§2 Второй отборочный этап

Включает задачи, для решения которых достаточно школьных знаний и умений программы 10-11 класса, навыков использовать школьные знания для решения новых задач. Включает 5 задач, целью которых является подготовка к финальному командному туру. Данный тур позволяет очертить область предметных знаний, необходимую для участия в профиле. Методические рекомендации к данному туру обозначают область знаний и навыков для самостоятельного изучения. Все задачи с уклоном в специфику тура, сочетают в себе математику и информатику. Задачи участники решали на сервере компании разработчиков через веб-интерфейс.

Задача2.1 "Окружность". Максимальная оценка: 5 балов

Условие:

Определить центр окружности (x0,y0), заданной на битовой матрице 400x400 (1 - окружность, 0 - нет окружности) и вычислить ее радиус (R) с точностью до дискрета матрицы. На битовой матрице может присутствовать незначительное количество (порядка 1% от числа единиц) случайных единиц, не лежащих на окружности.

Программа должна читать значения матрицы с потока stdin и выдавать результат (три целых числа, разделенных пробелами - x0 y0 R) на stdout.

То есть программа должна запускаться, как

solution.exe <input.dat >output.dat

Скорость выполнения программы - не более 10 секунд

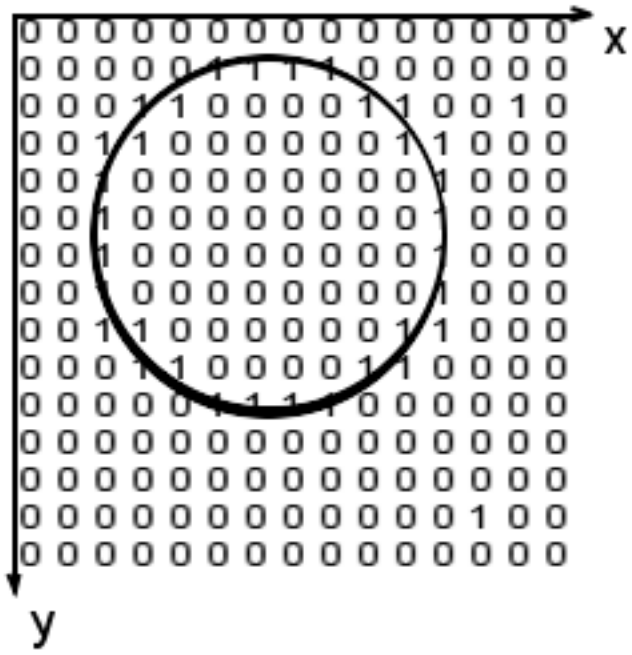
Формат входных данных:
Выглядит примерно так:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Реальный файл намного больше (400 столбцов и 400 строчек)

Формат выходных данных:
8 6 5

Иллюстрации:



Генератор примера:

```
//C
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
int main(int argn,char* argv[])
{
long R;
long x,y,x0,y0;
double phi;
int matrix[500][500];
srand(time(NULL));
if(argn>3)
{
x0=atol(argv[1]);
y0=atol(argv[2]);
R=atol(argv[3]);
}
for(x=0;x<400;x++)
for(y=0;y<400;y++)
matrix[x][y]=0;
for(phi=0;phi<2*3.1415926;phi+=0.001)
{
x=(int)(R*cos(phi)+x0);
y=(int)(R*sin(phi)+y0);
matrix[x][y]=1;
}
int count=0;
for(x=0;x<400;x++)
for(y=0;y<400;y++)
if(matrix[x][y]==1)
count++;
count=2;
int i;
for(i=0;i<count;i++)
{
matrix[rand()%(x0-R+1)][rand()%400]=1;
```

```
matrix[rand()%400][rand()%(y0-R+1)]=1;
}
for(y=0;y<400;y++)
{
for(x=0;x<400;x++)
{
printf("%d",matrix[x][y]);
if(x<400-1)
printf(" ");
}
printf("\n");
}
}
```

Решение:

```
// C
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define SIZEX 400
#define SIZEY 400
int main()
{
int matrix[SIZEX][SIZEY];
long x,y;
for(y=0;y<SIZEY&&!feof(stdin);y++)
for(x=0;(x<SIZEX)&&!feof(stdin);x++)
fscanf(stdin,"%d",&(matrix[x][y]));
if(y<SIZEY || x<SIZEX)
{
fprintf(stderr,"too small file\n"),exit(1);
}
double avrg_x=0,avrg_y=0,n=0;
for(y=0;y<SIZEY;y++)
for(x=0;x<SIZEX;x++)
{
avrg_x+=matrix[x][y]*x;
avrg_y+=matrix[x][y]*y;
n+=matrix[x][y];
}
avrg_x/=n;
avrg_y/=n;
avrg_x=round(avrg_x);
avrg_y=round(avrg_y);
printf("%ld %ld ",(long)avrg_x,(long)avrg_y);
n=0;
double avrg_r;
avrg_r=0;
for(y=0;y<SIZEY;y++)
for(x=0;x<SIZEX;x++)
{
avrg_r+=matrix[x][y]*sqrt((x-avrg_x)*(x-avrg_x)+(y-avrg_y)*(y-avrg_y));
n+=matrix[x][y];
}
avrg_r=round(avrg_r/n);
printf("%ld\n",(long)avrg_r);
}
```

Критерии оценки:

```
B
//Perl
```

```
#!/usr/bin/perl
$params[1]='131 101 97';
$params[2]='219 207 142';
$params[3]='171 220 95';
$params[4]='141 211 45';
$params[5]='310 313 54';
$sol=7;
for($sol=-3;$sol<=30;$sol++)
{
open RESFILE,">$sol/results.txt";
$ball=5;
$codename="solution_1_$sol";
for($i=1;$i<=5;$i++)
{
$fsrc=$params[$i];
`./generator $fsrc >test.dat`;
($solution_file,$rest)=split(/\s+/,`ls ./$sol/solution.*`);
print "$solution_file\n";
compile($solution_file);
$time=run($solution_file,"test.dat","sol");
$res=`cat sol`;
$res=~ s/[\n\r]//g;
@res_user=split(/\s+/,$res);
@res_src=split(/\s+/,$fsrc);
if(abs(@res_user[0]-@res_src[0])<=1 && abs(@res_user[1]-@res_src[1])<=1 &&
abs(@res_user[2]-@res_src[2])<=1)
{
print RESFILE "$codename $fsrc $res TIME:$time\n";
}
else
{
$ball=0;
if(-s "$fsrc.errors" >0)
{ print RESFILE  "$codename $fsrc COMPILE_ERROR TIME:$time\n";  }
else
{ print RESFILE  "$codename $fsrc ERROR TIME:$time\n"; }
}
`rm -f test.dat sol`;
}
print RESFILE  "\n sol_$sol Баллы: $ball\n";
close RESFILE;
}
sub compile()
{
$fname=shift;
`rm -f solution *.java *.class *.py *.txt output* input*`;
if($fname =~ /\.cpp/)
{
`rm -f solution`;
`g++ $fname -o solution -lm -std=c++11 2> $fname.errors`;
return;
}
if($fname =~ /\.java/)
{
`rm -f *.class *.java`;
`cp $fname solution.java`;
`javac solution.java  2> $fname.errors `;
return;
}
if($fname =~ /\.py/)
{
`rm -f solution.py`;
`cp $fname solution.py`;
return;
```

```
}
if($fname =~ /\.c/)
{
`rm -f solution`;
`gcc $fname -o solution -lm 2> $fname.errors`;
return;
}
}
sub run()
{
$fname=shift;
$src=shift;
$dest=shift;
if($fname =~ /\.cpp/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.java/)
{
`rm -f tm`;
`time -f %U java solution <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.py/)
{
`rm -f tm`;
`time -f %U python ./solution.py   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.c/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
return 1000;
}
```

Принци прешения

Можно задачу решать несколькими способами. В данном случае измеряется центр окружности методом центра масс: рассчитывается центр масс тела, где значение ячейки матрицы представляет собой вес. Радиус окружности рассчитывается, исходя из среднеквадратичного отклонения ненулевых точек от центра масс. Поскольку предполагается, что точки представляют собой окружность, мы можем так делать. Малое количество 'шумовых' точек не влияет на результат.

Задача 2.2 "Числа". Максимальная оценка 20 баллов

Условие:

По неизвестному каналу с шумами передается последовательность 1000 чисел, выбранных случайно из 6-значных простых чисел от 100003 до 199999 включительно. Числа разделены переводом строки.

За счет шумов в некоторых переданных числах возможна ошибка типа замены одного из символов на следующий за ним (инкремент по модулю 10) (1 заменяется на 2, 4 на 5, 9 на 0 и т.д.) без изменения остальных символов числа. Эта ошибка для каждого числа может возникнуть не более, чем 1 раз (т.е. если число передано ошибочно, то ошибка только в одном символе, в двух символах одновременно ошибки быть не может). Например 100003 может стать 101003 или 200003.

Найти элементы последовательности переданные с ошибками, по возможности исправить эти ошибки. Что невозможно исправить - отметить звездочкой (например 100004*). Максимальное число баллов - за полностью верный ответ, за каждую ошибку снимается 1 балл.

Скорость выполнения программы - не более 10 секунд

Программа должна читать исходный файл со стандартного потока stdin и передавать исправленный файл на стандартный поток stdout в аналогичном формате (числа разделены переводом строки)

То есть работать в виде:

solution.exe <input.dat >output.dat

Формат входных данных:

Выглядит примерно так:

100003
199999
149894
108769
128257
127973
144731
133571
146701
117281
150901
197713
172028
135757
119591
187687
159179
179083
160441
177043
133649
129461
132589
164839
244379
133981
157489
128659
134877
188311

Реальный файл намного больше (1000 строчек)

Генератор примера:

//generator.c

```
//файл primes.txt содержит все простые числа от 100003 до 199999
//C
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
long TEXT_LEN=1000;
int dig_pos[]={1,10,100,1000,10000,100000,1000000};
long primes_arr[100000];
long len;
```

```
int is_prime(long val)
{
long i;
for(i=0;i<len;i++)
{
if(primes_arr[i]==val)
return 1==1;
}
return 1==0;
}
main()
{
FILE* stream;
long prime;
long i=0;
long pos;
long strpos;
long nprime;
long digit;
srand(time(NULL));
stream=fopen("primes.txt","rt");
len=0;
for(;!feof(stream);)
{
fscanf(stream,"%ld",&prime);
if(prime>0)
{
primes_arr[len++]=prime;
if(!is_prime(prime))
fprintf(stderr,"prime %ld not in hash\n",prime);
}
}
fclose(stream);
len--;
for(i=0;i<TEXT_LEN;)
{
switch(i%100)
{
case 0: pos=0; break;
case 1: pos=len-1;break;
default:
pos=rand()%len;
break;
}
if(rand()%10 == 1)
{
int set1;
for(set1=0;set1<10;set1++)
{
strpos=rand()%6;
prime=primes_arr[pos];
digit=((prime/dig_pos[strpos])%10);
nprime=prime-digit*dig_pos[strpos];
digit=(digit+1)%10;
nprime=nprime+digit*dig_pos[strpos];
if(nprime && !is_prime(nprime))
{
printf ("%ld\n",nprime);
i++;
goto BRK;
break;
}
}
}
```

```
else
{
printf("%ld\n",primes_arr[pos]);
i++;
}
BRK:;
}
}
```

Решение:

//check_file.c

```
// C
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
long TEXT_LEN=1000;
int dig_pos[]={1,10,100,1000,10000,100000,1000000};
long primes_arr[100000];
long len;
int is_prime(long val)
{
long i;
for(i=0;i<len;i++)
{
if(primes_arr[i]==val)
return 1==1;
}
return 1==0;
}
main()
{
FILE* stream;
long nprime;
long prime;
long i=0;
long pos;
long strpos;
long digit;
char var_code[255];
long var_num;
stream=fopen("primes.txt","rt");
len=0;
for(;!feof(stream);)
{
fscanf(stream,"%ld",&prime);
if(prime>0)
{
primes_arr[len++]=prime;
if(!is_prime(prime))
{
fprintf(stdout,"not found: %ld\n",prime);
}
}
}
fclose(stream);
len--;
long cand_prime;
int res;
for(;!feof(stdin);)
{
res=fscanf(stdin,"%ld",&prime);
if(res>0 && prime>0 && !is_prime(prime))
{
```

```
sprintf(var_code,"%ld*",prime);
strpos=0;
var_num=0;
long found_prime=-1;
for(strpos=0;strpos<6;strpos++)
{
cand_prime=prime;
digit=((cand_prime/dig_pos[strpos])%10);
nprime=cand_prime-digit*dig_pos[strpos];
digit=digit-1;
if(digit<0)
digit=9;
nprime=nprime+digit*dig_pos[strpos];
if(is_prime(nprime))
{
sprintf(var_code,"%ld",nprime);
found_prime=nprime;
var_num++;
}
}
if(var_num>1)
{
sprintf(var_code,"%ld*",prime);
}
if(var_num==1)
{
sprintf(var_code,"%ld",found_prime);
}
if(var_num<1)
{
sprintf(var_code,"%ld*",prime);
}
printf("%s",var_code);
printf("\n");
}
else
{
if(res>0)
printf("%ld\n",prime);
}
}
}
```

Критерии оценки:

```
//Perl
#!/usr/bin/perl
for($i=1;$i<=10;$i++)
{
`./generator >test-$i.dat`;
`./check_file <test-$i.dat >test-$i.chk`;
`dos2unix test-$i.chk`;
}
for($sol=1;$sol<=38;$sol++)
{
$codename="solution_2_$sol";
open RESFILE,">$sol/results.txt";
$max_err=0;
$ball=20;
for($i=1;$i<=10;$i++)
{
`rm -f test.dat test.chk`;
`cp test-$i.dat test.dat`;
`cp test-$i.chk test.chk`;
$fsrc="itteration ".$i;
```

```
($solution_file,$rest)=split(/\s+/,`ls ./$sol/solution.*`);
print "$solution_file\n";
compile($solution_file);
`rm -f sol $solution_file.$i.log`;
$time=run($solution_file,"test.dat","sol")/2;
if(-s "sol">0)
{
`dos2unix sol`;
`diff test.chk sol > $solution_file.$i.log`;
$DIFF=split(/\n/,`cat $solution_file.$i.log`)/4;
if($DIFF>$max_err)
{
$max_err=$DIFF;
}
}
else
{
$DIFF = -1;
$ball=0;
print RESFILE "Программа не работает как требуется условиями задачи\n";
}
if(!$DIFF && $time>=0 & $time<10)
{
print RESFILE "$codename $fsrc ($DIFF) TIME:$time\n";
}
else
{
if($time>10)
{
$ball=0;
      print RESFILE  "Слишкоммедленноработает\n";
}
if(-s "$fsrc.errors" >0)
{ print RESFILE  "$codename $fsrc COMPILE_ERROR TIME:$time\n";
$ball=0;
}
else
{
print RESFILE  "$codename $fsrc ERROR $DIFF TIME:$time\n";
}
}
`rm -f test.dat sol`;
}
$ball-=$max_err;
if($ball<0)
{ $ball=0; }
print RESFILE  "\nМаксимальноошибок: $max_err\n solution_$sol Баллы: $ball";
close RESFILE;
}
sub compile()
{
$fname=shift;
`rm -f *.class *.java *.py solution output.* input.*`;
if($fname =~ /\.cpp/)
{
`rm -f solution`;
`g++ $fname -o solution -std=c++11 2>$fname.errors`;
#   exit;
return;
}
if($fname =~ /\.java/)
{
`rm -f *.class *.java`;
`cp $fname solution.java`;
```

```
`javac solution.java 2>$fname.errors`;
return;
}
if($fname =~ /\.py/)
{
`rm -f solution.py`;
`cp $fname solution.py`;
return;
}
if($fname =~ /\.c/)
{
`rm -f solution`;
`gcc $fname -o solution 2>$fname.errors`;
return;
}
}
sub run()
{
$fname=shift;
$src=shift;
$dest=shift;
if($fname =~ /\.cpp/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.java/)
{
`rm -f tm`;
if(-s "solution.class")
{ `time -f %U java solution <$src >$dest 2>tm`;}
else
{ `time -f %U java Main <$src >$dest 2>tm`;}
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.py/)
{
`rm -f tm`;
`time -f %U python3 ./solution.py   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.c/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
return 1000;
}
```

Принцип решения

Сначала проверяется, является число простым в интервале 100003-199999 включительно или нет. Если оно не является простым в этом интервале, то оно гарантировано искажено и его надо исправлять.

Решение задачи исправления сводится к перебору для каждого такого числа всех простых чисел, которые могут произвести это число после прохождения канала с помехами. Если существует только одно такое простое число - значит мы заменяем искаженное число на него. Если таких числе не существует или их больше одного, значит мы не можем уверенно исправить искаженное число и поэтому помечаем его *.

Задача 2.3 "Движение". Максимальная оценка 60 баллов

Условие:

Объект движется в плоскости XY равномерно по прямой параллельно оси X слева направо (от больших отрицательных к большим положительным значениям X, см.рисунок). Скорость объекта неизвестна и находится в пределах от 0.01 до 0.1 км/сек. В некий момент он въезжает в облако акустических излучателей, расположенных также в этой плоскости в квадратном полигоне размером [-30..30км,-30..30км], никогда не приближаясь к ним сильно близко (ближе 1 км).Каждый акустический излучатель излучает постоянно строго синхронизированную периодическую последовательность импульсов, представимую в виде:

$u_i(t)=A_i*\cos^2(t*w_i+B_i)$

Количество излучателей известно и равно 7.Координаты каждого излучателя известны с точностью 0.01км и перечислены ниже. Циклические частоты повторения импульсов w_i каждого излучателя различны,известны и перечислены ниже. Начальные фазы B_i и амплитуды A_i неизвестны и могут быть различны, но постоянны для каждого из излучателей. Скорость звука считать не зависящей от частоты звука и равной 0.3км/сек, спадание амплитуды принимаемого звука с расстоянием считать отсутствующим.

Исходный файл содержит запись звуковых последовательностей через 0.03 сек., принятой объектом с каждого излучателя, как функции времени. Длина файла (обычно >20 МБ) выбрана такой, что начало и конец последовательностей соответствует очень большой удаленности от облака излучателей. Первый столбец файла - время, остальные столбцы - данные от различных излучателей, разделеные пробелами. Данные в последовательные моменты времени разделены переводом строки.

Ваша программа на основе указанной информации и исходного файла записи звуковых последовательностей должна определить положение объекта в момент t=0 с точностью не хуже 2км и модуль его скорости с точностью не хуже 0.01км/с (в среднем, по результатам 5 испытаний на созданных организаторами трека тестовых файлах).

Время работы программы должно быть не более 5мин.

Программа должна читать исходные данные со стандартного потока stdin и передавать их на stdout.

То есть программа должна работать так:

solution.exe <input.dat >output2.dat

Ввод:

t u_1 u_2 u_3 u_4 u_5 u_6 u_7

...
...
...

Вывод:

x0 y0 Vx

Примеры входных и выходных файлов присоединены

Оценка

За верное вычисление скорости с точностью не хуже требуемой участник получает 10 баллов.
Дополнительные баллы начисляются за точность согласно формуле :
если d<0.001км/с Доп.баллы=10
если d>0.01км/с Доп.баллы=0
если 0.001км/с <d<0.01км/с Доп.Баллы=10* (0.01-d)/(0.01-0.001)
где d - достигнутая точность определения скорости.
За верное вычисление координат с точностью не хуже требуемой участник получает 20 баллов.Дополнительные баллы начислаются за точность согласно формуле :
если d<0.1км Доп.баллы=20
если d>2км Доп.баллы=0
если 0.1км <d< 2км Доп.Баллы=20* (2-d)/(2-0.1)

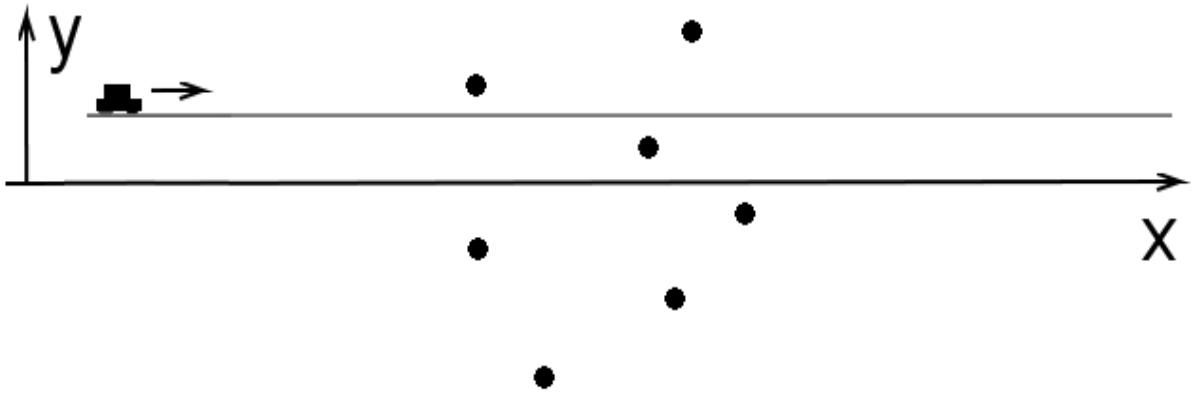
где d - достигнутая точность определения координат.

Таким образом, максимальное число баллов за задачу равно 60.

Характеристики излучателей:

	Положение x_i(км)	Положение y_i(км)	w_i(радиан/сек)
Излучатель 1	10	20	2
Излучатель 2	15	21	3
Излучатель 3	-10	-25	4
Излучатель 4	-7	-5	6
Излучатель 5	8	1	7
Излучатель 6	-17	-5	8
Излучатель 7	18	11	9

Иллюстрация:



Задача 2.4 "Перемешивание". Максимальная оценка 28 баллов.

Условие

Система передачи данных перемешивает в случайном порядке поступающие на ее вход данные блоками по n байт (n = 54) иередает результат на выход. Порядок байт внутри блока не меняется, меняется только порядок блоков.

Необходимо написать две программы (кодировщик и декодировщик) для организации устойчивой передачи данных через такой канал. Тестовые файлы выбираются организаторами трека и представляют собой 4 файла примерно одинаковой длины (9-10Мб каждый) в последовательности: аудиозапись (в формате WAV), документ OOffice без изображений (в формате ODT), текстовый файл (в формате TXT) и изображение (в формате BMP).

Программа-кодировщик, написанная участником должна иметь имя encode_blocks и читать файл ввода (имя - первый аргумент строки вызова) и передавать в файл вывода (имя - второй аргумент строки вызова), учитывая тип файла (значения от 1 до 4: 1-WAV, 2-ODT, 3-TXT, 4-BMP) т.е. работать при вызове:

encode_blocks ВходнойФайл ВыходнойФайл типФайла

Программа-декодировщик, написанная участником, должна иметь имя

decode_blocks и читать файл с файла, задаваемого в строке вызова первым аргументом и передавать его в файл, задаваемым вторым аргументом программы, третий аргумент программы - тип файла (1-WAV, 2-ODT, 3-TXT, 4-BMP).

Т.е. работать при вызове:

decode_blocks ВходнойФайл ВыходнойФайл типФайла

Тестовая программа (mix_blocks), перемешивающая поток данных, сделанная разработчиками задачи, читает файл, сгенерированный encode_blocks и передает в файл вывода для работы decode_blocks, т.е. работает при вызове:

mix_blocks ВходнойФайл ВыходнойФайл

Если размер входного файла не кратен размеру блока перемешивания (54 байт),в конец файла *(до его перемешивания)* дописываются нули, чтобы сделать размер исходного файла кратнымразмеру блока перемешивания. *(Но восстановить надо файл оригинальной длинны, без дополнительных нулей.)*

Тестирование решения представляет собой следующие операции:

1) Кодирование файла программой участника:

encode_blocks ВходнойФайл ЗакодированныйФайл типФайла

если работает дольше 1 минуты - задача не решена

2) Измерение размеров передаваемого файла ЗакодированныйФайл

3) пропуск закодированного файла через программу для перемешивания:

mix_blocks ЗакодированныйФайл ИскаженныйФайл

4) Декодирование файла программой участника:

decode_blocks ИскаженныйФайл РаскодированныйФайл типФайла

если работает дольше 1 минуты - задача не решена

5) Проверка идентичности файлов ВходнойФайл и РаскодированныйФайл

если ВходнойФайл и РаскодированныйФайл идентичны, задача считается решеннойи в зависимости от размера файла ЗакодированныйФайл начисляются баллы.

За каждый правильно переданный файл начисляется 4 балла. Дополнительные баллы начисляются за все файлы, кроме ODT за переданный объем данных согласно формуле:

если d<0.7*R Доп.баллы=4

если d>1.1*R Доп.баллы=0

если 0.7*R<d<1.1*R Доп.баллы = 4*(1.1*R-d)/(1.1*R-0.7*R)

где R - размер исходного файла, d - размер передаваемого файла.

Таким образом, правильно передавший все файлы может получить от 16 до 28 баллов.

Ограничение на быстродействие алгоритма шифрации-дешифрации - по одной минуте на каждый.

Если необходимы библиотеки или программы из дистрибутива, перечислите их в файле README.txt в формате вызовов менеджера пакетов apt-get

Формат входных данных:

Выглядит примерно так:

-3.000000e+03 7.633277e-01 4.078850e-01 7.480130e-02 3.000088e-01 1.933265e-01 2.810929e-02 1.140790e+00

-2.999970e+03 6.902201e-01 4.504225e-01 5.947929e-02 5.816752e-01 1.082141e-01 1.447245e-01 1.470185e+00

-2.999940e+03 6.119836e-01 4.808357e-01 4.314489e-02 8.677065e-01 3.655547e-02 3.052178e-01 1.423073e+00

-2.999910e+03 5.307278e-01 4.972847e-01 2.754262e-02 1.089970e+00 1.422651e-03 4.426967e-01 1.024084e+00

-2.999880e+03 4.486434e-01 4.987744e-01 1.433881e-02 1.195523e+00 1.412734e-02 4.998610e-01 4.818057e-01

-2.999850e+03 3.679437e-01 4.852147e-01 4.943637e-03 1.159223e+00 7.057899e-02 4.528849e-01 7.973762e-02

-2.999820e+03 2.908047e-01 4.574260e-01 3.605147e-04 9.897153e-01 1.526019e-01 3.213478e-01 2.807748e-02

-2.999790e+03 2.193062e-01 4.170892e-01 1.078921e-03 7.273779e-01 2.337870e-01 1.600734e-01 3.538328e-01

-2.999760e+03 1.553760e-01 3.666445e-01 7.022130e-03 4.346991e-01 2.879952e-01 3.627986e-02 8.867012e-01

-2.999730e+03 1.007379e-01 3.091435e-01 1.755540e-02 1.813950e-01 2.977731e-01 1.563342e-03 1.348103e+00

-2.999700e+03 5.686507e-02 2.480646e-01 3.155377e-02 2.780248e-02 2.599724e-01 7.039350e-02 1.496822e+00

-2.999670e+03 2.494045e-02 1.871029e-01 4.752222e-02 1.050731e-02 1.867638e-01 2.140824e-01 1.255107e+00

-2.999640e+03 5.824837e-03 1.299460e-01 6.375529e-02 1.336292e-01 1.017184e-01 3.727415e-01 7.493270e-01

-2.999610e+03 3.363474e-05 8.005173e-02 7.851928e-02 3.678404e-01 3.221825e-02 4.802428e-01

2.438984e-01
-2.999580e+03 7.722992e-03 4.043829e-02 9.023740e-02 6.573521e-01 6.403225e-04 4.917805e-01
3.055740e-03
-2.999550e+03 2.868558e-02 1.350210e-02 9.765814e-02 9.332025e-01 1.715178e-02 4.025458e-01
1.527097e-01
-2.999520e+03 6.235619e-02 8.726294e-04 9.998897e-02 1.129684e+00 7.643643e-02 2.497310e-01
6.146224e-01
-2.999490e+03 1.078270e-01 3.313889e-03 9.698094e-02 1.199995e+00 1.594064e-01 9.702839e-02
1.147309e+00
-2.999460e+03 1.638719e-01 2.067820e-02 8.895533e-02 1.127387e+00 2.393477e-01 8.083251e-03
1.472286e+00
-2.999430e+03 2.289798e-01 5.191512e-02 7.676926e-02 9.291562e-01 2.905218e-01 1.996732e-02
1.419657e+00
-2.999400e+03 3.013952e-01 9.513500e-02 6.172422e-02 6.525200e-01 2.964520e-01 1.277274e-01
1.016937e+00
-2.999370e+03 3.791656e-01 1.477233e-01 4.542703e-02 3.633736e-01 2.552290e-01 2.864498e-01
4.746639e-01
-2.999340e+03 4.601940e-01 2.064987e-01 2.961824e-02 1.305916e-01 1.801254e-01 4.299802e-01
7.633447e-02
-2.999310e+03 5.422957e-01 2.679057e-01 1.598623e-02 9.622466e-03 9.532230e-02 4.984961e-01
3.019212e-02
-2.999280e+03 6.232570e-01 3.282295e-01 5.986903e-03 2.928115e-02 2.812382e-02 4.634406e-01
3.603597e-01
-2.999250e+03 7.008949e-01 3.838209e-01 6.882026e-04 1.848849e-01 1.658468e-04 3.394245e-01
8.942282e-01
-2.999220e+03 7.731161e-01 4.313169e-01 6.560272e-04 4.393690e-01 2.045003e-02 1.781369e-01
1.352695e+00
-2.999190e+03 8.379733e-01 4.678443e-01 5.893813e-03 7.321153e-01 8.244546e-02 4.680134e-02
1.496078e+00
-2.999160e+03 8.937178e-01 4.911935e-01 1.584216e-02 9.933918e-01 1.661914e-01 1.575554e-04
1.249417e+00
-2.999130e+03 9.388464e-01 4.999518e-01 2.943858e-02 1.160962e+00 2.447243e-01 5.764634e-02
7.416646e-01
-2.999100e+03 9.721425e-01 4.935896e-01 4.523097e-02 1.194912e+00 2.927587e-01 1.953068e-01
2.382699e-01
-2.999070e+03 9.927082e-01 4.724916e-01 6.153270e-02 1.087154e+00 2.948290e-01 3.557630e-01
2.403741e-03
-2.999040e+03 9.999891e-01 4.379342e-01 7.660272e-02 8.633555e-01 2.502687e-01 4.721379e-01
1.573750e-01
-2.999010e+03 9.937888e-01 3.920079e-01 8.883157e-02 5.768261e-01 1.734249e-01 4.959273e-01
6.221660e-01
-2.998980e+03 9.742745e-01 3.374909e-01 9.691318e-02 2.958167e-01 8.903891e-02 4.172160e-01
1.153788e+00
-2.998950e+03 9.419724e-01 2.776813e-01 9.998444e-02 8.726370e-02 2.428059e-02 2.688102e-01
1.474312e+00
-2.998920e+03 8.977534e-01 2.161971e-01 9.771735e-02 8.443053e-04 1.981286e-07 1.125645e-01
1.416172e+00

Реальный файл намного больше (20МБайт)

Формат выходных данных:
-16.4000 12.2000 0.1133

Генератор примера:
//C
//Файл generator.c
```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <math.h>
#include "immitate.h"
#include "quasars.h"
#define QUASARS_TOTAL 1
double V_X=0.3;
double V_Y=0.0;
double  X0=-23;
double  Y0=9;
#define T0  0.0
quasars quasars_arr;
void trajectory(double t, double* x,double* y)
{
*x=X0+V_X*(t-T0);
*y=Y0+V_Y*(t-T0);
}
double q_signal(double t,int q_num)
{
double phase;
phase=t*quasars_arr.period[q_num]+quasars_arr.phase[q_num];
return quasars_arr.A[q_num]*cos(phase)*cos(phase);
}
double signal(double t,int i)
{
double x,y;
// int i;
double R;
double res;
res=0;
trajectory(t,&x,&y);
{
R=sqrt((x-quasars_arr.x[i])*(x-quasars_arr.x[i])
      +(y-quasars_arr.y[i])*(y-quasars_arr.y[i]));
if(R<1)
{
fprintf(stderr,"ERROR: trajectory too close to source\n");
exit(1);
}
res+=q_signal(t-R/LIGHT_SPEED,i);
}
return res;
}
int main(int argn, char* argv[])
{
double t;
double x,y;
double s;
if(argn>3)
{
X0=atof(argv[1]);
Y0=atof(argv[2]);
V_X=atof(argv[3]);
V_Y=0.0;//atof(argv[4]);
}
init_quasars(&quasars_arr);
int k;
for(t=-3000;t<3000;t+=0.03)
{
fprintf(stdout,"%le ",t);
for(k=0;k<QUASARS;k++)
{
s=signal(t,k);
fprintf(stdout,"%le",s);
if(k<QUASARS-1)
```

```
fprintf(stdout," ",s);
}
fprintf(stdout,"\n");
}
}
```
// Файл immitate.c
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
double Vd(double t,double Y0,double V0,double T0)
{
double res;
res=-V0*V0*(t-T0);
res/=sqrt(Y0*Y0+V0*V0*(t-T0)*(t-T0));
return res;
}
```
//Файл immitate.h
```
#ifndef __IMMITATE_H__
#define __IMMITATE_H__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define LIGHT_SPEED 0.3
double Vd(double t,double Y0,double V0,double T0);
#endif
```
//Файл quasars.c
```
#include "quasars.h"
void init_quasars(quasars* quasars_arr)
{
quasars_arr->x[0]=10; quasars_arr->y[0]=20;
quasars_arr->x[1]=15;  quasars_arr->y[1]=21;
quasars_arr->x[2]=-10;  quasars_arr->y[2]=-25;
quasars_arr->x[3]=-7;  quasars_arr->y[3]=-5;
quasars_arr->x[4]=8;  quasars_arr->y[4]=1;
quasars_arr->x[5]=-17;  quasars_arr->y[5]=-5;
quasars_arr->x[6]=18;  quasars_arr->y[6]=11;
quasars_arr->period[0]=2;  quasars_arr->phase[0]=0.1; quasars_arr->A[0]=1.;
quasars_arr->period[1]=3;  quasars_arr->phase[1]=-1.1; quasars_arr->A[1]=0.5;
quasars_arr->period[2]=4;  quasars_arr->phase[2]=0.6; quasars_arr->A[2]=0.1;
quasars_arr->period[3]=6;  quasars_arr->phase[3]=-2.6; quasars_arr->A[3]=1.2;
quasars_arr->period[4]=7;  quasars_arr->phase[4]=-1.6; quasars_arr->A[4]=0.3;
quasars_arr->period[5]=8;  quasars_arr->phase[5]=-0.6; quasars_arr->A[5]=0.5;
quasars_arr->period[6]=9;  quasars_arr->phase[6]=-0.6; quasars_arr->A[6]=1.5;
}
```
//Файл quasars.h
```
#ifndef __QUASARS_H__
#define __QUASARS_H__
#include <stdio.h>
#include <math.h>
#define QUASARS 7
typedef struct{
double A[100];
double x[100];
double y[100];
double period[100];
double phase[100];
}quasars;
void init_quasars(quasars* quasars_arr);
#endif
```

Решение:
//C

```
//Файл solution.c
#include <stdio.h>
#include <math.h>
#include "quasars.h"
#include "immitate.h"
#define FFT_LEN 1024
#define FFT_OFFSET ((long)(FFT_LEN/10))
#include "immitate.h"
#define TOTAL_QUASARS QUASARS
#define MAX_LENGTH 400000
long dI=25;
double pow2(double x)
{
return x*x;
}
long fill_arrays(double** src_full,double* time_full,long quasars_num,long
max_length)
{
long i,k;
double r;
for(i=0;!feof(stdin) && i<max_length;i++)
{
fscanf(stdin,"%le",&(time_full[i]));
for(k=0;k<quasars_num;k++)
{
fscanf(stdin,"%le",&(src_full[k][i]));
}
}
return i;
}
int init_arrays(double* src_full[],double** src_time,long quasars_num,long
max_length)
{
long k;
for(k=0;k<quasars_num;k++)
{
src_full[k]=(double*)calloc(max_length,sizeof(double));
if(src_full[k]==NULL) return -1;
}
*src_time=(double*)calloc(max_length,sizeof(double));
if(*src_time==NULL)
return -1;
return 0;
}
int init_calc_arrays(double** Time,double* FreqMax[],double* TimeMax[],long
quasars_num,long max_length)
{
long k;
// fprintf(stderr,"init arrays, stage 2\n") ;
*Time=(double*)calloc(max_length,sizeof(double));
for(k=0;k<quasars_num;k++)
{
FreqMax[k]=(double*)calloc(max_length,sizeof(double));
TimeMax[k]=(double*)calloc(max_length,sizeof(double));
}
return 0;
}


void get_individual_track_param(double* FreqMax[],double* TimeMax[],long
CountMax[],double known_w0,long k,double* Yopt,double* new_t,double* los_V)
{
double left_freq,right_freq;
double s,max_s,max_s_pos;
```

```
long i;
double exact_freq;
double exact_t;
double dlos_V;
double Ypos;
double sigma,sigma_opt=1e100;
double t0;
max_s=0;
right_freq=-1e100;
left_freq=-1e100;
max_s_pos=-1e100;
s=0;
for(i=0;i<CountMax[k];i++)
{
exact_freq=FreqMax[k][i];
exact_t=TimeMax[k][i];
//Detect los Velocity
if(!(right_freq>-1e100))
right_freq=exact_freq;
if(!(left_freq>-1e100))
left_freq=exact_freq;
if(left_freq<exact_freq)
left_freq=exact_freq;
if(right_freq>exact_freq)
right_freq=exact_freq;
//Detect Time of closest position
s+=(exact_freq-known_w0>0)?1:-1;
if(s>max_s)
{
max_s=s;
max_s_pos=exact_t;
}
}
dlos_V=((right_freq+left_freq)/2.-known_w0)/(right_freq+left_freq)*LIGHT_SPEED;
los_V[k]=-(right_freq-left_freq)/(right_freq+left_freq)*LIGHT_SPEED;
if(fabs(dlos_V)>0.01)
{
new_t[k]=TimeMax[k][0];
return;
}
else
{
}
new_t[k]=max_s_pos;
t0=max_s_pos;
sigma_opt=1e100;
{
for(Ypos=-50.;Ypos<50.;Ypos+=0.5)
{
sigma=0.;
for(i=0;i<CountMax[k];i++)
{
exact_freq=FreqMax[k][i];
exact_t=TimeMax[k][i];
sigma+=pow(exact_freq-
(Vd(exact_t,Ypos,los_V[k],t0)/LIGHT_SPEED*known_w0+known_w0),2.0);
}
if(sigma<sigma_opt)
{
sigma_opt=sigma;
Yopt[k]=Ypos;
new_t[k]=t0;
}
}
```

```
}
for(i=0;i<CountMax[k];i++)
{
exact_freq=FreqMax[k][i];
exact_t=TimeMax[k][i];
}
}
void trajectory(double t,double x0,double y0,double Vx,double Vy,double*
x,double* y)
{
*x=x0+Vx*t;
*y=y0+Vy*t;
return;
}
void full_fit(double** FreqMax,double** TimeMax,quasars quasars_arr,long
quasars_num,long* CountMax,double* los_V,double* x0_opt,double* y0_opt,double*
Vx_opt,double* Vy_opt,double xmin,double xmax,double ymin,double ymax,double
dr,double *new_t)
{
double sigma_opt=1e100, sigma;
double phi_opt;
double x0,y0,t0,phi;
long i,k;
double Vx,Vy;
double minimal_range[10], minimal_range_moment[10], range;
double exact_freq,exact_t;
double x1,y1;
double known_w0;
t0=0;
phi=0;
x0=-15;
y0=-15;
double avrg_losV=0;
double max_losV=0;
for(k=0;k<quasars_num;k++)
{
avrg_losV+=los_V[k];
max_losV=(max_losV>los_V[k])?max_losV:los_V[k];
}
avrg_losV/=(double)quasars_num;
phi=0;
for(x0=xmin;x0<xmax;x0+=dr)
for(y0=ymin;y0<ymax;y0+=dr)
{
sigma=0;
for(k=0;k<quasars_num;k++)
{
if(fabs(new_t[k]-TimeMax[k][0])<fabs(TimeMax[k][0]-TimeMax[k][CountMax[k]-
1])/10. || fabs(new_t[k]-TimeMax[k][CountMax[k]-1])<fabs(TimeMax[k][0]-
TimeMax[k][CountMax[k]-1])/10.)
{
continue;
}
Vx=fabs(los_V[k])*cos(phi);
Vy=fabs(los_V[k])*sin(phi);
known_w0=quasars_arr.period[k];
minimal_range[k]=1e100;
minimal_range_moment[k]=-1e100;
for(i=0;i<CountMax[k];i+=dI)
          {
            exact_freq=FreqMax[k][i];
            exact_t=TimeMax[k][i];
            trajectory(exact_t,x0,y0,Vx,Vy,&x1,&y1);
            range=sqrt(pow2(x1-quasars_arr.x[k])+pow2(y1-quasars_arr.y[k]));
```

```
            if(minimal_range[k]>range)
             {
              minimal_range[k]=range;
              minimal_range_moment[k]=exact_t;
             }
           }
for(i=0;i<CountMax[k];i+=dI)
          {
            exact_freq=FreqMax[k][i];
            exact_t=TimeMax[k][i];
            sigma+=pow2(exact_freq-
(Vd(exact_t,minimal_range[k],los_V[k],minimal_range_moment[k])/LIGHT_SPEED*known
_w0+known_w0));
          }
        }
if(sigma<sigma_opt)
{
*x0_opt=x0;
*y0_opt=y0;
phi_opt=phi;
sigma_opt=sigma;
sigma=0;
        k=0;
known_w0=quasars_arr.period[k];
}
}
*Vx_opt=avrg_losV*cos(phi_opt);
*Vy_opt=avrg_losV*sin(phi_opt);
}
int main()
{
double* FreqMax[10];
double* TimeMax[10];
long CountMax[10];
double* Time;
long k;
double* src_full[10];
double* src_time;
double src_re[FFT_LEN*2];
long i,j;
double t,old_t;
double avrg=0.,avrg_n=0.;
double dt;
double max_diff_t=-1e100;
i=0;
double max_diff=0;
long pos=0;
double los_V[10];
double Yopt[10];
double new_t[10];
quasars quasars_arr;
init_arrays(src_full,&src_time,(long)TOTAL_QUASARS,(long)MAX_LENGTH);
init_calc_arrays(&Time,FreqMax,TimeMax,(long)TOTAL_QUASARS,(long)MAX_LENGTH);
long length;
FILE* stream;


length=fill_arrays(src_full,src_time,(long)TOTAL_QUASARS,(long)MAX_LENGTH);
if(length>=MAX_LENGTH-1)
{
fprintf(stderr,"file too long to process\n");
exit(1);
}
double max_ampl[TOTAL_QUASARS];
```

```
double min_ampl[TOTAL_QUASARS];
double ampl;
for(k=0;k<TOTAL_QUASARS;k++)
{
max_ampl[k]=0;
min_ampl[k]=1e100;
for(i=0;i<length;i++)
{
if(max_ampl[k]<src_full[k][i])
max_ampl[k]=src_full[k][i];
if(min_ampl[k]>src_full[k][i])
min_ampl[k]=src_full[k][i];
}
ampl=(max_ampl[k]-min_ampl[k])/2.;
double c_,s_;
double ph_old,ph;
double w0,w0_old;
ph_old=0;
w0_old=0;
double dt=src_time[2]-src_time[1];
for(j=i=0;i<length;i++)
{
c_=(src_full[k][i]-ampl)/ampl;
s_=sqrt(1.-c_*c_);
ph=atan2(s_,c_);
w0=ph-ph_old;
ph_old=ph;
if(i>0 && i<length-1)
{
if(fabs((w0_old-w0)/dt)<0.1)
{
FreqMax[k][j]=fabs(w0)/(2.*dt);
TimeMax[k][j]=src_time[i];
j++;
}
}
w0_old=w0;
}
CountMax[k]=j;
}
init_quasars(&quasars_arr);
for(k=0;k<TOTAL_QUASARS;k++)
{
double known_w0=quasars_arr.period[k];
get_individual_track_param(FreqMax,TimeMax,CountMax,known_w0,k,Yopt,new_t,los_V)
;
}
double x0_opt;
double y0_opt;
double Vx_opt;
double Vy_opt;
full_fit(FreqMax,TimeMax,quasars_arr,TOTAL_QUASARS,CountMax,los_V,&x0_opt,&y0_op
t,&Vx_opt,&Vy_opt,-30,30,-30,30,1,new_t); //0.3 - losV
full_fit(FreqMax,TimeMax,quasars_arr,TOTAL_QUASARS,CountMax,los_V,&x0_opt,&y0_op
t,&Vx_opt,&Vy_opt,x0_opt-1.,x0_opt+1.,y0_opt-1.,y0_opt+1.,0.2,new_t); //0.3 -
losV
fprintf(stdout,"%2.4lf %2.4lf %2.4lf\n",x0_opt,y0_opt,Vx_opt);

}
```

Критериоценки:

//Perl

```
#!/usr/bin/perl
$params[1]='-16 12 0.111';
```

```
$params[2]='-11 -12 0.0422';
$params[3]='-7 22 0.0233';
$params[4]='17 -10 0.0144';
$params[5]='14 23 0.155';
# по всем участникам
for($sol=-2;$sol<=6;$sol++)
{
$sum_dv=0;
$sum_dr=0;
open RESFILE,">$sol/results";
$codename="solution_1_$sol";
for($i=1;$i<=5;$i++)
{
$fsrc=$params[$i];
`./generator $fsrc >test.dat`;
($solution_file,$rest)=split(/\s+/,`ls ./$sol/solution.*`);
print "$solution_file\n";
compile($solution_file);
$time=run($solution_file,"test.dat","sol");
$res=`cat sol`;
$res=~ s/[\n\r]//g;
@res_user=split(/\s+/,$res);
@res_src=split(/\s+/,$fsrc);
$dr=sqrt((@res_user[0]-@res_src[0])*(@res_user[0]-@res_src[0])+(@res_user[1]-
@res_src[1])*(@res_user[1]-@res_src[1]));
$dv=abs(@res_user[2]-@res_src[2]);
$sum_dv+=$dv;
$sum_dr+=$dr;
if(abs(@res_user[0]-@res_src[0])<=2 && abs(@res_user[1]-@res_src[1])<=2 &&
abs(@res_user[2]-@res_src[2])<=2)
{
print RESFILE "$codename $fsrc  your solution: $res GOOD!  TIME:$time dR: $dr
dV:$dv\n";
}
else
{
if(-s "$fsrc.errors" >0)
{ print RESFILE  "$codename $fsrc COMPILE_ERROR TIME:$time\n";
$sum_dv+=1000;
$sum_dr+=1000;
}
else
{ print RESFILE  "$codename $fsrc your solution: $res  ERROR TIME:$time  dR: $dr
dV:$dv\n"; }
}
`rm -f test.dat sol`;
}
$sum_dv/=5;
$sum_dr/=5;
$ball=0;
if($sum_dv<=0.01)
{$ball+=10;
if($sum_dv<0.001)
{
$ball+=10;
}
else
{
$ball+=10*(0.01-$sum_dv)/(0.01-0.001);
}
}
if($sum_dr<=2)
{$ball+=20;
if($sum_dr<0.1)
```

```
{
$ball+=20;
}
else
{
$ball+=20*(2-$sum_dr)/(2-0.1);
}
}
print RESFILE  "\n Средняя ошибка по скорости: $sum_dv\n";
print RESFILE  "\n Средняя ошибка по пространству: $sum_dr\n";
print RESFILE  "\n Баллы: $ball\n";
close RESFILE;
}
sub compile()
{
$fname=shift;
`rm -f *.class *.java *.py solution`;
if($fname =~ /\.cpp/)
{
`rm -f solution`;
`g++ $fname -o solution -std=c++11 2>$fname.errors`;
#   exit;
return;
}
if($fname =~ /\.java/)
{
`rm -f *.class *.java`;
`cp $fname solution.java`;
`javac solution.java 2>$fname.errors`;
return;
}
if($fname =~ /\.py/)
{
`rm -f solution.py`;
`cp $fname solution.py`;
return;
}
if($fname =~ /\.c/)
{
`rm -f solution`;
`gcc $fname -o solution 2>$fname.errors`;
return;
}
}
sub run()
{
$fname=shift;
$src=shift;
$dest=shift;
if($fname =~ /\.cpp/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.java/)
{
`rm -f tm`;
`time -f %U java solution <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.py/)
```

```
{
`rm -f tm`;
`time -f %U python ./solution.py   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
if($fname =~ /\.c/)
{
`rm -f tm`;
`time -f %U ./solution   <$src >$dest 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
return $time;
}
return 1000;
}
```

Принцип решения:

Решение задачи проводится на основе эффекта доплеровского сдвига частоты - изменение частоты сигнала, принимаемого движущимся приемником, пропорционально проекции скорости его движения на линию, соединяющую приемник и передатчик.

Поскольку по условиям задачи, запись сигналов идет издалека, для оценки скорости движения приемника можно просто посчитать частоту сигнала в начале файла и частоту сигнала в конце файла по каждому из каналов. Они позволят определить скорость с достаточно высокой точностью. Определив ее независимо по каждому из 7 передатчиков и усреднив полученные значения, можно определить скорость с достаточно высокой степенью точности.

Определение частоты можно проводить различными методами, самый простой - посчитать период функции на каком-то промежутке и взять от него обратную величину. Для точного определения скорости этого достаточно. Можно использовать преобразование Фурье (спектр). Можно использовать и другие способы.

Задача определения положения сложнее. Для этого надо получить формулу для временной зависимости проекции скорости на луч зрения приемник-передатчик, как функцию момента времени и координаты, в которых приемник наиболее близко подходит к передатчику. Эта формула достаточно легко получается из геометрии и описана в файле immitate.c. После этого надо определить параметры момента приближения и его координаты по данным наблюдений. Самый простой способ - просто перебрать по сетке все варианты координаты Y. Времени на проведение такого поиска достаточно. Можно придумать более сложные, но быстрые методы.

Однако здесь встает проблема того, что для точного определения положения надо очень точно знать частоту в каждый момент времени, а она в моменты максимального приближения приемника к передатчику меняется очень быстро. Поэтому простые способы ее определения, описанные выше, например по среднему периоду, не годятся. Один из пригодных способов - воспользоваться формулой Эйлера, и пользуясь тем, что амплитуда сигнала во времени не меняется, сделать вторую компоненту сигнала (мнимую) по известной формуле $\cos^2(x)+\sin^2(x)=1$ и посчитать фазу сигнала x по формуле Эйлера (или по формуле арктангенса). Производная фазы x по времени будет определять частоту сигнала даже при ее быстрых изменениях и поэтому работает устойчиво даже при предельных приближениях приемника к передатчику, надо только аккуратно отбросить переход фазы через 360 градусов, но это легко.

Как показало решение, такого подхода достаточно для решения задачи определения координат. Для увеличения точности сначала ведется грубое, а потом более точное определение координат, при этом подгонять доплеровское смещение надо сразу по всем передатчикам одновременно, чтобы обеспечить точность.

Задача 2.4 "Перемешивание". Максимальная оценка 28 баллов.

Условие:

Система передачи данных перемешивает в случайном порядке поступающие на ее вход данные блоками по n байт (n = 54) и передает результат на выход. Порядок байт внутри блока не меняется, меняется только порядок блоков.
Необходимо написать две программы (кодировщик и декодировщик) для организации устойчивой передачи данных через такой канал. Тестовые файлы выбираются организаторами трека и

представляют собой 4 файла примерно одинаковой длины (9-10Мб каждый) в последовательности: аудиозапись (в формате WAV), документ OOffice без изображений (в формате ODT), текстовый файл (в формате TXT) и изображение (в формате BMP).

Программа-кодировщик, написанная участником должна иметь имя encode_blocks и читать файл ввода (имя - первый аргумент строки вызова) и передавать в файл вывода (имя - второй аргумент строки вызова), учитывая тип файла (значения от 1 до 4: 1-WAV, 2-ODT, 3-TXT, 4-BMP) т.е. работать при вызове:

encode_blocks ВходнойФайл ВыходнойФайл типФайла

Программа-декодировщик, написанная участником, должна иметь имя decode_blocks и читать файл с файла, задаваемого в строке вызова первым аргументом ипередавать его в файл, задаваемым вторым аргументом программы, третий аргумент программы - тип файла (1-WAV, 2-ODT, 3-TXT, 4-BMP).Т.е. работать при вызове:decode_blocks ВходнойФайл ВыходнойФайл типФайла

Тестовая программа (mix_blocks), перемешивающая поток данных, сделаннаяразработчиками задачи, читает файл, сгенерированный encode_blocks и передает в файл вывода для работы decode_blocks, т.е. работает при вызове:

mix_blocks ВходнойФайл ВыходнойФайл

Если размер входного файла не кратен размеру блока перемешивания (54 байт),в конец файла *(до его перемешивания)* дописываются нули, чтобы сделать размер исходного файла кратнымразмеру блока перемешивания. *(Но восстановить надо файл оригинальной длинны, без дополнительных нулей.)*

Тестирование решения представляет собой следующие операции:

1) Кодирование файла программой участника:

encode_blocks ВходнойФайл ЗакодированныйФайл типФайла

если работает дольше 1 минуты - задача не решена

2) Измерение размеров передаваемого файла ЗакодированныйФайл

3) пропуск закодированного файла через программу для перемешивания:

mix_blocks ЗакодированныйФайл ИскаженныйФайл

4) Декодирование файла программой участника:

decode_blocks ИскаженныйФайл РаскодированныйФайл типФайла

если работает дольше 1 минуты - задача не решена

5) Проверка идентичности файлов ВходнойФайл и РаскодированныйФайл

Если ВходнойФайл и РаскодированныйФайл идентичны, задача считается решеннойи в зависимости от размера файла ЗакодированныйФайл начисляются баллы.

За каждый правильно переданный файл начисляется 4 балла. Дополнительные баллы начисляются за все файлы, кроме ODT за переданный объем данных согласно формуле:

если d<0.7*R Доп.баллы=4

если d>1.1*R Доп.баллы=0

если 0.7*R<d<1.1*R Доп.баллы = 4*(1.1*R-d)/(1.1*R-0.7*R)

где R - размер исходного файла, d - размер передаваемого файла.

Таким образом, правильно передавший все файлы может получить от 16 до 28 баллов.

Ограничение на быстродействие алгоритма шифрации-дешифрации - по одной минуте на каждый.

Если необходимы библиотеки или программы из дистрибутива, перечислите их в файле README.txt в формате вызовов менеджера пакетов apt-get

Генератор mix_blocks:

//C

//mix_blocks.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "code_decode.h"
#define BLOCK_SIZE FULL_BLOCK_LEN
#define MAX_BLOCKS 3000
int main(int argn, char* argv[])
{
long i,pos,len,j;
char* blocks[MAX_BLOCKS];
long num_blocks;
```

```
long transfer_bytes=0;
for(i=0;i<MAX_BLOCKS;i++)
blocks[i]=calloc(BLOCK_SIZE,sizeof(char));
char* tmp_block;
tmp_block=calloc(BLOCK_SIZE,sizeof(char));
FILE* in;
FILE* out;
in=fopen(argv[1],"rb");
out=fopen(argv[2],"wb");
// for(;!feof(stdin);)
for(;!feof(in);)
{
num_blocks=0;
for(i=0;!feof(in)&&i<MAX_BLOCKS;i++)
{
memset(blocks[i],0,BLOCK_SIZE);
len=fread(blocks[i],sizeof(char),BLOCK_SIZE,in);
num_blocks=i+1;
if(len<BLOCK_SIZE)
{
if(len==0)
num_blocks--;
break;
}
}
fprintf(stderr,"nb %ld\n",num_blocks);
if(num_blocks>1)
{
for(i=0;i<num_blocks-1;i++)
{
pos=i+(1+(rand()%(num_blocks-i-1)));
fprintf(stderr,"mix %ld with %ld\n",i,pos);
memcpy(tmp_block,blocks[pos],BLOCK_SIZE);
memcpy(blocks[pos],blocks[i],BLOCK_SIZE);
memcpy(blocks[i],tmp_block,BLOCK_SIZE);
}
for(i=0;i<num_blocks;i++)
{
fwrite(blocks[i],sizeof(char),BLOCK_SIZE,out);
transfer_bytes+=BLOCK_SIZE;
for(j=0;j<BLOCK_SIZE;j++)
blocks[i][j]=0;
}
}
}
fprintf(stderr,"bytes transfered:%ld\n",transfer_bytes);
fclose(out);
}
```

//code_decode.h

```
#ifndef __CODE_DECODE_H__
#define __CODE_DECODE_H__
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define FULL_BLOCK_LEN  54
#define MAX_FILE_SIZE   20000000
#endif
```

Решение:

//code_block.c

```
#include "code_decode.h"
#include "compr.h"
#include <string.h>
#include <unistd.h>
```

```
#include <fcntl.h>
int main(int argn,char* argv[])
{
char* tmp_file1;
char* tmp_file2;
long compr_len;
unsigned long code_block=0;
long i,j;
char* block;
int mode;
if(argn>3)
mode=atol(argv[3]);
else
mode=1;
FILE* in;
FILE* out;
tmp_file1=calloc(sizeof(char),40000000);
tmp_file2=calloc(sizeof(char),40000000);
long file_len,block_len;
block=calloc(DATA_BLOCK_LEN,sizeof(char));
code_block=0;
file_len=0;
in=fopen(argv[1],"rb");
out=fopen(argv[2],"wb");
for(i=0;!feof(in);)
{
block_len=fread(block,sizeof(char),DATA_BLOCK_LEN,in);
memcpy(tmp_file1+i,block,block_len);
i+=block_len;
}
compr_len=20000000;
zlib_compress(tmp_file1,tmp_file2,i,&compr_len);
file_len=0;
for(j=0;j<compr_len;j+=DATA_BLOCK_LEN)
{
block_len=DATA_BLOCK_LEN;
if(j+DATA_BLOCK_LEN<=compr_len)
memcpy(block,tmp_file2+j,DATA_BLOCK_LEN);
else
{
block_len=compr_len-j;
memcpy(block,tmp_file2+j,block_len);
}
file_len+=block_len;
for(i=block_len;i<DATA_BLOCK_LEN;i++)
block[i]=0;
fwrite(block,sizeof(char),DATA_BLOCK_LEN,out);
fwrite(&code_block,sizeof(char),CODE_BLOCK_LEN,out);
for(i=0;i<DATA_BLOCK_LEN;i++)
block[i]=0;
code_block++;
}
code_block=-1;
fprintf(stderr,"file len: %ld\n",file_len);
memcpy(block,&file_len,sizeof(long));
fwrite(block,sizeof(char),DATA_BLOCK_LEN,out);
fwrite(&code_block,sizeof(char),CODE_BLOCK_LEN,out);
fflush(out);
fclose(out);
}
```

//compr.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // for strlen
```

```
#include <assert.h>
#include "zlib.h"
int zlib_compress (char* src,char* dest,long len_src,long* len_dest)
{
z_stream defstream;
defstream.zalloc = Z_NULL;
defstream.zfree = Z_NULL;
defstream.opaque = Z_NULL;
// setup "a" as the input and "b" as the compressed output
defstream.avail_in = /*(uInt)*/len_src; // size of input, string + terminator
defstream.next_in = (Bytef *)src; // input char array
defstream.avail_out = /*(uInt)*/(*len_dest); // size of output
defstream.next_out = (Bytef *)dest; // output char array
// the actual compression work.
deflateInit(&defstream, Z_BEST_COMPRESSION);
deflate(&defstream, Z_FINISH);
deflateEnd(&defstream);
*len_dest=(long)((char*)defstream.next_out-dest);
}
int zlib_uncompress (char* src,char* dest,long len_src,long* len_dest)
{
z_stream infstream;
infstream.zalloc = Z_NULL;
infstream.zfree = Z_NULL;
infstream.opaque = Z_NULL;
// setup "b" as the input and "c" as the compressed output
infstream.avail_in = (uInt)(len_src); // size of input
infstream.next_in = (Bytef *)src; // input char array
infstream.avail_out = (uInt)(*len_dest); // size of output
infstream.next_out = (Bytef *)dest; // output char array
// the actual DE-compression work.
inflateInit(&infstream);
inflate(&infstream, Z_NO_FLUSH);
inflateEnd(&infstream);
*len_dest=(long)((char*)infstream.next_out-dest);
}
```

//compr.h

```
#ifndef __COMPR_H__
#define __COMPR_H__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // for strlen
#include <assert.h>
#include "zlib.h"
int zlib_compress (char* src,char* dest,long len_src,long* len_dest);
int zlib_uncompress (char* src,char* dest,long len_src,long* len_dest);
#endif
```

//decode_block.c

```
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include "code_decode.h"
#include "compr.h"
int main(int argn,char* argv[])
{
int mode;
if(argn>3)
mode=atol(argv[3]);
else
mode=1;
unsigned long code_block=0;
long i,j,length;
char* block;
unsigned char* file_content;
```

```
unsigned char* new_file_content;
long file_len;
long block_len;
FILE* in;
FILE* out;
in=fopen(argv[1],"rb");
out=fopen(argv[2],"wb");
file_content=calloc(MAX_FILE_SIZE+1,sizeof(char));
if(!file_content)
{fprintf(stderr,"no memory\n");exit(1);}
new_file_content=calloc(MAX_FILE_SIZE+1,sizeof(char));
if(!new_file_content)
{fprintf(stderr,"no memory\n");exit(1);}
block=calloc(FULL_BLOCK_LEN,sizeof(char));
if(!block)
{fprintf(stderr,"no memory\n");exit(1);}
length=fread(file_content,sizeof(char),MAX_FILE_SIZE+1,in);
if(length>MAX_FILE_SIZE)
{
fprintf(stderr,"file too long to process\n");exit(1);
}
long actual_len;
for( actual_len=i=0;i<length;i+=FULL_BLOCK_LEN)
{
memcpy(&code_block,file_content+i+DATA_BLOCK_LEN,CODE_BLOCK_LEN);
if(code_block==-1)
{
memcpy(&file_len,file_content+i,sizeof(long));
fflush(stderr);
}
else
{
int empty_block=1;
for(j=0;j<DATA_BLOCK_LEN;j++)
if(file_content[i+j]!=0)
{
empty_block=0;break;
}
if(empty_block==0)
{
memcpy(new_file_content+code_block*DATA_BLOCK_LEN,file_content+i,DATA_BLOCK_LEN)
;
actual_len+=DATA_BLOCK_LEN;
}
}
}
fprintf(stderr,"flen:%ld\n",file_len);
char *tmp1;
long new_len=20000000;
tmp1=calloc(sizeof(char),40000000);
zlib_uncompress(new_file_content,tmp1,actual_len,&new_len);
fwrite(tmp1,sizeof(char),new_len,out);
fclose(out);
}
```

Критериоценки:

//Perl

```
#!/usr/bin/perl
# файлы, использованные для тестов
$params[1]='test.bmp';
$params[2]='test.odt';
$params[3]='test.txt';
$params[4]='test.wav';
#по всем участникам
```

```
for($sol=-2;$sol<=15;$sol++)
{
$codename="solution_4_$sol";
open RESFILE,">$sol/results.txt";
print "rest $codename\n";
$ball=0;
for($i=1;$i<=4;$i++)
{
print "rest file $i\n";
`rm *.class decode_blocks* encode_blocks* *.java output* input* *.h `;
$fsrc=$params[$i];
($solution_file,$rest)=split(/\s+/,`ls ./$sol/encode_blocks.*`);
$sf=$solution_file;
$sf=~ s/encode/decode/;
compile($solution_file);
compile($sf);
`rm -f $solution_file.$i.log sol`;
`rm -f tmp1 tmp2`;
$time=run($solution_file,$fsrc,"sol",$i);
$size=(-s "tmp2")/(-s $fsrc);
if(-s "sol")
{
`diff $fsrc sol > $solution_file.$i.log`;
$DIFF=`cat $solution_file.$i.log`;
}
else
{
$DIFF = -1;
}
if(!$DIFF || $DIFF=='')
{
$count=4;
if($i!=2)
{
if($size>1.1)
{ $count+=0; }
else
{
if($size<0.7)
{ $count+=4; }
else
{ $count+=4*(1.1-$size)/(1.1-0.7); }
}
}
print RESFILE "$codename $fsrc ($DIFF) TIME:$time SIZE: $size COUNT: $count\n";
$ball+=$count;
}
else
{
if(-s "$fsrc.errors" >0)
{ print RESFILE  "$codename $fsrc COMPILE_ERROR TIME:$time SIZE:$size\n";  }
else
{ print RESFILE  "$codename $fsrc ERROR $DIFF TIME:$time SIZE:$size\n"; }
}
`rm -f test.dat sol`;
}
print RESFILE  "Баллы: $ball\n";
print RESFILE  "\n";
close RESFILE;
}
sub compile()
{
$fname=shift;
# `rm -f *.class *.java *.py solution`;
```

```
`rm -f solution`;
if($fname =~ /\.cpp/)
{
if($fname=~ /decode/)
{
`rm -f decode_blocks`;
`g++ $fname -o decode_blocks -lz -lm -std=c++11 2>$fname.errors`;
}
else
{
`rm -f encode_blocks`;
`g++ $fname -o encode_blocks -lz -lm -std=c++11 2>$fname.errors`;
}
#   exit;
return;
}
if($fname =~ /\.java/)
{
if($fname=~ /decode/)
{
`rm -f decode_blocks.java`;
#   `rm -f *.class *.java`;
`cp $fname decode_blocks.java`;
`javac decode_blocks.java 2>$fname.errors`;
}
else
{
`rm -f encode_blocks.java`;
#   `rm -f *.class *.java`;
`cp $fname encode_blocks.java`;
`javac encode_blocks.java`;
}
return;
}
if($fname =~ /\.py/)
{
if($fname=~ /decode/)
{
`rm -f decode_blocks.py`;
`cp $fname decode_blocks.py`;
#      print "cp $fname decode_blocks.py\n";
#      exit;
}
else
{
`rm -f encode_blocks.py`;
`cp $fname encode_blocks.py`;
#      print "cp $fname encode_blocks.py\n";
}
return;
}
if($fname =~ /\.c/)
{
if($fname=~ /decode/)
{
`rm -f decode_blocks`;
`gcc $fname -o decode_blocks -lz -lm 2>$fname.errors`;
}
else
{
`rm -f encode_blocks`;
`gcc $fname -o encode_blocks -lz -lm 2>$fname.errors`;
}
return;
```

```
}
}
sub run()
{
$fname=shift;
$src=shift;
$dest=shift;
$mode=shift;
if($fname =~ /\.cpp/)
{
`rm -f tm`;
if(-s "encode_blocks" > 0 && -s "decode_blocks" > 0 )
{
`time -f %U ./encode_blocks  $src tmp1 $mode 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
`rm -f tm`;
`./mix_blocks tmp1 tmp2 2>tm`;
`time -f %U ./decode_blocks  tmp2 $dest $mode  2>tm`;
# print STDERR "time -f %U ./decode_blocks  tmp2 $dest $mode  2>tm\n";
($time2,$rest)=split(/\s+/,`more tm`);
return $time+$time2;
}
else
{
return 1000;
}
}
if($fname =~ /\.java/)
{
`rm -f tm`;
`time -f %U java encode_blocks  $src tmp1 $mode  2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
`rm -f tm`;
`./mix_blocks tmp1 tmp2 2>tm`;
`time -f %U java decode_blocks  tmp2 $dest $mode  2>tm`;
($time2,$rest)=split(/\s+/,`more tm`);

return $time+$time2;
}
if($fname =~ /\.py/)
{
`rm -f tm`;
`time -f %U python3 ./encode_blocks.py  $src tmp1 $mode  2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
`rm -f tm`;
`./mix_blocks tmp1 tmp2 2>tm`;
`time -f %U python3 ./decode_blocks.py  tmp2 $dest $mode  2>tm`;
($time2,$rest)=split(/\s+/,`more tm`);
return $time+$time2;
}
if($fname =~ /\.c/)
{
`rm -f tm`;
if(-s "encode_blocks" > 0 && -s "decode_blocks" > 0 )
{
`time -f %U ./encode_blocks  $src tmp1 $mode 2>tm`;
($time,$rest)=split(/\s+/,`more tm`);
`rm -f tm`;
`./mix_blocks tmp1 tmp2 2>tm`;
`time -f %U ./decode_blocks  tmp2 $dest $mode  2>tm`;
($time2,$rest)=split(/\s+/,`more tm`);
return $time+$time2;
}
return 1000;
```

```
}
return 1000;
}
```

Принцип решения

Для решения необходимо было разбить файл на блоки и внутри каждого блока поставить его порядковый номер в файле. Например, можно было разбить исходный файл на блоки по 50 байт, а в оставшиеся 4 байта писать порядковый номер. Получившиеся блоки по 54 байта записывать. Получившийся файл получается примерно на 8% исходного. При сборке файла после прохождения перемешивающей программы надо было снова разбить файл на блоки по 54 байта, выделять из них номер блока и переставлять получившиеся 50-байтные куски по порядковым номерам блоков. Длину исходного файла можно было хранить в отдельном блоке (например в первом).

Для получения дополнительных баллов нужно было оптимизировать работу этой схемы по объему, например воспользоваться каким-то алгоритмом сжатия , проще всего - доступной библиотекой, например zlib. Таким образом, вы считываете файл в память, сжимаете его и после этого уже делите на блоки. После перемешивания снова делите не блоки, собираете их в нужном порядке и разжимаете получившиеся данные в файл. Можно было дополнительно аккуратно подобрать количество байт, требуемых для записи порядкового номера блока, оно зависит от максимального размера обрабатываемых файлов.

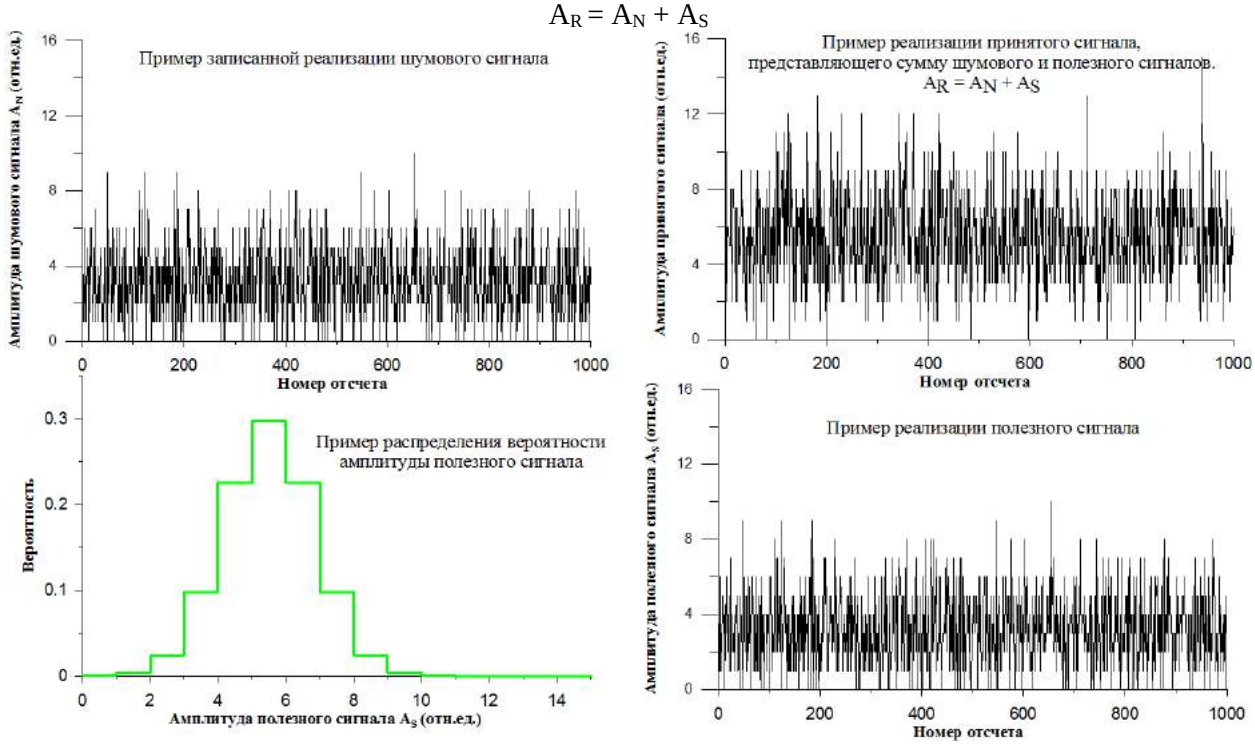
Задача 2.5. Максимальная оценка 35 баллов

Передача информации по нескольким каналам (теория вероятности)

Условие

Необходимо принять с некоторого удаленного объекта сообщение длиной M единиц. Для этого важно понять хватит ли для этого наших возможностей. В нашем распоряжении 5 каналов связи в каждом присутствует свой стационарный шум. Для анализа свойств шума в каждом канале была записана реализация шумового сигнала в N =1000 отсчетов. Также известно, что вероятность передачи данных (полезного сигнала) с удаленного объекта составляет 60%, т.е. 60% времени объект передает данные и 40% времени «молчит». Амплитуда полезного сигнала не постоянна, но для каждого канала известно распределение вероятности его амплитуды. В качестве иллюстрации на рисунке приведены: верхний левый - реализация шумового сигнала, нижний левый - распределение вероятности амплитуды полезного сигнала, нижний правый – пример реализации сигнала, соответствующий распределению на левом нижнем рисунке, правый верхний – пример принятого сигнала A_R, представляющего сумму шума A_N и полезного сигнала A_S:

$$A_R = A_N + A_S$$



Верхний левый рисунок – пример шумового сигнала, нижний левый рисунок – пример распределения вероятности амплитуды полезного сигнала, пример реализации сигнала, соответствующий распределению на левом нижнем рисунке, правый верхний – пример принятого сигнала, представляющего сумму шума и полезного сигнала.

Из рисунка видно, что сигналы в каналах очень слабые, таким образом, рассматривается задача обнаружения сигнала (есть сигнал/нет сигнала). Решение о том, что в принятом сигнале присутствует полезный сигнал, принимается на основе сравнения амплитуды принятого сигнала с пороговым уровнем, если амплитуда больше или равна порогу, считаем, что полезный сигнал присутствует и отсутствует в противном случае. Пороговый уровень должен быть таким, чтобы сумма вероятности пропуска сигнала и вероятности ложных тревог (вероятность превышения шумом порога) была минимальной.

Если использование только одного из представленных каналов не позволяет решить задачу, тогда нужно задействовать несколько каналов и принимать решение о наличии сигнала голосованием – при условии, что хотя бы в двух каналах сигнал выше порогового уровня, то принимаем решение - полезный сигнал присутствует. Для каждого канала известна потребляемая мощность. При использовании нескольких каналов суммируется и их энергопотребление.

Таким образом, необходимо проанализировать представленные шумовые сигналы и гистограммы распределения полезного сигнала, определить пороговый уровень и принять решение о том, какие каналы нужно задействовать, для приема сообщения длинной M=50 с наименьшими энергетическими затратами, так чтобы с вероятностью 99% в сообщении было не более 2 ошибок.

Языки программирования - Python,C,C++,Java

Примеры работы программы решающей предлагаемую задачу присоединены.

Максимальное число баллов - за полностью верный ответ, включающий:

1. Правильно определены номера каналов, использование которых позволит решить задачу с наименьшими энергетическими затратами; (3 балла)
2. Правильно определены пороговые уровни выбранных каналов, перечисленных в пункте 1; (10 баллов)
3. Правильно определена вероятность верного обнаружения единичного сигнала, при использовании всех выбранных каналов, перечисленных в пункте 1 (данную вероятность необходимо определить с точность до 4-го знака); (15 баллов)
4. Правильно определена вероятность того, что с использованием выбранных каналов (пункт 1), сообщение длины M будет передано с не более чем 2 ошибками (данную вероятность необходимо определить с точность до 4-го знака); (5 баллов)
5. Правильно рассчитана энергия, необходимая для передачи единицы сообщения (суммарная энергия всех задействованных каналов). (2 балла)

Штрафы:
В пункте (3):
за ошибку в четвертом знаке снимается 5 баллов.
за ошибку в третьем знаке снимается 15 баллов.
В пункте (4):
за ошибку в четвертом знаке снимается 2 балла.
за ошибку в третьем знаке снимается 5 баллов.

Скорость выполнения программы - не более 10 секунд

Программа должна читать исходный файл со стандартного потока stdin и передавать решение на стандартный поток stdout

Описание формата входных и выходных данных для задачи:
Входные данные (input.dat)
5 – кол-во каналов;

50 – длина передаваемого сообщения;
2 – допустимое количество ошибок;
0.6 - вероятность передачи данных (полезного сигнала);
4.0296 5.0000 3.0038 5.0006 4.0272 – 5 чисел - затраты энергии на передачу сигнала в каждом канале;
15 – количество уровней сигнала;
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 – значение уровней сигнала;
Далее 5 строк - Распределения уровня сигнала в каждом каналах (вероятность встретить сигнал с данным уровнем в данном канале, строка 1 – распределение вероятности в 1 канале, 2-я строка – распределение во втором канале и т.д.)
1000 - Длина шумовой реализации
Далее 5 строк - Реализация шумового сигнала для каждого канала (строка 1- реализация, полученная в канале 1, срока – реализация, полученная в канале 2 и т.д.)
Выходные данные (output.dat)
4 – необходимое кол-во каналов;
0 1 2 4 - номера используемых каналов;
4 5 4 4 – пороговый уровень для каждого из выбранных каналов;
0.996471 – вероятность верного обнаружения единичного сигнала, при использовании всех выбранных каналов;
0.999239 - Вероятность того что сообщение длины 50 будет получено с не более чем 2 ошибками;
16.060588 - Необходимая энергия для передачи единицы сообщения (суммарная энергия всех задействованных каналов)

Алгоритм решения задачи

1. Для каждого канала строим гистограмму распределения шума. Интеграл, полученной гистограммы нормируем на единицу. Полученное распределение обозначим $P_{Ni}(u)$, I – номер канала $i \ni [1,5]$.
2. Для каждого канала на основе распределений сигнала и шума строим распределение “сигнал+шум” $P_{SNi}(u) = \int_0^u P_{Si}(v)P_{Ni}(u-v)dv$. В дальнейшем пороговый уровень будет определяться на основе $P_{Ni}(u)$ и $P_{SNi}(u)$.
3. Используем т. Байесса для того чтобы для каждого канала найти пороговый уровень u_{bi}, при котором минимизируется вероятность ложных тревог и вероятность пропуска.
$$q_i + n_i \to min,$$
$q_i = \int_0^{u_{bi}} P_{SNi}(u)du$ – вероятность пропуска сигнала,
$p_i = 1 - \ q_i$; – вероятностьверного определения сигнала в канале при данном пороговом уровне u_{bi}.
$n_i = \int_{u_{bi}}^{\infty} P_{Ni}(u)du$ – вероятность ложной тревоги.
4. Находим вероятность правильного принятия решения о наличии сигнала, на основе, найденного порогового уровня.
Вероятность принятия правильного решения о наличии сигнала в канале: $P_i = \frac{p*p_i}{q*n_i+p*p_i}$, где p – вероятность присутствия сигнала, q =1-p.
5. Для каждого канала рассчитываем вероятность принятия сообщения из M отсчетов с не более чем двумя ошибками:
$$P_{Mi} = P_i^M + C_M^1 P_i^{M-1}(1-P_i) + C_M^2 P_i^{M-2}(1-P_i)^2$$
6. Если ни один из доступных каналов не обеспечивает $P_{Mi} > P_M$, тогда для приема используем несколько каналов, причем упорядочиваем их по убыванию и считаем, что если хотя бы в двух каналах сигнал превысит пороговый уровень, тогда принимаем решение о наличии сигнала, т.е. если рассматриваем три канала с вероятностями: P1, P2, P3, тогда:
$$P_{123} = P_1P_2P_3 + P_1P_2(1-P_3) + P_1(1-P_2)P_3 + (1-P_1)P_2P_3 = \sum_{j=1}^{3}\prod_{\substack{i=1\\ i\neq j}}^{3} P_i\left(1-P_j\right)$$
далее, если P_{123} больше максимальной P_i, проверяем возможность приема сообщения длинной M с не более чем двумя ошибками.
Если использование трех каналов не удовлетворяет данным требованиям, тогда подобным образом рассматриваем 4 канала, т.е.:

$$P_{1234} = P_1P_2P_3P_4 + \sum_{j=1}^{4}\prod_{\substack{i=1\\i\neq j}}^{4} P_i(1-P_j) + \sum_{j=1}^{4}\sum_{\substack{k=1\\k\neq j}}^{4}\prod_{\substack{i=1\\i\neq k\\i\neq j}}^{4} P_i(1-P_k)(1-P_j).$$

Если использование пяти каналов не удовлетворяет данным требованиям, тогда подобным образом рассматриваем 5 канала, т.е.:

$$P_{12345} = P_1P_2P_3P_4P_5 + \sum_{j=1}^{5}\prod_{\substack{i=1\\i\neq j}}^{5} P_i(1-P_j) + \sum_{j=1}^{5}\sum_{\substack{k=1\\k\neq j}}^{5}\prod_{\substack{i=1\\i\neq k\\i\neq j}}^{5} P_i(1-P_k)(1-P_j)$$
$$+ \sum_{j=1}^{5}\sum_{\substack{k=1\\k\neq j}}^{5}\sum_{\substack{m=1\\m\neq k\\m\neq j}}^{5}\prod_{\substack{i=1\\i\neq m\\i\neq k\\i\neq j}}^{5} P_i(1-P_m)(1-P_k)(1-P_j).$$

Если использование пяти каналов не удовлетворяет данным требованиям, тогда принимаем решение о том, что невозможно принять сообщение длинной M с не более чем двумя ошибками.

Решение на языке C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
int K;//Количество каналов
int N;//размер тестового массива
int M;//Длина контрольного сообщения
int merr;//Допустимое количество битых символов
int n;//Фактическое количество уровней сигнала
int n2;//Количество уровней сигнала
double Ps;//Вероятность того что в данной единицы времени будет передан сигнал
double Pn;
double PT=0.99;//Требуемая вероятность правильного решения задачи

void Initialization(const char* inFileName,int **&ts,double
**&GistRaspSig,double *&E);
void Solution(int **&ts,double **&GistRaspSig,double *&E,const char
*rezFileName);
void GenGistRasp_fromSig(int nn,double GistRasp[],int N,int sig[]);
void SumRasp(int nn,double Rasp1[],double Rasp2[],double Rasp12[]);
int BayesLimit(int nn,double GistRasp[],double GistRaspSig[]);
void VerUpPorog(int nn,int u,double RaspNoise[],double RaspSignal[],double
&pu,double &pl,double &pp);
void CalcP3chanal(int K,double pk[],int &K3,double **p3);
void CalcP4chanal(int K,double pk[],int &K4,double **p4);
void CalcP5chanal(int K,double pk[],int &K5,double **p5);
int fact(int n);
double power(double p,int n);
int Combination(int n,int k);
double CalcTotalP(double p,int Nerr,int M);
int Control_1_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]);
int Control_3_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]);
int Control_4_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]);
int Control_5_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]);
void AnalizSolution(FILE*output,int NControl,int NumberChanalControl[],int
PorogControl[],double PControl,double PtotalControl,double EControl);
int main(int argc, char** argv){
```

```
	if (argc < 3){
		printf("Run: control.exe input.dat output.dat\n");
		return -1;
	}
	const char* inFileName = argv[1]; //Входнойфайл
	const char* rezFileName = argv[2]; //Выходной файл, результаты анализа
	int **ts;//Реализация шумового сигнала для каждого канала
	double *E;//Требуемая энергия для приема одного отсчета в каждом канале
	double **GistRaspSig;//Гистограмма распределения полезного сигнала
	Initialization(inFileName,ts,GistRaspSig,E);//Чтениеданныхизфайла
inFileName
	Solution(ts,GistRaspSig,E,rezFileName);
	free(E);
	for(int i=0;i<K;i++){
		free(ts[i]);
		free(GistRaspSig[i]);
	}
	free(ts);
	free(GistRaspSig);
	return 0;
}
void Solution(int **&ts,double **&GistRaspSig,double *&E,const char
*rezFileName){
	FILE*output;
	double **GistRasp;//Гистограммараспределенияшума
	double **Rasp12;//Распределениесигнал + шум
	int *u= (int*)calloc(K,sizeof(int));//Порог
	double *pu = (double*)calloc(K,sizeof(double));//Вероятность правильного
выделения сигнала в смеси сигнал+шум
	double *pl = (double*)calloc(K,sizeof(double));//Вероятность ложных тревог
в отсутствие сигнала
	double *pp = (double*)calloc(K,sizeof(double));//Вероятность пропустить
сигнал в смеси сигнал+шум
	double *Pu = (double*)calloc(K,sizeof(double));//Вероятность правильного
выделения сигнала в ситуации, когда не известно есть сигнал или нет (в общем
случае)
	double *Pl = (double*)calloc(K,sizeof(double));//Вероятность ложных тревог
в отсутствие сигнала (в общем случае)
	GistRasp=(double**)calloc(K,sizeof(double*));
	Rasp12 = (double**)calloc(K,sizeof(double*));
	for(int i=0;i<K;i++){//Циклпоканалам
		GistRasp[i]=(double*)calloc(n2,sizeof(double));
	//Гистограммараспределенияшума
		Rasp12[i]  =(double*)calloc(n2,sizeof(double));
//Гистограммараспределениясигнал+шум
	}
//Блок подготовки к расчетам: определение порога и вероятности верного
определения единицы информации (начало)
	for(int i=0;i<K;i++){//Цикл по каналам
		GenGistRasp_fromSig(n2,GistRasp[i],N,ts[i]);//Строим гистограмму для
шума по заданной реализации
		SumRasp(n2,GistRasp[i],GistRaspSig[i],Rasp12[i]);//Распределение
суммы сигнала и шума
		u[i] = BayesLimit(n2,GistRasp[i],Rasp12[i]);//Уровень сигнала, при
котором суммарная вероятность ложных тревог и пропуска сигнала минимальна
	VerUpPorog(n2,u[i],GistRasp[i],GistRaspSig[i],pu[i],pl[i],pp[i]);//pu -
вероятность верного определения т.е. уровень сигнала больше порога - u
		Pu[i]=pu[i]*Ps/(pu[i]*Ps + pl[i]*Pn);//Вероятность верного
определения с учетом вероятности появления сигнала Ps
		Pl[i]=pl[i]*Pn/(pu[i]*Ps + pl[i]*Pn);//Вероятность ложных тревог с
учетом вероятности отсутствия сигнала Pn
	}
//Блок подготовки к расчетам: определение порога и вероятности верного
определения единицы информации (конец)
```

```
//Блокпроверкиканалов (начало)
      int (*ControlChanal[])(double [], double&, double&, double [], int [], int
[], char []) =
      {
            Control_1_Chanal, //Проверяем может ли с задачей справиться один
канал
            Control_3_Chanal, //Поиск группы из 3-х каналов, удовлетворяющих
условию задачи (если в каких либо двух из трех каналов порог превышен, то
считаем что сигнал есть)
            Control_4_Chanal, //Поиск группы из 4-х каналов, удовлетворяющих
условию задачи (если в каких либо двух из четырех каналов порог превышен, то
считаем что сигнал есть)
            Control_5_Chanal  //Проверка того поможет ли использование всех 5
каналов решить задачу
      };
//Результаты анализа сохраняем в файле rezFileName
      output=fopen(rezFileName,"w");
      char str[1000];
      int NC[10]={0};
      double PTotal,P1,Emin=0.0;
      int CC[10]={1,3,4,5};
      int i,nn=sizeof(ControlChanal)/sizeof(ControlChanal[0]);
      for(i=0;i<nn;i++)
            if(ControlChanal[i](Pu,P1,PTotal,E,u,NC,str)>0){
                  printf("the problem is solved!\n");
                  for(int j=0;j<CC[i];j++)
                        Emin+=E[NC[j]];
                  AnalizSolution(output,CC[i],NC,u,P1,PTotal,Emin);
                  break;
            }
      if(i==nn){
            printf("problem can not be solved\n");
            fprintf(output,"problem can not be solved\n");
      }
      fclose(output);
}

void AnalizSolution(FILE*output,int NControl,int NumberChanalControl[],int
PorogControl[],double PControl,double PtotalControl,double EControl){
      fprintf(output,"%d\n",NControl);
      for(int i=0;i<NControl;i++)
            fprintf(output,"%d ",NumberChanalControl[i]);
      fprintf(output,"\n");
      for(int i=0;i<NControl;i++)
            fprintf(output,"%d ",PorogControl[NumberChanalControl[i]]);
      fprintf(output,"\n");
      fprintf(output,"%lf\n",PControl);
      fprintf(output,"%lf\n",PtotalControl);
      fprintf(output,"%lf\n",EControl);
}
void GenGistRasp_fromSig(int nn,double GistRasp[],int N,int sig[]){
      for(int i=0;i<nn;i++)
            GistRasp[i]=0.0;
      for(int i=0;i<N;i++)
            GistRasp[sig[i]]++;
      for(int i=0;i<nn;i++)
            GistRasp[i]/=(double)N;
}
void SumRasp(int nn,double Rasp1[],double Rasp2[],double Rasp12[]){
      for(int i=0;i<nn;i++)
            for(int j=0;j<nn;j++)
                  if(i+j<nn)
                        Rasp12[i+j]=0.0;
      for(int i=0;i<nn;i++){
```

```
            for(int j=0;j<nn;j++){
                  if(i+j<nn)
                        Rasp12[i+j]+=Rasp1[i]*Rasp2[j];
                  else
                        Rasp12[nn-1]+=Rasp1[i]*Rasp2[j];
            }
      }
}
int BayesLimit(int nn,double GistRasp[],double GistRaspSig[]){
      int u,umin=0;
      double summin=10.0,sum;
      for(u=0;u<nn;u++){
            sum=0.0;
            for(int i=0;i<nn;i++)
                  sum+= (i<u? GistRaspSig[i] : GistRasp[i]);
            if(summin>sum){
                  summin=sum;
                  umin = u;
            }
      }
      return umin;
}
void VerUpPorog(int nn,int u,double RaspNoise[],double RaspSignal[],double
&p,double &pl,double &pp){
      p=0.0; //Вероятность того, что нами действительно зарегистрирован сигнал
      for(int i=0;i<nn;i++){
            for(int j=0;j<nn;j++){
                  if(i+j>=u)
                        p+=RaspNoise[i]*RaspSignal[j];
            }
      }
            pp=0.0;//Вероятность пропустить сигнал, т.е. вероятность того что
суммарный сигнал окажется ниже порога
      for(int i=0;i<nn;i++){
            for(int j=0;j<nn;j++){
                  if(i+j<u)
                        pp+=RaspNoise[i]*RaspSignal[j];
            }
      }
      pl=0.0;//Вероятность ложных тревог
      for(int i=u;i<nn;i++)
            pl+=RaspNoise[i];
}
int Control_1_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]){
      int i,j=0,I;
      double Ptotal,Emin;
      char ch[50];
      for(i=0;i<K;i++){
            Ptotal=CalcTotalP(Pu[i],merr,M);
            if(Ptotal>PT && (j==0 || E[i]<Emin)){
                  PTotal=Ptotal;
                  Emin=E[i];//Минимумэнергии
                  I=i;
                  j++;
            }
      }
      if(j>0){
            NC[0]=I;
            P1=Pu[I];
            char temp[200];
            sprintf(rez,"Для приема собщения можно воспользоваться каналом: %d
\n",I);
            sprintf(temp,"Пороговыйуровень: %d \n",u[I]); strcat(rez,temp);
```

```
            sprintf(temp,"Вероятность того что верно будет обнаружен единичный
сигнал: %lf \n",Pu[I]); strcat(rez,temp);
            sprintf(temp,"Вероятность того что сообщение будет передано с не
более чем %d ошибками: %lf \n",merr,PTotal); strcat(rez,temp);
            sprintf(temp,"Необходимая энергия для передачи единицы сообщения:
%lf \n",E[I]); strcat(rez,temp);
      }
      return j;
}
int Control_3_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]){
      int I,i,j,K3;
      const int K32=K*K;
      int *nk = (int*)calloc(K32,sizeof(int));
      double Ptotal,Emin;
      double **p3 = (double **)calloc(4,sizeof(double *)); //Вероятность верного
определения единицы информации при совместном анализе каналов
      for(int i=0;i<4;i++)
            p3[i]=(double *)calloc(K32,sizeof(double));
      CalcP3chanal(K,Pu,K3,p3);
      j=0;
      for(i=0;i<K3;i++){
            Ptotal=CalcTotalP(p3[3][i],merr,M);
            if(Ptotal>PT && (j==0 || E[(int)p3[0][i]] + E[(int)p3[1][i]] +
E[(int)p3[2][i]] < Emin)){
                  PTotal=Ptotal;
                  Emin=0;
                  for(int l=0;l<3;l++){
                        nk[l]=(int)p3[l][i];
                        Emin+=E[nk[l]];
                  }
                  I=i;
                  j++;
            }
      }

      if(j>0){
            NC[0]=nk[0]; NC[1]=nk[1]; NC[2]=nk[2];
            P1=p3[3][I];
            char temp[200];
            sprintf(rez,"Для приема собщения можно воспользоваться каналами: %d
%d %d \n",nk[0],nk[1],nk[2]);
            sprintf(temp,"Соответствующие пороговые уровни: %d %d %d
\n",u[nk[0]],u[nk[1]],u[nk[2]]); strcat(rez,temp);
            sprintf(temp,"Вероятность того, что верно будет обнаружен единичный
сигнал: %lf \n",p3[3][I]); strcat(rez,temp);
            sprintf(temp,"Вероятность того, что сообщение будет передано с не
более чем %d ошибками: %lf \n",merr,PTotal); strcat(rez,temp);
            sprintf(temp,"Необходимая энергия для приема единицы сообщения: %lf
\n",Emin); strcat(rez,temp);
      }
      for(int i=0;i<4;i++)
            free(p3[i]);
      free(p3);
      return j;
}
int Control_4_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]){
      int I,i,j,K4;
      const int K42=K*K;
      int *nk = (int*)calloc(K42,sizeof(int));
      double Ptotal,Emin;
      double **p4 = (double **)calloc(5,sizeof(double *)); //Вероятность верного
определения единицы информации при совместном анализе каналов
```

```
	for(int i=0;i<5;i++)
		p4[i]=(double *)calloc(K42,sizeof(double));
	CalcP4chanal(K,Pu,K4,p4);
	j=0;
	for(i=0;i<K4;i++){
		Ptotal=CalcTotalP(p4[4][i],merr,M);
		if(Ptotal>PT && (j==0 || E[(int)p4[0][i]] + E[(int)p4[1][i]] +
E[(int)p4[2][i]] + E[(int)p4[3][i]] < Emin)){
			PTotal=Ptotal;
			Emin=0;
			for(int l=0;l<4;l++){
				nk[l]=(int)p4[l][i];
				Emin+=E[nk[l]];
			}
			I=i;
			j++;
		}
	}
	if(j>0){
		NC[0]=nk[0]; NC[1]=nk[1]; NC[2]=nk[2];  NC[3]=nk[3];
		P1=p4[4][I];
		char temp[200];
		sprintf(rez,"Для приема собщения можно воспользоваться каналами: %d
%d %d %d \n",nk[0],nk[1],nk[2],nk[3]);
		sprintf(temp,"Соответствующие пороговые уровени: %d %d %d
%d\n",u[nk[0]],u[nk[1]],u[nk[2]],u[nk[3]]); strcat(rez,temp);
		sprintf(temp,"Вероятность того что верно будет обнаружен единичный
сигнал: %lf \n",p4[4][I]); strcat(rez,temp);
		sprintf(temp,"Вероятность того что сообщение будет передано с не
более чем %d ошибками: %lf \n",merr,PTotal); strcat(rez,temp);
		sprintf(temp,"Необходимая энергия для приема единицы сообщения: %lf
\n",Emin); strcat(rez,temp);
	}
	for(int i=0;i<5;i++)
		free(p4[i]);
	free(p4);
	free(nk);
	return j;
}
int Control_5_Chanal(double Pu[],double &P1,double &PTotal,double E[],int
u[],int NC[],char rez[]){
	int I,i,j,K5;
	const int K52=K*K;
	int *nk = (int*)calloc(K52,sizeof(int));
	double Ptotal,Emin;
	double **p5 = (double **)calloc(6,sizeof(double *)); //Вероятность верного
определения единицы информации при совместном анализе каналов
	for(int i=0;i<6;i++)
		p5[i]=(double *)calloc(K52,sizeof(double));
	CalcP5chanal(K,Pu,K5,p5);
	j=0;
	for(i=0;i<K5;i++){
		Ptotal=CalcTotalP(p5[5][i],merr,M);
		if(Ptotal>PT && (j==0 || E[(int)p5[0][i]] + E[(int)p5[1][i]] +
E[(int)p5[2][i]] + E[(int)p5[3][i]] + E[(int)p5[4][i]] < Emin)){
			PTotal=Ptotal;
			Emin=0;
			for(int l=0;l<5;l++){
				nk[l]=(int)p5[l][i];
				Emin+=E[nk[l]];
			}
			I=i;
			j++;
		}
```

```
	}
	if(j>0){
		NC[0]=nk[0]; NC[1]=nk[1]; NC[2]=nk[2]; NC[3]=nk[3]; NC[4]=nk[4];
		P1=p5[5][I];
		char temp[200];
		sprintf(rez,"Для приема собщения можно воспользоваться каналами: %d
%d %d %d %d\n",nk[0],nk[1],nk[2],nk[3],nk[4]);
		sprintf(temp,"Соответствующие пороговые уровни: %d %d %d %d
%d\n",u[nk[0]],u[nk[1]],u[nk[2]],u[nk[3]],u[nk[4]]); strcat(rez,temp);
		sprintf(temp,"Вероятность того что верно будет обнаружен единичный
сигнал: %lf \n",p5[5][I]); strcat(rez,temp);
		sprintf(temp,"Вероятность того что сообщение будет передано с не
более чем %d ошибками: %lf \n",merr,PTotal); strcat(rez,temp);
		sprintf(temp,"Необходимая энергия для приема единицы сообщения: %lf
\n",Emin); strcat(rez,temp);
	}
	for(int i=0;i<6;i++)
		free(p5[i]);
	free(p5);
	return j;
}
double CalcTotalP(double p,int Nerr,int M){
	double P=0.0;
	for(int i=0;i<=Nerr;i++){
		P+=Combination(M,i)*power(p,M-i)*power(1.0-p,i);
	}
	return P;
}
double power(double p,int n){
	if(n==0) return 1.0;
	else if(n%2==0)
			return power(p,n/2)*power(p,n/2);
		 else
			return p*power(p,n-1);
}

int fact(int n){
	if(n==0) return 1;
	else return n*fact(n-1);
}
int Combination(int n,int k){
	int x=1;
	for(int i=0;i<k;i++)
		x*=(n-i);
	x/=fact(k);
	return x;
}
void CalcP3chanal(int K,double pk[],int &K3,double **p3){
	int n=0;
	K3=K*(K-1)*(K-2)/6;
	for(int i=0;i<K;i++)
		for(int j=i+1;j<K;j++)
			for(int k=j+1;k<K;k++){
				p3[0][n]=i;
				p3[1][n]=j;
				p3[2][n]=k;
				p3[3][n]=pk[i]*pk[j]*pk[k]+pk[i]*pk[j]*(1.0-
pk[k])+pk[i]*(1.0-pk[j])*pk[k]+(1.0-pk[i])*pk[j]*pk[k];
				n++;
			}
}
void CalcP4chanal(int K,double pk[],int &K4,double **p4){
	int n=0;
	K4=K*(K-1)*(K-2)*(K-3)/24;
```

```
      for(int i=0;i<K;i++)
            for(int j=i+1;j<K;j++)
                  for(int l=j+1;l<K;l++)
                        for(int k=l+1;k<K;k++){
                              p4[0][n]=i;
                              p4[1][n]=j;
                              p4[2][n]=l;
                              p4[3][n]=k;
                              p4[4][n]=pk[i]*pk[j]*pk[l]*pk[k]+ //4
                                          pk[i]*pk[j]*pk[l]*(1.0-pk[k])+ //3
                                          pk[i]*pk[j]*(1.0-pk[l])*pk[k]+ //3
                                          pk[i]*(1.0-pk[j])*pk[l]*pk[k]+ //3
                                          (1.0-pk[i])*pk[j]*pk[l]*pk[k]+ //3
                                          pk[i]*pk[j]*(1.0-pk[l])*(1.0-pk[k])+
//2
                                          pk[i]*(1.0-pk[j])*pk[l]*(1.0-pk[k])+
//2
                                          (1.0-pk[i])*pk[j]*pk[l]*(1.0-pk[k])+
//2
                                          pk[i]*(1.0-pk[j])*(1.0-pk[l])*pk[k]+
//2
                                          (1.0-pk[i])*pk[j]*(1.0-pk[l])*pk[k]+
//2
                                          (1.0-pk[i])*(1.0-pk[j])*pk[l]*pk[k];
//2
                              n++;
                  }
//    printf("K4=%d\n",n);
}
void CalcP5chanal(int K,double pk[],int &K4,double **p5){
      int n=0;
      K4=K*(K-1)*(K-2)*(K-3)/24;
      for(int i=0;i<K;i++)
            for(int j=i+1;j<K;j++)
                  for(int l=j+1;l<K;l++)
                        for(int k=l+1;k<K;k++)
                              for(int m=k+1;m<K;m++){
                                    p5[0][n]=i;
                                    p5[1][n]=j;
                                    p5[2][n]=l;
                                    p5[3][n]=k;
                                    p5[4][n]=m;
                                    p5[5][n]=pk[i]*pk[j]*pk[l]*pk[k]*pk[m]    +
//5
                                                pk[i]*pk[j]*pk[l]*pk[k]*(1.0-
pk[m])+ //4
                                                pk[i]*pk[j]*pk[l]*(1.0-
pk[k])*pk[m]+ //4
                                                pk[i]*pk[j]*(1.0-
pk[l])*pk[k]*pk[m]+ //4
                                                pk[i]*(1.0-
pk[j])*pk[l]*pk[k]*pk[m]+ //4
                                                (1.0-
pk[i])*pk[j]*pk[l]*pk[k]*pk[m]+ //4
                                                pk[i]*pk[j]*pk[l]*(1.0-
pk[k])*(1.0-pk[m])+ //3
                                                pk[i]*pk[j]*(1.0-
pk[l])*pk[k]*(1.0-pk[m])+ //3
                                                pk[i]*(1.0-
pk[j])*pk[l]*pk[k]*(1.0-pk[m])+ //3
                                                (1.0-
pk[i])*pk[j]*pk[l]*pk[k]*(1.0-pk[m])+ //3
                                                pk[i]*pk[j]*(1.0-pk[l])*(1.0-
pk[k])*pk[m]+ //3
```

```
                                                    pk[i]*(1.0-pk[j])*pk[l]*(1.0-
pk[k])*pk[m]+ //3
                                                    (1.0-pk[i])*pk[j]*pk[l]*(1.0-
pk[k])*pk[m]+ //3
                                                    pk[i]*(1.0-pk[j])*(1.0-
pk[l])*pk[k]*pk[m]+ //3
                                                    (1.0-pk[i])*pk[j]*(1.0-
pk[l])*pk[k]*pk[m]+ //3
                                                    (1.0-pk[i])*(1.0-
pk[j])*pk[l]*pk[k]*pk[m]+ //3
                                                    (1.0-pk[i])*(1.0-pk[j])*(1.0-
pk[l])*pk[k]*pk[m]+ //2
                                                    (1.0-pk[i])*(1.0-
pk[j])*pk[l]*(1.0-pk[k])*pk[m]+ //2
                                                    (1.0-pk[i])*pk[j]*(1.0-
pk[l])*(1.0-pk[k])*pk[m]+ //2
                                                    pk[i]*(1.0-pk[j])*(1.0-
pk[l])*(1.0-pk[k])*pk[m]+ //2
                                                    (1.0-pk[i])*(1.0-
pk[j])*pk[l]*pk[k]*(1.0-pk[m])+ //2
                                                    (1.0-pk[i])*pk[j]*(1.0-
pk[l])*pk[k]*(1.0-pk[m])+ //2
                                                    pk[i]*(1.0-pk[j])*(1.0-
pk[l])*pk[k]*(1.0-pk[m])+ //2
                                                    (1.0-pk[i])*pk[j]*pk[l]*(1.0-
pk[k])*(1.0-pk[m])+ //2
                                                    pk[i]*(1.0-pk[j])*pk[l]*(1.0-
pk[k])*(1.0-pk[m])+ //2
                                                    pk[i]*pk[j]*(1.0-pk[l])*(1.0-
pk[k])*(1.0-pk[m]); //2
                                        n++;
                            }
}
void Initialization(const char* inFileName,int **&ts,double
**&GistRaspSig,double*&E){
      FILE*input;
      double *m,*cko;
      double *GistRasp;
      int **rez;
      input=fopen(inFileName,"r");
      fscanf(input,"%d",&K);
      fscanf(input,"%d",&M);
      fscanf(input,"%d",&merr);
      fscanf(input,"%lf",&Ps);
      Pn=1.0-Ps;
      E=(double*)calloc(K,sizeof(double));
      for(int i=0;i<K;i++)
            fscanf(input,"%lf ",&E[i]);
      fscanf(input,"%d",&n2);
      for(int i=0;i<n2;i++)
            fscanf(input,"%d",&i);
      GistRaspSig = (double**)calloc(K,sizeof(double*));
      for(int i=0;i<K;i++)
            GistRaspSig[i]=(double*)calloc(n2,sizeof(double));
      for(int i=0;i<K;i++)
            for(int j=0;j<n2;j++)
                  fscanf(input,"%lf",&GistRaspSig[i][j]);
      fscanf(input,"%d\n",&N);
      ts=(int**)calloc(K,sizeof(int*));
      rez=(int**)calloc(K,sizeof(int*));
      for(int i=0;i<K;i++){
            ts[i] = (int*)calloc(N,sizeof(int));
            rez[i] = (int*)calloc(N,sizeof(int));
      }
```

```
	for(int i=0;i<K;i++)
		for(int j=0;j<N;j<j++)
			fscanf(input,"%d",&ts[i][j]);//Реализация шумового сигнала
	fclose(input);
}
```

В таблице приведены номера задач второго этапа, элементы решения которых или полученное в результате решения понимание могут быть использованы для решения командных задач финала.

№ задачи 2-го этапа	**Знания и навыки, на выявление и развитие которых направлена задача**	**№ задачи на командном туре в финале, в которой применимо**
1	Нацелена на выявление и развитие навыков математического моделирования. Для решения задачи необходимы разделы информатики посвященные следующим темам: **работа с матрицами, обработка простых массивов данных, перевод графической информации в бинарный файл, работа с бинарными файлами.** Для решения задачи необходимы разделы математики посвященные следующим темам: **алгебраическая запись декартовой метрики для плоскости.**	4,5
2	Нацелена на выявление и развитие алгоритмического мышления, на понимание того, что такое ошибка передачи, когда она может быть восстановлена и когда не может. Для решения задачи необходимы разделы информатики посвященные следующим темам: **делимость чисел, алгоритмы перебора на определение делимости чисел, алгоритмы на определение однократной ошибки.** Для решения задачи необходимы разделы математики посвященные следующим темам: **теория чисел, простые числа, делимость.**	2, 5
3	Нацелена на развитие навыков математического моделирования, аппроксимации функций и решению обратных задач. Для решения задачи необходимы разделы информатики посвященные следующим темам: **способность работать с рядами данных, умение использовать преобразование Фурье, умение анализировать параметры длинного числового ряда, а также эффективно работать с алгебраической задачей.**	3, 5

	Для решения задачи необходимы разделы математики посвященные следующим темам: **работа с тригонометрическими функциями (способность записать сумму тригонометрических функций с учетом фазового сдвига), работа со спектрами функций (метод материалы трека), параметрическая задача из планиметрии (объекты движутся)**	
4	Нацелена на выявление развитие алгоритмического мышления и развитие навыков по передачи информации в каналах с существенными потерями данных. Для решения задачи необходимы разделы информатики посвященные следующим темам: **работа с массивами данных, работы с чтением/записью файлов, работа с организацией стека памяти** Для решения задачи необходимы разделы математики посвященные следующим темам: **теория чисел**	1, 2, 5
5	Нацелена на развитие навыков работы со статистикой и теорией вероятности, на понимание таких важных понятий как отношение сигнала/шум, условная вероятность, независимые и зависимые события. Требует в основном факультативных знаний, доступных школьнику. Материалы по данным разделам представлены в методических материалах к треку. Для решения задачи необходимы разделы информатики посвященные следующим темам: **методы численного корреляционного анализа** Для решения задачи необходимы разделы математики посвященные следующим темам: **теория информации, теория вероятности, теория случайного сигнала**	2, 5