«Мир информатики»

Журнал для тех, у кого информатика – любимый школьный предмет

Выпуск № 91, август 2024 г.

Спасибо вам, уважаемые коллеги!

За активную работу с учащимися по материалам интернет-журнала “Мир информатики” редакция журнала “Информатика в школе” выражает благодарность:

— Брунову Александру Сергеевичу, средняя школа села Ириновка, Новобурасский р-н Саратовской обл.;

— Букиной Елене Юрьевне, г. Зеленокумск Ставропольского края, школа № 1;

— Волковой Татьяне Петровне, Владимирская обл., г. Струнино, школа № 11;

— Волкову Юрию Павловичу, Владимирская обл., г. Струнино, школа № 11;

— Дикову Андрею Валентиновичу, г. Пенза, школа № 73;

— Долговой Галине Александровне, средняя школа села Восточное Нижегородской обл.;

— Евдокимовой Алевтине Ивановне, средняя школа поселка Осиновка, Алтайский край;

— Зудину Василию Павловичу, Ардатовский областной многопрофильный техникум, поселок Ардатов Нижегородской обл.;

— Зыбиной Анне Юрьевне, Барнаульский государственный педагогический колледж имени В.К. Штильке;

— Каменецкой Татьяне Сергеевне, Москва, школа № 1530 «Школа Ломоносова»;

— Кашаповой Рамиле Хабибовне, Республика Татарстан, г. Лениногорск, школа № 8;

— Комбаровой Светлане Ивановне, г. Воронеж, лицей № 2;

— Лазовской Жанне Георгиевне, Прилукская средняя школа, Республика Беларусь, Минская обл.;

— Лиготиной Жанне Васильевне, Барнаульский государственный педагогический колледж имени В.К. Штильке;

— Лукьяновой Наталии Владимировне, Барнаульский государственный педагогический колледж имени В.К. Штильке;

— Михайлину Станиславу Александровичу, Владимирская обл., г. Александров, школа № 4;

— Муравьевой Ольге Викторовне, средняя школа деревни Муравьево, Вологодская обл.;

— Мурашову Юрию Николаевичу Владимирская обл., г. Александров, школа № 13;

— Пархатской Анастасии Михайловне, Барнаульский государственный педагогический колледж имени В.К. Штильке;

— Николаевой Веронике Павловне, средняя школа имени Героя Советского Союза П.Х. Бухтулова, Чувашская Республика, село Янтиково;

— Плюсниной Наталье Петровне, Хабаровский край, г. Комсомольск-на-Амуре, школа № 38;

— Порываевой Нафисе Сабитовне, Республика Татарстан, Мамадышский р-н, совхоз “Мамадышский”, политехнический колледж;

— Рунову Даниилу Николаевичу, Владимирская обл., г. Струнино, школа № 10;

— Семенковичу Алексею Олеговичу, Владимирская обл., г. Александров, школа № 13;

— Сысоляпиной Виктории Александровне, Барнаульский государственный педагогический колледж имени В.К. Штильке;

— Фирсовой Марии Николаевне, Волгоградская обл., г. Фролово, школа № 1 имени А.М. Горького;

— Черновой Ларисе Ивановне, средняя школа села Сердар, Республика Марий Эл;

— Шитовой Любови Алексеевне, средняя школа села Горелово Тамбовской обл.

Язык программирования Python. Повторение действий в программе[1]

В предыдущем выпуске интернет-журнала «Мир информатики» были рассмотрены особенности одной из двух программных конструкций, обеспечивающих повторение действий в программе, – инструкции (цикла) **`for`**. В данном выпуске обсудим инструкцию (цикл) **`while`**.

Можно сказать, что рассмотренная до этого инструкция (цикл) **`for`** используется в программе, когда известно *количество повторений* каких-то действий или известен *набор значений*, для которых должны выполняться повторяющиеся действия. Однако эта информация при разработке программ не всегда известна.

Рассмотрим ряд задач.

Задача 1. Дано число 923 451. Получить на экране:

92345
9234
923
92
9
0

Инструкции `print()`, выводящие на экран конкретные числа и цифры (`print(9234)`, `print(9)`, `print(9, 2, 3, 4, sep = '')` и т. п.), не использовать.

Решение

Как, имея переменную *n* = 923 451, получить число 92 345, то есть отбросить последнюю цифру числа *n*? Ответ – разделить *n* нацело на 10. А число 9234? – разделить n нацело на 100. Учитывая это, можем составить программу решения задачи:

```
n = 923451
print(n//10)
print(n//100)
print(n//1000)
print(n//10000)
print(n//100000)
print(n//1000000)
```

Можно также сказать, что для решения задачи нужно 6 раз отбросить последнюю цифру меняющегося числа, начальное значение которого равно 923 451, и выводить полученное число на экран:

```
n = 923451
n = n//10
```

[1] Многократное выполнение одинаковых действий в программировании называют «цикл».

```
print(n)
n = n//10
print(n)
n = n//10
print(n)
n = n//10
print(n)
n = n//10
print(n)
n = n//10
print(n)
```

то есть можно использовать инструкцию цикла **`for`**:

```
for k in range(6):
    n = n//10
    print(n)
```

Предлагаем читателям самостоятельно решить следующую задачу.

Задача 2. Дано число 923 451. Получить на экране:

```
23451
3451
451
51
1
0
```

Инструкции `print()`, выводящие на экран конкретные числа и цифры (`print(3451)`, `print(1)` и т. п.), не использовать.

Задача 3. Дано натуральное (целое положительное) число *n*. Определить количество цифр в нем.

Идея решения такая.

Нужно многократно отбрасывать одну из цифр и при каждом шаге «отбрасывания» увеличивать значение переменной-счетчика на 1.

Какую цифру заданного числа можно отбросить? Ответ следует из решения задачи 1 – последнюю[2].

Следовательно, для решения задачи в программе надо многократно выполнить следующие действия:

1) отбросить последнюю цифру числа;

2) увеличить счетчик на 1.

Но сколько раз надо повторить указанные действия? – Неизвестно. Именно в таких случаях и применяется инструкция **`while`**.

Рассмотрим ее особенности.

Общий вид инструкции:

`while` <*условие*>**:**
    <*тело инструкции*>

где <*условие*> – условие, при котором выполняется <*тело инструкции*>.

[2] Для отбрасывания первой цифры необходимо знать, сколько цифр в записи числа, а это неизвестно.

Схема, иллюстрирующая работу инструкции **while**:

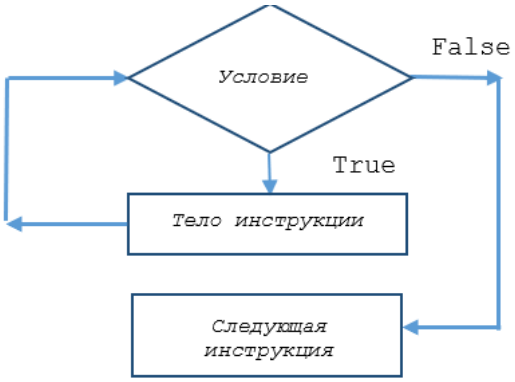


Рис. 1

Приведём полезное правило, с помощью которого можно установить условие, записываемое в такой инструкции:

1) необходимо определить условие, при котором *нельзя* или *не нужно* повторять действия;

2) записать в инструкция **while** условие, противоположное найденному на предыдущем этапе.

Применительно к задаче 3 (см. выше) инструкция **while** оформляется следующим образом:

```
while ? :
    #Отбрасывание последней цифры числа n
    n = n//10
    #Увеличение значения переменной-счетчика
    k = k + 1
```

Чтобы определить условие, обозначенное символом «?», вспомним приведенное чуть выше правило. Мы должны прекратить все действия, когда `n == 0` (см. задачу 1), значит, условие, записываемое после слова **while**, должно быть таким: `n > 0` (отрицательным значение *n* быть не может). Следовательно, можем оформить фрагмент так:

```
k = 0
while n > 0:
    n = n//10
    k = k + 1
```

Конечно, в данном случае условие, записываемое в программе после слова **while**, можно было определить, и не используя приведенного правила. В ряде случаев искомое условие не так очевидно. Вот пример задачи.

Задача 4. Даны два натуральных числа. Разработать программу для определения наибольшего общего делителя (НОД) заданных чисел.

Решение

Найти НОД двух натуральных чисел можно разными способами. Лучшим из них является использование алгоритма Евклида, названного так в честь его автора – древнегреческого математика III века до нашей эры. Суть алгоритма можно изобразить схемой:

$$\text{НОД}(a, b) \quad \begin{cases} = \text{НОД}(a - b, b) \text{ при } a > b \\ = \text{НОД}(a, b - a) \text{ при } b > a \\ = a \text{ (или } b\text{) при } a = b \end{cases}$$

Например, согласно схеме: НОД(26, 18) = НОД(8, 18) = НОД(8, 10) = НОД(8, 2) = НОД(6, 2) = НОД(4, 2) = НОД(2, 2) = 2.

Анализ показывает, что для нахождения НОД нужно многократно вычитать из большего из двух чисел меньшее, причем количество повторений неизвестно, то есть следует применить инструкцию **while**:

```
#Ввод чисел
a = int(input('Задайте первое число '))
b = int(input('Задайте второе число '))
#Определение НОД
while ? :
    if a > b:
        a = a - b
    else:
        b = b - a
НОД = a
#Вывод результата
print('НОД равен ', НОД)
```

Чтобы определить условие, обозначенное символом «?», также вспомним приведенное выше правило. Когда мы должны прекратить действия? Согласно алгоритму Евклида, когда `a == b`. Значит, вместо вопросительно знака должно быть записано условие `a!= b`.

Задания

1. Подумайте, чем в последней программе можно заменить многократные вычитания меньшего числа из большего, когда *a* много больше *b* или когда *b* много больше *a*, например при *a* = 348, *b* = 6.

2. Подумайте над вопросами:

1) может ли тело инструкции **while** не выполниться ни разу?

2) может ли тело инструкции **while** выполняться бесконечно (или до того момента, когда пользователь прервет выполнение одновременным нажатием клавиш <**Ctrl**> и <**C**> или закроет окно **Python Shell**)?

Задачи для разработки программ

1. Вывести на экран натуральные числа, не превышающие заданного числа *n*. Инструкцию **for** не использовать.

2. Напечатать все нечетные числа из интервала [10, 100]. Инструкцию **for** не использовать.

3. Дано натуральное число. Определить сумму его цифр.

4. Напечатать те натуральные числа, квадрат которых не превышает заданного числа *n*.

5. Дано число *n*. Из чисел 1, 4, 9, 16, 25, ... напечатать те, которые не превышают *n*.

6. Напечатать числа 1.0, 1.5, 2.0, …, 13.5. Инструкцию **for** не использовать.

7. Дано число `a` (0 < `a` ≤ 1). Из чисел $1, \frac{1}{2}, \frac{1}{3}$, ... напечатать те, которые не меньше `a`.

8. Дано число *a* (1 < *a* ≤ 1,5). Из чисел $1+\frac{1}{2}$, $1+\frac{1}{3}$, ... напечатать те, которые не меньше *a*.

Комментарий к выполнению

Числа $1+\frac{1}{2}$, $1+\frac{1}{3}$, ... представляют собой сумму $1 + \frac{1}{n}$ (n = 2, 3, …) и образуют убывающую последовательность.

9. Дана последовательность чисел. В ней число 0 – единственное и последнее. Ввести каждое число, а затем напечатать его на экране (с сообщением 'Вы ввели число:').

10. Дана последовательность вещественных чисел. Определить количество чисел, предшествующих первому числу 0. Числа, следующие за числом 0, вводить в программу не нужно.

11. Дана последовательность целых чисел. Первое число в последовательности нечетное. Найти сумму всех идущих подряд в начале последовательности нечетных чисел. Инструкцию **`if`** не использовать.

12. Дана последовательность целых из *n* целых чисел, в начале которой записано несколько равных между собой элементов. Определить количество таких элементов последовательности. Инструкцию **`if`** не использовать.

Инструкция **`while`** также может быть использована в программе для обеспечения некоторой паузы в работе программы. Составьте соответствующий вариант программы самостоятельно.

Обращаем внимание на одну особенность работы инструкции **`while`**. Как следует из рис. 2, условие проверяется только *перед* выполнением тела инструкции, но не проверяется в процессе выполнения. Неучет этого обстоятельства может привести к ошибкам. Очень наглядно это можно продемонстрировать в следующей задаче с исполнителем «Робот».

Задача. «Робот», находящийся на поле из клеток и умеющий перемещаться на одну клетку влево, вправо, вверх или вниз, находится в клетке **A** (рис. 2) и выполняет фрагмент программы:

```
while снизу свободно:
  вниз
  вниз
```

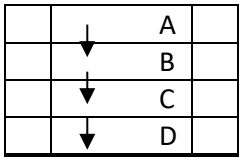


Рис. 3

Проследим работу инструкции согласно схеме на рис. 1.

1. Исходное положение – «Робот» в клетке **A**. Проверяется условие снизу свободно – оно истинно – выполнится тело инструкции **`while`**, и «Робот», сместившись дважды вниз, окажется в клетке **C**.

2. Опять проверяется условие снизу свободно – оно также истинно, и начнется выполнение тела инструкции – «Робот» сначала сместится в клетку **D**, но вторая инструкция вниз выполниться не сможет (появится сообщение об ошибке).

Конечно, «Робот» – условный исполнитель. Но аналогичные ситуации могут иметь место в «настоящих» компьютерных программах. Помните об этом!

Дополнение 1

Инструкцию **`while`** называют «инструкцией цикла с предусловием», или просто «циклом с предусловием», потому что проверка условия проводится сначала, до выполнения тела цикла (см. схему на рис. 1). Иными словами, действия повторяются, пока соблюдается заданное условие.

Часто нужно *прекратить* повторения, когда становится истинным некоторое условие. Во многих языках программирования в таких случаях используется так называемый «цикл с постусловием», в котором условие проверяется *после* выполнения тела цикла. Это значит, что действия в теле инструкции цикла при этом выполняются как минимум один раз. Такая

ситуация полезна, например, в случаях, когда нужно проверить правильность вводимых в программу значений. Например, в программе нахождения корней квадратного уравнения $ax^2 + bx + c = 0$ пользователь должен задать коэффициенты *a, b* и *c*, при этом коэффициент *a* не должен быть равен нулю.

В языке Python нет цикла с постусловием, но его можно организовать с помощью цикла **while** так:

```
print('Задайте значение коэффициента a уравнения:')
a = float(input())
while a == 0:
    print('Коэффициент a не должен быть равен нулю!')
    print('Задайте значение коэффициента a уравнения:')
    a = float(input())
```

Пользователь программы вводит значение коэффициента *a*, и в случае ввода ошибочного значения действия повторяются. Если введенное значение правильное, инструкция **while** больше выполняться не будет.

Однако приведенный вариант оформления не очень хорош, потому что нам пришлось два раза написать две одни и те же инструкции. Можно поступить иначе:

```
while True:
    print('Задайте значение коэффициента a уравнения:')
    a = float(input())
    if a != 0:
        break
```

Цикл, который начинается с заголовка **while True**, будет выполняться бесконечно, потому что условие **True** всегда истинно (см. рис. 1). Выйти из такого цикла можно только с помощью специальной инструкции `break` (досрочный выход из цикла). В данном случае она сработает тогда, когда станет истинным условие `a != 0`, то есть тогда, когда пользователь введет допустимое значение.

Конечно, инструкция `break` может быть применена и для досрочного выхода из цикла **for**. Это удобно делать, например, в задачах поиска некоторого заданного значения в наборе значений. В таких случаях инструкция **for** оформляется следующим образом:

```
for <всех значений в наборе>:
    if <условие поиска значения> истинно:
        break
…
```

В следующих разделах цикл **for** с инструкцией `break` будет использоваться часто.

Контрольные вопросы

1. Почему инструкцию **while** называют «циклом с предусловием»?
2. В чем особенность программной конструкции, которую называют «цикл с постусловием»? Как такой цикл можно организовать в программе на Python?
3. Для чего используется инструкция `break`?

Задания

13. Подготовьте фрагмент программы, в котором пользователь должен ввести четное число. В случае ввода нечетного числа на экран должно выводиться сообщение об ошибке, после чего действия должны повторяться до ввода правильного значения.
14. Подготовьте фрагмент программы, в которой пользователь должен ввести установленный пароль в виде целого числа. В случае ввода неправильного пароля на экран должно выводиться сообщение об ошибке, после чего действия должны повторяться до ввода правильного значения. После этого на экран должно выводиться некоторое приветствие.

15. Подготовьте фрагмент программы, в котором должны вводиться 10 чисел. Если будет введено число 0, ввод должен прекратиться.

Другие задачи для разработки программ

16. Вывести первое натуральное число, квадрат которого больше заданного значения n. Известно, что это число не больше 100.

Комментарий к выполнению. Здесь, в отличие от задачи 4, предложенной для разработки программ, выводить надо только одно число, которое будет удовлетворять заданному условию, после чего рассмотрение чисел прекратить.

17. Дано число `a` (`0 < a ≤ 1`). Из чисел 1, $\frac{1}{2}$, $\frac{1}{3}$, ... найти первое число, которое меньше `a`.

Преобразование одной инструкции цикла в другую

Часто в программах[3] необходимо заменить одну инструкцию цикла (**`for`** или **`while`**) на другую. Обсудим, всегда ли это возможно.

Если проанализировать схему на рис. 1 в предыдущем выпуске, то можно увидеть, что для инструкции **`for`**:

1) известно значение величины *a* до начала выполнения тела инструкции (`a == <начальное значение>`);

2) величина *a* увеличивается с шагом, равным 1 (в общем случае этот шаг известен – он задается в функции `range()`);

3) условием окончания работы инструкции является `a == <конечное значение>`.

Это означает, что мы знаем начальное значение параметра, знаем, как оно меняется и при каких его значениях надо повторять тело цикла[4], то есть можно вместо инструкции **`for`** использовать инструкцию **`while`**.

Например, приведенный в пункте 1 фрагмент с инструкцией **`for`**:

```
for a in range(10, 21):
    print(a)
```

можно заменить на следующий:

```
a = 10
while a != 21:
    print(a)
    a = a + 1
```

А наоборот – можно ли инструкцию **`while`** заменить на **`for`**? Вспомним задачу о количестве цифр некоторого натурального числа:

```
k = 0
while n > 0:
    n = n//10
    k = k + 1
```

В данном случае количество повторений тела инструкции **`while`** неизвестно (оно равно количеству цифр введенного числа, то есть зависит от исходных данных), то есть мы не можем применить инструкцию **`for`**? Иногда это возможно. Например, вместо фрагмента

```
n = 1
while n <= 31:
```

[3] Например, в программах, в которых используются списки (массивы).

[4] Соответствующее условие будет противоположным условию окончания работы цикла **`for`**.

```
    print(n * n)
    n = n + 2
```

можем написать в программе:

```
for n in range(1, 32, 2):
    print(n * n)
```

Итак, схему, иллюстрирующую возможность замены одной инструкции цикла на другую, можно представить в виде:

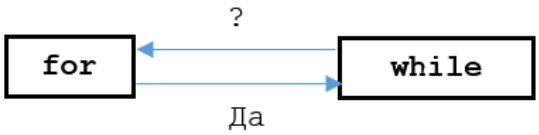


Рис. 3

Вывод: в некоторых случаях для реализации повторяющихся действий в программе может быть использована любая из двух разновидностей инструкции цикла (**for** или **while**), в некоторых – только одна (инструкция **while**).

Примечание. Сделанный вывод справедлив для случая, когда в инструкции for записана функция `range()`. При других вариантах оформления инструкции **for** заменить ее на инструкцию **while** нельзя.

Внимание! Начинающие программисты, заменяя инструкцию **for** на инструкцию **while** (или только оформляя тело инструкции **while**), часто забывают менять значение величины, от которой зависит заданное в инструкции условие. Например, если в программе вывода на экран приветствия

```
for k in range(10):
    print('Привет!')
```

вместо инструкции **for** использовать инструкцию **while** и забыть увеличить переменную *k* в теле цикла:

```
k = 0
while k < 10:
    print('Привет!')
```

то инструкция **while** будет выполняться бесконечно долго (условие, записанное в ней, никогда не станет ложным). В этом случае говорят, что «программа зациклилась».

Правильный вариант:

```
k = 0
while k < 10:
    print('Привет!')
    k = k + 1
```

Контрольные вопросы

1. Всегда ли можно инструкцию цикла **for** заменить на инструкцию **while**? А наоборот?
2. Что такое «зацикливание программы». В каком случае оно может произойти?

Задания на разработку программ

18. Имеется фрагмент программы, обеспечивающий вывод на экран «столбиком» всех целых чисел от 100 до 80, в виде инструкции **for**. Оформите этот фрагмент с использованием инструкции **while**.

19. Имеется фрагмент программы:

```
n = 21
```

```
while n <= 151:
    print(2 * n)
    n = n + 10
```

Оформите соответствующий фрагмент с использованием инструкции **for**.

20. Имеется фрагмент программы:

```
n = 2
while n <= 12:
    print(n)
    n = n + 0.5
```

Оформите соответствующий фрагмент с использованием инструкции **for**.

Комментарий к выполнению. Числа 0, 0.5, 1.0, 1.5, …, 10.0 можно представить как $0 \cdot 0.5$, $1 \cdot 0.5$, $2 \cdot 0.5$, $3 \cdot 0.5$, …, $20 \cdot 0.5$.

Дополнение 2 «“Обобщающие” задачи»

Рассмотрим ряд задач, в программах решения которых обобщаются вопросы, рассмотренные в данном разделе.

Задача 1. Дано целое число *n*, не меньшее 2. Найти все его натуральные делители, не превышающие числа *m* ($m \le n$).

Решение

Используем инструкцию **for** и рассмотрим все числа *vdel*, которые могут делителем числа *n*. Если встретится делитель, меньший или равный *m*, то выводим его на экран. Соответствующая программа:

```
#Способ 1
n = int(input('n = '))
m = int(input('m = '))
for vdel in range(1, n + 1):
    if n % vdel == 0:    #Встретился делитель числа n
        #Сравниваем его с m
        if vdel <= m:
            print(vdel)
```

Можно две инструкции if объединить, записав сложное условие (см. предыдущий выпуск нашего журнала):

```
for vdel in range(2, n + 1):
    if n % vdel == 0 and vdel <= m:
        print(vdel)
```

Недостатком первого способа решения задачи является то, что проверки чисел проводятся даже после нахождения всех требуемых делителей. Этот недостаток можно устранить, используя инструкцию `break` в случае, когда проверяемое число *vdel* станет больше *m*:

```
#Способ 2
…
for vdel in range(1, n + 1):
    if n % vdel == 0 and vdel <= m:
        print(vdel)
    if vdel > m:
        break
```

Можно также рассмотреть только числа *vdel*, не большие *m*, применив вместо инструкции цикла **for** инструкцию **while** (нам известно, какие числа *vdel* надо проверять, то

есть условие повторения действий). Как заменить инструкцию **for** на инструкцию **while**, описано выше.

```
#Способ 3
…
vdel = 1
while vdel <= m:
    if n % vdel == 0:
        print(vdel)
    vdel = vdel + 1
```

Задача 2. Дано целое число *n*, не меньшее 2. Найти его наименьший натуральный делитель, отличный от 1.

Решение

Начнем с использования инструкции **for** – рассмотрим все числа, которые могут быть делителем числа *n*. Так как нам нужно найти первый делитель, больший 1, то введем в программу переменную *vstr* логического типа или принимающую значения 0 или 1, которая будет определять, встретился ли уже нужный нам делитель (чтобы не выводить следующих).

Сначала:

```
vstr = False   #Или    vstr = 0
```

Если встретится делитель числа *n*, то следует с помощью переменной *vstr* проверить, является ли он первым (минимальным). Если является, то нужно:

1) вывести его на экран;

2) переменной vstr присвоить значение **True**:

```
vstr = True #(минимальный делитель найден)
```

Итак, программа:

```
#Способ 1
n = int(input('n = '))
vstr = False   #Или      vstr = 0
#Рассматриваем все числа, большие 1,
#которые могут быть делителем числа n
for vdel in range(2, n + 1):
    #Проверяем, является ли число vdel делителем числа n
    if n % vdel == 0: #Если является,
        #проверяем - это первый делитель? (До этого не было?)
        #Если не было
        if vstr == False:     #Или     if vstr == 0:
            #Встретился искомый делитель
            #Выводим его на экран
            print(vdel)
            #Искомый делитель найден –
            #меняем значение переменой vstr
            vstr = True       #Или vstr = 1
```

Недостатком описанного способа является то, что проверки чисел проводятся даже после нахождения нужного делителя. Этот недостаток можно устранить, заменив инструкцию **for** на инструкцию **while**, а в качестве условия продолжения цикла использовать условие vstr == **False**:

```
#Способ 2
n = int(input('n = '))
vdel = 2          #Начальные
vstr = False      #значения
while vstr == False:
```

```
    if n % vdel == 0:
        print(vdel)
        vstr = True
    #Переходим к следующему числу
    vdel = vdel + 1
```

При решении задачи третьим способом используется аналог инструкции цикла с постусловием (см. выше):

```
#Способ 3
n = int(input('n = '))
vdel = 2
while True:
    if n % vdel == 0:
        print(vdel)
        break
    vdel = vdel + 1
```

Инструкцию `break` можно использовать и в теле инструкции **`for`** (см. первый способ решения) после нахождения искомого делителя:

```
#Способ 4
n = int(input('n = '))
for vdel in range(2, n + 1):
    if n % vdel == 0:  #Встретился искомый делитель
        print(vdel)
        break          #Выход из цикла
```

Можно также учесть, что у четного числа *n* искомый в задаче делитель равен 2, а у нечетного числа – все делители нечетные. Соответствующие варианты программ разработайте самостоятельно.

Задача 3. Дано целое число *n*, не меньшее 2. Найти его наименьший натуральный делитель, больший заданного числа *m* ($m \le n$).

Здесь отличие от задачи 2 в том, что проверка чисел начинается не с 2, а с $m + 1$.

Обсудим три рассмотренные задачи. В них были известны числа *vdel*, которые надо проверять (то есть известно число повторений действий), или известно условие, при котором надо продолжать или прекратить проверки. Поэтому в программах можно было использовать обе инструкции цикла (**`for`** и **`while`**), при необходимости с инструкцией `break`. Однако так бывает не всегда. Приведем пример.

Задача 4. Вывести на экран все числа последовательности Фибоначчи, не превышающие числа *m*.

Ясно, что в данной задаче, в отличие от задач, предложенных в пункте 1, количество повторений действий по расчету очередного числа последовательности неизвестно. Но известно условие, при котором надо проводить расчеты, – очередное число последовательности не должно быть больше *m*. Значит, можем в программе применить инструкцию **`while`**:

```
m = int(input('m = '))
#Выводим первые 2 числа последовательности
print('1 1 ', end ='')
#Определяем и выводим остальные числа
pred = 1
predpred = 1
while pred + predpred <= m:   #Обратите внимание на условие
```

```
ocher = pred + predpred
print(ocher, end = ' ')
predpred = pred
pred = ocher
```

Так же надо поступать и в других аналогичных случаях.

Возможны также «промежуточные варианты», в которых число повторений (или интервал чисел для расчетов) явно не задано, но его можно рассчитать. Примером такой задачи является задача 4, предложенная на стр. 5. Напомним её: «Напечатать те натуральные числа, квадрат которых не превышает заданного числа *n*».

Подумаем, какое последнее (максимальное) число должно быть выведено. Для ответа составим таблицу:

n	Последнее число	Обоснование
81	9	$9^2 = n$
82	9	$9^2 < n$
99	9	$9^2 < n$
100	10	$10^2 = n$
101	10	$10^2 < n$

Из нее следует, что в общем случае последнее число, квадрат которого не превышает *n*, равно целой части квадратного корня из *n*. Поэтому можем использовать в программе инструкцию **`for`**:

```
import math
n = int(input('n = '))
for k in range(1, int(math.sqrt(n)) + 1):
    print(k)
```

Подумайте, почему использована функция `int`.

Программа решения задачи упрощается (не нужно определять максимальное число для проверки, подключать модуль `math` и использовать функции `sqrt()` и `int()`) и становится логичной, если заменить инструкцию **`for`** на инструкцию **`while`**:

```
n = int(input('n = '))
k = 1
while k * k <= n:
    print(k)
    k = k + 1
```

Аналогично решается задача 5, предложенная на стр. 5:

```
import math
n = int(input('n = '))
for k in range(1, int(math.sqrt(n)) + 1):
    print(k * k)
```

или

```
k = 1
while k * k <= n:
    print(k * k)
    k = k + 1
```

В программах рассматриваются необходимые числа *k*, а на экран выводятся их квадраты.

Задание

Разработайте[5] разные варианты программ решения:

а) задач 7 и 8, предложенных на стр. 5;

б) задач, предложенных в конце дополнения 1.

Другие задачи для разработки программ

21. Имеется монотонно возрастающая последовательность вещественных чисел y, рассчитываемых по закону:

$$y = \frac{(x^2 + 100)}{(x + 200)} \text{ при } 1 \le x \le 100$$ (x – целое число).

Напечатать все числа последовательности, меньшие заданного числа m ($0{,}52 \le m \le 33{,}7$).

22. Имеется последовательность значений:

$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \ldots$

Напечатать в виде вещественных чисел все числа этой последовательности, меньшие заданного числа m ($0{,}5 < m < 1$).

Книжная полка

Учебник по языку программирования Python

Издательство ДМК Пресс предлагает книгу «Основы программирования на языке Python. Второе издание» https://dmkpress.com/catalog/computer/programming/python/978-5-97060-641-4/)

Эта книга для тех, кто хочет научиться программировать на языке программирования Python. Ее отличие в том, что в ней главное – не описание языка программирования, которое представлено в большинстве книг по программированию. В данной книге рассматриваются особенности разработки программ, описываются методы решения типовых задач, встречающихся при разработке, распространенные алгоритмы, даются советы и т. п. Это учебник, в котором в доступной форме излагаются вопросы, с которыми сталкивается человек, начинающий осваивать этот непростой, но захватывающий и очень интересный процесс – программирование. В первой части книги (главы 1–6) рассматриваются особенности разработки компьютерных программ и соответствующие инструкции языка Python. После их изучения читатель сможет разработать ряд простейших игр, описанных в главе 7. В завершающей части книги (главы 8–16) рассматриваются основные структуры данных языка Python (строки, списки, словари), вопросы, связанные с работой с файлами, а также с совершенствованием программ на основе использования функций. По всем темам, рассмотренным в книге, приводятся большое число примеров и задач, подробная методика их решения и соответствующие программы с комментариями. Дается ряд полезных советов. В приложении 2 описаны особенности разработки программ на языке Python с графическим пользовательским интерфейсом. В приложении 3 приводятся ответы к заданиям и программы решения задач, предложенных в книге для самостоятельной работы.

Через сайт издательства её можно приобрести (даже с учётом стоимости доставки) гораздо дешевле, чем в магазинах.

[5] Если, конечно, вы еще не сделали этого…

Методика перевода целых недесятичных чисел в десятичную систему счисления

При выполнении заданий ЕГЭ и ОГЭ по информатике часто требуется проводить перевод недесятичных чисел в десятичную систему счисления. Так как на экзаменах можно пользоваться компьютером, то указанную задачу можно решать, используя электронную таблицу Microsoft Excel или аналогичную программу.

Возможны два способа такого перевода:

1) с использованием так называемой «развёрнутой формы записи чисел»;

2) по так называемой «схеме Горнера».

Применительно к десятичным числам развёрнутую форму записи можно проиллюстрировать следующими примерами:

$29 = 2 \times 10 + 9 \times 1 = = 2 \times 10^1 + 9 \times 10^0$

$506 = 5 \times 100 + 0 \times 10 + 5 \times 1 = 5 \times 10^2 + 0 \times 10^1 + 6 \times 10^0$

$2173 = 2 \times 1000 + 1 \times 100 + 7 \times 10 + 3 \times 1 = 2 \times 10^3 + 1 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$.

Значения 1 (10^0), 10 (10^1), 100 (10^2) и т. д. называют «весом», или «весомостью», того или иного разряда числа. Видно, что весомость разряда в данном случае равна 10^n, где n — номер разряда при условии, что разряды нумеруются справа налево, начиная с нуля.

Аналогичным образом развёрнутую форму записи можно применять и к недесятичным числам.

Перевод целого числа из системы счисления с основанием p в десятичную систему, основанный на развёрнутой форме записи, можно провести следующим образом.

Пусть $p = 2$, исходное число равно 110101_2.

1. Пронумеруем разряды заданного числа справа налево, начиная с нуля:

1	1	0	1	0	1
5	*4*	*3*	*2*	*1*	*0*

2. Начиная с крайнего правого разряда, выпишем числа 1, 2, 4, … (очередное число равно удвоенному «предыдущему»):

1	1	0	1	0	1
5	*4*	*3*	*2*	*1*	*0*
32	**16**	**8**	**4**	**2**	**1**

3. Сложим те из чисел, оформленных полужирным начертанием, для которых соответствующая цифра заданного числа равна 1:

$32 + 16 + 4 + 1 = 53$.

Полученная сумма и будет равна искомому числу. Итак, $110101_2 = 53_{10}$.

Примечание. Так как числа 1, 2, 4, … это весомости разрядов (см. выше), то можно сказать, что для перевода двоичного числа в десятичное необходимо сложить весомости тех двоичных разрядов, в которых записана цифра 1.

Приведём ещё пример.

Пусть $p = 4$, а исходное число равно 1302_4.

1. Пронумеруем разряды заданного числа справа налево, начиная с нуля:

1	3	0	2
3	*2*	*1*	*0*

2. Начиная с крайнего правого разряда, запишем числа 1, 4, 16, … (очередное число в 4 раза больше «предыдущего»):

1	3	0	2
3	*2*	*1*	*0*
64	**16**	**4**	**1**

Можно считать, что числа, оформленные полужирным начертанием, есть весомости разрядов в четверичной системе счисления ($1 = 4^0$, $4 = 4^1$, $16 = 4^2$, …).

3. Искомый результат получить как сумму произведений всех цифр заданного числа на весомость соответствующего разряда:

$1 \times 64 + 3 \times 16 + 0 \times 4 + 2 \times 1 = 114_{10}$.

Последнее правило можно распространить на перевод в десятичную систему счисления целого *p*-ичного числа при любом основании *p*, равном от 2 до 9.

При основаниях *p*, больших 10, нужно каждую цифру заданного числа представить в виде десятичного числа, после чего определить указанную сумму произведений.

Пример. Перевести шестнадцатеричное число B0F9 в десятичную систему.

Решение. $B0F9_{16} = [11_{10}][0][15_{10}][9] = 11 \times 16 + 15 \times 16 + 9 = 45305_{10}$.

Оформим лист программы Microsoft Excel, на котором можно переводить в десятичную систему целые недесятичные числа по описанной методике.

Лист будет иметь следующий вид[6]:

	A	B	C	D	E	F	G	H	I
1	Цифры числа								
2	Весомость разрядов								
3	Результат перевода								

Для случая перевода двоичных чисел весомости разрядов можно записать вручную (начиная с последней цифры!):

	A	B	C	D	E	F	G	H	I
1	Цифры числа	1	1	0	1	0	1		
2	Весомость разрядов	31	16	8	4	2	1		
3	Результат перевода								

При других основаниях системы счисления это удобней сделать следующим образом (см. фрагмент листа ниже):

- в строке 2 под последней цифрой записать 1;
- в строке 2 под предпоследней цифрой записать формулу =G2 * …, где вместо многоточия должно быть указано основание системы счисления;
- указанную формулу распространить (скопировать) на остальные ячейки строки 2.

	A	B	C	D	E	F	G	H	I
1	Цифры числа	1	2	0	6	0	3		
2	Весомость разрядов					=G2 * 7	1		
3	Результат перевода								

[6] При выполнении заданий на ЕГЭ по информатике с целью экономии времени тексты в столбце A можно не записывать.

Во всех случаях искомое значение десятичного числа в ячейке B3 определяется с помощью функции СУММПРОИЗВ[7] (с особенностями её оформления ознакомьтесь самостоятельно).

Теперь – об использовании схемы Горнера.

Пусть, например, надо перевести в десятичную систему счисления пятеричное число 31402.

Запишем в строке 1 цифры заданного числа и его первую цифру повторим в строке 2:

	A	B	C	D	E	F	G	H	I
1	Цифры числа	3	1	4	0	2			
2		3							

В ячейке C2 запишем формулу =B2 * 5 + C1, которую распространим (скопируем) на остальные ячейки строки 2:

	A	B	C	D	E	F	G	H	I
1	Цифры числа	3	1	4	0	2			
2		3	16	84	420	2102			

Последнее число строки 2 будет искомым десятичным значением.

Можно также в ячейку B2 первую цифру не вводить, а записать в ней формулу =A2 * 5 + B1, которую также распространить вправо.

По сути, вычисления по схеме Горнера представляют собой другой порядок расчётов промежуточных значений по развёрнутой форме записи (см. выше). Если исходного цифры числа, начиная с последней, обозначить a_0, a_1, , …, a_n, то развёрнутая форма будет иметь вид [1–2]:

$$a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p + a_0$$

где p – основание системы счисления.

Это выражение можно представить следующим образом:

$$a_0 + p(a_1 + p(a_2 + \ldots + p(a_{n-1} + pa_n)\ldots))$$

Именно по этой формуле (начиная с pa_n) проводились наши расчёты (убедитесь в этом!).

Предлагаем читателям сравнить два описанных метода перевода и выбрать лучший…

Дополнение (не связанное с электронными таблицами, но полезное)

Перевод в десятичную систему чисел, представленных в двоичной, восьмеричной и шестнадцатеричной системах счисления[8], удобно проводить с помощью стандартной программы Калькулятор операционной системы Windows в режиме Программист[9]:

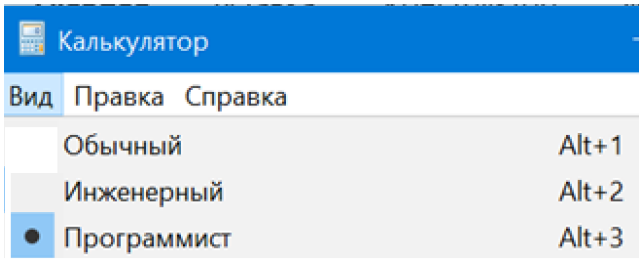


Рис. Д1

[7] В других программах – электронных таблицах используется соответствующая функция.

[8] А также – обратный перевод!

[9] Напомним, что Единый государственный экзамен по информатике проводится в компьютерной форме.

После выбора режима нужно в появившемся окне установить переключатель, например, в пункт Oct (соответствующий восьмеричной системе), после чего в поле ввода ввести восьмеричное число (рис. Д2) и установить переключатель в пункт Dec – в поле ввода отразится искомое десятичное число (рис. Д3).

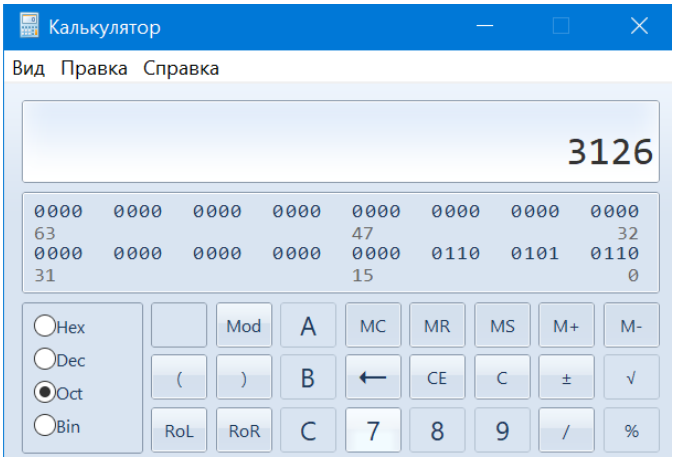


Рис. Д2

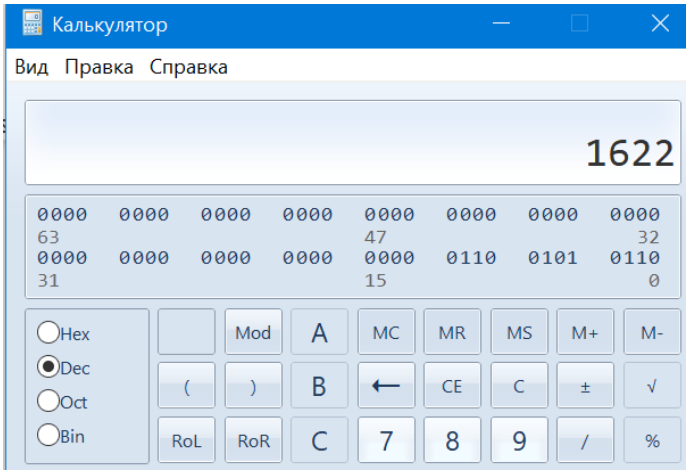


Рис. Д3

Литература

1. Андреева Е.В., Босова Л.Л., Фалина И.Н. Математические основы информатики. М.: Бином. Лаборатория знаний, 2005.

2. Схема Горнера // Википедия. https://ru.wikipedia.org/wiki_Схема_Горнера

Методика выполнения заданий 9 демонстрационных вариантов ОГЭ по информатике

1. Общие вопросы

В Спецификации контрольных измерительных материалов для проведения в 2024 году основного государственного экзамена (ОГЭ) по информатике [1] в качестве проверяемого результата обучения применительно к заданию 9 указывается «Умение анализировать информацию, представленную в виде схем».

Тема в федеральном компоненте ГОС ООО – «Формализация описания реальных объектов и процессов, моделирование объектов и процессов».

Уровень сложности – повышенный.

Максимальный балл за задание – 1.

Примерное время выполнения задания (мин.) – 4.

В демонстрационных вариантах контрольных измерительных материалов ОГЭ по информатике нескольких последних лет и в Открытом банке заданий ОГЭ заданий задания 9 имели следующий общий вид:

1) «На рисунке <*приводился рисунок*> представлена схема дорог, связывающих города <*приводился перечень городов*>. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города <*указывался город*> в город <*указывался город*>?»

или

2) «На рисунке <*приводился рисунок*> представлена схема дорог, связывающих города <*приводился перечень городов*>. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города <*указывался город*> в город <*указывался город*>, проходящих через город <*указывался город*>?».

Задание первого вида условно будем называть «задания типа 1», второго – «задания типа 2».

2. Один из методов выполнения задания

Указанные задания могут быть выполнены различными способами. Один из них, по нашему мнению – эффективный, заключается в следующем [2].

Представьте, что однажды Чеширскому Коту[10] нужно было подсчитать количество маршрутов, которыми он сможет добраться из пункта A в пункт H на графе (см. раздел 4), схема которого показана на рис. 1.

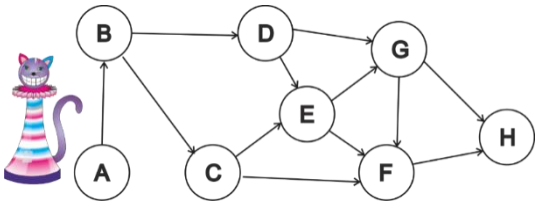


Рис. 1

Чтобы многократно не ходить к цели и обратно, Кот придумал оригинальный метод подсчета. Он отправил из пункта A по маршруту… свою копию. Далее он поступал так.

[10] Чеширский Кот – персонаж книги Льюиса Кэрролла «Алиса в Стране чудес».

В тех пунктах, в которые приходит одна дорога, по которой пришла одна копия, а выходят две дороги (например, в пункте B), он добавлял ещё одну свою копию, каждая из которых шла по своей дороге:

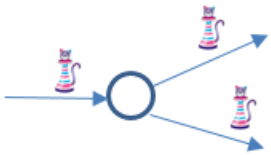


Рис. 2

Если же в таком случае выходят три и более дорог, то и число копий увеличивается соответственно (см. рис. 3).

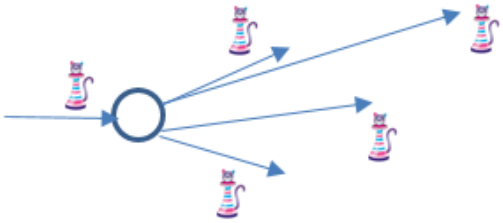


Рис..3

Когда в какой-то пункт пришли две копии нашего «умника», а выходящих дорог тоже две (см. рис. 4), то и число копий удваивается, и каждая пара идет по своей дороге.

Рис..4

В общем случае, когда в некоторый пункт пришли (по всем дорогам) *n* копий котов, а из него выходят несколько дорог, то по каждой из них должны уйти по *n* копий.

При таком решении каждый кот-копия будет идти своей дорогой и не будет котов, прошедших по одному и тому же маршруту. А число котов, пришедших в точку H, и будет равно искомому числу маршрутов.

Итак, пункт A – начало маршрута, из которого кот отправил в путешествие свою первую копию. Из A выходит единственная дорога, которая приведет ее в пункт B. Далее дороги расходятся, и из пункта B далее выйдут два кота – один отправится в направлении пункта C, другой – в пункт D. В каждом из этих пунктов произойдет то же самое – число выходящих копий удвоится. Поэтому в пункт E придут два кота, а выйдут – $2 \times 2 = 4$ (два из них отправятся в пункт G и два кота – в пункт F). В результате в пункт G придут два кота, вышедшие из E, и один кот из пункта D. Запишем на графе количество котов, пришедших в тот или иной пункт. У первого пункта (на «старте») стоит единица, а далее числа получаются в результате сложения чисел, соответствующих пунктам, из которых можно попасть в данный (см. рис. 5).

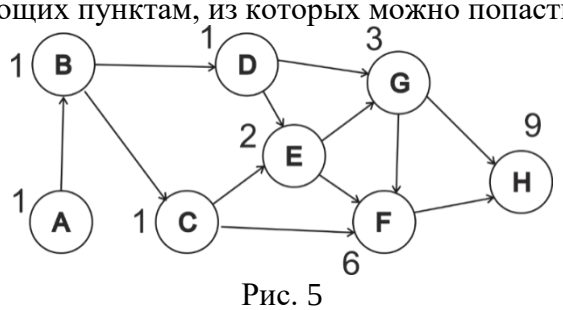


Рис. 5

Из рис. 5 следует, что к финишу придут 9 котов-копий, то есть искомое количество маршрутов перемещения из пункта A в пункт H равно 9**.**

Можно также, начиная с первого пункта, записывать число копий, пришедших в тот или иной пункт, в таблицу:

Пункт	A	B	C	D	E	F	G	H
Число копий	1	1	1	1	2	…		

Можно также записывать на графе, на его «ребрах» (в данном случае – на дорогах, соединяющих пункты), число, равное количеству копий котов, идущих по данной дороге – см. рис. 6. Обратите внимание – в пункт E пришли 2 кота (из пунктов C и D), поэтому на дорогах, выходящих из этого пункта, записаны числа 2. Аналогично определяется число котов на дорогах из пунктов G (1 +2 = 3) и F (1 + 2 + 3 = 6). Искомое число будет равно сумме чисел на дорогах, ведущих в пункт H, то есть также 9. Какой метод удобнее – решать вам.

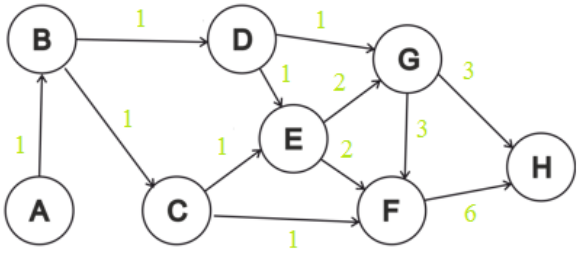


Рис. 6

Конечно, можно «забыть» о Чеширском Коте и записывать не количество копий Кота, а «обычные», но соответствующие, числа.

Теперь о заданиях второго типа, указанного в начале раздела. Его отличие в том, что в путях, количество которых должно быть определено, должен обязательно присутствовать заданный «промежуточный» город.

В этом случае задача разбивается на две подзадачи:

1) определения количества путей, ведущих из исходного города в «промежуточный»;

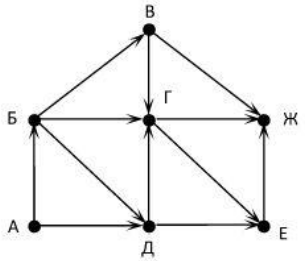
2) определения количества путей, ведущих из «промежуточного» города в пункт назначения.

Если первое количество равно N_1, а второе – N_2, то искомое в основной задаче число путей равно $N_1 \times N_2$ (каждый из путей первой подзадачи можно объединить с каждый из путей второй подзадачи).

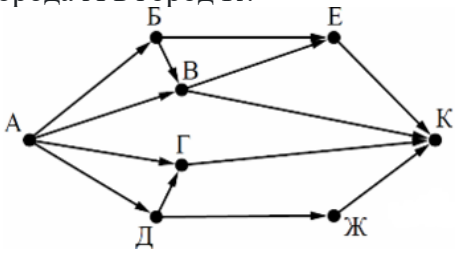
Аналогично решаются задачи, в которых фигурируют два и более «промежуточных» города.

3. Задания для самостоятельной работы

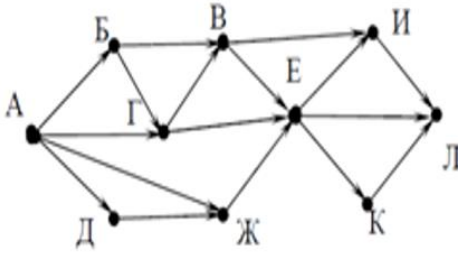
1. На рисунке приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город Ж?



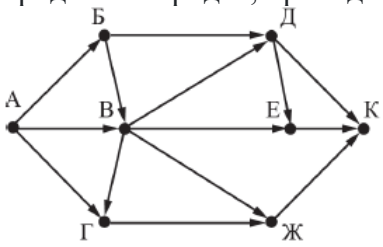
2. На рисунке приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж и К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?



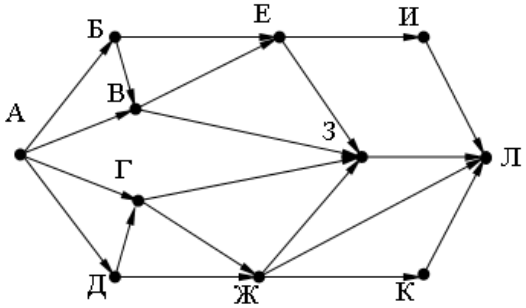
3. На рисунке приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К и Л. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город Л?



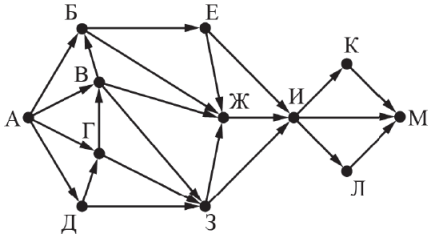
4. На рисунке представлена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К, проходящих через город В?



5. На рисунке представлена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К, Л. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город Л, проходящих через город Ж?



6. На рисунке представлена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К, Л, М. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город М, проходящих через город Ж?



Литература

1. Спецификация контрольных измерительных материалов для проведения в 2022 году основного государственного экзамена по информатике / ФГБНУ «Федеральный институт педагогических измерений». ОГЭ. Демоверсии, спецификации, кодификаторы 2022. Информатика и ИКТ. https://fipi.ru/oge/demoversii-specifikacii-kodifikatory#!/tab/173801626-5

2. Мирончик Е.А. Чеширский Кот путешествует по графу // Первое сентября. Информатика, 2015, № 10.

Внимание! Конкурс!

Конкурс № 67 «Отмеривание воды»

В качестве заданий этого конкурса, который будет проходить *в несколько туров*, а его итоги будут подводиться *с учётом всех туров в целом*, будут предложены задания следующего типа.

Представьте себе, что имеется исполнитель, которого назовем Водомером, так как он занимается отмериванием того или иного количества воды. У него есть источник воды (река, озеро или т. п.), количество воды в котором неограниченно, и две или три ёмкости (банки, ведра или т. п.): A и B (или A, B и C). При двух ёмкостях система команд, которые может выполнять Водомер, такая:

К1. Наполнить A
К2. Наполнить B
К3. Перелить из A в B
К4. Перелить из B в A
К5. Вылить из A
К6. Вылить из B

при трёх:

К1. Наполнить A
К2. Наполнить B
К3. Наполнить C
К4. Перелить из A в B
К5. Перелить из A в C
К6. Перелить из B в A
К7. Перелить из B в C
К8. Перелить из C в A
К9. Перелить из C в B
К10. Вылить из A
К11. Вылить из B
К12. Вылить из B

Составьте алгоритмы решения следующих задач, которые должен решить Водомер. Если иное не оговорено особо, задача должна решаться за минимально возможное количество действий[11].

[11] Обращаем внимание на то, что среди предложенных задач имеются задачи с одинаковыми исходными данными и требуемым результатом, но отличающиеся условиями получения результата (минимальное число операций или минимально возможное количество используемой воды — см. комментарий к задаче 9).

Алгоритм решения задач, пожалуйста, оформите в виде таблицы (приведены условные значения):

№ пп	Обозначение команды	В чём она заключается	Станет объём воды в ёмкости	
			A	B
Сначала			0	0
1	K1	Наполнить A	5	0
2				
3				
4				

Тур 1

1. Объем емкости A равен 3 л, емкости B — 4 л, надо отмерить 2 л.

2. Объем емкости A равен 3 л, емкости B — 10 л, надо отмерить 5 л.

3. Объем емкости A равен 4 л, емкости B — 10 л, надо отмерить 2 л.

4. Объем емкости A равен 5 л, емкости B — 10 л, емкости C — 12 л, надо отмерить 4 л.

5. Объем емкости A равен 5 л, емкости B — 10 л, емкости C — 12 л, надо отмерить 1 л.

6. Объем емкости A равен 4 л, емкости B — 5 л, емкости C — 10 л, надо отмерить 7 л.

7. Объем емкости A равен 3 л, емкости B — 5 л, емкости C — 8 л, надо отмерить 7 л.

8. Объем емкости A равен 3 л, емкости B — 5 л, емкости C — 10 л, надо отмерить 9 л.

9. Объем емкости A равен 3 л, емкости B — 5 л, емкости C — 11 л, надо отмерить 9 л, использовав при этом минимально возможное количество воды (здесь и в аналогичных задачах далее имеется в виду общий объем воды, используемой при выполнении команд Заполнить A, Заполнить B, Заполнить C).

10. Объем емкости A равен 3 л, емкости B — 6 л, емкости C — 11 л, надо получить 4 л в емкости C.

Ответы присылайте по адресу: mirinfo@infojournal.ru. Укажите в письме свои фамилию и имя, населенный пункт, номер школы и фамилию, имя и отчество учителя информатики. Срок представления ответов — 30 сентября 2024 г.

Уважаемые читатели!

Сообщаем, что очередные номера нашего интернет-журнала публикуются в период с 15 по 20 число каждого месяца, кроме июля.