

Условия задач заключительного этапа

Задача 1. Парольная комбинация

Для входа в систему используется пароль, состоящий из трёх шестизначных чисел, расположенных следующим образом:

xxxxxx-xxxxxx-xxxxxx

Известно, что пароль состоит из 3-х *неповторяющихся простых чисел* в диапазоне от 100 000 до 100 300. При этом *последняя цифра первого числа равна первой цифре второго числа, а последняя цифра второго числа равна первой цифре третьего числа.*

Пример:

xxxxx3-3xxxx7-7xxxxx

Задержка между попытками входа в систему равна 1 секунде. За какое максимальное количество времени (в секундах) можно гарантированно получить пароль, если на ввод пароля время не тратится?

Решение

Для начала необходимо найти все простые числа в диапазоне от 100 000 до 100 300. *Простым* является такое число, которое делится только на себя и на 1. Сделать это можно перебором, проверяя, не делится ли проверяемое число на какое-либо от 2-х до самого числа (либо половины числа).

Для ускорения напишем программу, которая выводит на экран все простые числа в указанном диапазоне.

Листинг программы на языке Python.

```
# Функция, определяющая, является ли число простым
# ПАРАМЕТР:
# n - проверяемое число
# ВОЗВРАЩАЕТ:
# True - число n простое
# False - число n непростое
def isprime(n):
    if n == 1:
        return True
    for d in range(2, n//2):
        if n % d == 0:
            return False
    return True

# Цикл перебора чисел от 100 000 до 100 300
# Для каждого числа вызывается функция isprime()
# Если функция вернула True - число добавляется
# в массив list_input
# Счетчик Count считает количество простых чисел
Count = 0
list_input = []
for x in range(100000,100300):
```

```

    if isprime(x) == True:
        list_input.append(x)
        Count += 1
print(list_input)
print("Total:", Count)

```

Листинг программы на языке С.

```

// Функция, определяющая, является ли число простым
// ПАРАМЕТР:
// n - проверяемое число
// ВОЗВРАЩАЕТ:
// true - число n простое
// false - число n непростое
bool isprime(int n)
{
    if (n == 1)
        return true;
    for (int d = 2; d < n / 2; d++)
        if (n % d == 0)
            return false;
    return true;
}

int main()
{
    // Цикл перебора чисел от 100000 до 100300
    // Для каждого числа вызывается функция isprime()
    // Если функция вернула true - число добавляется
    // в массив
    // Счетчик Count считает количество простых чисел
    int Count = 0;
    int list_input[100] = { 0 };
    // Формирование массива простых чисел
    for (int x = 100000; x < 100300; x++)
    {
        if (isprime(x) == true)
        {
            list_input[Count] = x;
            Count += 1;
        }
    }
    // Вывод на экран
    for (int i = 0; i < Count; i++)
    {
        printf("%d ", list_input[i]);
    }
    printf("\nTotal: %d\n", Count);
    return 0;
}

```

Результат работы программы:

```

100003 100019 100043 100049 100057 100069 100103 100109 100129
100151 100153 100169 100183 100189 100193 100207 100213 100237
100267 100271 100279 100291 100297

```

Total: 23

Всего 23 простых чисел в диапазоне от 11 до 99. Теперь необходимо выделить все возможные комбинации, удовлетворяющие условию *«последняя цифра первого числа равна первой цифре второго числа, а последняя цифра второго числа равна первой цифре третьего числа»*.

Сделать это можно двумя способами.

Способ 1. Перебор.

В цикле перебираем всевозможные числа и проверяем два условия:

- все числа разные;
- последняя цифра первого числа равна первой цифре второго числа, а последняя цифра второго числа равна первой цифре третьего числа.

Если все условия выполнены, то увеличиваем значение счетчика. В результате счетчик будет содержать количество комбинаций, удовлетворяющих заданию. Поскольку задержка между вводом паролей равна 1 секунде, а после ввода последней комбинации задержки нет, то ответ – количество найденных комбинаций минус 1.

Пример программы представлен в листингах ниже.

Листинг программы на языке Python.

```
# Функция возвращает список всех комбинаций чисел,
# удовлетворяющих заданному условию паролей
# ПАРАМЕТР:
#     list_input - список чисел, из которых будет
#                 формироваться комбинация
# ВОЗВРАЩАЕТ:
#     list_combination - список комбинаций
#                 из 3-х чисел, удовлетворяющих условию
def combination(list_input):
    list_combination = []
    for a in list_input:
        for b in list_input:
            for c in list_input:
                if a != b and b != c and a != c:
                    if str(a)[-1] == str(b)[0] and
                       str(b)[-1] == str(c)[0]:
                        list_combination.append(str(a) +
                                                "-" + str(b) + "-" + str(c))
    return list_combination

# Заполнение списка всеми простыми числами
# в заданном диапазоне
list_input = []
for x in range(100000,100300):
    if isprime(x) == True:
        list_input.append(x)
# Список, содержащий все комбинации,
# удовлетворяющие заданному условию
result_list = combination(list_input)
# Вывод всех комбинаций
for i in result_list:
```

```

    print(i, end=", ")
# Вывод количества комбинаций
print("Total:", len(result_list))

```

Листинг программы на языке C.

```

// Функция выводит на экран все комбинации,
// удовлетворяющие условию
// ПАРАМЕТРЫ:
//     list_input - массив чисел, из которых будут
//                 строится комбинации
//     count - количество чисел в массиве list_input
// ВОЗВРАЩАЕТ:
//     выводит на экран комбинации, удовлетворяющие
//     условию
//     выводит на экран количество комбинаций
void combination(int list_input[], int count)
{
    int last_digit1;
    int first_digit2;
    int last_digit2;
    int first_digit3;
    int total = 0;
    for (int i = 0; i < count; i++)
        for (int j = 0; j < count; j++)
            for (int k = 0; k < count; k++)
                {
                    if ( list_input[i] != list_input[j] &&
                        list_input[j] != list_input[k] &&
                        list_input[i] != list_input[k] )
                        {
                            // последняя цифра числа
                            last_digit1 = list_input[i] % 10;
                            // первая цифра числа
                            first_digit2 = list_input[j]/100000;
                            last_digit2 = list_input[j] % 10;
                            first_digit3 = list_input[k]/100000;
                            if ( last_digit1 == first_digit2 &&
                                last_digit2 == first_digit3 )
                                {
                                    printf("%d-%d-%d\n",
                                        list_input[i],
list_input[j],
                                        list_input[k]);
                                    total++;
                                }
                        }
                }
    // Вывод общего количества комбинаций
    printf("Total: %d\n", total);
}

```

В результате выполнения программы получим следующие данные:

```

100151-100271-100003, 100151-100271-100019,
100151-100271-100043, 100151-100271-100049,
100151-100271-100057, 100151-100271-100069,

```

100151-100271-100103, 100151-100271-100109,
100151-100271-100129, 100151-100271-100153,
100151-100271-100169, 100151-100271-100183,
100151-100271-100189, 100151-100271-100193,
100151-100271-100207, 100151-100271-100213,
100151-100271-100237, 100151-100271-100267,
100151-100271-100279, 100151-100271-100291,
100151-100271-100297, 100151-100291-100003,
100151-100291-100019, 100151-100291-100043,
100151-100291-100049, 100151-100291-100057,
100151-100291-100069, 100151-100291-100103,
100151-100291-100109, 100151-100291-100129,
100151-100291-100153, 100151-100291-100169,
100151-100291-100183, 100151-100291-100189,
100151-100291-100193, 100151-100291-100207,
100151-100291-100213, 100151-100291-100237,
100151-100291-100267, 100151-100291-100271,
100151-100291-100279, 100151-100291-100297,
100271-100151-100003, 100271-100151-100019,
100271-100151-100043, 100271-100151-100049,
100271-100151-100057, 100271-100151-100069,
100271-100151-100103, 100271-100151-100109,
100271-100151-100129, 100271-100151-100153,
100271-100151-100169, 100271-100151-100183,
100271-100151-100189, 100271-100151-100193,
100271-100151-100207, 100271-100151-100213,
100271-100151-100237, 100271-100151-100267,
100271-100151-100279, 100271-100151-100291,
100271-100151-100297, 100271-100291-100003,
100271-100291-100019, 100271-100291-100043,
100271-100291-100049, 100271-100291-100057,
100271-100291-100069, 100271-100291-100103,
100271-100291-100109, 100271-100291-100129,
100271-100291-100151, 100271-100291-100153,
100271-100291-100169, 100271-100291-100183,
100271-100291-100189, 100271-100291-100193,
100271-100291-100207, 100271-100291-100213,
100271-100291-100237, 100271-100291-100267,
100271-100291-100279, 100271-100291-100297,
100291-100151-100003, 100291-100151-100019,
100291-100151-100043, 100291-100151-100049,
100291-100151-100057, 100291-100151-100069,
100291-100151-100103, 100291-100151-100109,
100291-100151-100129, 100291-100151-100153,
100291-100151-100169, 100291-100151-100183,
100291-100151-100189, 100291-100151-100193,
100291-100151-100207, 100291-100151-100213,
100291-100151-100237, 100291-100151-100267,
100291-100151-100271, 100291-100151-100279,
100291-100151-100297, 100291-100271-100003,
100291-100271-100019, 100291-100271-100043,
100291-100271-100049, 100291-100271-100057,
100291-100271-100069, 100291-100271-100103,
100291-100271-100109, 100291-100271-100129,
100291-100271-100151, 100291-100271-100153,

100291-100271-100169, 100291-100271-100183,
 100291-100271-100189, 100291-100271-100193,
 100291-100271-100207, 100291-100271-100213,
 100291-100271-100237, 100291-100271-100267,
 100291-100271-100279, 100291-100271-100297
 Total: 126

Всего таких комбинаций будет 126, следовательно, максимальное затраченное время будет равно

$$(126 - 1) * 1 \text{ сек} = 125 \text{ сек.}$$

Способ 2. Аналитический.

Можно заметить, что все числа из диапазона 100 000–100 300 начинаются на цифру «1». Это означает, что на место первого и второго числа подойдут только те числа, которые заканчиваются на «1». Шаблон комбинаций будет следующий:

$$1xxxx1-1xxxx1-1xxxxx$$

Анализируя полученные простые числа, всего 3 числа подходят на первую и вторую позицию: 100151, 100271, 100291. Учитывая невозможность повторения чисел возможны следующие варианты:

- на 1-ю позицию возможно 3 варианта,
- на вторую – 2 варианта,
- на третью – оставшиеся простые числа за вычетом использованных на первой и второй позиции (23–2=21).

Общее число комбинаций равно:

$$3 * 2 * 21 = 126.$$

Следовательно, максимальное затраченное время будет равно

$$(126 - 1) * 1 \text{ сек} = 125 \text{ сек.}$$

Ответ: минимальное время, за которое можно гарантированно получить пароль, равно **125 секунд.**

Задача 2. Сетевой график

Был получен фрагмент сетевого трафика пользователя при взаимодействии с игровым сервером. Известно, что сервер работает по протоколу UDP и его порт назначения равен 3365.

Структура UDP-дейтаграммы представлена ниже:

2 байта	2 байта	2 байта	2 байта	...
UDP-порт отправителя	UDP-порт получателя	Длина UDP-дейтаграммы	Контрольная сумма	Данные

Длина UDP-дейтаграммы включает в себя размер заголовка и размер данных в байтах.

Весь сетевой трафик шифруется методом «двоичного гаммирования», то есть путём выполнения операции «побитового исключающего ИЛИ» между байтами

UDP-дейтаграммы (включая заголовок) и байтами, полученными циклическим повторением последовательности некоторого ключа.

Дамп трафика:

```
A3 67 AA 90 A7 AD 8F 36 E0 F0 F3 95 F4 F0 E4 E7 E2 E1 87 F6 E8
F1 E2 B5 AA 90 A3 67 A7 BB BB 05 EE F1 E3 E4 E3 B5 A3 67 AA 90 A7
A7 C6 1C E4 FA E3 F0 87 98 87 FA EC B5
```

Определите, что сервер ответил на запрос пользователя, если известно, что для шифрования используется 2-байтовый ключ. В ответе укажите только содержимое поля данных UDP-дейтаграммы.

Решение

Из условия задачи известно, что взаимодействие происходит между клиентом и сервером. Можно сделать предположение, что первая дейтаграмма идет в направлении от клиента серверу. Порт клиента неизвестен, однако известен порт сервера. Следовательно, в первой дейтаграмме поле «UDP-порт получателя» (3-й и 4-й байты) будет равно $3365_{10} = 0D25_{16}$.

Весь сетевой трафик зашифрован методом «двоичного гаммирования». В основе «двоичного гаммирования» лежит операция «Исключающее ИЛИ» (XOR, ^). Таблица истинности операции «Исключающее ИЛИ»:

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Если T – это открытый текст, K – ключ, а C – зашифрованный текст, то верно следующее:

$$\begin{aligned} T \wedge K &= C \\ C \wedge K &= T \end{aligned}$$

При этом, $T \wedge C = K$.

Ели известны открытый текст и зашифрованный текст, то можно определить ключ шифрования.

3-й и 4-й байты первой дейтаграммы равны AA 90 – это зашифрованный текст. В этих байтах хранится значение 0D 25 – это открытый текст. Следовательно, ключ можно получить следующим образом:

$$K = AA\ 90 \wedge 0D\ 25.$$

Распишем данную операция поразрядно.

Двоичный код	Шестнадцатеричный код
10101010 10010000	AA 90
00001101 00100101	0D 25
10100111 10110101	A7 B5

Ключ шифрования – A7 B5.

Далее необходимо определить размер первой дейтаграммы. Поле «длина дейтаграммы» – это 5-й и 6-й байты – A7 AD. Однако это поле зашифровано, но нам известен ключ. Результат расшифрования поля:

Двоичный код	Шестнадцатеричный код
10100111 10101101	A7 AD
10100111 10110101	A7 B5
00000000 00011000	00 18

Длина первой дейтаграммы равна $0018_{16} = 24_{10}$.

Выделим и расшифруем поле данных первой дейтаграммы. Если длина всей дейтаграммы 24 байта, а длина заголовка равна $2(\text{порт отправителя}) + 2(\text{порт получателя}) + 2(\text{длина}) + 2(\text{контрольная сумма}) = 8$ байт,

то длина поля данных равна $24 - 8 = 16$ байт.

Зашифрованные данные:

E0 F0 F3 95 F4 F0 E4 E7 E2 E1 87 F6 E8 F1 E2 B5

Используя ключ, расшифруем эти байты:

E0 F0 F3 95 F4 F0 E4 E7 E2 E1 87 F6 E8 F1 E2 B5
 A7 B5 A7 B5 A7 B5 A7 B5 A7 B5 A7 B5 A7 B5 A7 B5
 47 45 54 20 53 45 43 52 45 54 20 43 4F 44 45 00

Каждый полученный байт представляет собой код символа в ASCII-таблице. В результате получаем текстовое сообщение: «GET SECRET CODE».

Следующая дейтаграмма начинается с 25-го байта:

AA90 A367 A7BB BB05 EE F1 ...

По логике взаимодействия вторая дейтаграмма – это ответ сервера. Значит, в поле «UDP-порт отправителя» должно стоять значение 0D25. В зашифрованном трафике в этом поле стоит значение AA90.

Используя вычисленный ранее ключ, можно расшифровать первые 2 байта дейтаграммы:

$$AA90 \wedge A7B5 = 0D25$$

Предположение подтвердилось – данная дейтаграмма отправлена от сервера клиенту. Осталось определить длину дейтаграммы (5-й и 6-й байты).

В зашифрованном виде поля длины дейтаграммы равно A7BB. Используя ключ A7B5 расшифруем это поле.

Двоичный код	Шестнадцатеричный код
10100111 10111011	A7 BB
10100111 10110101	A7 B5
00000000 00001110	00 0E

Длина второй дейтаграммы равна $000E_{16} = 14_{10}$.

Выделим и расшифруем поле данных второй дейтаграммы. Если длина всей дейтаграммы 14 байт, а длина заголовка равна 8 байт, то длина поля данных равна $14 - 8 = 6$ байт.

Зашифрованные данные: EE F1 E3 E4 E3 B5.

Используя ключ, расшифруем эти байты:

```

EE F1 E3 E4 E3 B5
A7 B5 A7 B5 A7 B5
-----
49 44 44 51 44 00

```

Каждый полученный байт представляет собой код символа в ASCII-таблице. В результате получаем текстовое сообщение: «*IDDQD*».

Рассмотрим третью дейтаграмму:

```
A367 AA90 A7A7 C61C E4 FA E3 F0 87 98 87 FA EC B5
```

Анализируя поля «UDP-порт источника» и «UDP-порт назначения», можно сделать вывод, что данная дейтаграмма отправлена от клиента серверу (порт назначения равен AA90, что при расшифровании даст 0D25 – порт сервера).

Определим размер третьей дейтаграммы. Поле «длина дейтаграммы» – это 5-й и 6-й байты – A7 A7. Используя известный ключ, расшифруем это поле.

Двоичный код	Шестнадцатеричный код
10100111 10100111	A7 A7
10100111 10110101	A7 B5
00000000 00010010	00 12

Длина первой дейтаграммы равна $0012_{16} = 18_{10}$.

Выделим и расшифруем поле данных третьей дейтаграммы. Если длина всей дейтаграммы 18 байт, а длина заголовка 8 байт, то длина поля данных равна $28-8=10$ байт.

Зашифрованные данные:

```
E4 FA E3 F0 87 98 87 FA EC B5
```

Используя ключ, расшифруем эти байты:

```

E4 FA E3 F0 87 98 87 FA EC B5
A7 B5 A7 B5 A7 B5 A7 B5 A7 B5
-----
43 4F 44 45 20 2D 20 4F 4B 00

```

Каждый полученный байт представляет собой код символа в ASCII-таблице. В результате получаем текстовое сообщение: «*CODE – OK*».

Ответ: сервер отправил клиенту текстовое сообщение **IDDQD**. Ключ шифрования **A7B5**.

Задача 3. Вирус

Имеется система, представляющая собой файл-серверную архитектуру, состоящую из 1 файл-сервера и 6 ПК. На файл-сервере хранится 30 файлов-приложений (*file1.exe, file2.exe, ..., fileN.exe*). Известно, что один из файлов заражён вредоносным кодом, который после попадания на клиентское устройство выводит его из строя через 1 час. Каждый ПК может копировать с файл-сервера любое количество файлов.

За какое *минимальное количество часов* можно точно определить зараженный файл. Ответ обоснуйте.

Решение

Для определения вредоносного файла необходимо каждый из них пронумеровать следующим образом.

1. Номер каждого файла представим в виде двоичного 6-разрядного разложения:

file1.exe – 000001 (двоичное представление числа 1)

file2.exe – 000010 (двоичное представление числа 2)

file3.exe – 000011 (двоичное представление числа 3)

...

file30.exe – 11110 (двоичное представление числа 30)

Такое разложение позволяет каждому файлу предоставить уникальную последовательность нулей и единиц.

2. Всего 6 ПК. Распределим их номера: 0,1,2,3,4,5.

3. Каждому ПК соответствует разряд в последовательности представления файла:

ПК0 – младший разряд (0-й),

ПК1 – второй справа разряд (1-й),

...

ПК5 – старший разряд (5-й)

4. На каждый ПК копируем только те файлы, в соответствующем разряде которого стоит «1»:

на ПК0 – file1.exe, file3.exe, file5.exe, ... , file29.exe

на ПК1 – file2.exe, file3.exe, file6.exe, file7.exe, ...

на ПК2 – file4.exe, file5.exe, file6.exe, file7.exe, ...

...

на ПК5 – file16.exe, file17.exe, file18.exe, ... , file30.exe.

5. Ровно через 1 час после копирования какие-то ПК выйдут из строя. Их номера определяют номер зараженного файла следующим образом: если ПК i -й не поврежден, то на i -м разряде номера файла ставим «0». Если ПК i -й поврежден, то на i -м разряде номера файла ставим «1».

В результате получится последовательность, переведя которую в десятичный формат можно однозначно определить зараженный файл.

Ответ: 1 час.

Задача 4. Стеганография

На web-странице содержатся ссылки на скачивание файлов-изображений с указанием контрольных сумм их содержимого. Известно, что в одном из них спрятано секретное текстовое сообщение. Определите это изображение и скрытое в нем секретное текстовое сообщение.

Файлы и контрольные суммы их содержимого (MD5):

d812f179df6b60943dbfd69c4e613aaf Chrysanthemum.jpg

4e427c78ecb620aecce46bb006d247e5 Desert.jpg

6dbafbc2e49df3c3cfe7515d5c6cac72 Hydrangeas.jpg

f037c82a48be22696142c59b4eea3298	Jellyfish.jpg
e3614da88d1511dfb8a05dd3ec24999d	Koala.jpg
46b851500907f4ccdfa75c0a29dd8dcf	Lighthouse.jpg
2022b59db2185282fd753f6320e782c4	Penguins.jpg
902b6ea2111efb87b19056655165c72d	Tulips.jpg

К задаче прилагается:

- 1) 8 файлов-изображений (*.jpg),
- 2) программа подсчета контрольной суммы по алгоритму MD5 (fciv.exe),
- 3) файл с описанием формата RAR-архива.

Решение

1. С помощью программы fciv.exe определим MD5-суммы для файлов и сравним их с данными в задаче.
2. Можно заметить, что для файла *Jellyfish.jpg* контрольная сумма не совпадает.
3. Предполагаем, зачем дано описание формата RAR-файла. Возможно, это намек на то, что в файле с изображением содержится скрытый RAR-архив.
4. Из описания RAR-архива понятно, что любой архив начинается с метки 0x5052. Используя HEX-редактор анализируем побайтовое содержимое файла *Jellyfish.jpg* и осуществляем поиск метки 0x5052. Метка найдена.
5. Копируем содержимое файла, начиная с метки, в новый файл, который назовем *архив.rar*
ИЛИ
можно переименовать файл *Jellyfish.jpg* в *архив.rar*.
6. В архиве содержится файл *Сообщение.docx*, открыв который, читаем сообщение – «Система обнаружения вторжений (IDS)».

**Ответ: Система обнаружения вторжений (IDS)
(файл – Jellyfish.jpg).**

Задача 5. Разработка процессора

При разработке процессора инженеры решили реализовать поддержку виртуальной памяти, когда программа оперирует виртуальным адресом, а этот адрес пересчитывается процессором в физический адрес ячейки – байта в оперативной памяти. Инженеры решили использовать структуру виртуального адреса, состоящего из трёх частей:

Индекс-1	Индекс-2	Индекс-3
----------	----------	----------

Вся память делится на непересекающиеся блоки одинаковой длины – страницы.

Индекс-1 содержит номер таблицы, в которой содержится информация о страницах.

Индекс-2 содержит номер страницы из таблицы с номером *Индекс-1*.

Индекс-3 определяет номер байта на странице с номером *Индекс-2* из таблицы с номером *Индекс-1* (см. рисунок).

Каждая таблица занимает ровно 1 страницу, размер таблиц фиксирован. Каждая запись в таблице имеет фиксированный размер и состоит из следующих полей:

- физический адрес страницы в оперативной памяти;
- заполнитель до кратности байту.

Общий диапазон адресов физической памяти, который должен адресоваться процессором – 1 Гб, размер страницы – 1 Кб.

Укажите минимально необходимые размерности полей Индекс-1, Индекс-2, Индекс-3, а также общую размерность виртуального адреса для возможности адресации объёма оперативной памяти в 1 Гб?

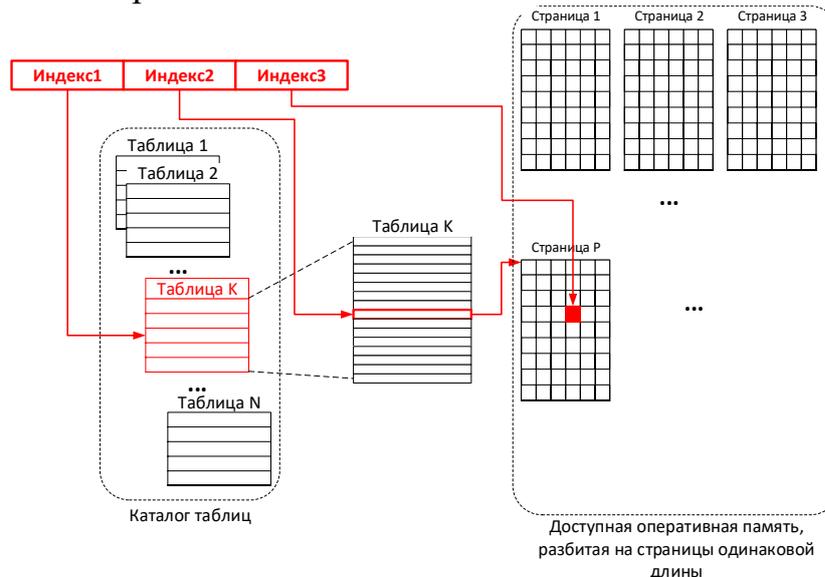


Рисунок. Организация виртуальной памяти

Решение

Вся память разбита на страницы. Страница занимает 1 Кб = 1024 байта. Значит, для адресации байта внутри страницы требуется 10 бит ($2^{10} = 1024$).

Индекс-3 = 10 бит.

В таблице одна запись занимает физический адрес и заполнитель. Для адресации 1 Гб адресного пространства (1 073 741 824 байт) нужен диапазон значений физического адреса $[0:2^{30}-1]$, т.е. 30 двоичных разрядов при линейном строении адреса. Соответственно, одна запись в таблице занимает $30(\text{адрес}) + 2(\text{заполнитель}) = 32$ двоичных разряда (4 байта). Размер таблицы – 1 Кб. Всего в таблице может быть $1024 / 4 = 256$ записей. Для адресации 256 записей необходимо 8 двоичных разрядов ($2^8 = 256$).

Индекс-2 = 8 бит.

В 1 Гб помещается $1\,073\,741\,824 / 1024 = 1\,048\,576$ страниц размером 1 Кб. Учитывая, что в одной таблице 256 записей, всего таблиц должно быть $1\,048\,576 / 256 = 4096$. Для адресации 4096 таблиц понадобится 12 двоичных разрядов ($2^{12} = 4096$).

Индекс-1 = 12 бит.

Итого, общая размерность виртуального адреса будет

$$\text{Индекс-1} + \text{Индекс-2} + \text{Индекс-3} = 12 + 8 + 10 = 30 \text{ бит.}$$

Ответ: **Индекс-1 – 12 бит,**
 Индекс-2 – 8 бит,
 Индекс-3 – 10 бит.
 Всего – 30 бит.