (recon == a), ('TestFullDivision failed: a='
                              + `a` + ', b=' + `b` + ', c='
                              + `c` + ', d=' + `d` + ', recon=',
recon)

    def TestInverse(self):
        """
        This function tests the Inverse function by generating
        a random non-zero polynomials a(x) and checking if
        a * Inverse(a) == 1.
        """

        a = self.GetRandomElement(nonZero=1)
        aInv = self.Inverse(a)
        prod = self.Multiply(a,aInv)
        assert 1 == prod, ('TestInverse failed:' + 'a=' + `a` + ',
aInv='
                           + `aInv` + ', prod=' + `prod`,
                           'gen=' + `self.generator`)

class LUT:
    """
    Lookup table used to speed up some finite field operations.
    """
    pass


class FElement:

    def __init__(self,field,e):
        """
```

```
        The constructor takes two arguments, field, and e where
        field is an FField object and e is an integer representing
        an element in FField.

        The result is a new FElement instance.
        """
        self.f = e
        self.field = field

    def __add__(self,other):
        assert self.field == other.field
        return FElement(self.field,self.field.Add(self.f,other.f))

    def __mul__(self,other):
        assert self.field == other.field
        return
FElement(self.field,self.field.Multiply(self.f,other.f))

    def __mod__(self,o):
        assert self.field == o.field
        return FElement(self.field,
                        self.field.FullDivision(self.f,o.f,

self.field.FindDegree(self.f),

self.field.FindDegree(o.f))[1])

    def __floordiv__(self,o):
        assert self.field == o.field
        return FElement(self.field,
                        self.field.FullDivision(self.f,o.f,

self.field.FindDegree(self.f),

self.field.FindDegree(o.f))[0])

    def __div__(self,other):
        assert self.field == other.field
        return FElement(self.field,self.field.Divide(self.f,other.f))

    def __str__(self):
        return self.field.ShowPolynomial(self.f)

    def __repr__(self):
        return self.__str__()

    def __eq__(self,other):
        assert self.field == other.field
        return self.f == other.f

def FullTest(testsPerField=10,sizeList=None):
    """
    This function runs TestInverse and TestFullDivision for
testsPerField
    random field elements for each field size in sizeList.  For
example,
    if sizeList = (1,5,7), then thests are run on GF(2), GF(2^5), and
    GF(2^7).  If sizeList == None (which is the default), then every
```

```
    field is tested.
    """

    if (None == sizeList):
        sizeList = gPrimitivePolys.keys()
    for i in sizeList:
        F = FField(i)
        for j in range(testsPerField):
            F.TestInverse()
            F.TestFullDivision()


fields_doc = """
Roughly speaking a finite field is a finite collection of elements
where most of the familiar rules of math work.  Specifically, you
can add, subtract, multiply, and divide elements of a field and
continue to get elements in the field.  This is useful because
computers usually store and send information in fixed size chunks.
Thus many useful algorithms can be described as elementary operations
(e.g. addition, subtract, multiplication, and division) of these
chunks.

Currently this package only deals with fields of characteristic 2.
That
is all fields we consider have exactly 2^p elements for some integer
p.
We denote such fields as GF(2^p) and work with the elements
represented
as p-1 degree polynomials in the indeterminate x.  That is an element
of
the field GF(2^p) looks something like

     f(x) = c_{p-1} x^{p-1} + c_{p-2} x^{p-2} + ... + c_0

where the coefficients c_i are in binary.

Addition is performed by simply adding coefficients of degree i
modulo 2.  For example, if we have two field elements f and v
represented as f(x) = x^2 + 1 and v(x) = x + 1 then s = f + v
is given by (x^2 + 1) + (x + 1) = x^2 + x.  Multiplication is
performed modulo a p degree generator polynomial g(x).
For example, if f and v are as in the above example, then s = s * v
is given by (x^2 + 1) + (x + 1) mod g(x).  Subtraction turns out
to be the same as addition for fields of characteristic 2.  Division
is defined as f / v = f * v^-1 where v^-1 is the multiplicative
inverse of v.  Multiplicative inverses in groups and fields
can be calculated using the extended Euclid algorithm.

Roughly speaking the intuition for why multiplication is
performed modulo g(x), is because we want to make sure s * v
returns an element in the field.  Elements of the field are
polynomials of degree p-1, but regular multiplication could
yield terms of degree greater than p-1.  Therefore we need a
rule for 'reducing' terms of degree p or greater back down
to terms of degree at most p-1.  The 'reduction rule' is
taking things modulo g(x).

For another way to think of
```

```
taking things modulo g(x) as a 'reduction rule', imagine
g(x) = x^7 + x + 1 and we want to take some polynomial,
f(x) = x^8 + x^3 + x, modulo g(x).  We can think of g(x)
as telling us that we can replace every occurence of
x^7 with x + 1.  Thus f(x) becomes x * x^7 + x^3 + x which
becomes x * (x + 1) + x^3 + x = x^3 + x^2 .  Essentially, taking
polynomials mod x^7 by replacing all x^7 terms with x + 1 will
force down the degree of f(x) until it is below 7 (the leading power
of g(x).  See a book on abstract algebra for more details.
"""

design_doc = """
The FField class implements a finite field calculator for fields of
characteristic two.  This uses a representation of field elements
as integers and has various methods to calculate the result of
adding, subtracting, multiplying, dividing, etc. field elements
represented AS INTEGERS OR LONGS.

The FElement class provides objects which act like a new kind of
numeric type (i.e. they overload the +,-,*,%,//,/ operators, and
print themselves as polynomials instead of integers).

Use the FField class for efficient storage and calculation.
Use the FElement class if you want to play around with finite
field math the way you would in something like Matlab or
Mathematica.

-----------------------------------------------------------------
                          WHY PYTHON?

You may wonder why a finite field calculator written in Python would
be useful considering all the C/C++/Java code already written to do
the same thing (and probably faster too).  The goals of this project
are as follows, please keep them in mind if you make changes:

o  Provide an easy to understand implementation of field operations.
   Python lends itself well to comments and documentation.  Hence,
   we hope that in addition to being useful by itself, this project
   will make it easier for people to implement finite field
   computations in other languages.  If you've ever looked at some
   of the highly optimized finite field code written in C, you will
   understand the need for a clear reference implementation of such
   operations.

o  Provide easy access to a finite field calculator.
   Since you can just start up the Python interpreter and do
   computations, a finite field calculator in Python lets you try
   things out, check your work for other algorithms, etc.
   Furthermore since a wealth of numerical packages exist for python,
   you can easily write simulations or algorithms which draw upon
   such routines with finite fields.

o  Provide a platform independent framework for coding in Python.
   Many useful error control codes can be implemented based on
   finite fields.  Some examples include error/erasure correction,
   cyclic redundancy checks (CRCs), and secret sharing.  Since
   Python has a number of other useful Internet features being able
   to implement these kinds of codes makes Python a better framework
```

for network programming.

o Leverages Python arbitrary precision code for large fields.
If you want to do computations over very large fields, for example GF(2^p) with p > 31 you have to write lots of ugly bit field code in most languages. Since Python has built in support for arbitrary precision integers, you can make this code work for arbitrary field sizes provided you operate on longs instead of ints. That is if you give as input numbers like 0L, 1L, 1L << 55, etc., most of the code should work.

---

BASIC DESIGN

The basic idea is to index entries in the finite field of interest using integers and design the class methods to work properly on this representation. Using integers is efficient since integers are easy to store and manipulate and allows us to handle arbitrary field sizes without changing the code if we instead switch to using longs.

Specifically, an integer represents a bit string

```
c = c_{p-1} c_{p-2} ... c_0.
```

which we interpret as the coefficients of a polynomial representing a field element

```
f(x) = c_{p-1} x^{p-1} + c_{p-2} x^{p-2} + ... + c_0.
```

---

FUTURE

In the future, support for fields of other characteristic may be added (if people want them). Since computers have built in parallelized operations for fields of characteristic two (i.e. bitwise and, or, xor, etc.), this implementation uses such operations to make most of the computations efficient.

"""

```
license_doc = """
  This code was originally written by Emin Martinian
(emin@allegro.mit.edu).
  You may copy, modify, redistribute in source or binary form as long
  as credit is given to the original author. Specifically, please
  include some kind of comment or docstring saying that Emin Martinian
  was one of the original authors. Also, if you publish anything
based
  on this work, it would be nice to cite the original author and any
  other contributers.

  There is NO WARRANTY for this software just as there is no warranty
  for GNU software (although this is not GNU software). Specifically
  we adopt the same policy towards warranties as the GNU project:

  BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT
```

```
WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER
EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK
AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY
SERVICING,
REPAIR OR CORRECTION.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR
DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT
LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED
BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY
OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.
"""

testing_doc = """
The FField class has a number of built in testing functions such as
TestFullDivision, TestInverse.  The simplest thing to
do is to call the FullTest method.

>>> import ffield
>>> ffield.FullTest(sizeList=None,testsPerField=100)

# To decrease the testing time you can either decrease the
testsPerField
# or you can only test the field sizes you care about by doing
something
# like sizeList = [2,7,20] in the ffield.FullTest command above.

If any problems occur, assertion errors are raised.  Otherwise
nothing is returned.  Note that you can also use the doctest
package to test all the python examples in the documentation
by typing 'python ffield.py' or 'python -v ffield.py' at the
command line.
"""


# The following code is used to make the doctest package
# check examples in docstrings.

__test__ = {
    'testing_doc' : testing_doc
}
```

```
# Copyright Emin Martinian 2002.  See below for license terms.
# Version Control Info: $Id: genericmatrix.py,v 1.5 2008-01-05
22:08:44 emin Exp $

"""
This package implements the GenericMatrix class to provide matrix
operations for any type that supports the multiply, add, subtract,
and divide operators.  For example, this package can be used to
do matrix calculations over finite fields using the ffield package
available at http://martinian.com.

The following docstrings provide detailed information on various
topics:

  GenericMatrix.__doc__   Describes the methods of the GenericMatrix
                          class and how to use them.

  license_doc             Describes the license and lack of warranty
                          for this code.

  testing_doc             Describes some tests to make sure the code
works.

"""

import operator

class GenericMatrix:

    """
    The GenericMatrix class implements a matrix with works with
    any generic type supporting addition, subtraction, multiplication,
    and division.  Matrix multiplication, addition, and subtraction
    are implemented as are methods for finding inverses,
    LU (actually LUP) decompositions, and determinants.  A complete
    list of user callable methods is:

    __init__
    __repr__
    __mul__
    __add__
    __sub__
    __setitem__
    __getitem__
    Size
    SetRow
    GetRow
    GetColumn
    Copy
    MakeSimilarMatrix
    SwapRows
    MulRow
    AddRow
    AddCol
    MulAddRow
    LeftMulColumnVec
```

```
    LowerGaussianElim
    Inverse
    Determinant
    LUP

    A quick and dirty example of how to use the GenericMatrix class
    for matricies of floats is provided below.

>>> import genericmatrix
>>> v = genericmatrix.GenericMatrix((3,3))
>>> v.SetRow(0,[0.0, -1.0, 1.0])
>>> v.SetRow(1,[1.0, 1.0, 1.0])
>>> v.SetRow(2,[1.0, 1.0, -1.0])
>>> v
<matrix
  0.0 -1.0  1.0
  1.0  1.0  1.0
  1.0  1.0 -1.0>
>>> vi = v.Inverse()
>>> vi
<matrix
  1.0  0.0  1.0
 -1.0  0.5 -0.5
 -0.0  0.5 -0.5>
>>> (vi * v) - v.MakeSimilarMatrix(v.Size(),'i')
<matrix
 0.0 0.0 0.0
 0.0 0.0 0.0
 0.0 0.0 0.0>

# See what happens when we try to invert a non-invertible matrix

>>> v[0,1] = 0.0
>>> v
<matrix
  0.0  0.0  1.0
  1.0  1.0  1.0
  1.0  1.0 -1.0>
>>> abs(v.Determinant())
0.0
>>> v.Inverse()
Traceback (most recent call last):
     ...
ValueError: matrix not invertible

# LUP decomposition will still work even if Inverse() won't.

>>> (l,u,p) = v.LUP()
>>> l
<matrix
 1.0 0.0 0.0
 0.0 1.0 0.0
 1.0 0.0 1.0>
>>> u
<matrix
  1.0  1.0  1.0
  0.0  0.0  1.0
  0.0  0.0 -2.0>
```

```
>>> p
<matrix
 0.0 1.0 0.0
 1.0 0.0 0.0
 0.0 0.0 1.0>
>>> p * v - l * u
<matrix
 0.0 0.0 0.0
 0.0 0.0 0.0
 0.0 0.0 0.0>

# Operate on some column vectors using v.
# The LeftMulColumnVec methods lets us do this without having
# to construct a new GenericMatrix to represent each column vector.
>>> v.LeftMulColumnVec([1.0,2.0,3.0])
[3.0, 6.0, 0.0]
>>> v.LeftMulColumnVec([1.0,-2.0,1.0])
[1.0, 0.0, -2.0]

# Most of the stuff above could be done with something like matlab.
# But, with this package you can do matrix ops for finite fields.
>>> XOR = lambda x,y: x^y
>>> AND = lambda x,y: x&y
>>> DIV = lambda x,y: x
>>> m =
GenericMatrix(size=(3,4),zeroElement=0,identityElement=1,add=XOR,mul=A
ND,sub=XOR,div=DIV)
>>> m.SetRow(0,[0,1,0,0])
>>> m.SetRow(1,[0,1,0,1])
>>> m.SetRow(2,[0,0,1,0])
>>> # You can't invert m since it isn't square, but you can still
>>> # get the LUP decomposition or solve a system of equations.
>>> (l,u,p) = v.LUP()
>>> p*v-l*u
<matrix
 0.0 0.0 0.0
 0.0 0.0 0.0
 0.0 0.0 0.0>
>>> b = [1,0,1]
>>> x = m.Solve(b)
>>> b == m.LeftMulColumnVec(x)
1

    """


    def __init__(self, size=(2,2), zeroElement=0.0,
identityElement=1.0,
                 add=operator.__add__, sub=operator.__sub__,
                 mul=operator.__mul__, div = operator.__div__,
                 eq = operator.__eq__, str=lambda x:`x`,
                 equalsZero = None,fillMode='z'):
        """
        Function:     __init__(size,zeroElement,identityElement,
                               add,sub,mul,div,eq,str,equalsZero
fillMode)

        Description:  This is the constructor for the GenericMatrix
```

```
                    class.  All arguments are optional and default
                    to producing a 2-by-2 zero matrix for floats.
                    A detailed description of arguments follows:

            size: A tuple of the form (numRows, numColumns)
                  zeroElement: An object representing the additive
                  identity (i.e. 'zero') for the data
                  type of interest.

            identityElement: An object representing the
multiplicative
                             identity (i.e. 'one') for the data
                             type of interest.

            add,sub,mul,div: Functions implementing basic arithmetic
                             operations for the type of interest.

            eq: A function such that eq(x,y) == 1 if and only if x ==
y.

            str: A function used to produce a string representation
of
                 the type of interest.

            equalsZero: A function used to decide if an element is
                        essentially zero.  For floats, you could use
                        lambda x: abs(x) < 1e-6.

            fillMode: This can either be 'e' in which case the
contents
                      of the matrix are left empty, 'z', in which
case
                      the matrix is filled with zeros, 'i' in which
                      case an identity matrix is created, or a two
                      argument function which is called with the row
                      and column of each index and produces the value
                      for that entry.  Default is 'z'.
        """
        if (None == equalsZero):
            equalsZero = lambda x: self.eq(self.zeroElement,x)

        self.equalsZero = equalsZero
        self.add = add
        self.sub = sub
        self.mul = mul
        self.div = div
        self.eq = eq
        self.str = str
        self.zeroElement = zeroElement
        self.identityElement = identityElement
        self.rows, self.cols = size
        self.data = []


        def q(x,y,z):
            if (x):
                return y
            else:
```

```
                return z

        if (fillMode == 'e'):
            return
        elif (fillMode == 'z'):
            fillMode = lambda x,y: self.zeroElement
        elif (fillMode == 'i'):
            fillMode = lambda x,y:
q(self.eq(x,y),self.identityElement,
                                      self.zeroElement)

        for i in range(self.rows):

self.data.append(map(fillMode,[i]*self.cols,range(self.cols)))

    def MakeSimilarMatrix(self,size,fillMode):
        """
        MakeSimilarMatrix(self,size,fillMode)

        Return a matrix of the given size filled according to fillMode
        with the same zeroElement, identityElement, add, sub, etc.
        as self.

        For example, self.MakeSimilarMatrix(self.Size(),'i') returns
        an identity matrix of the same shape as self.
        """
        return GenericMatrix(size=size,zeroElement=self.zeroElement,
                             identityElement=self.identityElement,
                             add=self.add,sub=self.sub,
                             mul=self.mul,div=self.div,eq=self.eq,

str=self.str,equalsZero=self.equalsZero,
                             fillMode=fillMode)


    def __repr__(self):
        m = 0
        # find the fattest element
        for r in self.data:
            for c in r:
                l = len(self.str(c))
                if l > m:
                    m = l
        f = '%%%ds' % (m+1)
        s = '<matrix'
        for r in self.data:
            s = s + '\n'
            for c in r:
                s = s + (f % self.str(c))
        s = s + '>'
        return s

    def __mul__(self,other):
        if (self.cols != other.rows):
            raise ValueError, "dimension mismatch"
        result = self.MakeSimilarMatrix((self.rows,other.cols),'z')

        for i in range(self.rows):
```

```
            for j in range(other.cols):
                result.data[i][j] = reduce(self.add,
                                           map(self.mul,self.data[i],
                                               other.GetColumn(j)))
        return result

    def __add__(self,other):
        if (self.cols != other.rows):
            raise ValueError, "dimension mismatch"
        result = self.MakeSimilarMatrix(size=self.Size(),fillMode='z')
        for i in range(self.rows):
            for j in range(other.cols):
                result.data[i][j] =
self.add(self.data[i][j],other.data[i][j])
        return result

    def __sub__(self,other):
        if (self.cols != other.cols or self.rows != other.rows):
            raise ValueError, "dimension mismatch"
        result = self.MakeSimilarMatrix(size=self.Size(),fillMode='z')
        for i in range(self.rows):
            for j in range(other.cols):
                result.data[i][j] = self.sub(self.data[i][j],
                                             other.data[i][j])
        return result

    def __setitem__ (self, (x,y), data):
        "__setitem__((x,y),data) sets item row x and column y to
data."
        self.data[x][y] = data

    def __getitem__ (self, (x,y)):
        "__getitem__((x,y)) gets item at row x and column y."
        return self.data[x][y]

    def Size (self):
        "returns (rows, columns)"
        return (len(self.data), len(self.data[0]))

    def SetRow(self,r,result):
        "SetRow(r,result) sets row r to result."

        assert len(result) == self.cols, ('Wrong # columns in row: ' +
                                          'expected ' + `self.cols` +
', got '
                                          + `len(result)`)
        self.data[r] = list(result)

    def GetRow(self,r):
        "GetRow(r) returns a copy of row r."
        return list(self.data[r])

    def GetColumn(self,c):
        "GetColumn(c) returns a copy of column c."
        if (c >= self.cols):
            raise ValueError, 'matrix does not have that many columns'
        result = []
        for r in self.data:
```

```
            result.append(r[c])
        return result

    def Transpose(self):
        oldData = self.data
        self.data = []
        for r in range(self.cols):
            self.data.append([])
            for c in range(self.rows):
                self.data[r].append(oldData[c][r])
        rows = self.rows
        self.rows = self.cols
        self.cols = rows

    def Copy(self):
        result = self.MakeSimilarMatrix(size=self.Size(),fillMode='e')

        for r in self.data:
            result.data.append(list(r))
        return result

    def SubMatrix(self,rowStart,rowEnd,colStart=0,colEnd=None):
        """
        SubMatrix(self,rowStart,rowEnd,colStart,colEnd)
        Create and return a sub matrix containg rows
        rowStart through rowEnd (inclusive) and columns
        colStart through colEnd (inclusive).
        """
        if (not colEnd):
            colEnd = self.cols-1
        if (rowEnd >= self.rows):
            raise ValueError, 'rowEnd too big: rowEnd >= self.rows'
        result = self.MakeSimilarMatrix((rowEnd-rowStart+1,colEnd-
colStart+1),
                                        'e')

        for i in range(rowStart,rowEnd+1):

result.data.append(list(self.data[i][colStart:(colEnd+1)]))

        return result

    def UnSubMatrix(self,rowStart,rowEnd,colStart,colEnd):
        """
        UnSubMatrix(self,rowStart,rowEnd,colStart,colEnd)
        Create and return a sub matrix containg everything except
        rows rowStart through rowEnd (inclusive)
        and columns colStart through colEnd (inclusive).
        """
        result = self.MakeSimilarMatrix((self.rows-(rowEnd-rowStart),
                                         self.cols-(colEnd-
colStart)),'e')

        for i in range(0,rowStart) + range(rowEnd,self.rows):
            result.data.append(list(self.data[i][0:colStart] +
                                    self.data[i][colEnd:]))

        return result
```

```
    def SwapRows(self,i,j):
        temp = list(self.data[i])
        self.data[i] = list(self.data[j])
        self.data[j] = temp

    def MulRow(self,r,m,start=0):
        """
        Function: MulRow(r,m,start=0)
        Multiply row r by m starting at optional column start (default
0).
        """
        row = self.data[r]
        for i in range(start,self.cols):
            row[i] = self.mul(row[i],m)

    def AddRow(self,i,j):
        """
        Add row i to row j.
        """
        self.data[j] = map(self.add,self.data[i],self.data[j])

    def AddCol(self,i,j):
        """
        Add column i to column j.
        """
        for r in range(self.rows):
            self.data[r][j] =
self.add(self.data[r][i],self.data[r][j])

    def MulAddRow(self,m,i,j):
        """
        Multiply row i by m and add to row j.
        """
        self.data[j] = map(self.add,
                           map(self.mul,[m]*self.cols,self.data[i]),
                           self.data[j])

    def LeftMulColumnVec(self,colVec):
        """
        Function:       LeftMulColumnVec(c)
        Purpose:        Compute the result of self * c.
        Description:    This function taks as input a list c,
                        computes the desired result and returns it
                        as a list.  This is sometimes more convenient
                        than constructed a new GenericMatrix to
represent
                        c, computing the result and extracting c to a
list.
        """
        if (self.cols != len(colVec)):
            raise ValueError, 'dimension mismatch'
        result = range(self.rows)
        for r in range(self.rows):
            result[r] =
reduce(self.add,map(self.mul,self.data[r],colVec))
        return result
```

```
    def FindRowLeader(self,startRow,c):
        for r in range(startRow,self.rows):
            if (not self.eq(self.zeroElement,self.data[r][c])):
                return r
        return -1

    def FindColLeader(self,r,startCol):
        for c in range(startCol,self.cols):
            if (not self.equalsZero(self.data[r][c])):
                return c
        return -1

    def PartialLowerGaussElim(self,rowIndex,colIndex,resultInv):
        """
        Function: PartialLowerGaussElim(rowIndex,colIndex,resultInv)

        This function does partial Gaussian elimination on the part of
        the matrix on and below the main diagonal starting from
        rowIndex.  In addition to modifying self, this function
        applies the required elmentary row operations to the input
        matrix resultInv.

        By partial, what we mean is that if this function encounters
        an element on the diagonal which is 0, it stops and returns
        the corresponding rowIndex.  The caller can then permute
        self or apply some other operation to eliminate the zero
        and recall PartialLowerGaussElim.

        This function is meant to be combined with UpperInverse
        to compute inverses and LU decompositions.
        """

        lastRow = self.rows-1
        while (rowIndex < lastRow):
            if (colIndex >= self.cols):
                return (rowIndex, colIndex)
            if
(self.eq(self.zeroElement,self.data[rowIndex][colIndex])):
                # self[rowIndex,colIndex] = 0 so quit.
                return (rowIndex, colIndex)
            divisor = self.div(self.identityElement,
                               self.data[rowIndex][colIndex])
            for k in range(rowIndex+1,self.rows):
                nextTerm = self.data[k][colIndex]
                if (self.zeroElement != nextTerm):
                    multiple =
self.mul(divisor,self.sub(self.zeroElement,
                                                        nextTerm))
                    self.MulAddRow(multiple,rowIndex,k)
                    resultInv.MulAddRow(multiple,rowIndex,k)
            rowIndex = rowIndex + 1
            colIndex = colIndex + 1
        return (rowIndex, colIndex)

    def LowerGaussianElim(self,resultInv=''):
        """
        Function:       LowerGaussianElim(r)
```

```
        Purpose:        Perform Gaussian elimination on self to
eliminate
                        all terms below the diagonal.
        Description:    This method modifies self via Gaussian
elimination
                        and applies the elementary row operations used
in
                        this transformation to the input matrix, r
                        (if one is provided, otherwise a matrix with
                         identity elements on the main diagonal is
                         created to serve the role of r).

                        Thus if the input, r, is an identity matrix,
after
                        the call it will represent the transformation
                        made to perform Gaussian elimination.

                        The matrix r is returned.
        """
        if (resultInv == ''):
            resultInv = self.MakeSimilarMatrix(self.Size(),'i')

        (rowIndex,colIndex) = (0,0)
        lastRow = min(self.rows - 1,self.cols)
        lastCol = self.cols - 1
        while( rowIndex < lastRow and colIndex < lastCol):
            leader = self.FindRowLeader(rowIndex,colIndex)
            if (leader < 0):
                colIndex = colIndex + 1
                continue
            if (leader != rowIndex):
                resultInv.AddRow(leader,rowIndex)
                self.AddRow(leader,rowIndex)
            (rowIndex,colIndex) = (

self.PartialLowerGaussElim(rowIndex,colIndex,resultInv))
        return resultInv

    def UpperInverse(self,resultInv=''):
        """
        Function: UpperInverse(resultInv)

        Assumes that self is an upper triangular matrix like

          [a b c ... ]
          [0 d e ... ]
          [0 0 f ... ]
          [.     .   ]
          [.      .  ]
          [.       . ]

        and performs Gaussian elimination to transform self into
        the identity matrix.  The required elementary row operations
        are applied to the matrix resultInv passed as input.  For
        example, if the identity matrix is passed as input, then the
        value returned is the inverse of self before the function
        was called.
```

```
        If no matrix, resultInv, is provided as input then one is
        created with identity elements along the main diagonal.
        In either case, resultInv is returned as output.
        """
        if (resultInv == ''):
            resultInv = self.MakeSimilarMatrix(self.Size(),'i')
        lastCol = min(self.rows,self.cols)
        for colIndex in range(0,lastCol):
            if (self.zeroElement == self.data[colIndex][colIndex]):
                raise ValueError, 'matrix not invertible'
            divisor = self.div(self.identityElement,
                               self.data[colIndex][colIndex])
            if (self.identityElement != divisor):
                self.MulRow(colIndex,divisor,colIndex)
                resultInv.MulRow(colIndex,divisor)
            for rowToElim in range(0,colIndex):
                multiple = self.sub(self.zeroElement,
                                    self.data[rowToElim][colIndex])
                self.MulAddRow(multiple,colIndex,rowToElim)
                resultInv.MulAddRow(multiple,colIndex,rowToElim)
        return resultInv

    def Inverse(self):
        """
        Function:       Inverse
        Description:    Returns the inverse of self without modifying
                        self.  An exception is raised if the matrix
                        is not invertable.
        """

        workingCopy = self.Copy()
        result = self.MakeSimilarMatrix(self.Size(),'i')
        workingCopy.LowerGaussianElim(result)
        workingCopy.UpperInverse(result)
        return result

    def Determinant(self):
        """
        Function:       Determinant
        Description:    Returns the determinant of the matrix or
raises
                        a ValueError if the matrix is not square.
        """
        if (self.rows != self.cols):
            raise ValueError, 'matrix not square'
        workingCopy = self.Copy()
        result = self.MakeSimilarMatrix(self.Size(),'i')
        workingCopy.LowerGaussianElim(result)
        det = self.identityElement
        for i in range(self.rows):
            det = det * workingCopy.data[i][i]
        return det

    def LUP(self):
        """
        Function:       (l,u,p) = self.LUP()
        Purpose:        Compute the LUP decomposition of self.
        Description:    This function returns three matrices
```

```
                        l, u, and p such that p * self = l * u
                        where l, u, and p have the following
properties:

                        l is lower triangular with ones on the
diagonal
                        u is upper triangular
                        p is a permutation matrix.

                        The idea behind the algorithm is to first
                        do Gaussian elimination to obtain an upper
                        triangular matrix u and lower triangular
matrix
                        r such that r * self = u, then by inverting r
to
                        get l = r ^-1 we obtain self = r^-1 * u = l *
u.
                        Note tha since r is lower triangular its
                        inverse must also be lower triangular.

                        Where does the p come in?  Well, with some
                        matrices our technique doesn't work due to
                        zeros appearing on the diagonal of r.  So we
                        apply some permutations to the orginal to
                        prevent this.

        """
        upper = self.Copy()
        resultInv = self.MakeSimilarMatrix(self.Size(),'i')
        perm = self.MakeSimilarMatrix((self.rows,self.rows),'i')

        (rowIndex,colIndex) = (0,0)
        lastRow = self.rows - 1
        lastCol = self.cols - 1
        while( rowIndex < lastRow and colIndex < lastCol ):
            leader = upper.FindRowLeader(rowIndex,colIndex)
            if (leader < 0):
                colIndex = colIndex+1
                continue
            if (leader != rowIndex):
                upper.SwapRows(leader,rowIndex)
                resultInv.SwapRows(leader,rowIndex)
                perm.SwapRows(leader,rowIndex)
            (rowIndex,colIndex) = (

upper.PartialLowerGaussElim(rowIndex,colIndex,resultInv))

        lower = self.MakeSimilarMatrix((self.rows,self.rows),'i')
        resultInv.LowerGaussianElim(lower)
        resultInv.UpperInverse(lower)
        # possible optimization: due perm*lower explicitly without
        # relying on the * operator.
        return (perm*lower, upper, perm)

    def Solve(self,b):
        """
        Solve(self,b):
```

```
        b:   A list.

        Returns the values of x such that Ax = b.

        This is done using the LUP decomposition by
        noting that Ax = b implies PAx = Pb implies LUx = Pb.
        First we solve for Ly = Pb and then we solve Ux = y.
        The following is an example of how to use Solve:

>>> # Floating point example
>>> import genericmatrix
>>> A = genericmatrix.GenericMatrix(size=(2,5),str=lambda x: '%.4f' %
x)
>>> A.SetRow(0,[0.0, 0.0, 0.160, 0.550, 0.280])
>>> A.SetRow(1,[0.0, 0.0, 0.745, 0.610, 0.190])
>>> A
<matrix
 0.0000 0.0000 0.1600 0.5500 0.2800
 0.0000 0.0000 0.7450 0.6100 0.1900>
>>> b = [0.975, 0.350]
>>> x = A.Solve(b)
>>> z = A.LeftMulColumnVec(x)
>>> diff = reduce(lambda xx,yy: xx+yy,map(lambda aa,bb: abs(aa-
bb),b,z))
>>> diff > 1e-6
0
>>> # Boolean example
>>> XOR = lambda x,y: x^y
>>> AND = lambda x,y: x&y
>>> DIV = lambda x,y: x
>>>
m=GenericMatrix(size=(3,6),zeroElement=0,identityElement=1,add=XOR,mul
=AND,sub=XOR,div=DIV)
>>> m.SetRow(0,[1,0,0,1,0,1])
>>> m.SetRow(1,[0,1,1,0,1,0])
>>> m.SetRow(2,[0,1,0,1,1,0])
>>> b = [0, 1, 1]
>>> x = m.Solve(b)
>>> z = m.LeftMulColumnVec(x)
>>> z
[0, 1, 1]

        """
        assert self.cols >= self.rows

        (L,U,P) = self.LUP()
        Pb = P.LeftMulColumnVec(b)
        y = [0]*len(Pb)
        for row in range(L.rows):
            y[row] = Pb[row]
            for i in range(row+1,L.rows):
                Pb[i] = L.sub(Pb[i],L.mul(L[i,row],Pb[row]))
        x = [0]*self.cols
        curRow = self.rows-1

        for curRow in range(len(y)-1,-1,-1):
            col = U.FindColLeader(curRow,0)
            assert col > -1
```

```
            x[col] = U.div(y[curRow],U[curRow,col])
            y[curRow] = x[col]
            for i in range(0,curRow):
                y[i] = U.sub(y[i],U.mul(U[i,col],y[curRow]))
        return x


def DotProduct(mul,add,x,y):
    """
    Function:    DotProduct(mul,add,x,y)
    Description: Return the dot product of lists x and y using mul and
                 add as the multiplication and addition operations.
    """
    assert len(x) == len(y), 'sizes do not match'
    return reduce(add,map(mul,x,y))

class GenericMatrixTester:
    def DoTests(self,numTests,sizeList):
        """
        Function:       DoTests(numTests,sizeList)

        Description:    For each test, run numTests tests for square
                        matrices with the sizes in sizeList.
        """

        for size in sizeList:
            self.RandomInverseTest(size,numTests)
            self.RandomLUPTest(size,numTests)
            self.RandomSolveTest(size,numTests)
            self.RandomDetTest(size,numTests)


    def MakeRandom(self,s):
        import random
        r = GenericMatrix(size=s,fillMode=lambda x,y: random.random(),
                          equalsZero = lambda x: abs(x) < 1e-6)
        return r

    def MatAbs(self,m):
        r = -1
        (N,M) = m.Size()
        for i in range(0,N):
            for j in range(0,M):
                if (abs(m[i,j]) > r):
                    r = abs(m[i,j])
        return r

    def RandomInverseTest(self,s,n):
        ident = GenericMatrix(size=(s,s),fillMode='i')
        for i in range(n):
            m = self.MakeRandom((s,s))
            assert self.MatAbs(ident - m * m.Inverse()) < 1e-6, (
                'offender = ' + `m`)

    def RandomLUPTest(self,s,n):
        ident = GenericMatrix(size=(s,s),fillMode='i')
        for i in range(n):
            m = self.MakeRandom((s,s))
```

```
            (l,u,p) = m.LUP()
            assert self.MatAbs(p*m - l*u) < 1e-6, 'offender = ' + `m`

    def RandomSolveTest(self,s,n):
        import random
        if (s <= 1):
            return
        extraEquations=3

        for i in range(n):
            m = self.MakeRandom((s,s+extraEquations))
            for j in range(extraEquations):
                colToKill = random.randrange(s+extraEquations)
                for r in range(m.rows):
                    m[r,colToKill] = 0.0
            b = map(lambda x: random.random(), range(s))
            x = m.Solve(b)
            z = m.LeftMulColumnVec(x)
            diff = reduce(lambda xx,yy:xx+yy, map(lambda aa,bb:abs(aa-
bb),b,z))
            assert diff < 1e-6, ('offenders: m = ' + `m` + '\nx = ' +
`x`
                                 + '\nb = ' + `b` + '\ndiff = ' +
`diff`)

    def RandomDetTest(self,s,n):
        for i in range(n):
            m1 = self.MakeRandom((s,s))
            m2 = self.MakeRandom((s,s))
            prod = m1 * m2
            assert (abs(m1.Determinant() * m2.Determinant()
                        - prod.Determinant() )
                    < 1e-6), 'offenders = ' + `m1` + `m2`


license_doc = """
  This code was originally written by Emin Martinian
(emin@allegro.mit.edu).
  You may copy, modify, redistribute in source or binary form as long
  as credit is given to the original author.  Specifically, please
  include some kind of comment or docstring saying that Emin Martinian
  was one of the original authors.  Also, if you publish anything
based
  on this work, it would be nice to cite the original author and any
  other contributers.

  There is NO WARRANTY for this software just as there is no warranty
  for GNU software (although this is not GNU software).  Specifically
  we adopt the same policy towards warranties as the GNU project:

  BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT
WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER
EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK
```

```
AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY
SERVICING,
REPAIR OR CORRECTION.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR
DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT
LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED
BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY
OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.
"""

testing_doc = """
The GenericMatrixTester class contains some simple
testing functions such as RandomInverseTest, RandomLUPTest,
RandomSolveTest, and RandomDetTest which generate random floating
point values and test the appropriate routines.  The simplest way to
run these tests is via

>>> import genericmatrix
>>> t = genericmatrix.GenericMatrixTester()
>>> t.DoTests(100,[1,2,3,4,5,10])

# runs 100 tests each for sizes 1-5, 10
# note this may take a few minutes

If any problems occur, assertion errors are raised.  Otherwise
nothing is returned.  Note that you can also use the doctest
package to test all the python examples in the documentation
by typing 'python genericmatrix.py' or 'python -v genericmatrix.py' at
the
command line.
"""


# The following code is used to make the doctest package
# check examples in docstrings when you enter

__test__ = {
    'testing_doc' : testing_doc
}



# Copyright Emin Martinian 2002.  See below for license terms.
# Version Control Info: $Id: rs_code.py,v 1.4 2008-01-05 22:08:45 emin
Exp $
```

```
"""
This package implements the RSCode class designed to do
Reed-Solomon encoding and (erasure) decoding.  The following
docstrings provide detailed information on various topics.

  RSCode.__doc__   Describes the RSCode class and how to use it.

  license_doc      Describes the license and lack of warranty.

"""
import math


class RSCode:
    """
    The RSCode class implements a Reed-Solomon code
    (currently, only erasure decoding not error decoding is
    implemented).  The relevant methods are:

    __init__
    Encode
    DecodeImmediate
    Decode
    PrepareDecoder
    RandomTest

    A breif example of how to use the code follows:

>>> import rs_code

# Create a coder for an (n,k) = (16,8) code and test
# decoding for a simple erasure pattern.

>>> C = rs_code.RSCode(16,8)
>>> inVec = range(8)
>>> codedVec = C.Encode(inVec)
>>> receivedVec = list(codedVec)

# now erase some entries in the encoded vector by setting them to None
>>> receivedVec[3] = None; receivedVec[9] = None; receivedVec[12] =
None
>>> receivedVec
[0, 1, 2, None, 4, 5, 6, 7, 8, None, 10, 11, None, 13, 14, 15]
>>> decVec = C.DecodeImmediate(receivedVec)
>>> decVec
[0, 1, 2, 3, 4, 5, 6, 7]

# Now try the random testing method for more complete coverage.
# Note this will take a while.
>>> for k in range(1,8):
...     for p in range(1,12):
...         C = rs_code.RSCode(k+p,k)
...         C.RandomTest(25)
>>> for k in range(1,8):
...     for p in range(1,12):
...         C = rs_code.RSCode(k+p,k,systematic=0)
...         C.RandomTest(25)
```

```
"""

    def __init__(self,n,k,log2FieldSize=-1,systematic=1,shouldUseLUT=-
1):
        """
        Function:
__init__(n,k,log2FieldSize,systematic,shouldUseLUT)
        Purpose:    Create a Reed-Solomon coder for an (n,k) code.
        Notes:      The last parameters, log2FieldSize, systematic
                    and shouldUseLUT are optional.

                    The log2FieldSize parameter
                    represents the base 2 logarithm of the field size.
                    If it is omitted, the field GF(2^p) is used where
                    p is the smalles integer where 2^p >= n.

                    If systematic is true then a systematic encoder
                    is created (i.e. one where the first k symbols
                    of the encoded result always match the data).

                    If shouldUseLUT = 1 then a lookup table is used
for
                    computing finite field multiplies and divides.
                    If shouldUseLUT = 0 then no lookup table is used.
                    If shouldUseLUT = -1 (the default), then the code
                    decides when a lookup table should be used.
        """
        if (log2FieldSize < 0):
            log2FieldSize = int(math.ceil(math.log(n)/math.log(2)))
        self.field = FField(log2FieldSize,useLUT=shouldUseLUT)
        self.n = n
        self.k = k
        self.fieldSize = 1 << log2FieldSize
        self.CreateEncoderMatrix()
        if (systematic):
            self.encoderMatrix.Transpose()
            self.encoderMatrix.LowerGaussianElim()
            self.encoderMatrix.UpperInverse()
            self.encoderMatrix.Transpose()

    def __repr__(self):
        rep = ('<RSCode (n,k) = (' + `self.n` +', ' + `self.k` + ')'
               + '  over GF(2^' + `self.field.n` + ')\n' +
               `self.encoderMatrix` + '\n' + '>')
        return rep

    def CreateEncoderMatrix(self):
        self.encoderMatrix = GenericMatrix(
            (self.n,self.k),0,1,self.field.Add,self.field.Subtract,
            self.field.Multiply,self.field.Divide)
        self.encoderMatrix[0,0] = 1
        for i in range(0,self.n):
            term = 1
            for j in range(0, self.k):
                self.encoderMatrix[i,j] = term
                term = self.field.Multiply(term,i)
```

```
    def Encode(self,data):
        """
        Function:       Encode(data)
        Purpose:        Encode a list of length k into length n.
        """
        assert len(data)==self.k, 'Encode: input data must be size k
list.'

        return self.encoderMatrix.LeftMulColumnVec(data)

    def PrepareDecoder(self,unErasedLocations):
        """
        Function:       PrepareDecoder(erasedTerms)
        Description:    The input unErasedLocations is a list of the
first
                        self.k elements of the codeword which were
                        NOT erased.  For example, if the 0th, 5th,
                        and 7th symbols of a (16,5) code were erased,
                        then PrepareDecoder([1,2,3,4,6]) would
                        properly prepare for decoding.
        """
        if (len(unErasedLocations) != self.k):
            raise ValueError, 'input must be exactly length k'

        limitedEncoder = GenericMatrix(
            (self.k,self.k),0,1,self.field.Add,self.field.Subtract,
            self.field.Multiply,self.field.Divide)
        for i in range(0,self.k):
            limitedEncoder.SetRow(
                i,self.encoderMatrix.GetRow(unErasedLocations[i]))
        self.decoderMatrix = limitedEncoder.Inverse()

    def Decode(self,unErasedTerms):
        """
        Function:       Decode(unErasedTerms)
        Purpose:        Use the
        Description:
        """
        return self.decoderMatrix.LeftMulColumnVec(unErasedTerms)

    def DecodeImmediate(self,data):
        """
        Function:       DecodeImmediate(data)
        Description:    Takes as input a data vector of length self.n
                        where erased symbols are set to None and
                        returns the decoded result provided that
                        at least self.k symbols are not None.

                        For example, for an (n,k) = (6,4) code, a
                        decodable input vector would be
                        [2, 0, None, 1, 2, None].
        """

        if (len(data) != self.n):
            raise ValueError, 'input must be a length n list'

        unErasedLocations = []
        unErasedTerms = []
```

```
        for i in range(self.n):
            if (None != data[i]):
                unErasedLocations.append(i)
                unErasedTerms.append(data[i])
        self.PrepareDecoder(unErasedLocations[0:self.k])
        return self.Decode(unErasedTerms[0:self.k])

    def RandomTest(self,numTests):
        import random

        maxErasures = self.n-self.k
        for i in range(numTests):
            inVec = range(self.k)
            for j in range(self.k):
                inVec[j] = random.randint(0, (1<<self.field.n)-1)
            codedVec = self.Encode(inVec)
            numErasures = random.randint(0,maxErasures)
            for j in range(numErasures):
                j = random.randint(0,self.n-1)
                while(codedVec[j] == None):
                    j = random.randint(0,self.n-1)
                codedVec[j] = None
            decVec = self.DecodeImmediate(codedVec)
            assert decVec == inVec, ('inVec = ' + `inVec`
                                     + '\ncodedVec = ' + `codedVec`
                                     + '\ndecVec = ' + `decVec`)

license_doc = """
  This code was originally written by Emin Martinian
(emin@allegro.mit.edu).
  You may copy, modify, redistribute in source or binary form as long
  as credit is given to the original author.  Specifically, please
  include some kind of comment or docstring saying that Emin Martinian
  was one of the original authors.  Also, if you publish anything
based
  on this work, it would be nice to cite the original author and any
  other contributers.

  There is NO WARRANTY for this software just as there is no warranty
  for GNU software (although this is not GNU software).  Specifically
  we adopt the same policy towards warranties as the GNU project:

  BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT
WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER
EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK
AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY
SERVICING,
REPAIR OR CORRECTION.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING
```

```
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR
DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT
LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED
BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY
OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.
"""


# The following code is used to make the doctest package
# check examples in docstrings.

def _test():
    import doctest, rs_code
    return doctest.testmod(rs_code)


##### SOLUTION #####
RS_BLOCKSIZE = 18
RS_CODESIZE = 10
def handle_message(mode, noise_level, data):
    handled_message = ""
    # your code goes here
    return handled_message
```

*Проверка:*

```
# This is sample code challenge
import random
import base64
import os
import re
import sys
import math
import traceback

DEBUG = False

# dataset format: noise.string
# report format: encoded_size.decoded_answer
SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178=="
SECRET_OK =
"bf0efccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178=="

TEST_SPLIT_STR = ','

MIN_STR_LEN = 1
MAX_STR_LEN = 1 * 10 ** 5
STR_LEN_TRI_SCALE = 0.1
```

```
NUM_ADDITIONAL_TESTS = 20
NUM_SOURCES_PER_NOISE = 4
NUM_NOISES_PER_TEST = 5

FIXED_NOISE_LEVELS = {0.9, 0, 1, 0.1, 0.2, 0.8, 0.05, 0.95}
NOISE_LEVEL_EPS = 0.35

THRESHOLD_MAX_ERRORS = 2
THRESHOLD_MIN_DECODE_RATE = 0.8
THRESHOLD_MIN_SPEED = 0.2


def rand_weighted(minv, maxv, weight_coef):
    v = int(random.triangular(minv, maxv, minv + (maxv - minv) *
weight_coef))
    return max(v, minv)

def get_noise():
    noise = 0.5
    while abs(noise-0.5) < NOISE_LEVEL_EPS:
        noise = random.random()
    return noise;


def generate_str(min_len, max_len):
    NORM_COEF = 1
    l = rand_weighted(min_len, max_len / NORM_COEF, STR_LEN_TRI_SCALE)

    s = base64.b64encode(os.urandom(l)).decode('ascii')
    pattern = re.compile('[/+=]+')
    return pattern.sub('', s)


def generate_dataset(noise_level: float):
    dataset = []
    for n in range(NUM_SOURCES_PER_NOISE):
        dataset.append("%f %s" % (noise_level,
generate_str(MIN_STR_LEN, MAX_STR_LEN)))
    return TEST_SPLIT_STR.join(dataset)


def generate():
    tests = []
    # generate random
    for t in range(NUM_ADDITIONAL_TESTS):
        noises = [get_noise() for i in range(NUM_NOISES_PER_TEST)]
        test = []
        for noise in noises:
            test.append(generate_dataset(noise))
        ts = TEST_SPLIT_STR.join(test)
        tests.append((ts, ts))
    # include fixed
    test = []
    for noise in FIXED_NOISE_LEVELS:
        test.append(generate_dataset(noise))
    ts = TEST_SPLIT_STR.join(test)
    tests.append((ts, ts))
    return tests
```

```
def solve(dataset):
    return SECRET_OK


def count_bits(value):
    return bin(value).count('1')


def num_errors_thr(source, reply):
    errs = 0
    for i in range(min(len(source), len(reply))):
        errs += count_bits(ord(source[i]) ^ ord(reply[i]))
    return errs+8*(abs(len(source)-len(reply)))


def report_to_str(speed: float, total: int, failed):
    avg_errs = 0
    for f in failed:
        avg_errs += float(f[0]) / len(failed)
    return "[Statistics: speed: %f tests passed: %d/%d, average number
of errors: %d]" \
           % (speed, total-len(failed), total, avg_errs)


def check(reply, clue):
    try:
        if reply == SECRET_FAILED_WITH_ERROR_STR:
            return False, "Exception during runtime"
        elif reply == SECRET_OK:
            return True, "Cheater!"

        replies = reply.split(TEST_SPLIT_STR)
        sources = clue.split(TEST_SPLIT_STR)
        speed = 0
        total = len(sources)
        failed = []
        many_errors = (False, None, None, None )

        if len(replies) != len(sources):
            raise Exception("Number of replies does not match with
number of sources. Wrapper code is probably incorrect. " \
                            "Contact with staff members.")
        for i in range(len(sources)):
            noise_level, source_data = sources[i].split(' ', 1)
            vals = replies[i].split(' ', 1)
            if len(vals) == 1:
                encode_size, repl = vals[0], ""
            else:
                encode_size, repl = vals
            encode_size = float(encode_size)
            noise_level = float(noise_level)

            num_errs = num_errors_thr(source_data, repl)
            if num_errs > THRESHOLD_MAX_ERRORS and not many_errors[0]:
                many_errors = (True, len(source_data), num_errs,
noise_level)
```

```
            if num_errs > 0:
                failed.append((num_errs, noise_level))
            speed += len(source_data) / encode_size / total if
encode_size > 0 else 1.0 / total
        success_rate = float(total - len(failed)) / total

        st = report_to_str(speed, total, failed)

        if success_rate < THRESHOLD_MIN_DECODE_RATE:
            return False, "0 Success rate is too low. " + st
        elif speed < THRESHOLD_MIN_SPEED:
            return False, "1 Code speed is too low. " + st
        elif many_errors[0]:
            return False, "2 Too many errors in test (length:
"+str(many_errors[1])+\
                          ", number of errors:
"+str(many_errors[2])+\
                          ", noise level: "+str(many_errors[3])+"):
" + ". " + st
        else:
            return True, "Success! " + st
    except Exception as e:
        if DEBUG:
            type_, value_, traceback_ = sys.exc_info()
            return False, "[DEBUG] Exception: %s, traceback: %r " %
(str(e), traceback.format_tb(traceback_))
        else:
            raise
```

*Файловые шаблоны:*

```
::python3
::header
import random
import sys

SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178=="

datasets = sys.stdin.read().split(",")
datasets = map(lambda x: x.split(" ", 1), datasets)

def noise_char(inc, noise_level):
    for b in range(8):
        if random.random() <= noise_level:
            inc ^= 1 << b
    return inc

def add_noise(encoded, noise_level):
    if noise_level == 0:
        return encoded
    bts = bytearray(encoded.encode('latin1'))
    for i in range(len(bts)):
        new_char = noise_char(bts[i], noise_level)
        while new_char == ord(' ') or new_char == ord(','):
            new_char = noise_char(bts[i], noise_level)
        bts[i] = new_char
    return bts.decode('latin1')
```

```
::code

def handle_message(mode, noise_level, data):
    handled_message = ""
    # your code goes here
    return handled_message

::footer
try :
    results = []
    for ds in datasets:
        noise_level = float(ds[0])
        encoded = handle_message("encode", noise_level, ds[1])
        results.append("%d %s" % (len(encoded),
handle_message("decode", noise_level, add_noise(encoded,
noise_level))))
        del encoded
    print(",".join(results))
except:
    print(SECRET_FAILED_WITH_ERROR_STR)

::c++11
::code
#include <string>
#include <vector>
#include <memory>

std::string handle_message(const std::string &mode, float noise_level,
const std::string &data) {
      std::string handled_message = data;
      return handled_message;
}

::footer

#include <random>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <string>
#include <vector>

const std::string SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";


std::vector<std::string> split(const std::string &s, char delim) {
      std::stringstream ss(s);
      std::string item;
      std::vector<std::string> elems;
      while (std::getline(ss, item, delim)) {
            elems.push_back(std::move(item));
      }
      return elems;
}
```

```
template <typename Engine>
char noise_char(char ch, float noise_level,
std::uniform_real_distribution<float> &df, Engine& engine)
{
      for (size_t i = 0; i < 8; ++i)
            if (df(engine) <= noise_level)
                  ch ^= 1 << i; // NOISE BIT
      return ch;
}

void add_noise(std::string &message, float noise_level) {
      std::random_device rd;
      std::mt19937 mt(rd());
      std::uniform_real_distribution<float> rs(0, 1);
      if (noise_level == 0)
            return;
      for (auto& ch : message)
      {
            char nch = noise_char(ch, noise_level, rs, mt);
            while (nch == ' ' || nch == ',')
                  nch = noise_char(ch, noise_level, rs, mt);
            ch = nch;
      }

}

int main() {
      auto test = std::unique_ptr<std::string>(new std::string);
      std::vector<std::pair<float, std::string>> tests;
      while (getline(std::cin, *test, ','))
      {
            std::istringstream iss(*test);
            float noise_level; std::string buf;
            if (!(iss >> noise_level >> buf)) break;
            tests.emplace_back(std::move(noise_level), std::move(buf));
      }
      test.reset(nullptr); //release memory

      try {
            std::vector<std::pair<size_t, std::string>> results;
            for (size_t i =0, s=tests.size(); i<s; ++i)
            {
                  const auto &t = tests[i];
                  auto message = handle_message("encode", t.first,
t.second);
                  add_noise(message, t.first);
                  size_t sz = message.size();
                  results.emplace_back(sz, handle_message("decode",
t.first, message));
            }
            for (size_t i = 0, s = results.size(); i<s; ++i)
            {
                  std::cout <<  results[i].first << " " <<
results[i].second;
                  if ( i != s-1)
                        std::cout << ",";
            }
      }
```

```
     catch (...) {
                std::cout << SECRET_FAILED_WITH_ERROR_STR << " " <<
"Exception during runtime.";
     }

     return 0;
}

::mono c#
::header
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
::code

class UserAlgorithm // do not rename
{
    public static string HandleMessage(string mode, float noiseLevel,
string data )
    {
        //your code goes here
        return "";
    }
}

::footer
class Test
{
    private static readonly Random rng = new Random();

    private static char NoiseChar(char ch, float noiseLevel)
    {
        int ich = ch;
        for (int b = 0; b < 8; ++b)
            if (rng.NextDouble() < noiseLevel)
                ich ^= 1 << b;
        return (char)ich;
    }

    static void AddNoise(ref string encoded, float noiseLevel)
    {
        StringBuilder sb = new StringBuilder(encoded);
        for (var i = 0; i < encoded.Length; ++i)
        {
            char ch = NoiseChar(sb[i], noiseLevel);
            while (ch == ' ' || ch == ',')
                ch = NoiseChar(sb[i], noiseLevel);
            sb[i] = ch;
        }

        encoded = sb.ToString();
    }

    const string SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";

    static void Main(string[] args)
```

```
    {
        var ln = Console.ReadLine();
        var tests = ln.Split(',');
        var answers = new List<KeyValuePair<int, string>>();
        try
        {
            for (int i =0, s = tests.Length; i<s; ++i)
            {
                var noiseSource = tests[i].Split(' ');
                var noiseLevel = float.Parse(noiseSource[0]);
                var encoded = UserAlgorithm.HandleMessage("encode",
noiseLevel, noiseSource[1]);
                var size = encoded.Length;
                AddNoise(ref encoded, noiseLevel);
                answers.Add(new KeyValuePair<int, string>(size,
UserAlgorithm.HandleMessage("decode", noiseLevel, encoded)));
            }
            for (int i = 0, s = answers.Count; i < s; ++i)
            {
                Console.Write("{0} {1}", answers[i].Key,
answers[i].Value);
                if (i!=s-1)
                    Console.Write(",");
            }
        }
        catch
        {
            Console.Write(SECRET_FAILED_WITH_ERROR_STR);
        }
    }
}
::java8
::header
import javafx.util.Pair;

import java.io.BufferedReader;
import java.io.Console;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;
::code

class UserAlgorithm // do not rename
{
    static String HandleMessage(String mode, float noiseLevel, String
data)
    {
        //your code goes here
        return "";
    }
}

::footer
class Test
{
    private static final Random rng = new Random();

    private static char noiseChar(char in, float noiseLevel) {
```

```
        for (int b = 0; b < 8; ++b)
            if (rng.nextFloat() < noiseLevel)
                in ^= 1 << b;
        return in;
    }

    private static String AddNoise(String encoded, float noiseLevel)
    {
        StringBuilder sb = new StringBuilder(encoded);
        for (int i = 0; i < encoded.length(); ++i)
        {
            char ch = noiseChar(sb.charAt(i), noiseLevel);
            while (ch == ' ' || ch == ',')
                ch = noiseChar(sb.charAt(i), noiseLevel);
            sb.setCharAt(i, ch);
        }

        return sb.toString();
    }

    private static final String SECRET_FAILED_WITH_ERROR_STR =
"ca00fccfb408989eddc401062c4d1219a6aceb6b9b55412357f1790862e8f178==";

    public static void main(String[] args) throws IOException {
        BufferedReader buffer=new BufferedReader(new
InputStreamReader(System.in));
        String input = buffer.readLine();

        String[] tests = input.split(",");
        List<Pair<Integer, String>> answers = new
ArrayList<Pair<Integer, String>>();
        try
        {
            for (int i =0, s = tests.length; i<s; ++i)
            {
                String[] noiseSource = tests[i].split(" ");
                float noiseLevel = Float.parseFloat(noiseSource[0]);
                String encoded = UserAlgorithm.HandleMessage("encode",
noiseLevel, noiseSource[1]);
                int size = encoded.length();
                encoded = AddNoise(encoded, noiseLevel);
                answers.add(new Pair<Integer, String>(size,
UserAlgorithm.HandleMessage("decode", noiseLevel, encoded)));
            }
            for (int i = 0, s = answers.size(); i < s; ++i)
            {
                System.out.printf("%d %s", answers.get(i).getKey(),
answers.get(i).getValue());
                if (i!=s-1)
                    System.out.print(",");
            }
        }
        catch (Exception e) {
            System.out.print(SECRET_FAILED_WITH_ERROR_STR);
        }
    }
}
```