Факультатив по программированию на языке C

План занятий

№	Тема	Описание
1	Введение в курс	Основы работы с Linux. Написание и компиляция простейших программ с использованием gcc. Правила написания кода.
2	Ввод данных. Библиотеки	Работа со вводом/выводом. Статические и динамические библиотеки. Разбиение программы на отдельные файлы. Make файлы
3	Хранение данных. Типы данных	Хранение процесса в памяти компьютера. Стек, куча. Типы данных. Преобразования типов.
4	Хранение данных. Указатели, ссылки Динамическая работа с памятью	Указатели, ссылки. Передача аргументов в функцию по ссылке/указателю. Динамическое выделение памяти. Работа с динамическими массивами – двумерные/одномерные.
5	Хранение данных	Структуры, объединения, битовые операции, битовые поля. Односвязные списки
6	Обработка данных	Переполнение данных. Правильное преобразование типов и пример ошибок, связанных с этим. Битовые операции – сдвиги, логические операции. Битовые поля.
7	Компиляция	Компиляция, Gdb и отладка.
8	Программирование под встраиваемые ОС	Перенос проекта под микрокомпьютер Raspberry Pi

Что мы уже прошли?

1)Основы работы с командной строкой
2)Компиляция программ
3)Разбиение программы на модули
4)Защита от многократного включения .h файлов

Что мы пройдем сегодня?

1)Make сборка
2)Компиляция
3)Чуть-чуть посмотрим ассемблер
4)Создание библиотек
5)Работа с файлами
6)Работа с git
7)Начнем наш проект…

Дерево языка

Дерево языка

Разбиение программы на модули

lib.h

```
#ifndef FILE_H
#define FILE_H
extern void summ(int, int, int *);
extern void mult(int, int, int *);
#endif
```

lib.c

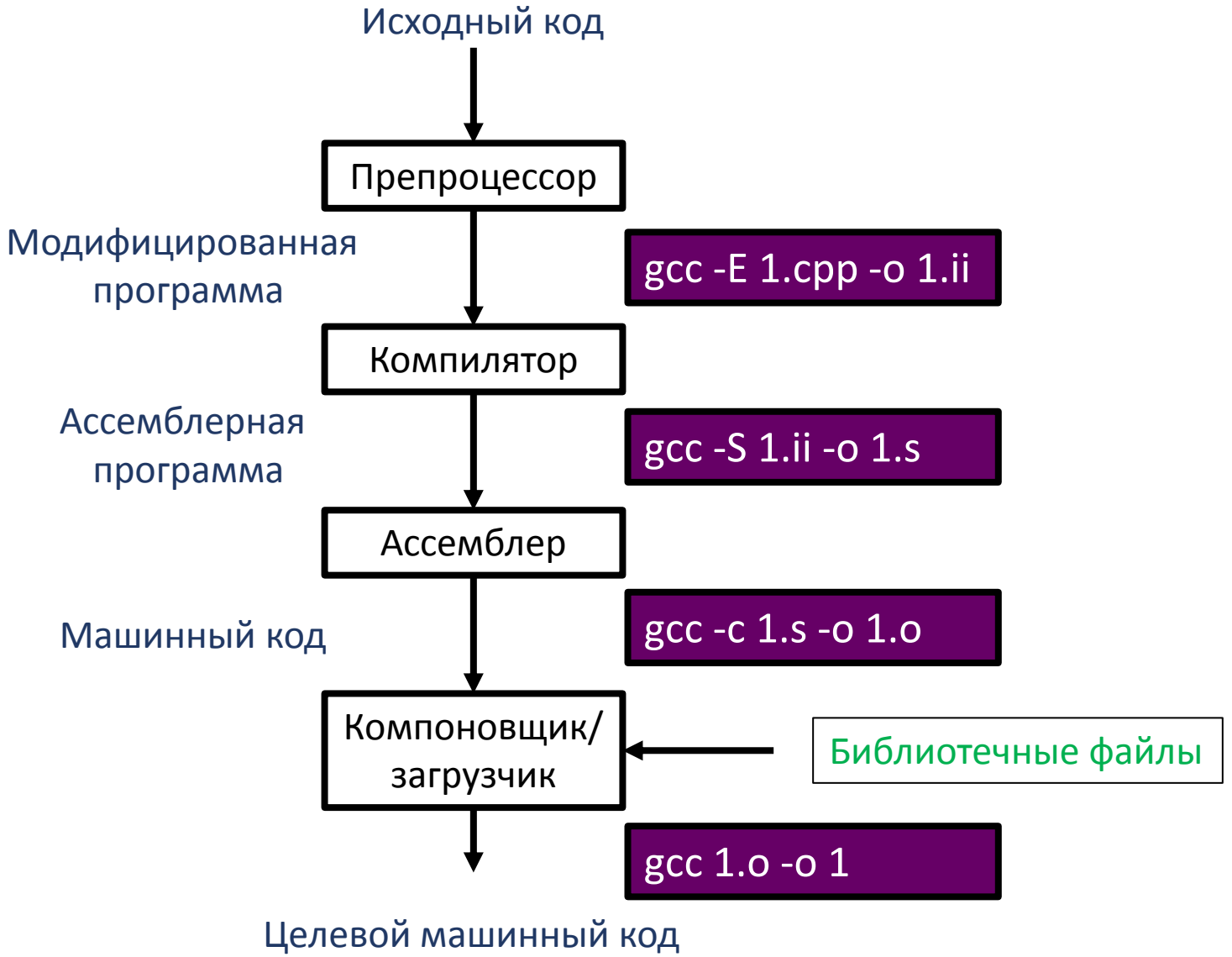
```
void summ(int x, int y, int *sum)
{
        *sum = x + y;
}

void mult(int x, int y, int *sum)
{
        *sum = x * y;
}
```

main.c

```
//#include "stdio.h"
#include "lib.h"
#define X 0
int main()
{
        int sum = X;
        summ(10,10,&sum);
        if(sum>10){
            mult(10,10,&sum);
        }
//      printf("Res = %d\n", sum);
        return 0;
}
```

```
gcc main.c –c –m32 -Werror -Wall -g -o main.o
gcc lib.c -c –m32 -Werror -Wall -g -o lib.o
gcc main.o lib.o -o main
./main
```

Этапы компиляции

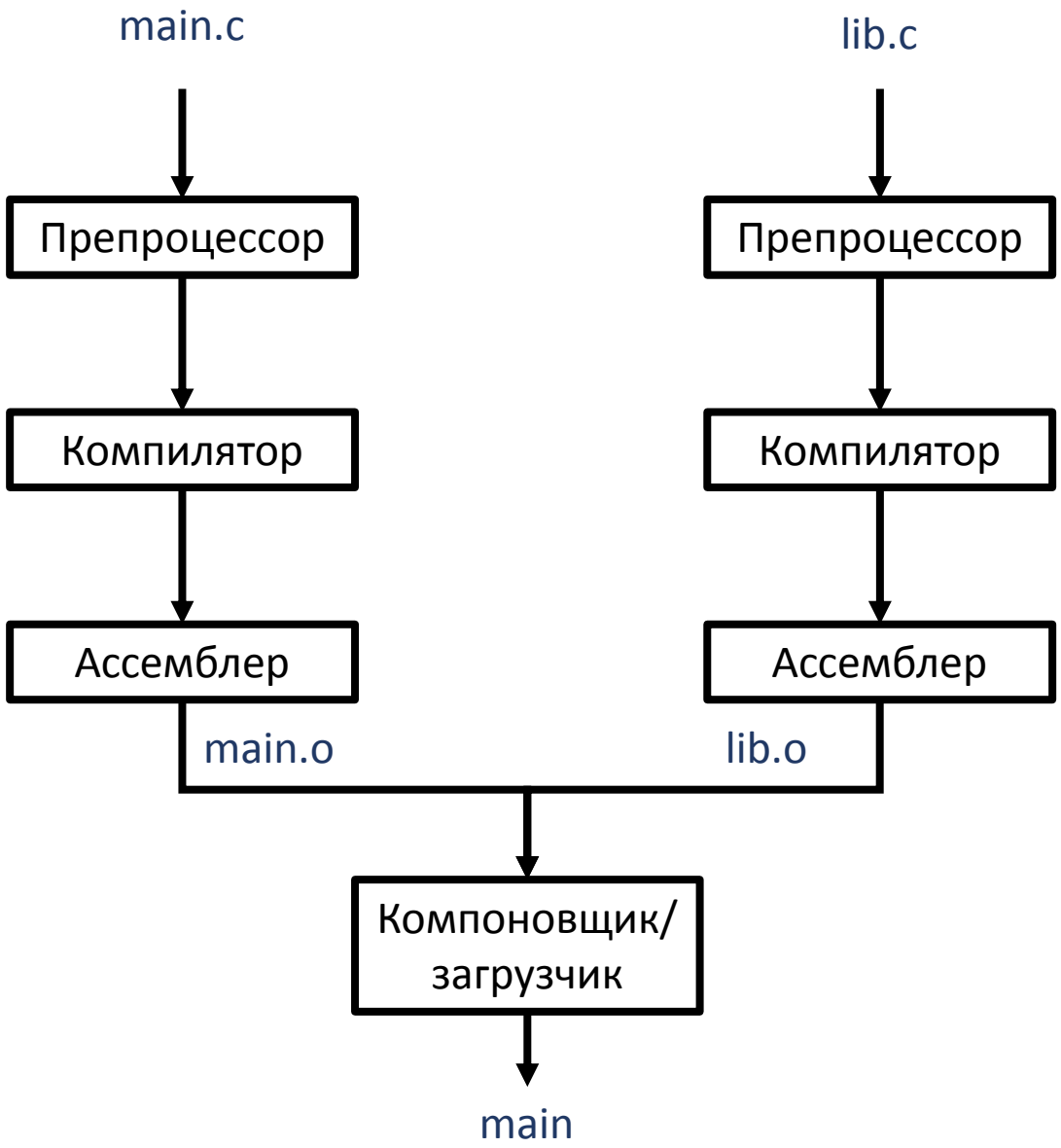


Кратко об ассемблере

Регистр	Назначение
%eax	хранение результатов промежуточных вычислений
%ebx	хранения адреса (указателя) на некоторый объект в памяти
%ecx	счетчик
%edx	хранения результатов промежуточных вычислений
%esp	содержит адрес вершины стека
%ebp	указатель базы кадра стека
%esi	индекс источника
%edi	индекс приёмника

Команда	Назначение
mov *источник, назначение*	копирование *источника* в *назначение*
lea *источник*, *назначение*	помещает адрес *источника* в *назначение*
add *источник*, приёмник	*приёмник* = *приёмник* + *источник*
sub *источник*, приёмник	*приёмник* = *приёмник* - *источник*
push *источник*	поместить в стек
pop *назначение*	извлечь из стека
cmp *операнд_2*, *операнд_1*	*операнд_1 – операнд_2* и устанавливает флаги
jle *метка*	Переход если <=

Этапы компиляции



Make файлы

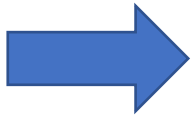
цель: зависимости
[tab] команда

makefile

```
all: clean
	gcc -c *.c
	gcc *.o -o main
clean:
	rm -f *.o
```

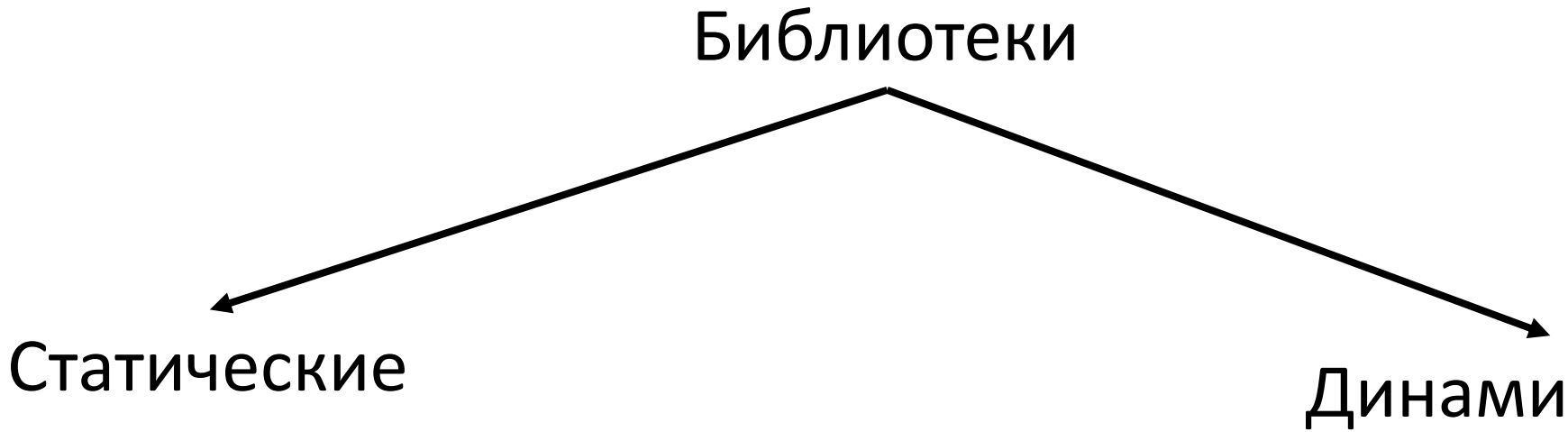
```
gcc main.c -c -o main.o
gcc lib.c -c -o lib.o
gcc main.o lib.o -o main
```

```
CFLAGS=-Wall -g -Werror -m32
all: clean
	gcc -c $(CFLAGS) *.c
	gcc *.o $(CFLAGS) -o main
clean:
	rm -f *.o
```



```
make
```

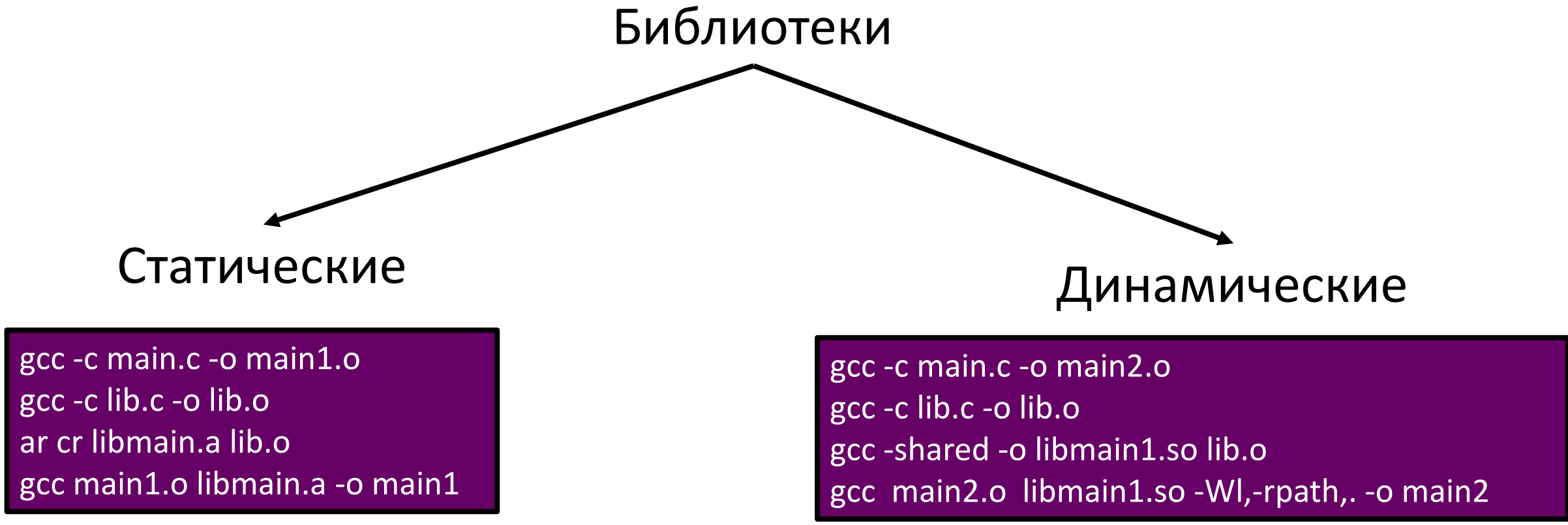
Библиотеки



Подключаются к программе во время **компоновки**.

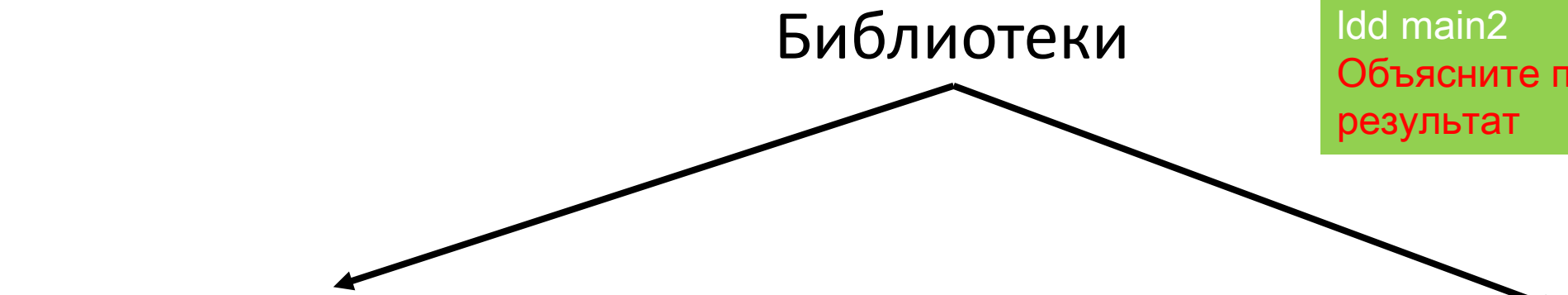
Подключаются к программе во время **выполнения программы**

Статические библиотеки



Статические библиотеки

Выполните команды
ldd main1
ldd main2
Объясните полученный результат




```
gcc -c main.c -o main1.o
gcc -c lib.c -o lib.o
ar cr libmain.a lib.o
gcc main1.o libmain.a -o main1
```

```
gcc -c main.c -o main2.o
gcc -c lib.c -o lib.o
gcc -shared -o libmain1.so lib.o
gcc  main2.o  libmain1.so -Wl,-rpath,. -o main2
```

Работа с файлами

FILE * **fopen**(const char * fname, const char * modeopen);
int **fgetc**(FILE * filestream);
int **fputc**(int character, FILE * filestream);
int **fclose**(FILE * filestream);

"r" — открыть файл для чтения (файл должен существовать);
"w" — открыть пустой файл для записи; если файл существует, то его содержимое теряется;
"a" — открыть файл для записи в конец (для добавления); файл создается, если он не существует;
"r+" — открыть файл для чтения и записи (файл должен существовать);
"w+" — открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряется;
"a+" — открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Основные определения из курса ОС

Процесс – программа во время исполнения и все её элементы: адресное пространство, глобальные переменные , регистры, стек, счетчик команд, состояние, открытые файлы, дочерние процессы и т. д

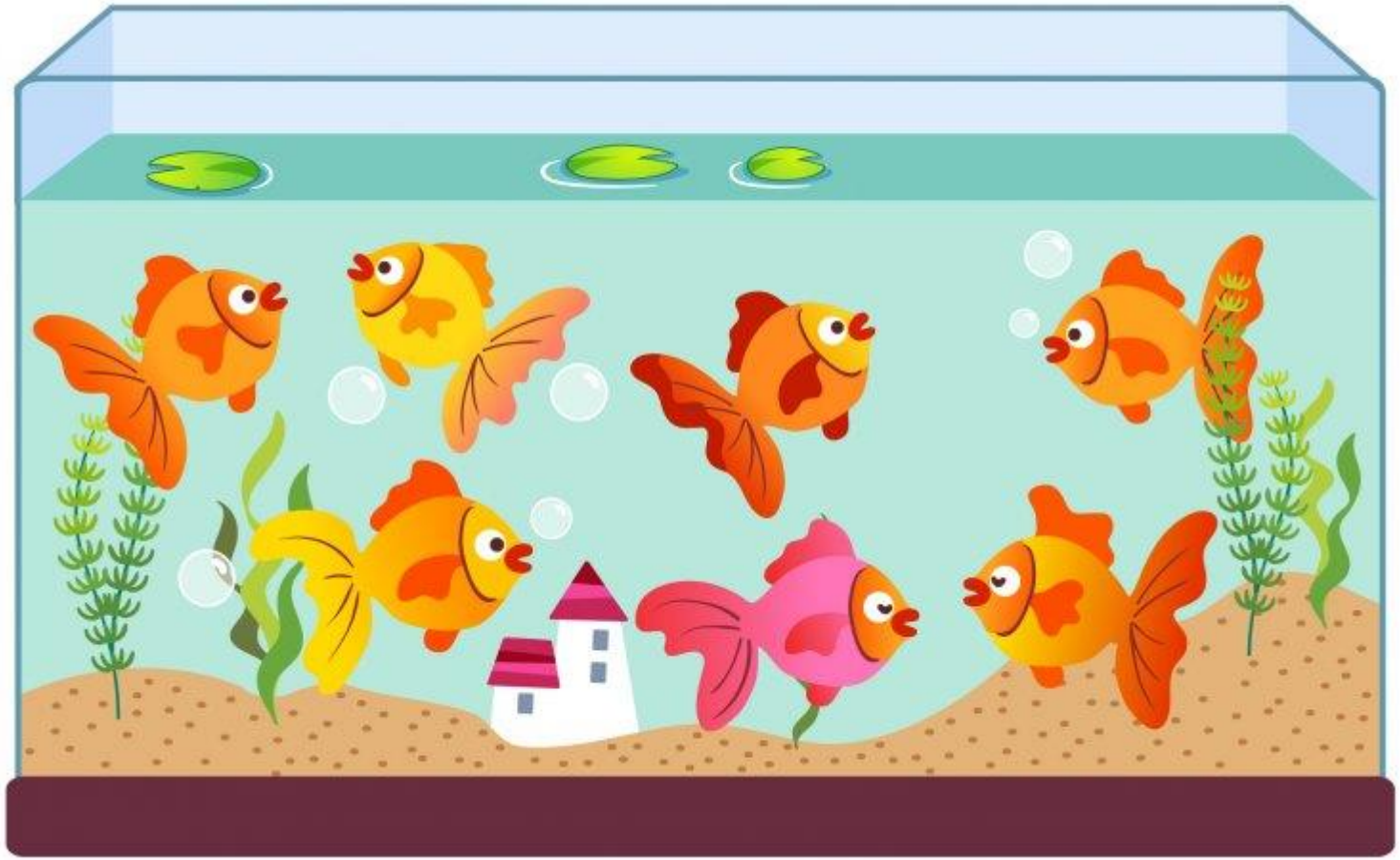
Поток - самостоятельная цепочка последовательно выполняемых операторов программы, соответствующих некоторой подзадаче

Прерывание – событие, при котором меняется последовательность команд, выполняемых процессором

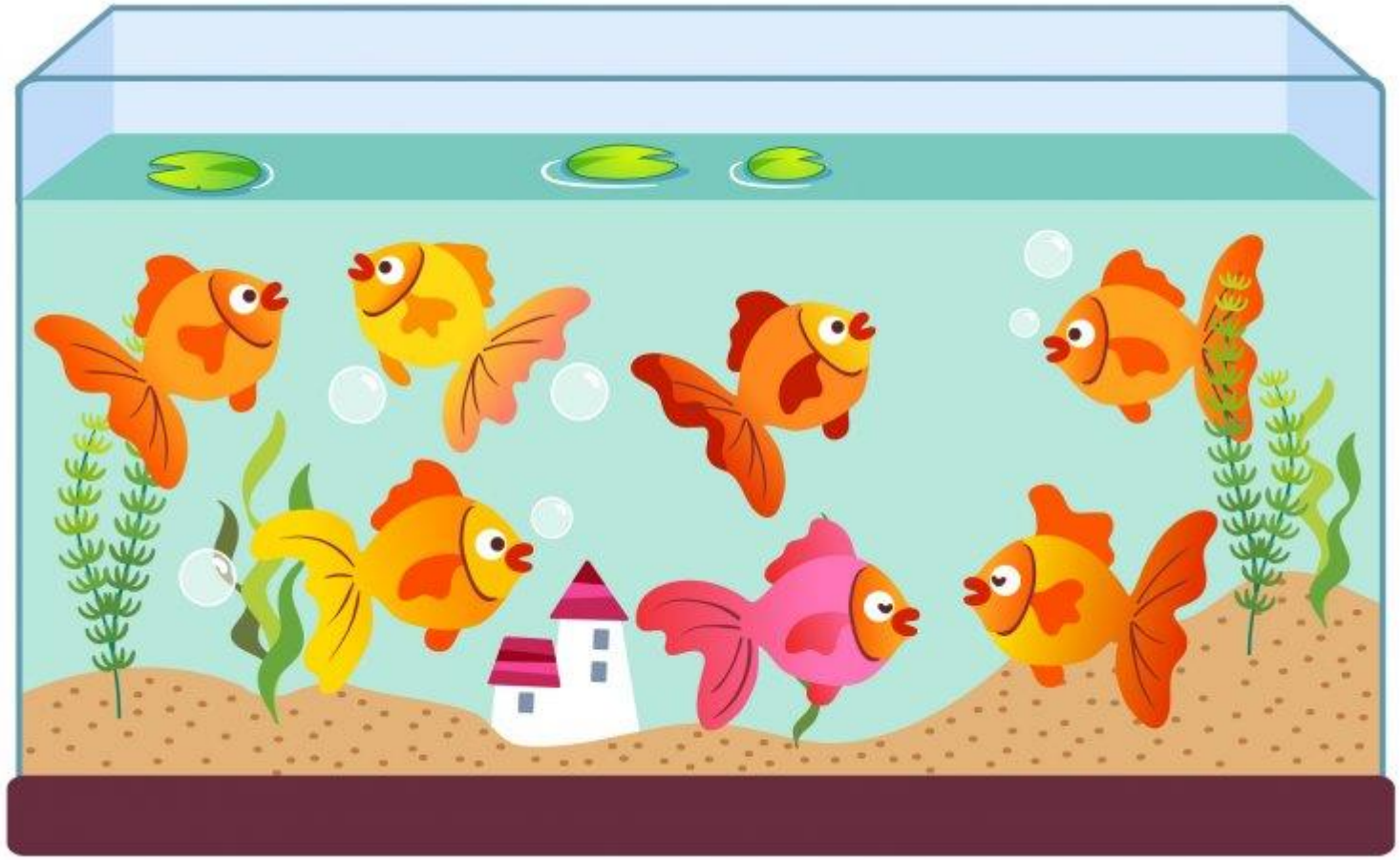
Системный вызов – это интерфейс для получения услуг операционной системы

Файл – поименованная совокупность данных

Основные определения из курса ОС



Основные определения из курса ОС



Аквариум - процесс

Рыбки - потоки

Корм - ресурсы

Стандартный ввод и вывод

```
extern FILE *stdin; /* Standard input stream. */
extern FILE *stdout; /* Standard output stream. */
extern FILE *stderr; /* Standard error output stream. */
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt, args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt, args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

```
int vsprintf(char *buf, const char *fmt, va_list args)
{
        return vsnprintf(buf, INT_MAX, fmt, args);
}
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt,
      args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

```
int vsnprintf(char *buf, size_t size, const char
*fmt, va_list args)
{
....
                        switch (spec.type) {
                        case FORMAT_TYPE_CHAR: {
                                .......
                                }
                                break;
                        }
                        case FORMAT_TYPE_STR:
.......
}
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt, args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

```
static int puts(const char *s)
 {
while (*s)
putchar(*s++);
return 0;
}
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt, args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

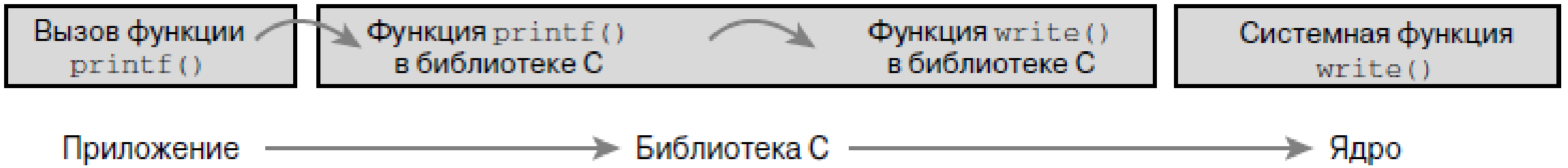
```
void __section(".inittext") putchar(int ch)
{
      if (ch == '\n’)
      putchar('\r'); /* \n -> \r\n */
       bios_putchar(ch);
       if (early_serial_base != 0)
       serial_putchar(ch);
}
```

Стандартный ввод и вывод

```
int printf(const char *fmt, ...)
 {
      char printf_buf[1024];
      va_list args; int printed;
      va_start(args, fmt);
      printed = vsprintf(printf_buf, fmt, args);
      va_end(args);
      puts(printf_buf);
      return printed;
}
```

```
static void __section(".inittext")
bios_putchar(int ch)
{
       struct biosregs ireg;
      initregs(&ireg);
      ireg.bx = 0x0007;
      ireg.cx = 0x0001;
      ireg.ah = 0x0e;
      ireg.al = ch;
      intcall(0x10, &ireg, );
 }
```

Стандартный ввод и вывод



ssize_t **write**(int fd, const void *buf, size_t nbytes);

write(fd1, buf, strlen(buf));

Перенаправление потоков ввода/вывода

write.c

```
#include <stdio.h>
int main(){
        printf("world!");
        return 0;
}
```

read.c

```
#include <stdio.h>

int main(){
        char address[100];
        printf("Hello ");
        scanf("%s", address);
        printf("%s\n", address);
        return 0;
}
```

```
./read > log.txt
./write > text.txt
./read  < text.txt
./read > log.txt < text.txt
./write | ./read
```

Git

Git - это консольная утилита, для отслеживания и ведения истории изменения файлов, в вашем проекте

С помощью Git-а вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий

Репозитории возможно хранить в интернете

Git

```
sudo apt install git
git init
git add File.txt
git commit -m "first commit“
git branch -M main
git remote add origin
https://github.com/you_repository/you_project
git push -u origin master
```

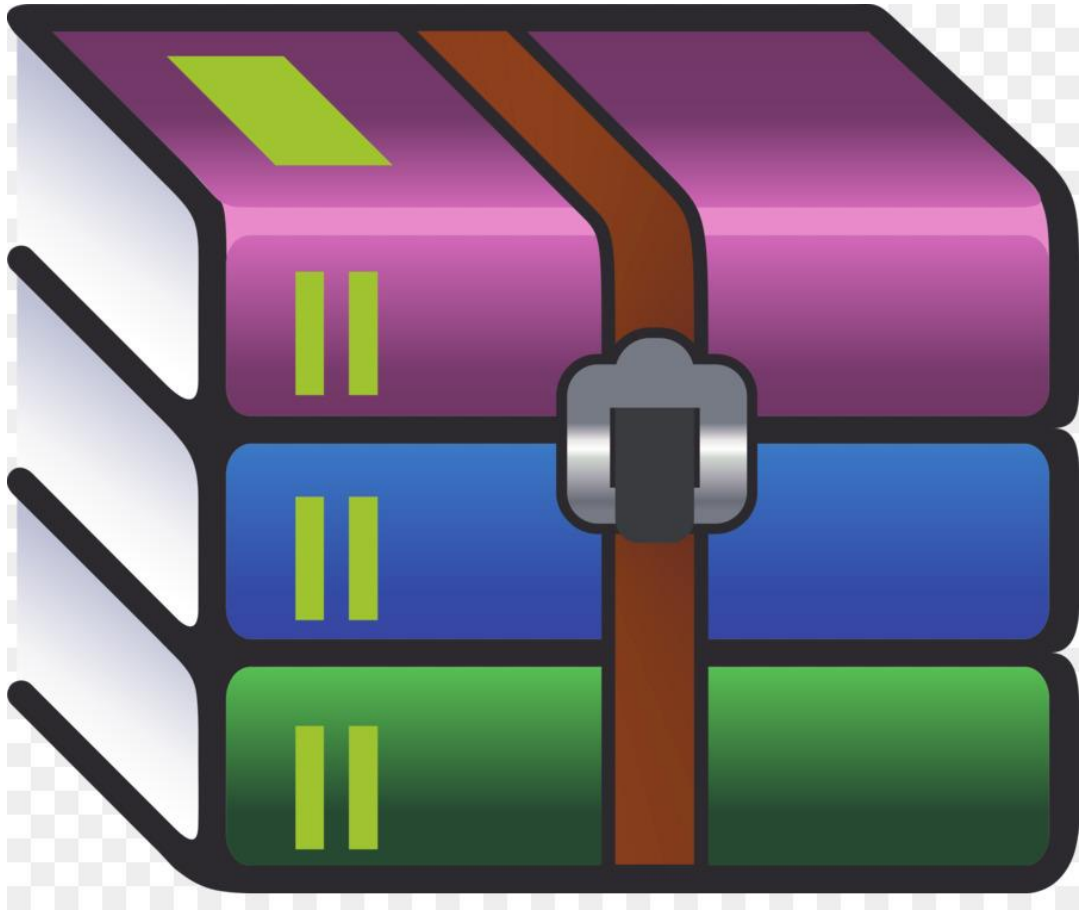
Git

Для работы
git add .
git status
git commit -m "1 commit"
git diff

Для отката
git log --oneline
git checkout 4beac58 .

Для залива на удаленный репозиторий
git remote add origin
https://github.com/you_repository/you_project
git remote -v
git push -u origin master

Практическая часть



Задание

Реализовать программу, кодирующая и сжимающая файлы по алгоритму Хаффмана

Реализовать программу для декодирования файлов

Основы теории информации

Информация (Information) — содержание сообщения или сигнала; сведения, рассматриваемые в процессе их передачи или восприятия, позволяющие расширить знания об интересующем объекте

Информация – первоначально — сведения, передаваемые одними людьми другим людям устным, письменным или каким-нибудь другим способом

Информация - как коммуникацию, связь, в процессе которой устраняется неопределенность. (К. Шеннон)

Мера информации

Пусть X – источник дискретных сообщений. Число различных состояний источника – N.
Переходы из одного состояния в другое не зависят от предыдущих состояний, а вероятности перехода в эти состояния $p_j=P\{X = x_j\}$

Тогда за меру количества информации примем следующую величину:

$$H(X) = -\sum_{k=1}^{N} p_k \log(p_k)$$

Эта величина называется **энтропией**

Энтропия

Свойства энтропии

1) Энтропия неотрицательна
2) Энтропия нескольких независимых файлов равна сумме энтропий каждого из них

Энтропия

Свойства энтропии

1) Энтропия неотрицательна
2) Энтропия нескольких независимых файлов равна сумме энтропий каждого из них

При каком условии энтропия максимальна?

Энтропия

Свойства энтропии

1) Энтропия неотрицательна
2) Энтропия нескольких независимых файлов равна сумме энтропий каждого из них
3) Максимально возможное значение энтропии равно log(N)

Энтропия

Свойства энтропии

1) Энтропия неотрицательна
2) Максимально возможное значение энтропии равно log(N)
3) Энтропия нескольких независимых файлов равна сумме энтропий каждого из них

Битовые затраты – среднее число бит приходящееся на один символ сообщения

$$R=\sum_{k=1}^{N} p_k R_k$$

R_k - число бит в коде символа x_k

Пример

Пусть нам пришло следующее сообщение: «мамамылараму» Рассчитаем энтропию сообщения и битовые затраты. Будем считать, что один символ кодируется 1 байтом.

Пример

1) Рассчитаем вероятности появления символов

мамамылараму

Символ	**м**	**а**	**ы**	**л**	**р**	**у**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**

Пример

1) Рассчитаем энтропию и битовые затраты

$$H(X) = -\sum_{k=1}^{N} p_k \log(p_k)$$

$$H(X) = -\sum_{k=1}^{6} p_k \log(p_k) = -\left(\frac{1}{3}\log\frac{1}{3} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12}\right) \sim 2{,}25$$

Пример

1) Рассчитаем энтропию

$$H(X) = -\sum_{k=1}^{N} p_k \log(p_k)$$

$$H(X) = -\sum_{k=1}^{6} p_k \log(p_k) = -\left(\frac{1}{3}\log\frac{1}{3} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12} + \frac{1}{12}\log\frac{1}{12}\right) \sim \mathbf{2,25}$$

$$R = \sum_{k=1}^{N} p_k R_k$$

$$H(X) = \sum_{k=1}^{6} p_k \log(p_k) = \left(\frac{1}{3} * 8 + \frac{1}{3} * 8 + \frac{1}{12} * 8 + \frac{1}{12} * 8 + \frac{1}{12} * 8 + \frac{1}{12} * 8\right) = \mathbf{8}$$

Идея сжатия

Давайте заменим стандартный равномерный ASCII код на неравномерный так, чтобы часто встречающимся символам соответствовали более короткие кодовые последовательности. Если средние битовые затраты будут меньше, чем 8 бит, то сжатие удалось!

Идея сжатия

Тогда произведем замену!

Символ	**м**	**а**	**ы**	**л**	**р**	**У**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**
Код	0	1	00	01	10	11	

Идея сжатия

Тогда произведем замену!

Символ	**м**	**а**	**ы**	**л**	**р**	**У**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**
Код	0	1	00	01	10	11	

Сработает ли такой вариант?

Идея сжатия

Тогда произведем замену!

Символ	**м**	**а**	**ы**	**л**	**р**	**У**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**
Код	0	1	00	01	10	11	

Наш код: 0101000011101011

ллыыурру

Сработает ли такой вариант?

Префиксные коды

Введем правило:
Ни один код не может быть началом другого.

Поэтому будем использовать **префиксные коды**

Символ	**м**	**а**	**ы**	**л**	**р**	**У**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**
Код	0	10	110	1110	11110	11111	

Префиксные коды

Введем правило:
Ни один код не может быть началом другого.

Поэтому будем использовать **префиксные коды**

Символ	**м**	**а**	**ы**	**л**	**р**	**У**	**Σ**
Количество	4	4	1	1	1	1	**12**
Вероятность	1/3	1/3	1/12	1/12	1/12	1/12	**1**
Код	0	10	110	1110	11110	11111	

Подобный вариант **избыточен**!

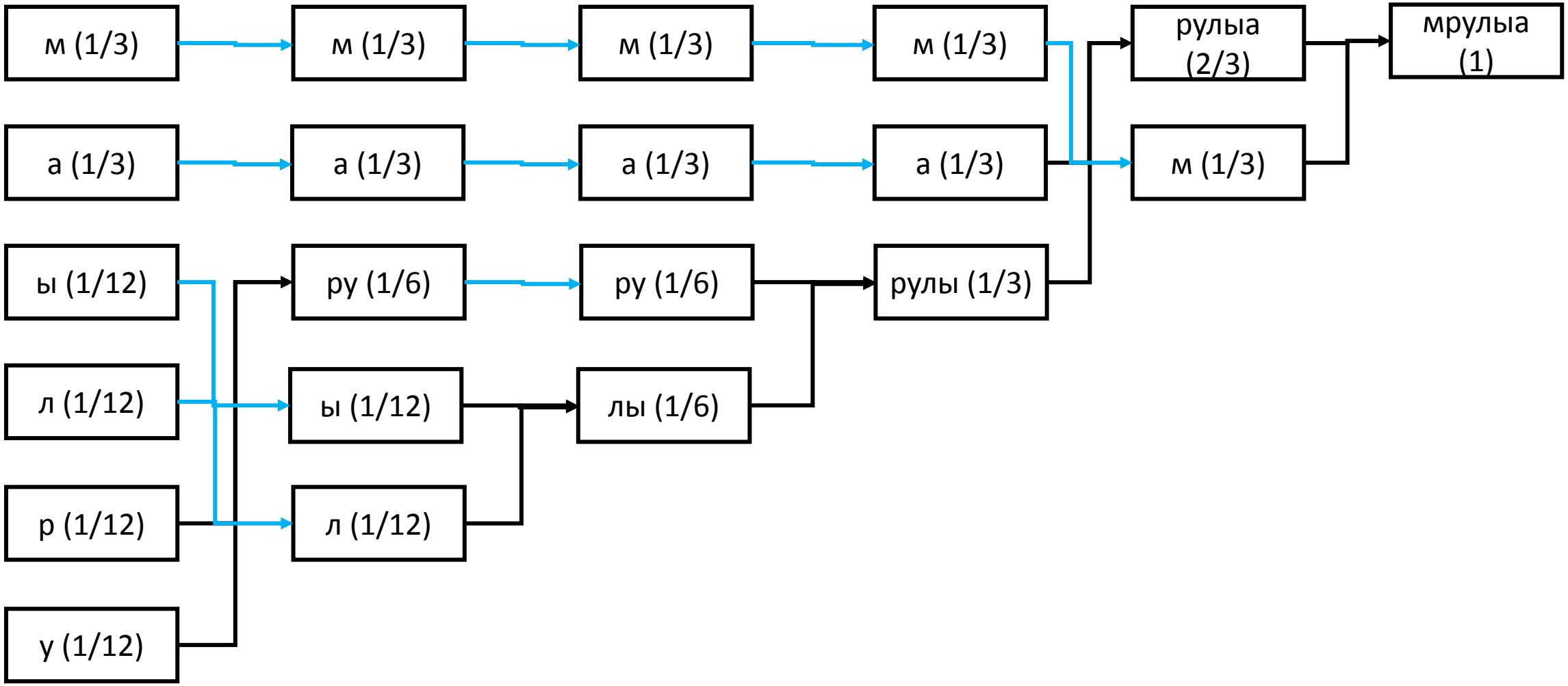
R ~ 2,42

Алгоритм Хаффмана

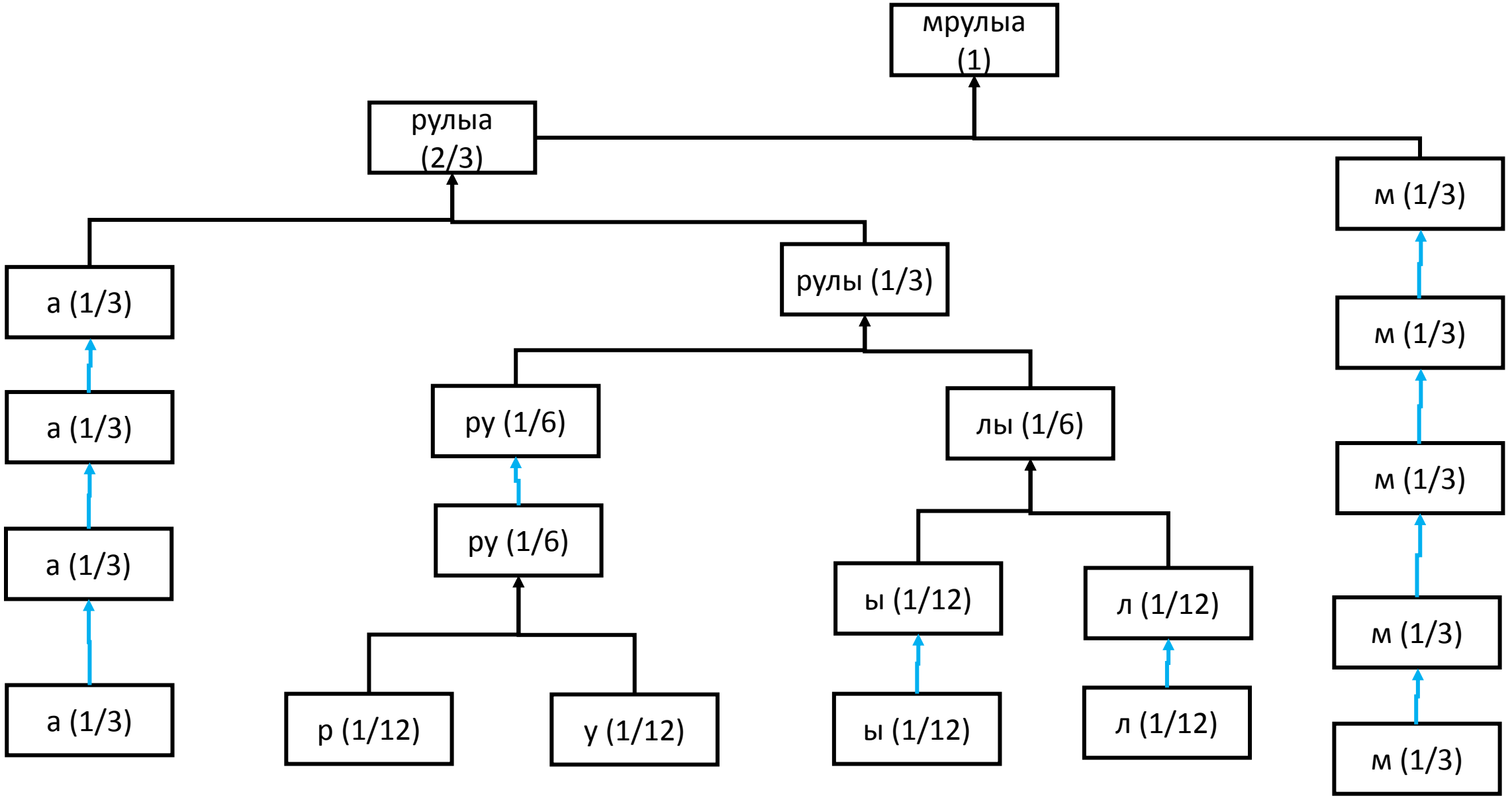
На вход алгоритма подается таблица символов

1. Построение дерева Хаффмана
1.1 Упорядочиваем таблицу символов в порядке убывания вероятностей
1.2 Два последних символа, имеющих наименьшие вероятности появления объединяются в новый символ
Если есть еще символы, то возвращаемся на 1.1
2. Построение битового кода
Для каждого узла дерева строим по два ребра, приписываем одному из них 1, другому 0

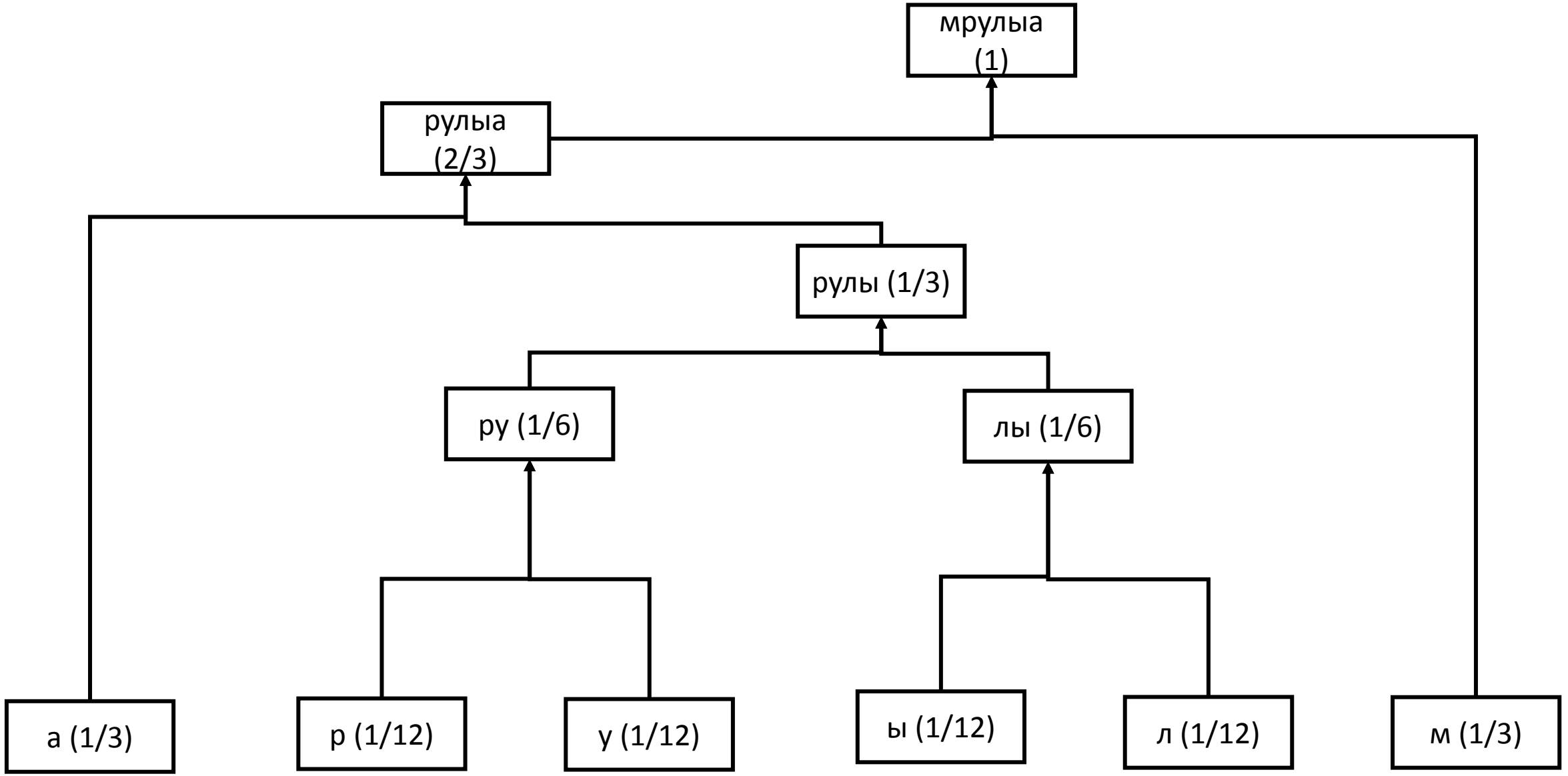
Алгоритм Хаффмана



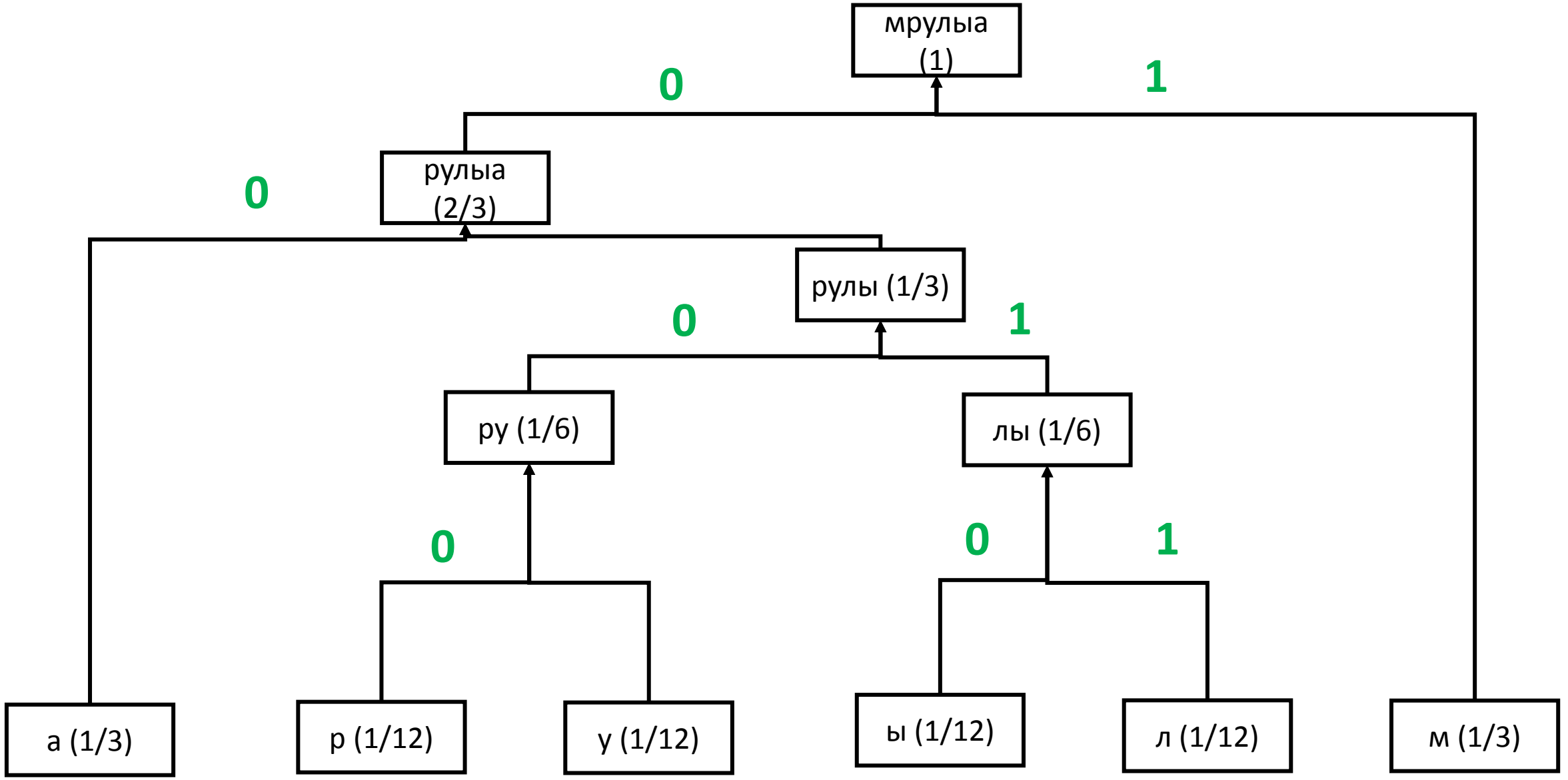
Алгоритм Хаффмана



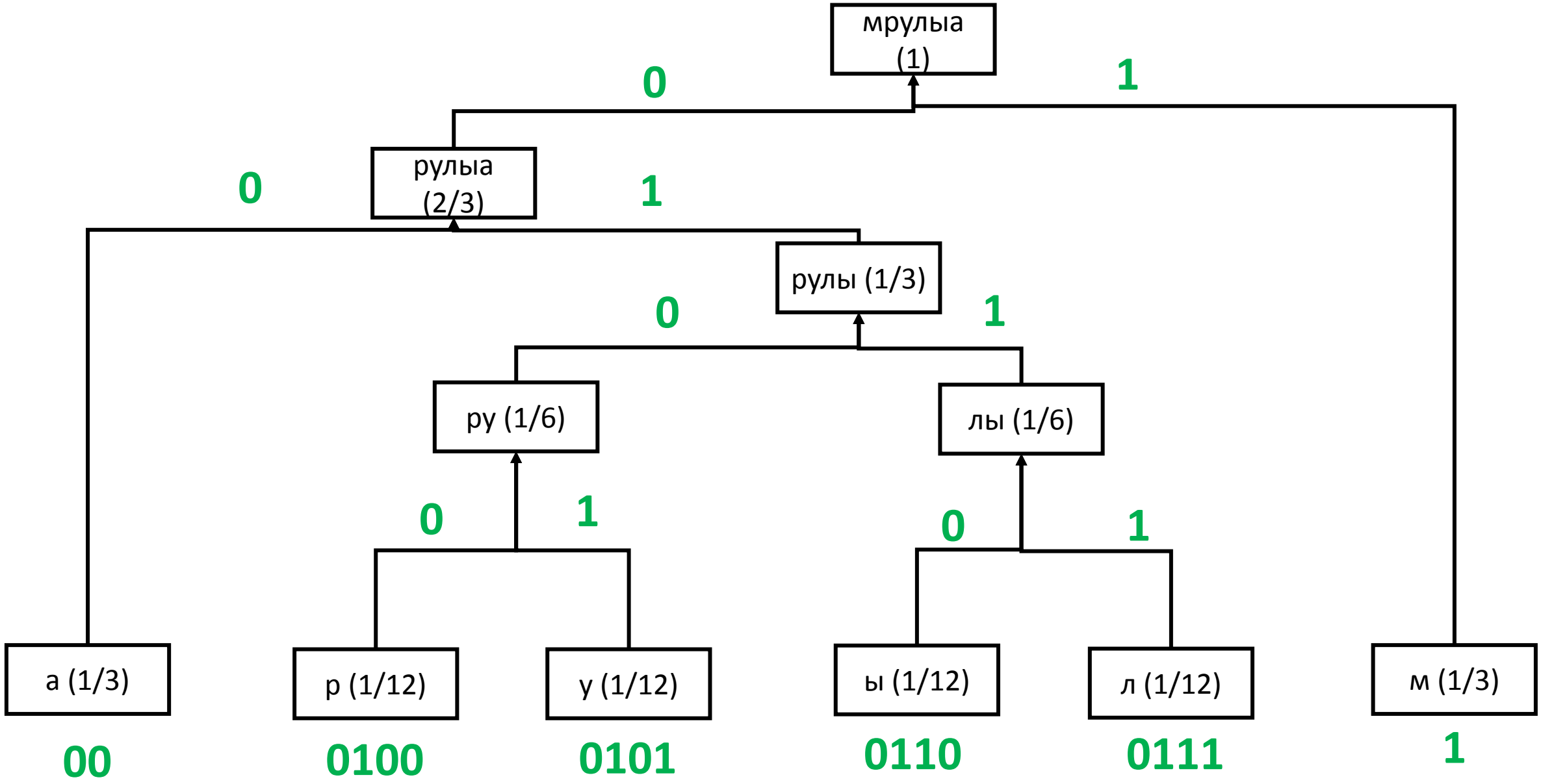
Алгоритм Хаффмана



Алгоритм Хаффмана



Алгоритм Хаффмана



Расчет битовых затрат

Без кодирования	R = 8
Кодирование «в лоб»	R ~ 2,42
Метод Хаффмана	R ~ 2,33
Энтропия	H ~ 2,25

Задача на сегодня

На основе примера работы с текстовыми файлами написать программу, выполняющую следующие действия:

1) Открытие файла в бинарном виде и посимвольное чтение потока байтов
2) Расчет гистограммы появлений символов
3) Функция расчета энтропии по полученной в п. 2 гистограмме

```
git clone https://github.com/SergeyBalabaev/Elective
```

Задача на сегодня

- В программе должен быть выделен отдельный модуль для работы с файлом
- Функция расчета энтропии также должна располагаться в отдельном модуле. По желанию сделайте его динамической библиотекой
- Для компиляции воспользуйтесь make

- Каждый символ должен храниться в структуре со следующим интерфейсом:

```
struct symbol
{
    unsigned char ch;
    float freq;
};
```

Спасибо за внимание!