



DOI: 10.32517/2221-1993-2024-23-2-73-80

К. Ю. Поляков

Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия;
школа № 174, Санкт-Петербург, Россия

ПРОГРАММНОЕ РЕШЕНИЕ ЗАДАЧ ЕГЭ ПО ИНФОРМАТИКЕ НА IP-АДРЕСА И МАСКИ ПОДСЕТЕЙ*

Аннотация

В статье рассмотрены подходы к программному решению задач ЕГЭ по информатике на IP-адреса и маски подсетей методом перебора. Задачи этого типа включены в демонстрационный вариант контрольно-измерительных материалов 2024 года. Хотя составители заданий предлагают решать задачи этого типа аналитически, все шаги решения успешно автоматизируются. В статье приводятся различные варианты решения типовых задач на языках Python и PascalABC.NET, в том числе рассматривается использование модуля *ipaddress* из стандартной библиотеки языка Python и специальных классов для работы с сетевыми адресами из модуля *school* для PascalABC.NET. Приемы, предложенные в статье, могут быть полезны для учащихся, хорошо владеющих программированием, позволяя им сэкономить время при решении данного класса заданий ЕГЭ.

Ключевые слова: информатика, ЕГЭ, КИМ, IP-адрес, маска подсети.

1. Введение

В демонстрационный вариант контрольно-измерительных материалов (КИМ) ЕГЭ по информатике 2024 года вновь включены задачи на IP-адреса и маски подсетей. В настоящей статье речь пойдет про задание 13 в текущей модели КИМ [1]. Задачи на эту тему ранее уже входили в состав КИМ (см., например, [2]), но в 2021 году были исключены в связи с переходом на компьютерную форму сдачи экзамена.

В рассматриваемых задачах используется бесклассовая адресация в рамках протокола IPv4. IP-адрес представляет собой 32-битное целое число и обычно записывается в виде четырех значений октетов (8-битных блоков, байтов) в десятичной системе счисления, разделенных точками. Например, адрес 3232243872 записывается как 192.168.32.160.

IP-адрес содержит две части: адрес подсети и адрес хоста. Чтобы отделить одну часть от другой, используют *маску* — 32-битное число, в котором единичные биты определяют адрес подсети в IP-адресе, а нулевые — адрес хоста. Для адресации в сетях используют только непрерывные маски, которые имеют структуру «сначала единицы, потом нули». Это значит, что старшие биты IP-адреса, соответствующие единичным битам маски, относятся к адресу подсети, а остальные (младшие) биты адреса — к адресу хоста.

Вместо маски для краткости часто указывают только количество единичных битов в ней. Например, 192.168.32.160/28 — это подсеть с адресом 192.168.32.160 и маской 255.255.255.240, в которой 28 единиц и 4 нуля.

Рассмотрим задачу из демонстрационного варианта КИМ 2024 года.

* Материалы к статье можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_02-2024/

Контактная информация

Поляков Константин Юрьевич, доктор тех. наук, профессор, Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия; *адрес:* 190121, Россия, г. Санкт-Петербург, ул. Лощманская, д. 3; *учитель информатики, школа № 174, Санкт-Петербург, Россия; адрес:* 191015, Россия, г. Санкт-Петербург, ул. Кирочная, д. 54; *e-mail:* kpolyakov@mail.ru

K. Yu. Polyakov

Saint-Petersburg State Marine Technical University;
School 174, Saint-Petersburg, Russia

PROGRAM-BASED SOLUTION OF TASKS ON IP ADDRESSES AND SUBNET MASKS IN THE UNIFIED STATE EXAM IN INFORMATICS

Abstract

The article presents a brute-force programming-based approach to solving tasks on IP addresses and subnet masks included in the newest demonstration version of the test materials for the Unified State Exam in informatics. Although authors of the tasks suggest to solve tasks of this type analytically, all the solution steps can be successfully automated. The article presents various options for solving typical tasks in Python and PascalABC.NET, including the use of *ipaddress* module from the Python standard library and special classes for handling network addresses from *school* module for PascalABC.NET. The techniques proposed in the article can be useful for students who have a good knowledge of programming, allowing them to save time when solving this class of USE tasks.

Keywords: informatics, Unified State Exam, USE, test materials, IP address, subnet mask.

Задача 1 [1].

Сеть задана IP-адресом 192.168.32.160 и маской сети 255.255.255.240. Сколько в этой сети IP-адресов, для которых сумма единиц в двоичной записи IP-адреса четна?

Решение.

Согласно спецификации [1], это задача повышенного уровня, для решения которой требуется умение использовать маску подсети. Ее решение несложно найти аналитически, не используя программу; именно этот вариант и предусмотрен авторами КИМ. Ответ зависит только от маски подсети, точнее, от количества нулей в ней. В данном случае маска 255.255.255.240 содержит 4 нуля, так что в последних четырех битах IP-адресов этой подсети может быть закодировано $2^4 = 16$ различных значений. Половина из этих 4-битных значений содержит четное количество единиц, а половина — нечетное. Поэтому ответ на вопрос задачи — 8.

Основная тема настоящей статьи — программное решение задачи 1 и подобных ей полным перебором вариантов. Такой подход может быть полезен для тех учеников, которые хорошо владеют программированием и хотят сэкономить время для решения более сложных задач высокого уровня. При этом менее вероятны ошибки по невнимательности, например, неверное определение количества нулей в маске подсети.

2. Общая структура программы

Определим класс задач, к которому относится задача 1. Некоторая подсеть задана адресом и маской в рамках протокола IPv4. Требуется определить количество IP-адресов в этой подсети, которые удовлетворяют некоторому условию. При этом, как следует из ответа в [1], первый и последний адреса в пространстве адресов подсети — адрес всей подсети и широковещательный адрес — тоже следует учитывать наравне с обычными адресами хостов.

Общую структуру программы можно записать на псевдокоде следующим образом:

```
count=0
для всех ip в подсети:
    если valid(ip):
        count+=1
print(count)
```

Здесь **valid** — это логическая функция, которая возвращает истинное значение (**True**), если IP-адрес, переданный ей в качестве аргумента, удовлетворяет условию задачи, и ложное значение (**False**) в противном случае.

Таким образом, для построения работающей программы нужно:

- 1) организовать перебор всех IP-адресов в заданной подсети;
- 2) составить функцию **valid**, которая проверяет соответствие IP-адреса заданному условию.

Сначала приведем решение задачи 1 с помощью встроенных средств языков программирования Python и PascalABC.NET [11], а затем используем дополнительные модули для работы с IP-адресами.

3. Преобразование IP-адреса в число

Прежде всего преобразуем IP-адрес, заданный в виде значений четырех октетов, в целое число. Поскольку каждая часть адреса вида 192.168.32.160 представляет собой значение одного 8-битного байта, фактически этот адрес записан в системе счисления с основанием $2^8 = 256$. Тогда получить числовое значение IP-адреса можно с помощью развернутой формы записи числа [6]:

$$A = 192 \cdot 256^3 + 168 \cdot 256^2 + 32 \cdot 256^1 + 160.$$

Будем считать, что изначально IP-адрес записан в символьной строке, например, так (язык Python):

```
s='192.168.32.160'
```

С помощью метода **split** разобьем эту строку на части, разделенные точками, и преобразуем каждую часть в целое число:

```
n=[int(p) for p in s.split('.')]

```

Числовое значение IP-адреса найдем с помощью развернутой формы записи числа:

```
ip=n[0]*256**3+n[1]*256**2+n[2]*256+n[3]
print(ip)

```

Эти операции имеет смысл оформить в виде функции **ip2int**, которая принимает символьную запись IP-адреса и возвращает его целочисленное значение:

```
def ip2int(s):
    n=[int(p) for p in s.split('.')]
    ip=n[0]*256**3+n[1]*256**2+n[2]*256+n[3]
    return ip
print(ip2int('192.168.32.160'))

```

Последняя строка в этом фрагменте — это основная программа, которая вызывает функцию **ip2int** и сразу выводит ее результат на экран.

Напишем аналогичную функцию на языке PascalABC.NET.

Преобразуем строку **s** в массив чисел:

```
var n:=s.Split('.').Select(p->p.ToInteger)
    .ToArray;
```

Метод **split** разбивает строку на части, используя точку в качестве разделителя. Затем с помощью метода **Select** мы преобразуем каждый из фрагментов в целое число. Результат работы метода **Select** — это последовательность, которую метод **ToArray** преобразует в числовой массив. Познакомиться с этими методами подробно можно в [7].

Программа на языке PascalABC.NET, выполняющая те же действия, что и программа на Python, записывается более многословно:

```
function ip2int(s: string): longword;
begin
    var n:=s.Split('.').Select(p->p.ToInteger)
        .ToArray;
    result:=n[0]*256*256*256+n[1]*256*256+n[2]*
        256+n[3];
end;
print(ip2int('192.168.32.160'));
```

Функция должна возвращать беззнаковое значение типа `longword` (или 64-битное значение типа `int64`), потому что 32-битный результат типа `integer` окажется отрицательным, если первый октет IP-адреса будет больше, чем 127.

Числовое значение IP-адреса можно получить также с помощью схемы Горнера [6]. Например, для адреса 192.168.32.160 получаем:

$$A = ((192 \cdot 256 + 168) \cdot 256 + 32) \cdot 256 + 160.$$

Этот алгоритм удобно реализовать в теле функции с помощью цикла:

```
def ip2int(s):
    ip=0
    for p in s.split('.'):
        ip=ip*256+int(p)
    return ip
```

Интерфейс функции, т. е. ее входные и выходные данные, не изменился, поэтому вызывается она так же, как и ранее.

Аналогичная функция на PascalABC.NET выглядит следующим образом:

```
function ip2int(s: string): longword;
begin
    result:=0;
    foreach var p in s.Split('.') do
        result:=256*result+p.ToInteger;
    end;
```

Отметим, что эти функции могут быть использованы для преобразования в целое число не только IP-адреса, но и маски подсети, которая записывается в такой же форме — как десятичные значения четырех октетов, разделенные точками.

4. Преобразование IP-адреса в двоичный код

Для того чтобы определить количество единиц и нулей в IP-адресе или маске подсети, нужно получить двоичный код этого числа.

Пусть в переменной `net` записан некоторый числовой IP-адрес, например:

```
net=ip2int('192.168.32.160')
```

Простейший вариант перевода значения переменной `net` в двоичную систему счисления с помощью функции `bin` или f-строки:

```
net2a=bin(net)[2:]
net2b=f'{net:b}'
```

может привести к неверному ответу, если значение первого октета будет меньше, чем 128 (например, для адреса 5.168.32.160). В этом случае длина полученной битовой цепочки окажется меньше, чем 32, и количество нулей в ней будет определяться неверно. Поэтому нужно использовать формат `032b`, что означает «дополнить результат нулями слева до длины 32 символа»:

```
net2 = f'{net:032b}'
```

В языке PascalABC.NET эта идея реализуется с помощью функции `Bin` из модуля `school` (перевод числа в двоичную систему счисления) и метода расширения `PadLeft` (дополнить символами слева до нужной длины строки):

```
uses school;
...
var net2:=Bin(net).PadLeft(32,'0');
```

5. Перебор IP-адресов

Как отмечалось выше, для решения задачи нам необходимо сгенерировать все IP-адреса, которые входят в заданную подсеть. Пусть в переменной `net` хранится числовой адрес подсети, а в переменной `zeros` — количество нулей в маске подсети:

```
net=ip2int('192.168.32.160')
mask=ip2int('255.255.255.240')
zeros=f'{mask:032b}'.count('0')
```

Тогда все IP-адреса, относящиеся к данной подсети, получаются добавлением целых чисел из полуинтервала $[0; 2^{\text{zeros}})$ к адресу подсети, который записан в переменной `net`. Поэтому цикл для подсчета всех IP-адресов, удовлетворяющих условию, можно написать так:

```
count=0
for i in range(2**zeros):
    if valid(net+i):
        count+=1
print(count)
```

Для задачи 1 функция `valid` возвращает результат `True` (истина), если в двоичной записи IP-адреса четное число единиц:

```
def valid(ip):
    return f'{ip:032b}'.count('1') % 2 == 0
```

Заметим, что при изменении условия отбора нужно переписать только функцию `valid`, вся остальная программа не требует изменений.

Приведем полную программу для решения задачи 1:

```
def ip2int(s):
    ip=0
    for p in s.split('.'):
        ip=ip*256+int(p)
    return ip
def valid(ip):
    return f'{ip:032b}'.count('1') % 2 == 0
net=ip2int('192.168.32.160')
mask=ip2int('255.255.255.240')
zeros=f'{mask:032b}'.count('0')
count=0
for i in range(2**zeros):
    if valid(net+i):
        count+=1
print(count)
```

Аналогичная программа на языке PascalABC.NET выглядит следующим образом:

```
##
uses school;
function ip2int(s: string): longword;
begin
    result:=0;
    foreach var p in s.Split('.') do
        result:=256*result+p.ToInteger;
    end;
function valid(ip: integer):=
    Bin(ip).PadLeft(32, '0').CountOf('1')
        .IsEven;
var net:=ip2int('192.168.32.160');
var mask:=ip2int('255.255.255.240');
var zeros:=Bin(mask).CountOf('0');
var count:=0;
for var i:=0 to Round(2**zeros)-1 do
    if valid(net+i) then
        count += 1;
print(count);
```

6. Модуль `ipaddress` (Python)

Модуль `ipaddress` входит в стандартную библиотеку Python и устанавливается вместе с интерпретатором. Поэтому можно быть уверенным в том, что на экзамене ученики смогут его использовать.

Главное преимущество этого модуля — простая организация перебора всех адресов, принадлежащих подсети.

Сначала создаем объект, описывающий всю подсеть. Для этого вызывается функция `ip_network` из модуля `ipaddress`:

```
from ipaddress import ip_network
net=ip_network(
    '192.168.32.160/255.255.255.240')
```

Аргумент этой функции — символьная строка, содержащая IP-адрес подсети и ее маску, разделенные знаком /. Функция `ip_network` возвращает объект класса `IPv4Network`, который описывает такую подсеть. Теперь с помощью цикла `for` можно выполнить перебор всех адресов этой подсети:

```
for ip in net:
    ...
```

Полная программа для решения задачи 1 с использованием модуля `ipaddress` может быть записана так:

```
from ipaddress import ip_network
def valid(ip):
    return f'{ip:b}'.count('1') % 2 == 0
net=ip_network(
    '192.168.32.160/255.255.255.240')
count=0
for ip in net:
    if valid(ip): count+=1
print(count)
```

Обратите внимание на функцию `valid`, где для преобразования адреса в двоичный формат применен формат `b`, а не `032b`, как было раньше. Дело в том, что метод,

преобразующий IP-адрес (объект класса `IPv4Address`) в двоичное представление, всегда дополняет битовую строку слева нулями до длины 32.

Заметим, что цикл, в котором определяется значение переменной `count`, можно заменить вычислением длины списка, составленного из подходящих IP-адресов, для которых функция `valid` вернула результат `True`:

```
count=len([ip for ip in net if valid(ip)])
```

При желании можно сократить программу, оставив только строки с импортом модуля `ipaddress` и выводом результата:

```
from ipaddress import ip_network
print(len([ip
    for ip in ip_network(
        '192.168.32.160/255.255.255.240')
    if f'{ip:b}'.count('1') % 2 == 0]))
```

Поскольку нас интересует только *количество* подходящих адресов, а не сами адреса, можно заменить каждый из них единицей и сложить все эти единицы, вызвав функцию `sum`:

```
from ipaddress import ip_network
print(sum(1
    for ip in ip_network(
        '192.168.32.160/255.255.255.240')
    if f'{ip:b}'.count('1') % 2 == 0)))
```

В языке Python можно складывать и логические значения. При этом `True` автоматически преобразуется в 1, а `False` — в 0.

В следующем варианте программы суммируются логические значения, полученные при проверке заданного условия для каждого IP-адреса:

```
from ipaddress import ip_network
print(sum(f'{ip:b}'.count('1') % 2 == 0
    for ip in ip_network(
        '192.168.32.160/255.255.255.240')))
```

Использование функций модуля `ipaddress` значительно ускоряет написание программы, что позволит подготовленным ученикам решить задачу быстрее, чем с помощью аналитического метода.

С подробной информацией о функциях модуля `ipaddress` можно познакомиться в официальной документации [10].

7. Модуль `school` (PascalABC.NET)

В модуль `school` в версиях PascalABC.NET не ниже 3.9.0.3360 от 21.09.2023 добавлен калькулятор сетей TCP/IP [8]. Применительно к решению задач ЕГЭ его возможности аналогичны возможностям модуля `ipaddress` в Python.

Приведем сразу полную программу для решения задачи 1, использующую модуль `school`:

```
##
uses school;
var net:=new CalcIP('192.168.32.160',
    '255.255.255.240');
var count:=0;
foreach var ip in net.GenAddrBin do
```



```
if ip.CountOf('1').IsEven then
    count+=1;
print(count);
```

CalcIP — это класс, который описывает подсеть. При создании объекта этого класса нужно передать конструктору две символьных строки: IP-адрес подсети и ее маску. У этого объекта есть метод **GenAddrBin**, который строит последовательность символьных строк, содержащую все 32-битные двоичные адреса, принадлежащие подсети (включая адрес самой подсети и широковещательный адрес). Цикл **foreach** позволяет перебрать все элементы такой последовательности; с помощью переменной-счетчика **count** мы считаем те из них, в которых четное число единиц.

Вместо цикла можно сразу вывести ответ, применив метод **Count** для последовательности, а затем метод **Print** для вывода результата:

```
net.GenAddrBin
    .Count( ip->ip.CountOf('1').IsEven )
    .Print;
```

Аргумент метода **Count** — это лямбда-функция (см. [7]), которая возвращает истинное значение, если число единиц в двоичной записи IP-адреса четное.

8. Примеры

В этом разделе мы разберем несколько примеров типовых задач на IP-адреса и маски подсетей, которые могут встретиться в КИМ ЕГЭ. Задачи по этой теме можно найти в открытом банке заданий ЕГЭ [9], в пособии [4], а также в сети Интернет [3, 5].

Задача 2.

Для узла с IP-адресом 115.98.181.167 адрес сети равен 115.98.181.160. Определите наименьшее возможное значение последнего (самого правого) байта маски.

Решение.

При полном переборе нужно проверить все варианты числовых значений четвертого байта маски. Для каждого из них определим номер сети, соответствующий известному адресу узла и маске. Если полученный адрес сети совпадает с заданным (115.98.181.160), выводим ответ.

Можно организовать цикл перебора так:

```
b=0
for p in range(7, -1, -1):
    b+=2**p
    if valid(b):
        print(b)
        break
```

Переменная **b** обозначает четвертый байт маски. Сначала его значение равно 0, затем по очереди добавляем одну за другой степени числа 2 в порядке убывания: 2^7 , 2^6 и т. д. до 2^0 . Значение **b** при этом изменяется следующим образом:

0, 128, 192, 224, 240, 248, 252, 254, 255.

Фактически на каждой итерации цикла старший значащий ноль в двоичной записи байта заменяется

на единицу. Заметим, что значение **b=0** (сеть с маской 255.255.255.0) не проверяется: последний байт адреса сети 115.98.181.160 ненулевой, поэтому четвертый байт маски точно больше нуля.

Логическая функция **valid** возвращает значение **True**, если сеть, полученная с данным значением **b**, соответствует условию задачи. Значения **b** в цикле перебираются в порядке возрастания, поэтому после получения первого (минимального) подходящего значения **b** работу цикла можно прервать.

В функции **valid** используем возможности модуля **ipaddress**:

```
from ipaddress import ip_network
def valid(b):
    netIP = '115.98.181.160'
    hostIP = '115.98.181.167'
    net = ip_network( f'{hostIP}/'
                     f'255.255.255.{b}', 0 )
    return str(net.network_address) == netIP
```

Для переданного значения **b** функция **ip_network** строит объект **net**, описывающий сеть. Обратите внимание на второй аргумент при вызове этой функции, равный нулю. Дело в том, что здесь сеть задается нестандартно: через адрес узла (в переменной **hostIP**), а не через адрес сети. Нулевой второй аргумент подавляет сообщение об ошибке и сообщает функции **ip_network** о том, что нужно построить адрес соответствующей сети, который будет доступен через свойство **network_address**. Если он совпадает с заданным адресом сети (значением переменной **netIP**), то решение найдено.

Аналогичное решение на языке PascalABC.NET использует возможности модуля **school**:

```
##
uses school;
function valid(b: integer): boolean;
begin
    var netIP:='115.98.181.160';
    var hostIP:='115.98.181.167';
    var net:=new CalcIP(hostIP,
                       '$255.255.255.{b}' );
    result:=(net.Network.Value=netIP)
end;
var b:=0;
for var p:=7 downto 0 do begin
    b+=round(2**p);
    if valid(b) then begin
        println(b);
        break
    end
end;
```

Можно решить задачу иначе, сразу построив список всех возможных значений четвертого байта маски:

```
bAll=[0, 128, 192, 224, 240, 248, 252, 254, 255]
```

Согласно условию, задача сводится к тому, чтобы найти минимальное из этих значений, которое при поразрядной конъюнкции с числом 167 (это младший байт адреса узла) даст 160 — младший байт адреса сети. Поэтому решение можно записать очень кратко:

```
print(min(b for b in bAll if 167 & b == 160))
```

Модуль `ipaddress` в этом случае использовать не нужно.

Программа на PascalABC.NET тоже укладывается в две строки:

```
##
var bAll:=|0, 128, 192, 224, 240, 248, 252,
    254|;
bAll.Where(b->(167 and b=160)).Min.Println;
```

Сначала из массива `bAll` с помощью метода `Where` отбираются те значения, при которых выполняется условие задачи, затем выводится минимальное из них. Заметим, что в этом варианте решения проверка выполняется для всех возможных значений `b`. Для ускорения можно учесть, что массив `bAll` отсортирован по возрастанию. Поэтому вместо метода `Min` лучше использовать метод `First`, который остановит вычисления при получении первого подходящего ответа.

Задача 3.

Для узла с IP-адресом 193.57.165.211 адрес сети равен 193.57.160.0. Каково наибольшее возможное количество единиц в разрядах маски?

Решение.

Для решения методом «грубой силы» можно применить ту же идею, что и при решении задачи 2. Перебор здесь ведется по количеству единиц в маске, которое будем хранить в переменной `ones`. Поскольку нас интересует *наибольшее* количество единиц, значения переменной `ones` перебираются в порядке убывания:

```
for ones in range(31,0,-1):
    if valid(ones):
        print(ones)
        break
```

Логическая функция `valid` строит объект, описывающий сеть с выбранной маской, и возвращает `True`, если адрес этой сети совпадает с заданным:

```
from ipaddress import ip_network
def valid(ones):
    netIP='193.57.160.0'
    hostIP='193.57.165.211'
    net=ip_network(f'{hostIP}/{ones}', 0)
    return str(net.network_address) == netIP
```

Обратите внимание, что при вызове функции `ip_network` маска подсети задается не в виде четырех значений октетов, а количеством единиц.

Аналогичное решение возможно и на языке PascalABC.NET:

```
##
uses school;
function valid(ones: integer): boolean;
begin
    var netIP:='193.57.160.0';
    var hostIP:='193.57.165.211';
    var net:=new CalcIP(hostIP, ones);
    result:=(net.Network.Value=netIP)
end;
for var ones:=31 downto 1 do
```

```
if valid(ones) then begin
    println(ones);
    break
end;
```

Попробуем найти более простое решение, не использующее модуль `ipaddress`.

При сравнении IP-адресов сети и узла легко заметить, что их первые два байта одинаковы, а третьи байты различны. Это значит, что первые два байта маски равны 255 и для решения задачи достаточно перебрать только все возможные значения третьего байта маски. Их можно предварительно объединить в список `bAll`:

```
bAll=[0, 128, 192, 224, 240, 248, 252, 254, 255]
```

В результате поразрядной конъюнкции третьего байта маски `b` со значением 165 (это третий байт адреса узла) должно получиться число 160 — третий байт IP-адреса сети.

Заметим, что индекс элемента в списке `bAll` равен количеству единиц в третьем байте маски. Например, элемент со значением $192 = 11000000_2$, содержащим две единицы в двоичной записи, имеет в массиве `bAll` индекс 2. Первые два байта маски равны $255 = 11111111_2$, они дают сразу 16 единиц. Поэтому маска с третьим байтом `b` содержит $16+i$ единиц, где i — индекс выбранного значения `b` в списке `bAll`.

Краткое решение, основанное на этих идеях, может быть получено с помощью функции `max`, которая применяется к генераторному выражению:

```
print(max(16+i for i, b in enumerate(bAll)
        if 165 & b == 160))
```

Функция `enumerate` строит последовательность кортежей, добавляя индексы к значениям элементов списка. На каждой итерации цикла `for` очередной кортеж сразу распаковывается в переменные `i` и `b`. Генераторное выражение (аргумент функции `max`) отбирает те кортежи, которые соответствуют условию задачи. Функция `max` находит максимальное из значений $16+i$, где i — первый элемент каждого кортежа с подходящим значением `b`. Сумма $16+i$ — это количество единиц в маске: 16 единиц в первых двух байтах и i единиц в третьем.

Краткое решение на PascalABC.NET использует аналогичные методы `Numerate` (добавление нумерации), `Where` (отбор нужных элементов последовательности), `Max` (выбор максимального значения) и `Println` (вывод на экран):

```
##
var bAll:=|0, 128, 192, 224, 240, 248, 252,
    254|;
bAll.Numerate(0)
    .Where(\(i,b)->(165 and b = 160))
    .Max(\(i,b)->16+i).Println;
```

Задача 4.

Узел с IP-адресом 173.97.129.92 принадлежит сети с маской 255.255.252.0. Сколько в этой сети IP-адресов, для которых в двоичной записи IP-адреса суммарное количество единиц в двух правых байтах не меньше суммарного количества единиц в двух левых байтах?

Решение.

Эта задача решается так же, как и задача 1. Удобно сначала вычислить количество единиц в первых двух байтах (переменная `left1` в программе); это число одинаково для всех IP-адресов данной подсети:

```
from ipaddress import ip_network
left1=f'{173:b}{97:b}'.count('1')
print(sum(f'{ip:b}'[16:].count('1') >= left1
        for ip in ip_network(
            '173.97.129.92/255.255.252.0',0)))
```

Краткое решение на языке PascalABC.NET:

```
##
uses school;
var net:=new CalcIP('173.97.129.92',
    '255.255.252.0');
var left1:=net.Network.Value2[:17]
    .CountOf('1');
net.GenAddrBin.Count(ip ->
    ip[17:].CountOf('1') >= left1)
    .Println;
```

Метод `Network` определяет адрес подсети, а метод `Value2` — его двоичное представление, из которого выделяются первые 16 символов для вычисления значения `left1`. Нужно учитывать, что по умолчанию нумерация символов строки в PascalABC.NET начинается с 1, а не с 0, поэтому два первых байта — это символы строки с индексами от 1 до 16 включительно, т. е. срез `[:17]`.

Задача 5.

Узел с IP-адресом 63.254.104.115 принадлежит сети с маской 255.255.A.0, где A — некоторое допустимое для записи маски число. Определите минимальное значение A, при котором для всех IP-адресов этой сети в двоичной записи IP-адреса суммарное количество единиц в двух левых байтах не меньше суммарного количества единиц в двух правых байтах.

Решение.

В этой задаче нужно перебрать все допустимые значения A, которые можно сразу объединить в список:

```
aAll=[0, 128, 192, 224, 240, 248, 252, 254, 255]
```

Основная программа имеет стандартную структуру: перебор значений A, пока не будет найдено первое подходящее:

```
for A in aAll:
    if valid(A):
        print(A)
        break
```

Проверка выполняется в логической функции `valid`, которая возвращает значение `True`, если сеть при данном значении A удовлетворяет условию задачи:

```
from ipaddress import ip_network
def valid(A):
    net=ip_network(f'63.254.104.115/'
        f'255.255.{A}.0',0)
    left1=f'{63:b}{254:b}'.count('1')
```

```
for ip in net:
    if not(left1>=f'{ip:b}'[16:].count('1')):
        return False
return True
```

В цикле для каждого IP-адреса, принадлежащего сети, проверяется условие «суммарное количество единиц в левых двух байтах не менее суммарного количества единиц в правых двух байтах». Если хотя бы для одного адреса условие не выполняется, функция сразу возвращает `False`. Если условие верно для всех IP-адресов и цикл завершился нормально, то значение A — подходящее и функция вернет результат `True`.

Можно также использовать генераторное выражение и функции `all` и `min`:

```
from ipaddress import ip_network
hostIP='63.254.104.115'
allA=[0, 128, 192, 224, 240, 248, 252, 254,
    255]
print(min(A for A in allA
    if all((s:=f'{ip:b}')[:16].count('1') >=
        s[16:].count('1')
        for ip in ip_network(
            f'{hostIP}/255.255.{A}.0',0))))
```

В таком виде решение становится похоже на ребус, который сложно понять с первого взгляда. Самое важное, что вместе с увеличением сложности кода повышается вероятность ошибки. Поэтому вряд ли стоит рекомендовать использование таких приемов во время экзамена.

Приведем также решение на PascalABC.NET:

```
##
uses school;
function valid(A: integer): boolean;
begin
    var net:=new CalcIP('63.254.104.115',
        '$255.255.{A}.0');
    result:=True;
    foreach var ip in net.GenAddrBin do
        if not(ip[:17].CountOf('1') >= ip[17:]
            .CountOf('1')) then begin
            result:=False;
            break
        end;
    end;
end;
var allA:=|0, 128, 192, 224, 240, 248, 252,
    254, 255|;
foreach var A in allA do
    if valid(A) then begin
        println(A);
        break
    end;
```

Задача 6.

Узел с IP-адресом 64.32.A.153, где A — некоторое допустимое для записи IP-адреса число, принадлежит сети с маской 255.255.224.0. Определите минимальное значение A, при котором для всех IP-адресов этой сети в двоичной записи IP-адреса суммарное количество единиц в двух левых байтах не больше суммарного количества единиц в двух правых байтах.

Решение.

Эта задача похожа на предыдущую. Главное отличие состоит в том, что значение *A* может быть любым целым числом от 0 до 255 включительно. Поэтому перебор значений *A* удобно сделать в цикле по диапазону

```
for A in range(256):
    if valid(A):
        print(A)
        break
```

Функция `valid` аналогична одноименной функции из решения предыдущей задачи:

```
def valid(A):
    net=ip_network(f'64.32.{A}.153/'
                  f'255.255.224.0',0)
    left1 = f'{64:b}{32:b}'.count('1')
    for ip in net:
        if not(left1 <=
               f'{ip:b}'[16:].count('1')):
            return False
    return True
```

Приведем аналогичное решение на PascalABC.NET:

```
##
uses school;
function valid(A: integer): boolean;
begin
    var net:=new CalcIP('$64.32.{A}.153',
        '255.255.224.0');
    result:=True;
    foreach var ip in net.GenAddrBin do
        if not(ip[17].CountOf('1') <= ip[17:]
            .CountOf('1')) then begin
            result:=False;
            break
        end
    end;
end;
foreach var A in 0..255 do
    if valid(A) then begin
```

```
        println(A);
        break
    end;
```

9. Заключение

В настоящей статье рассмотрены задачи на IP-адреса и маски подсетей, которые включены в КИМ ЕГЭ по информатике 2024 года. Приводятся программные решения таких задач методом полного перебора на языках Python и PascalABC.NET. Такой подход может быть полезен для учащихся, хорошо владеющих программированием.

Список источников

1. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2024 года по информатике. https://doc.fipi.ru/ege/demoversii-specifikacii-kodifikatory/2024/inf_11_2024.zip
2. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2020 года по информатике. https://doc.fipi.ru/ege/demoversii-specifikacii-kodifikatory/2020/inf_ege_2020.zip
3. Кабанов А. М. Демонстрационная версия станции КЕГЭ. <https://kompege.ru>
4. Крылов С. С., Чуркина Т. Е. ЕГЭ-2024. Информатика. Типовые экзаменационные варианты. 20 вариантов. М.: Национальное образование, 2024. 250 с.
5. Поляков К. Ю. Материалы для подготовки к ЕГЭ по информатике (2024). <https://kpolyakov.spb.ru/school/ege.htm>
6. Поляков К. Ю., Еремин Е. А. Информатика. 10 класс (базовый и углубленный уровни). В 2 ч.: учебник. Ч. 1. М.: БИНОМ. Лаборатория знаний, 2019. 352 с.
7. Осипов А. В. PascalABC.NET: Введение в современное программирование. 573 с. <https://pascalabc.net/downloads/OsipovBook/КнигаДляСайта.pdf>
8. Осипов А. В. PascalABC.NET: Калькулятор сетей TCP/IP в библиотеке School. https://teletype.in/@ralex061/k_YLcCFpIVE
9. Открытый банк заданий ЕГЭ // ФИПИ. <https://ege.fipi.ru/bank>
10. `ipaddress` — IPv4/IPv6 manipulation library — Python 3.12.3 documentation. <https://docs.python.org/3/library/ipaddress.html>
11. PascalABC.NET. Современное программирование на языке Pascal. <https://pascalabc.net>