Задачи с решениями
(8-10 класс)

Задача 1. Подбор пароля

Вариант 1

Для входа в систему используется пароль, который состоит из трех двузначных чисел. Двузначные числа образуют между собой примитивную пифагорову тройку (сумма квадратов двух чисел равна квадрату третьего числа, каждое из чисел натуральное, числа взаимно простые). Известно, что *первым* всегда стоит *наименьшее* двузначное число.

Сколько максимально времени потребуется для подбора пароля, если ввод пароля занимает 1 секунду, а задержка между вводом паролей составляет 1 секунду? Ответ дать в виде числа секунд.

Решение

Необходимо найти количество троек чисел (a, b, c) таких, что $c^2 = a^2 + b^2$ или $b^2 = a^2 + c^2$, при этом, $10 \le a,b,c \le 99$. Известно, что первое число наименьшее, то есть $a < b$ и $a < c$.

Числа a,b,c должны быть взаимнопростыми, то есть НОД(a,b) = НОД(b,c) = НОД(a,c) = 1.

Для реализации функции НОД (наибольший общий делитель) можно воспользоваться алгоритмом Евклида: вычитать из большего числа меньшее, пока они не сравняются. Результатом и будет НОД.
Пример реализации функции НОД на языке программирования С приведен в листинге 1.1-1.

Листинг 1.1-1 – Реализация функции НОД на языке программирования С

```
/// поиск НОД для 2-х чисел
/// PARAMS
///    a - первое число
///    b - второе число
/// RETURN
///    вычисленный НОД
int NOD(int a, int b)
{
  while (a != b)
  {
    if (a > b)
      a = a - b;
    else
      b = b - a;
  }
  return a;
}
```

Общий цикл перебора чисел и поиска требуемых троек на языке программирования С++ приведен в листинге 1.1-2.

Листинг 1.1-2 – Реализация цикла перебора на языке программирования С++

```
int main()
{
  int a, b, c; // искомые числа
  int count = 0; // счетчик числа найденных троек

  for (a = 10; a < 100; a++)
    for (b = 10; b < 100; b++)
      for (c = 10; c < 100; c++)
      {
        if (
          // условие на пифагоровы тройки
          ( c * c == a * a + b * b || b * b == a * a + c * c )
          // условие на первое минимальное число
          && a < b && a < c
          // условие на взаимнопростые числв
```

```
            && NOD(a,b) == 1 && NOD(b,c) == 1 && NOD(a,c) == 1
             )
          {
            count++; // увеличение счетчика на 1
            // вывод на экран найденной тройки
            cout << count << " - (" << a << "," << b << "," <<c<<")"<< endl;
          }
        }
}
```

В результате программа выведет следующие комбинации:

```
1 - (11,60,61)
2 - (11,61,60)
3 - (12,35,37)
4 - (12,37,35)
5 - (13,84,85)
6 - (13,85,84)
7 - (16,63,65)
8 - (16,65,63)
9 - (20,21,29)
10 - (20,29,21)
11 - (28,45,53)
12 - (28,53,45)
13 - (33,56,65)
14 - (33,65,56)
15 - (36,77,85)
16 - (36,85,77)
17 - (39,80,89)
18 - (39,89,80)
19 - (48,55,73)
20 - (48,73,55)
21 - (65,72,97)
22 - (65,97,72)
```

Под условия задачи подходит 22 комбинации. Учитывая, что время ввода комбинации 1 секунда и задержка между соседними комбинациями 1 секунда, общее время, требуемое на ввод всех комбинаций T = 22 + 21 = 43 секунды (после ввода последней комбинации нет задержки).

Ответ: 43 секунды.

Вариант 2

Для входа в систему используется пароль, который состоит из трех двузначных чисел. Двузначные числа образуют между собой примитивную пифагорову тройку (сумма квадратов двух чисел равна квадрату третьего числа, каждое из чисел натуральное, числа взаимно простые). Известно, что *первым* всегда стоит *наибольшее* двузначное число.

Сколько максимально времени потребуется для подбора пароля, если ввод пароля занимает 1 секунду, а задержка между вводом паролей составляет 1 секунду? Ответ дать в виде числа секунд.

Решение

Необходимо найти количество троек чисел (a, b, c) таких, что $a^2 = b^2 + c^2$, при этом, $10 \le a,b,c \le 99$. Известно, что первое число наибольшее, то есть a > b и a > c.

Числа a,b,c должны быть взаимнопростыми, то есть НОД(a,b) = НОД(b,c) = НОД(a,c) = 1.

Для реализации функции НОД (наибольший общий делитель) можно воспользоваться

алгоритмом Евклида: брать остаток от деления большего числа на меньшее, пока не будет нулевой остаток. Результатом и будет НОД.

Пример реализации функции НОД на языке программирования Python приведен в листинге 1.2-1.

Листинг 1.2-1 – Реализация функции НОД на языке программирования Python

```
##
# поиск НОД для 2-х чисел
# PARAMS
#   a - первое число
#   b - второе число
# RETURN
#   вычисленный НОД
##
def NOD(a: int, b: int):
  while a != 0 and b != 0:
    if a > b:
      a = a % b
    else:
      b = b % a
  return a + b
```

Общий цикл перебора чисел и поиска требуемых троек на языке программирования Python приведен в листинге 1.2-2.

Листинг 1.2-2 – Реализация цикла перебора на языке программирования Python

```
# счетчик числа найденных троек
cnt = 0

for a in range(10, 100):
  for b in range(10, 100):
    for c in range(10, 100):
      # условие на первое максимальное число
      if a > b and a > c:
        # условие на пифагорову тройку
        if a * a == b * b + c * c:
          # условие на взаимнопростые числа
          if NOD(a, b) == 1 and NOD(a, c) == 1 and NOD(b, c) == 1:
            print(a, b, c)
            cnt += 1 # увеличение счетчика найденных троек

# вывод общего числа троек
print('Total=' + str(cnt))
```

В результате программа выведет следующие комбинации:

```
29 20 21
29 21 20
37 12 35
37 35 12
53 28 45
53 45 28
61 11 60
61 60 11
65 16 63
65 33 56
65 56 33
65 63 16
73 48 55
73 55 48
```

```
85 13 84
85 36 77
85 77 36
85 84 13
89 39 80
89 80 39
97 65 72
97 72 65
Total=22
```

Под условия задачи подходит 22 комбинации. Учитывая, что время ввода комбинации 1 секунда и задержка между соседними комбинациями 1 секунда, общее время, требуемое на ввод всех комбинаций T = 22 + 21 = 43 секунды (после ввода последней комбинации нет задержки).

Ответ: 43 секунды.

Задача 2. Несанкционированный вход

Вариант 1

Администратору был предоставлен журнал аудита входа в информационную систему. Формат записи журнала:

`Логин ТипОперации Время`

`ТипОперации` принимает одно из значений: Вход или Выход.

`Время` записано в формате *ДД.ММ.ГГГГ ЧЧ:ММ:СС*.

Каждая запись начинается с новой строки. В качестве разделителя между полями записи используется знак табуляции (`TAB`).

Система подразумевает, что при корректной работе у пользователей не может быть более одной открытой сессии.

Известно, что была попытка несанкционированного подключения к системе с использованием логина одного из пользователей.

Определите время несанкционированного подключения и логин скомпрометированного пользователя.

К задаче прилагается:

файл журнала audit_v1.log.txt.

Решение

Под несанкционированным входом подразумевается ситуация, когда у одного пользователя более одного открытого сеанса. То есть, в журнале присутствует две записи «Вход», между которыми нет записи «Выход» для одного пользователя.

Для упрощения поиска таких записей существует 2 подхода: использовать MS Excel с настроенными фильтрами и сортировкой или написать программу, которая в автоматическом режиме будет осуществлять разбор записей журнала и определение пользователей с несколькими открытыми сеансами.

Пример функционирования такой программы:

1) Из файла журнала берется очередная запись (строка).
2) Из строки выделяется *Логин* и *ТипОперации* (по разделительным символам).
3) Если *ТипОперации* – «*Вход*», добавляем Логин в массив пользователей с открытым сеансом. Если такой пользователь в массиве уже есть – обнаружен несанкционированный вход.
4) Если *ТипОперации* – «*Выход*», удаляем Логин из массива пользователей с открытым сеансом.

Пример реализации программы на языке программирования C++ приведен в листинге 2.1-1.

Листинг 2.1-1 – Реализация поиска несанкционированной записи в журнале на языке

программирования C++

```
#include <locale>
#include <iostream>
#include <fstream>
#include <string>
#include <map>
using namespace std;
int main()
{
    // открытие файла на чтение
    ifstream fin("audit_v1.log.txt");

    string line;
    int tabInd, tabInd2;
    string login, action;

    // словарь открытых сеансов
    map<string, string> sessions;

    // в цикле считывание очередной строки,
    // пока не конец файла
    while (!getline(fin, line).eof())
    {
        // индекс первого разделителя (для поля Логин)
        tabInd = line.find('\t');
        // индекс второго разделителя (между ними типОперации)
        tabInd2 = line.find('\t', tabInd+1);
        // копируем Логин из строки журнала
        login = line.substr(0,tabInd);
        // копируем ТипОперации из строки журнала
        action = line.substr(tabInd + 1, tabInd2 - tabInd - 1);

        // поиск в словаре по Логину
        auto it = sessions.find(login);
        // Запись в словаре не найдена - добавляем новую
        if (it == sessions.end())
            sessions[login] = action;
        else
        // запись в словаре найдена
        {
            // если сохраненный ТипОперации совпадает с текущей - ошибка!
            // вывод на экран информации о несанкционированном доступе
            if (sessions[login] == action)
                cout << "Bad login: " << line << endl;
            else
                // иначе удаляем запись из словаря
                sessions.erase(it);
        }
    } // while
}
```

В результате программа выведет следующий результат:

```
Bad login: user27 Вход 14.01.2021 11:00:24
```

Осуществив поиск в файле журнала по логину *user27*, можно найти следующие записи:

```
user27              Вход 14.01.2021 05:41:54
user27              Вход 14.01.2021 11:00:24
user27              Выход 14.01.2021 13:24:46
user27              Выход 14.01.2021 22:05:28
```

В результате можно сделать вывод, что была попытка несанкционированного подключения от имени пользователя user27 14.01.2021 05:41:54 или 14.01.2021 11:00:24.

Ответ: user27 14.01.2021 05:41:54 или 14.01.2021 11:00:24.

Вариант 2

Администратору был предоставлен журнал аудита входа в информационную систему. Формат записи журнала:

`Логин ТипОперации Время`

`ТипОперации` принимает одно из значений: Вход или Выход.

`Время` записано в формате *ДД.ММ.ГГГГ ЧЧ:ММ:СС*.

Каждая запись начинается с новой строки. В качестве разделителя между полями записи используется знак табуляции (`TAB`).

Система подразумевает, что при корректной работе у пользователей не может быть более одной открытой сессии.

Известно, что была попытка несанкционированного подключения к системе с использованием логина одного из пользователей.

Определите время несанкционированного подключения и логин скомпрометированного пользователя.

К задаче прилагается:

файл журнала audit_v2.log.txt.

Решение

Под несанкционированным входом подразумевается ситуация, когда у одного пользователя более одного открытого сеанса. То есть, в журнале присутствует две записи «Вход», между которыми нет записи «Выход» для одного пользователя.

Для упрощения поиска таких записей существует 2 подхода: использовать MS Excel с настроенными фильтрами и сортировкой или написать программу, которая в автоматическом режиме будет осуществлять разбор записей журнала и определение пользователей с несколькими открытыми сеансами.

Пример функционирования такой программы:

1) Из файла журнала берется очередная запись (строка).
2) Из строки выделяется *Логин* и *ТипОперации* (по разделительным символам).
3) Если *ТипОперации* – «*Вход*», добавляем Логин в массив пользователей с открытым сеансом. Если такой пользователь в массиве уже есть – обнаружен несанкционированный вход.
4) Если *ТипОперации* – «*Выход*», удаляем Логин из массива пользователей с открытым сеансом.

Пример реализации программы на языке программирования Python приведен в листинге 2.2-1.

Листинг 2.2-1 – Реализация поиска несанкционированной записи в журнале на языке программирования Python

```
# словарь открытых сессий
sessions = {}

# открытие файла журнала и считывание содержимого в txt
fi = open("audit_v2.log.txt")
txt = fi.read()
txt = txt.split("\n")

# цикл построчно
for el in txt:
    #  пропуск пустых строчек
    if not el:
```

```
            continue
        # разбиение строки по символу TAB
        el = el.split("\t")
        # el[0] - это Логин
        # el[1] - это ТипОперации
        # убираем лишние пробелы в начале и в конце
        el[0] = el[0].strip()

        # поиск логина (el[0]) в словарь сеансов (sessions)
        # нашли Логин и ТипОперации совпадает - несанкционированная запись
        if el[0] in sessions and sessions[el[0]] == el[1]:
            print('Bad login: ' + ' '.join(el))
        # не нашли Логин - добавляем в словарь сеансов
        elif el[0] not in sessions:
            sessions[el[0]] = el[1]
        # нашли Логин, но ТипОперации не совпадает - удаляем его
        else:
            del sessions[el[0]]
```

В результате программа выведет следующий результат:

```
Bad login: operator16 Вход 10.02.2021 11:56:21
```

Осуществив поиск в файле журнала по логину *operator16*, можно найти следующие записи:

```
operator16          Вход 10.02.2021 10:09:52
operator16          Вход 10.02.2021 11:56:21
operator16          Выход 10.02.2021 15:30:51
operator16          Выход 10.02.2021 16:39:47
```

В результате можно сделать вывод, что была попытка несанкционированного подключения от имени пользователя operator16 10.02.2021 10:09:52 или 10.02.2021 11:56:21.

Ответ: operator16 10.02.2021 10:09:52 или 10.02.2021 11:56:21.

Задача 3. Цифровая панель

Вариант 1

Для передачи сообщения используется цифровая клавиатура (см. рисунок) и следующий алгоритм шифрования: каждый символ кодируется последовательностью из 3-х цифр. При этом, последовательность не может начинаться с 0 и 9, двигаться между клавишами можно только по правилам шахматного коня.

1	2	3
4	5	6
7	8	9
	0	

Рисунок. Цифровая панель

Алфавит какой длины можно использовать при таком алгоритме шифрования?

Решение

Длина алфавита определяется количеством различных комбинаций из трех цифр, которые можно получить из панели с учетом правил и ограничений их построения. Правило шахматного коня определяет порядок перехода от одной цифры к другой (см. рисунок 3.1-1)

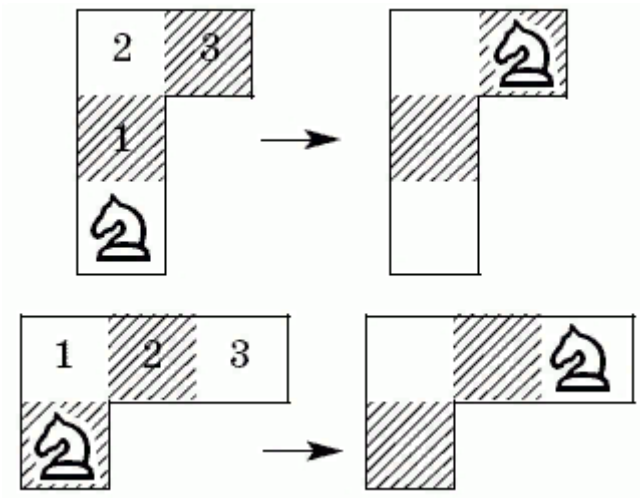


Рисунок 3.1-1 – Правило шахматного коня

Существует два варианта решения:

1. Перебор всех комбинаций вручную.
2. Автоматическое построение комбинаций с использованием графа переходов.

Способ 1.

Для поиска всех возможных комбинаций необходимо построить варианты перехода из каждой цифры клавиатуры. Возможные переходы:

1: 6, 8
2: 9,7
3: 4, 8
4: 0, 3, 9
5: пусто
6: 0, 1, 7
7: 2, 6
8: 1, 3
9: 2, 4
0: 6,4

Далее необходимо построить таблицу со всеми возможными комбинациями

№ комбинации	**1-я цифра**	**2-я цифра**	**3-ф цифра**
1.	0	6	0
2.	0	6	1
3.	0	6	7
4.	0	4	0
5.	0	4	3
6.	0	4	9
7.	1	6	0
8.	1	6	1
9.	1	6	7
10.	1	8	1
11.	1	8	3
12.	2	9	2
13.	2	9	4
14.	2	7	2
15.	2	7	6
16.	3	4	0
17.	3	4	3

18.	3	4	9
19.	3	8	1
20.	3	8	3
21.	4	0	6
22.	4	0	4
23.	4	3	4
24.	4	3	8
25.	4	9	2
26.	4	9	4
27.	6	0	6
28.	6	0	4
29.	6	1	6
30.	6	1	8
31.	6	7	2
32.	6	7	6
33.	7	2	9
34.	7	2	7
35.	7	6	0
36.	7	6	1
37.	7	6	7
38.	8	1	6
39.	8	1	8
40.	8	3	4
41.	8	3	8
42.	9	2	9
43.	9	2	7
44.	9	4	0
45.	9	4	3
46.	9	4	9

Всего 46 комбинаций. В задании указано, что комбинация не может начинаться с 0 (6 комбинаций) и с 9 (5 комбинаций). Ответ: 46–6–5 = 35 комбинаций.

Способ 2.

Необходимо задать матрицу переходов (двумерный массив), после чего осуществить перебор всех возможных трехзначных путей.

Пример задания матрицы на основе словаря на языке программирования C++ представлен в листинге 3.1-1.

Листинг 3.1-1 – Матрица переходов на основе словаря на языке программирования C++

```
// Словарь переходов
// ЦИФРА --> массив возможных переходов
map<int, vector<int>> table = { {0, {4, 6}},     // переходы из 0
                  {1, {6, 8}},                   // переходы из 1
                  {2, {7, 9}},                   // переходы из 2
                  {3, {4, 8}},                   // переходы из 3
                  {4, {3, 9, 0}},                // переходы из 4
                  {5, {}},                       // переходы из 5
                  {6, {1, 7, 0}},                // переходы из 6
                  {7, {2, 6}},                   // переходы из 7
                  {8,{1, 3}},                    // переходы из 8
                  {9, {2, 4}}                    // переходы из 9
};
```

Функцию подсчета числа комбинаций можно сделать рекурсивной. На вход функции подается первая цифра и длина комбинации. Функция рекурсивно вычисляет количество таких

комбинаций на основе матрицы переходов. Реализация функции на языке программирования C++ представлена в листинге 3.1-2.

Листинг 3.1-2 – Реализация функции подсчета числа комбинаций на языке программирования C++

```
/// Функция подсчета числа комбинаций,
/// начинающихся с цифры startNum и длиной len
/// PARAMS
///    startNum - первая цифра комбинации
///    len - длина комбинации
/// RETURN
///    количество комбинаций заданной длины,
///    начинающихся с startNum
int count(int startNum, int len)
{
  // счетчик числа комбинаций
  int cnt = 0;
  // если последовательность нулевой длины - их 0
  if (len == 0)
    return 0;

  // если последовательность длины 1 - их 1 (сама цифра startNum)
  if (len == 1)
    return 1;
  else
  {
    // перебираем все возможные числа, в которые можно перейти
    // из startNum - это указано в массиве table[startNum]
    auto numbersToGo = table[startNum];
    // считаем и складываем количество таких комбинаций,
    // длина которых уже на 1 цифру меньше
    for (int i = 0; i < numbersToGo.size(); i++)
      cnt += count(numbersToGo[i], len - 1);

    return cnt;
  }
}
```

Фрагмент общей программы на языке программирования C++, запускающей в цикле подсчет числа комбинаций, представлен в листинге 3.1-3.

Листинг 3.1-3 – Фрагмент программы подсчета общего числа комбинаций на языке программирования C++

```
int main()
{
  // счетчик комбинаций
  int cnt = 0;
  // перебираем все начальные цифры
  for (auto it = table.begin(); it != table.end(); it++)
  {
    cnt += count(it->first, 3);
  }
  // вычитаем комбинации, начинающиеся с 0 и с 9
  cnt = cnt - count(0, 3) - count(9, 3);
  cout << "Total: " << cnt << endl;
}
```

В результате выполнения программа выдает на экран:

```
Total: 35
```

Ответ: 35.

Вариант 2

Для передачи сообщения используется цифровая клавиатура (см. рисунок) и следующий алгоритм шифрования: каждый символ кодируется последовательностью из 3-х цифр. При этом, последовательность не может начинаться с 1 и 9, двигаться между клавишами можно только по правилам шахматного коня.

1	2	3
4	5	6
7	8	9
	0	

Рисунок. Цифровая панель

Алфавит какой длины можно использовать при таком алгоритме шифрования?

Решение

Длина алфавита определяется количеством различных комбинаций из трех цифр, которые можно получить из панели с учетом правил и ограничений их построения. Правило шахматного коня определяет порядок перехода от одной цифры к другой (см. рисунок 3.2-1)

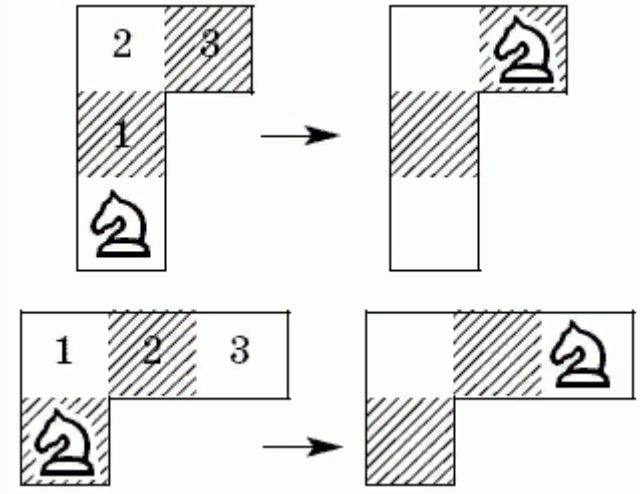


Рисунок 3.2-1 – Правило шахматного коня

Существует два варианта решения:

1. Перебор всех комбинаций вручную.
2. Автоматическое построение комбинаций с использованием графа переходов.

Способ 1.

Для поиска всех возможных комбинаций необходимо построить варианты перехода из каждой цифры клавиатуры. Возможные переходы:

1: 6, 8
2: 9,7
3: 4, 8
4: 0, 3, 9
5: пусто
6: 0, 1, 7
7: 2, 6
8: 1, 3
9: 2, 4
0: 6,4

Далее необходимо построить таблицу со всеми возможными комбинациями

№ комбинации	**1-я цифра**	**2-я цифра**	**3-я цифра**
1.	0	6	0
2.	0	6	1
3.	0	6	7

4.	0	4	0
5.	0	4	3
6.	0	4	9
7.	1	6	0
8.	1	6	1
9.	1	6	7
10.	1	8	1
11.	1	8	3
12.	2	9	2
13.	2	9	4
14.	2	7	2
15.	2	7	6
16.	3	4	0
17.	3	4	3
18.	3	4	9
19.	3	8	1
20.	3	8	3
21.	4	0	6
22.	4	0	4
23.	4	3	4
24.	4	3	8
25.	4	9	2
26.	4	9	4
27.	6	0	6
28.	6	0	4
29.	6	1	6
30.	6	1	8
31.	6	7	2
32.	6	7	6
33.	7	2	9
34.	7	2	7
35.	7	6	0
36.	7	6	1
37.	7	6	7
38.	8	1	6
39.	8	1	8
40.	8	3	4
41.	8	3	8
42.	9	2	9
43.	9	2	7
44.	9	4	0
45.	9	4	3
46.	9	4	9

Всего 46 комбинаций. В задании указано, что комбинация не может начинаться с 1 (5 комбинаций) и с 9 (5 комбинаций). Ответ: 46–5–5 = 36 комбинаций.

Способ 2.

Необходимо задать матрицу переходов (двумерный массив), после чего осуществить перебор всех возможных трехзначных путей.

Пример задания матрицы на основе словаря на языке программирования Python

представлен в листинге 3.2-1.

Листинг 3.2-1 – Матрица переходов на основе словаря на языке программирования Python

```
# матрица переходов
turns = {0: [4, 6],   # переход из 0
      1: [6, 8],      # переход из 1
      2: [7, 9],      # переход из 2
      3: [4, 8],      # переход из 3
      4: [3, 9, 0],   # переход из 4
      5: [],          # переход из 5
      6: [1, 7, 0],   # переход из 6
      7: [2, 6],      # переход из 7
      8: [1, 3],      # переход из 8
      9: [2, 4]       # переход из 9
     }
```

Подсчет числа комбинаций можно реализовать следующим образом:

1) Организовать цикл, в котором перебирать все возможные цифры на первом месте комбинации.
2) Для каждой первой цифры из матрицы переходов получить массив цифр для второй позиции в комбинации.
3) В цикле перебирать цифры для второй позиции, для каждой из которых определять количество цифр для третьей позиции в комбинации. Полученное число добавлять в общий список числа комбинаций.

Пример реализации такой программы на языке программирования Python представлен в листинге 3.2-2.

Листинг 3.2-2 – Реализации программы подсчета общего числа комбинаций на языке программирования Python

```
# счетчик комбинаций
cnt = 0
# сама комбинация
combo = [0, 0, 0]

# перебираем первую цифру
for a in range(10):
    # пропускаем цифры 1 и 9
    if a == 1 or a == 9:
        continue
    combo[0] = a
    # массив для второй цифры
    array2 = turns[a]
    # перебираем все комбинации для второй цифры
    # из матрицы переходов
    for b in array2:
        combo[1] = b
        # массив для третьей цифры
        array3 = turns[b]
        # добавляем к результату число переходов из второй цифры
        cnt += len(array3)
        # перебираем все комбинации для третьей цифры
        # из матрицы переходов для вывода на экран
        for c in array3:
            combo[2] = c
            print(combo)
# вывод общего числа комбинаций на экран
print(cnt)
```

В результате выполнения программа выводит на экран следующее.

```
[0, 4, 3]
[0, 4, 9]
[0, 4, 0]
[0, 6, 1]
[0, 6, 7]
[0, 6, 0]
[2, 7, 2]
[2, 7, 6]
[2, 9, 2]
[2, 9, 4]
[3, 4, 3]
[3, 4, 9]
[3, 4, 0]
[3, 8, 1]
[3, 8, 3]
[4, 3, 4]
[4, 3, 8]
[4, 9, 2]
[4, 9, 4]
[4, 0, 4]
[4, 0, 6]
[6, 1, 6]
[6, 1, 8]
[6, 7, 2]
[6, 7, 6]
[6, 0, 4]
[6, 0, 6]
[7, 2, 7]
[7, 2, 9]
[7, 6, 1]
[7, 6, 7]
[7, 6, 0]
[8, 1, 6]
[8, 1, 8]
[8, 3, 4]
[8, 3, 8]
36
```

Ответ – 36 комбинаций.

Ответ: 36.

Задача 4. Вирус

Вариант 1

В одном институте спроектировали сеть, схематично изображенная на рисунке. Каждый маленький треугольник на рисунке обозначает компьютер. Треугольники с общей стороной соответствуют компьютерам, которые соединены между собой напрямую.

Нарушитель решает заразить один из компьютеров сети вирусом. Вирус распространяется по сети от заражённого компьютера ко всем соседним незаражённым. Однако при передаче на новый компьютер код вируса сжимается в три раза. Когда вирус сжимается до размера 1 Кб, он больше не может передаваться на соседние устройства, но компьютер, на котором он находится, считается заражённым.

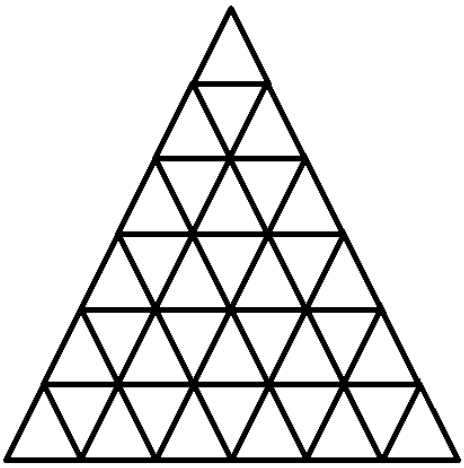


Рисунок. Схема сети

Антивирусная система сигнализирует опасность в случае, когда больше половины компьютеров заражены.

Какое максимальное количество компьютеров сможет заразить вирус без обнаружения антивирусом, если на первом заражённом компьютере вирус имеет размер 243Кб. В ответе укажите максимальное количество зараженных компьютеров и компьютер, на который нарушитель должен скопировать вирус.

Решение

Всего в сети 36 компьютеров. Заразить можно половину (18 компьютеров), чтобы антивирус не смог обнаружить. Размер вируса 243 Кб. Учитывая, что при каждом копировании размер вируса сжимается в 3 раза, возможно следующее количество переходов (заражений) компьютеров:

1 шаг – начальный (243 Кб)
2 шаг – 81 Кб
3 шаг – 27 Кб
4 шаг – 9 Кб
5 шаг – 3 Кб
6 шаг – 1 Кб

После 6-го шага вирус перестанет распространяться. Для получения ответа необходимо найти точку заражения такую, что после 6 шагов распространения зараженных компьютеров было максимальное количество, но не более 18.

Для поиска точки заражения возможно 2 варианта решения:

– ручной поиск места и подсчет числа зараженных компьютеров (с учетом симметричности фигуры);

– автоматизированный поиск точки заражения с использованием программы.

Способ 1.

Необходимо рассмотреть 3 пограничных варианта:

1) заражение центрального компьютера (В1);
2) заражение углового компьютера (В2);
3) заражение крайнего компьютера, расположенного в середине последнего ряда треугольника (В3).

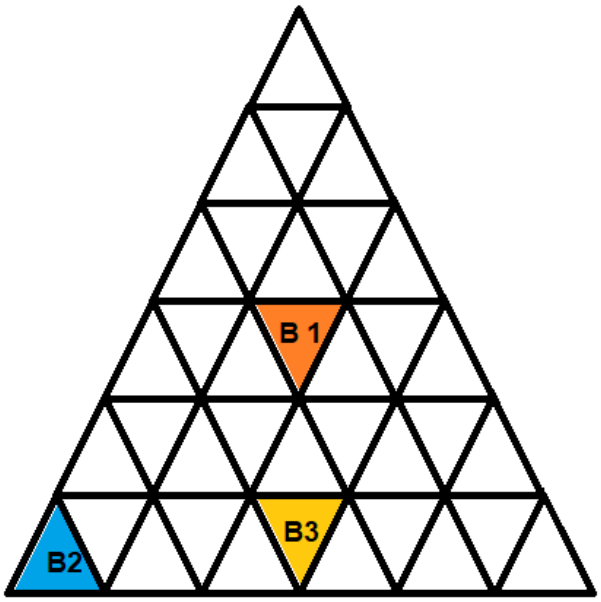
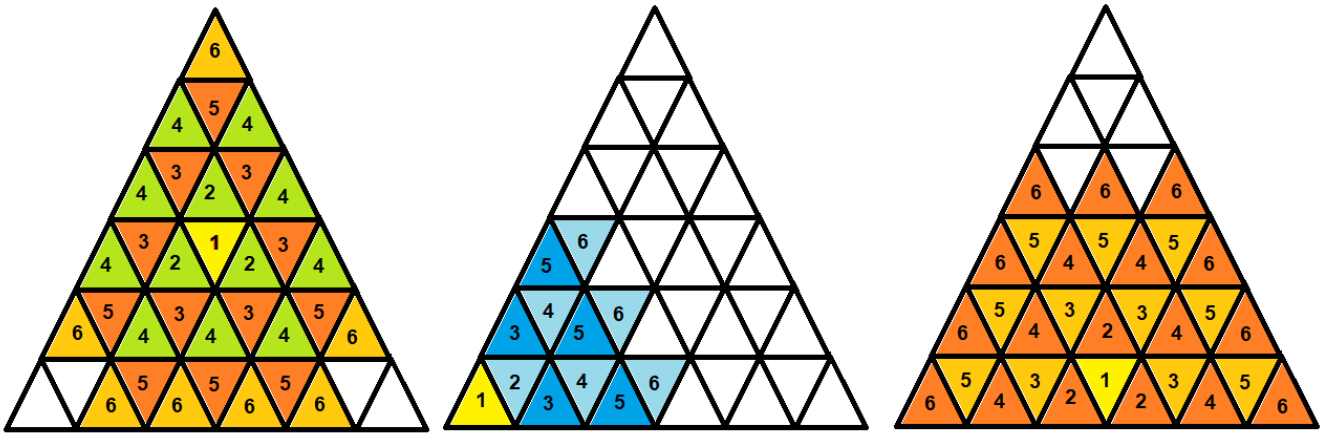


Рисунок 4.1-1 – Варианты начальной точки распространения вируса

Результаты распространения вирусов с точек В1, В2, В3 приведены на рисунке 4.1-2.



В1 В2 В3

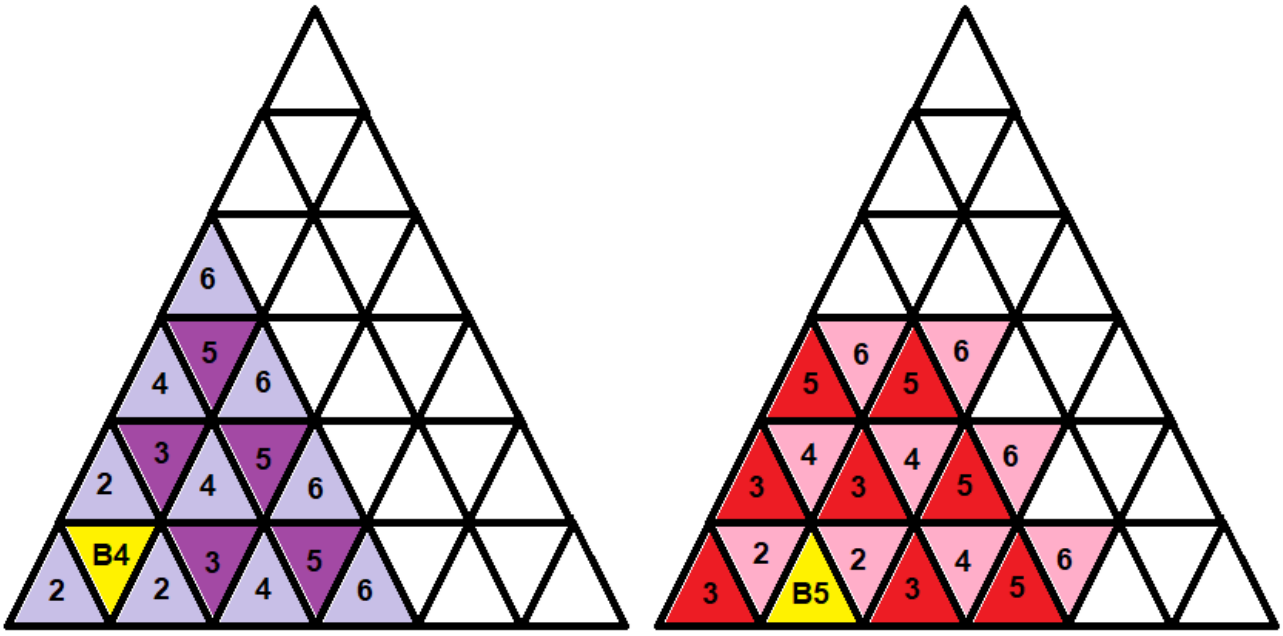
Рисунок 4.1-2 – Результаты распространения вируса из точек В1, В2, В3

При начале распространения вируса из точки В1 заражаются 32 компьютера – антивирус обнаружит.

При начале распространения вируса из точки В2 заражаются 12 компьютеров – антивирус не обнаружит, но это меньше 18.

При начале распространения вируса из точки В3 заражаются 30 компьютеров – антивирус обнаружит.

Необходимо искать другие точки заражения, расположенные рядом с В2. Рассмотрим 2 варианта: В4 и В5 (см. рисунок 4.1-3).



B4 B5

Рисунок 4.1-3 – Результаты распространения вируса из точек B4 и B5

При начале распространения вируса из точки B4 заражаются 16 компьютеров – антивирус не обнаружит.

При начале распространения вируса из точки B5 заражаются 18 компьютеров – антивирус не обнаружит, это ровно половина компьютеров.

Ответ, удовлетворяющий условиям – B5, при котором заражаются ровно половина компьютеров (18). С учетом симметричности фигуры вирус необходимо скопировать на любой из компьютеров, указанных на рисунке 4.1-4 цветом и буквой «C».

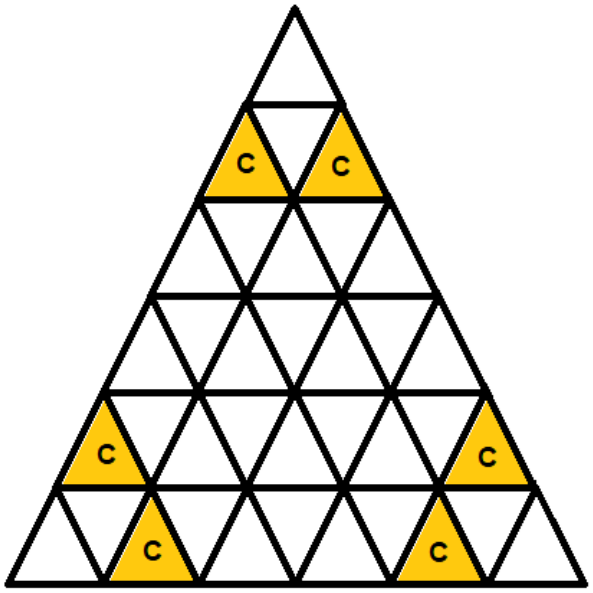


Рисунок 4.1-4 – Место заражения вирусом первого компьютера («C»)

Способ 2.

Для автоматизации поиска числа зараженных компьютеров и места заражения необходимо задать структуру сети в виде графа. Предлагается реализовать матрицу переходов, где каждой вершине соответствует список смежных вершин, то есть тех компьютеров, которые он может заразить. Однако прежде необходимо пронумеровать вершины (компьютеры) (см. рисунок 4.1-5).

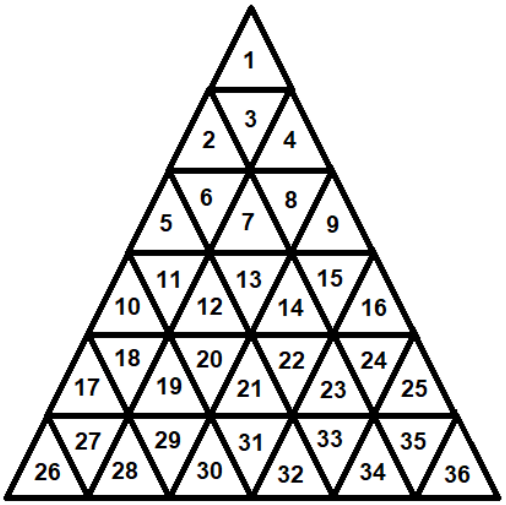


Рисунок 4.1-5 – Нумерация компьютеров (вершин графа)

Теперь матрица переходов будет выглядеть следующим образом (см. листинг 4.1-1).

Листинг 4.1-1 – Реализация матрицы переходов на языке программирования C++

```
map<int, vector<int>> matrix = {
                    {1, {3}},
                    {2, {3,6}},
                    {3, {1,2,4}},
                    {4, {3,8}},
                    {5, {6,11}},
                    {6, {2,5,7}},
                    {7, {6,8,13}},
                    {8, {4,7,9}},
                    {9, {8,15}},
                    {10, {11,18}},
                    {11, {5,10,12}},
                    {12, {11,13,20}},
                    {13, {7,12,14}},
                    {14, {13,15,22}},
                    {15, {9,14,16}},
                    {16, {15,24}},
                    {17, {18,27}},
                    {18, {10,17,19}},
                    {19, {18,20,29}},
                    {20, {12,19,21}},
                    {21, {20,22,31}},
                    {22, {14,21,23}},
                    {23, {22,24,33}},
                    {24, {16,23,25}},
                    {25, {24,35}},
                    {26, {27}},
                    {27, {17,26,28}},
                    {28, {27,29}},
                    {29, {19,28,30}},
                    {30, {29,31}},
                    {31, {21,30,32}},
                    {32, {31,33}},
                    {33, {23,32,34}},
                    {34, {33,35}},
                    {35, {25,34,36}},
                    {36, {35}}
};
```

Алгоритм работы самой программы следующий:

1) В цикле перебираем все вершины от 1 до 36.
2) Обнуляем массив инфицированных узлов и добавляем очередную вершину (шаг 1 заражения).
3) В цикле 5 раз (шаги 2-6 заражения) просматриваем массив инфицированных узлов, и для каждого инфицированного узла добавляем туда всех его соседей из матрицы переходов.
4) Подсчитываем количество зараженных узлов в массиве инфицированных.

Для того, чтобы один и тот же узел не добавлялся несколько раз в массив инфицированных узлов, можно воспользоваться контейнером «множество», в котором каждое значение может храниться только в одном экземпляре.

Пример реализации такого алгоритма приведен в листинге 4.1-2.

Листинг 4.1-2 – Реализация алгоритма перебора всех узлов и поиска числа зараженных компьютеров на языке программирования C++

```
int main()
{
  // множество инфицированных узлов (массив без повторений)
  set<int> infected;

  // множество инфицированных узлов (массив без повторений)
  // на очередном шаге
  set<int> step_infected;

  // множество соседей узла
  set<int> neighbors;

  // цикл по всем вершинам
  for (int i = 1; i <= 36; i++)
  {
    // очистка множества инфицированных узлов
    infected.clear();
    step_infected.clear();

    // инфицирование первого узла
    infected.insert(i);

    // цикл для 2,3,4,5,6 волны инфицирования
    for (int s = 1; s < 6; s++)
    {
      // цикл по уже инфицированным узлам
      // заражаем всех их соседей
      // и добавляем в массив инфицированных
      for (auto it = infected.begin(); it != infected.end(); it++)
      {
        // соседи текущего зараженного узла
        neighbors = matrix[*it];
        // добавляем всех соседей в массив инфицированных узлов
        // на текущем шаге
        step_infected.insert(neighbors.begin(), neighbors.end());
      }
      // добавляем все инфицированные узлы на текущем шаге
      // в общий массив инфицированных узлов
      infected.insert(step_infected.begin(), step_infected.end());
    }
    // вывод на экран финформации
    cout << "Start: " << i << " total: " << infected.size() << endl;
  }
}
```

Результаты выполнения программы:

```
Start: 1 total: 12
Start: 2 total: 18
Start: 3 total: 16
Start: 4 total: 18
Start: 5 total: 24
Start: 6 total: 23
Start: 7 total: 26
Start: 8 total: 23
Start: 9 total: 24
Start: 10 total: 24
Start: 11 total: 30
Start: 12 total: 30
Start: 13 total: 32
Start: 14 total: 30
Start: 15 total: 30
Start: 16 total: 24
Start: 17 total: 18
Start: 18 total: 23
Start: 19 total: 26
Start: 20 total: 32
Start: 21 total: 30
Start: 22 total: 32
Start: 23 total: 26
Start: 24 total: 23
Start: 25 total: 18
Start: 26 total: 12
Start: 27 total: 16
Start: 28 total: 18
Start: 29 total: 23
Start: 30 total: 24
Start: 31 total: 30
Start: 32 total: 24
Start: 33 total: 23
Start: 34 total: 18
Start: 35 total: 16
Start: 36 total: 12
```

Проанализировав информацию, можно сделать вывод, что наилучшие условия для заражения узлов без обнаружения антивирусом – это 18 узлов (ровно половина), при этом разместить вирус необходимо в одном из узлов с номерами 2,4,17,25,28,34 (рисунок 4.1-6).

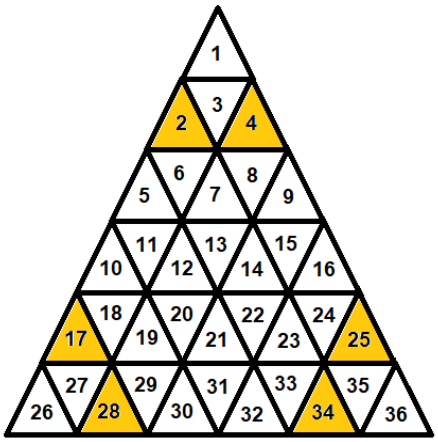


Рисунок 4.1-4 – Место заражения вирусом первого компьютера («С»)

Ответ: 18 узлов, 6 вариантов начального заражения.

Вариант 2

В одном институте спроектировали сеть, схематично изображенная на рисунке. Каждый маленький треугольник на рисунке обозначает компьютер. Треугольники с общей стороной соответствуют компьютерам, которые соединены между собой напрямую.

Нарушитель решает заразить один из компьютеров сети вирусом. Вирус распространяется по сети от заражённого компьютера ко всем соседним незаражённым. Однако при передаче на новый компьютер код вируса сжимается в три раза. Когда вирус сжимается до размера 1 Кб, он больше не может передаваться на соседние устройства, но компьютер, на котором он находится, считается заражённым.

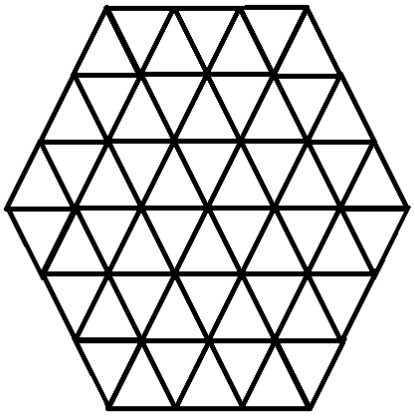


Рисунок. Схема сети

Антивирусная система сигнализирует опасность в случае, когда больше половины компьютеров заражены.

Какое максимальное количество компьютеров сможет заразить вирус без обнаружения антивирусом, если на первом заражённом компьютере вирус имеет размер 81Кб. В ответе укажите максимальное количество зараженных компьютеров и компьютер, на который нарушитель должен скопировать вирус.

Решение

Всего в сети 54 компьютера. Заразить можно половину (27 компьютеров), чтобы антивирус не смог обнаружить. Размер вируса 81 Кб. Учитывая, что при каждом копировании размер вируса сжимается в 3 раза, возможно следующее количество переходов (заражений) компьютеров:

1 шаг – начальный (81 Кб)
2 шаг – 27 Кб
3 шаг – 9 Кб
4 шаг – 3 Кб
5 шаг – 1 Кб

После 5-го шага вирус перестанет распространяться. Для получения ответа необходимо найти точку заражения такую, что после 5 шагов распространения зараженных компьютеров было максимальное количество, но не более 27.

Для поиска точки заражения возможно 2 варианта решения:

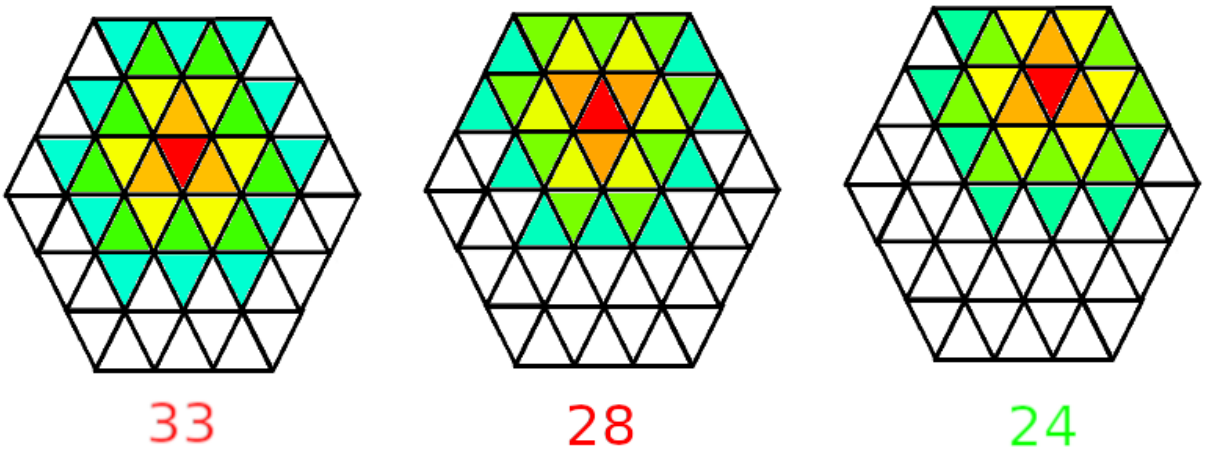
– ручной поиск места и подсчет числа зараженных компьютеров (с учетом симметричности фигуры);

– автоматизированный поиск точки заражения с использованием программы.

Способ 1.

Можно рассмотреть 3 варианта, учитывая симметричность фигуры:

1) заражение центрального компьютера (B1);
2) заражение компьютера на расстоянии 1 от центрального (B2);
3) заражение компьютера на расстоянии 2 от центрального (B3).



B1 B2 B3

Рисунок 4.2-1 – Результаты распространения вируса из точек B1, B2, B3

При начале распространения вируса из точки B1 заражаются 33 компьютера – антивирус обнаружит.

При начале распространения вируса из точки B2 заражаются 28 компьютеров – антивирус обнаружит.

При начале распространения вируса из точки B3 заражаются 24 компьютеров – антивирус не обнаружит, но это меньше половины (27).

Очевидно, что при рассмотрении точек заражения ближе к границам фигуры, количество зараженных компьютеров будет меньше. Поэтому необходимо рассмотреть еще один вариант (B4) и убедиться, что больше 24 и меньше 27 компьютеров заразить нет возможности (рисунок 4.2-2).

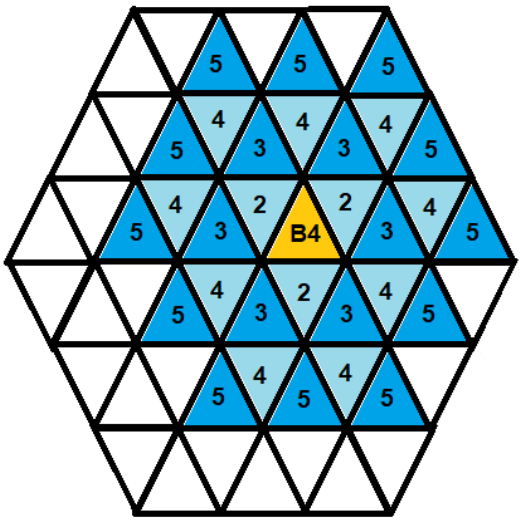


Рисунок 4.2-2 – Результаты распространения вируса из точки B4

При начале распространения вируса из точки B4 заражается 31 компьютер – антивирус обнаружит.

С учетом симметричности фигуры ответом будет заражение 24 компьютеров без обнаружения антивирусом. Узел, на который необходимо скопировать вирус, указан на рисунке 4.2-3.

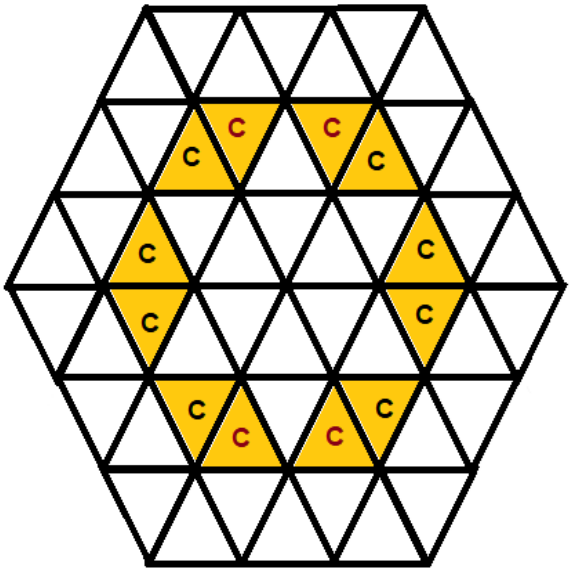


Рисунок 4.2-3 – Место заражения вирусом первого компьютера («С»)

Способ 2.

Для автоматизации поиска числа зараженных компьютеров и места заражения необходимо задать структуру сети в виде графа. Однако прежде необходимо пронумеровать вершины (компьютеры) (см. рисунок 4.2-4). Предлагается нумеровать строки и столбцы для каждого узла.

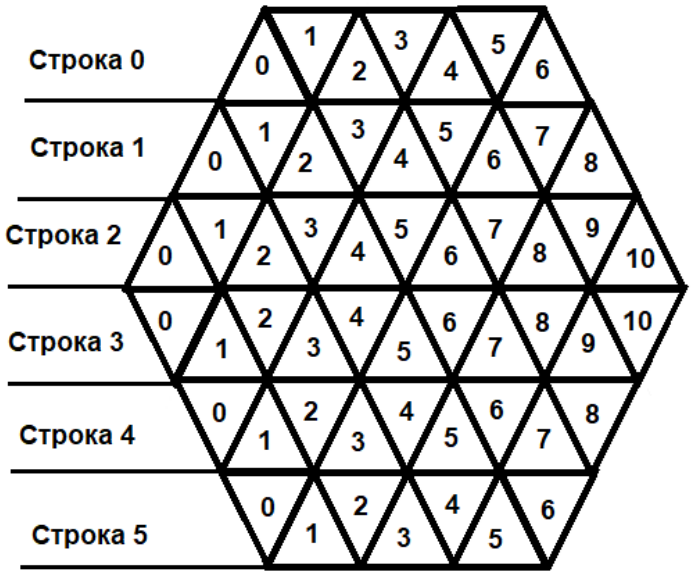


Рисунок 4.2-4 – Нумерация узлов

Легко заметить зависимости в номерах соседних узлов:

1) Соседи справа есть у всех, кроме крайних правых узлов (номера которых последние в строке). Его номер равен номеру столбца узла плюс 1.
2) Соседи слева есть у всех, кроме крайних левых узлов (номера которых равны 0). Его номер равен номеру столбца узла минус 1.
3) Соседи сверху (кроме первой строки 0) есть:
   - у узлов с нечетными номерами в верхней половине – номер строки минус 1 и номер столбца минус 1;
   - у узлов с четными номерами в нижней половине – номер строки минус 1 и номер столбца минус 1 (или номер столбца такой же, если длина верхней строки такая же).

4) Соседи снизу (кроме последней строки 5) есть:
   - у узлов с четными номерами в верхней половине – номер строки плюс 1 и номер столбца плюс 1 (или номер столбца такой же, если длина нижней строки такая же);
   - у узлов с нечетными номерами в нижней половине – номер строки плюс 1 и номер столбца минус 1.

Вместо того, чтобы вручную задавать матрицу переходов или матрицу смежности (где число строк и число столбцов равно числу вершин – а это 54), можно написать функцию, которая возвращает массив соседних узлов. Пример такой функции приведен в листинге 4.2-1.

Листинг 4.2-1 – Функция получения массива соседних связанных узлов на языке программирования Python

```
# Длины строк
StrLen = [7,9,11,11,9,7]

## Функция получения массива соседних связанныъх узлов
# PARAMS
#   nStr - номер строки искомого узла (нумерация с 0)
#   nStb - номер столбца искомого узла (нумерация с 0)
# RETURN
#   массив соседей в формате [ [nStr1, nStb1], [nStr2, nStb2],...]
##
def neighbors(nStr ,nStb):
    res = []
    # если не первый узел в строке - сосед слева есть
    if nStb != 0:
        res.append([nStr, nStb - 1])
    # если не последний узел в строке - сосед справа есть
    if nStb != StrLen[nStr] - 1:
        res.append([nStr, nStb + 1])

    # если строка из верхней половины и не первая и номер узла нечетный,
    # ИЛИ
    # строка из нижней половины и номер узла четный,
    # то сосед сверху есть
    if ( (nStr != 0 and nStr < len(StrLen)//2 and nStb % 2 == 1)
         or (nStr >= len(StrLen)//2 and nStb % 2 == 0) ):
        # если верхняя строка больше, то столбец + 1
        if StrLen[nStr - 1] > StrLen[nStr]:
            res.append([nStr - 1, nStb + 1])
        # если верхняя строка меньше, то столбец - 1
        elif StrLen[nStr - 1] < StrLen[nStr]:
            res.append([nStr - 1, nStb - 1])
        # если верхняя строка такой же длины, то номер столбца такой же
        else:
            res.append([nStr - 1, nStb])

    # если строка из верхней половины и номер узла четный,
    # ИЛИ
    # строка из нижней половины и не последняя и номер узла нечетный,
    # то сосед сверху есть
    if ( (nStr < len(StrLen)//2 and nStb % 2 == 0)
      or (nStr >= len(StrLen)//2 and nStr != len(StrLen)-1 and nStb%2==1)
      ):
        # если нижняя строка больше, то столбец + 1
        if StrLen[nStr + 1] > StrLen[nStr]:
            res.append([nStr + 1, nStb + 1])
        # если нижняя строка меньше, то столбец - 1
        elif StrLen[nStr + 1] < StrLen[nStr]:
```

```
            res.append([nStr + 1, nStb - 1])
        # если нижняя строка такой же длины, то номер столбца такой же
        else:
            res.append([nStr + 1, nStb])
    # возвращаем массив соседей
    return res
```

Алгоритм работы самой программы следующий:

1) В цикле перебираем все строки от 0 до 5 включительно.
2) Для каждой строки в цикле перебираем узлы.
3) Обнуляем массив инфицированных узлов и добавляем очередную вершину (шаг 1 заражения).
4) В цикле 4 раз (шаги 2-5 заражения) просматриваем массив инфицированных узлов, и для каждого инфицированного узла добавляем туда всех его соседей, получаемых с помощью функции `neighbors()`.
5) После завершения цикла по шагам заражения подсчитываем количество зараженных узлов в массиве инфицированных, выводим на экран информацию и переходим к следующему узлу (п.3).

Пример реализации такого алгоритма приведен в листинге 4.2-2.

Листинг 4.2-2 – Реализация алгоритма перебора всех узлов и поиска числа зараженных компьютеров на языке программирования Python

```
## Функция инфицирования узлов
# PARAMS
#   массив уже инфицированных узлов
# RETURN
#   массив инфицированных + новых зараженных узлов
#   (всех соседей из массива инфицированных + сами инфицированные узлы)
def infect(infected):
    # создаем копию массива, чтобы не испортить оригинал
    res = copy.copy(infected)
    # цикл по всем узлам из массива инфицированных
    for node in infected:
        # цикл по всем соседям очередного узла node
        for infNode in neighbors(node[0],node[1]):
            # если такого узла еще нет в списке инфицированных -
            # добавляем его
            if infNode not in res:
                res.append(infNode)
    # возвращаем результат
    return res

output=[]
# цикл по строкам
for i in range(0,len(StrLen)):
    # цикл по узлам в строке
    for j in range(0,StrLen[i]):
        # обнудение массива инфицированных
        infected = []
        # добавление первого зараженного узла (очередной)
        infected.append([i,j])
        # шаги заражения
        for step in range(1,5):
            # добавляем всех соседей в массив инфицированных
            infected = infect(infected)
        # вывод на экран информации и статистики
```

```
        print(f"[{i},{j}] - total: {len(infected)}")
        output.append({'start':[i,j],'count':len(infected)})
# сортировка итоговой информации
output.sort(key=lambda x: x['count'])
# вывод на экран в отсортированном виде
for i in output:
    print(i)
```

Результат работы программы:

```
{'start': [0, 0], 'count': 15}
{'start': [0, 1], 'count': 15}
{'start': [0, 5], 'count': 15}
{'start': [0, 6], 'count': 15}
{'start': [2, 0], 'count': 15}
{'start': [2, 10], 'count': 15}
{'start': [3, 0], 'count': 15}
{'start': [3, 10], 'count': 15}
{'start': [5, 0], 'count': 15}
{'start': [5, 1], 'count': 15}
{'start': [5, 5], 'count': 15}
{'start': [5, 6], 'count': 15}
{'start': [0, 3], 'count': 17}
{'start': [1, 0], 'count': 17}
{'start': [1, 8], 'count': 17}
{'start': [4, 0], 'count': 17}
{'start': [4, 8], 'count': 17}
{'start': [5, 3], 'count': 17}
{'start': [0, 2], 'count': 19}
{'start': [0, 4], 'count': 19}
{'start': [1, 1], 'count': 19}
{'start': [1, 7], 'count': 19}
{'start': [2, 1], 'count': 19}
{'start': [2, 9], 'count': 19}
{'start': [3, 1], 'count': 19}
{'start': [3, 9], 'count': 19}
{'start': [4, 1], 'count': 19}
{'start': [4, 7], 'count': 19}
{'start': [5, 2], 'count': 19}
{'start': [5, 4], 'count': 19}
{'start': [1, 2], 'count': 24}
{'start': [1, 3], 'count': 24}
{'start': [1, 5], 'count': 24}
{'start': [1, 6], 'count': 24}
{'start': [2, 2], 'count': 24}
{'start': [2, 8], 'count': 24}
{'start': [3, 2], 'count': 24}
{'start': [3, 8], 'count': 24}
{'start': [4, 2], 'count': 24}
{'start': [4, 3], 'count': 24}
{'start': [4, 5], 'count': 24}
{'start': [4, 6], 'count': 24}
{'start': [1, 4], 'count': 28}
{'start': [2, 3], 'count': 28}
{'start': [2, 7], 'count': 28}
{'start': [3, 3], 'count': 28}
{'start': [3, 7], 'count': 28}
{'start': [4, 4], 'count': 28}
{'start': [2, 4], 'count': 31}
{'start': [2, 5], 'count': 31}
```

```
{'start': [2, 6], 'count': 31}
{'start': [3, 4], 'count': 31}
{'start': [3, 5], 'count': 31}
{'start': [3, 6], 'count': 31}
```

Проанализировав информацию, можно сделать вывод, что наилучшие условия для заражения узлов без обнаружения антивирусом – это 24 узла, при этом разместить вирус необходимо в одном из узлов с координатами [1,2], [1,3], [1,5], [1,6], [2,2], [2,8], [3,2], [3,8], [4,2], [4,3], [4,5], [4,6] (всего 12 вариантов). Подробнее см. рис. 4.2-3 и 4.2-4.

Ответ: 24 узла, 12 вариантов начального заражения.

Задача 5. Web-сайт

Вариант 1

Пользователь хранит на сервере секретное слово, доступ к которому можно получить, авторизовавшись через web-сайт. Сервер выдаст секретное слово только в том случае, если ему будет отправлена верная зашифрованная последовательность, сформированная из логина и пароля. Чтобы не забыть логин и пароль, пользователь оставил себе подсказки на сайте.

Также известно, что:

1. Логин и пароль имеют одинаковую длину.
2. Логин состоит только из латинских букв, пароль состоит только из цифр.

Определите секретное слово.

К задаче прилагается:
папка с содержимым web-страницы.

Решение

Задача предполагает 2 способа решения:
1) аналитический;
2) анализ исходного кода (reverse-engineering).

Способ 1.
На открывшейся web-странице есть следующие активные поля (рисунок 5.1-1):
–логин (1);
–пароль (2);
– ссылка «Помнишь меня?» (3);
–ссылка «Забыли код?» (4).

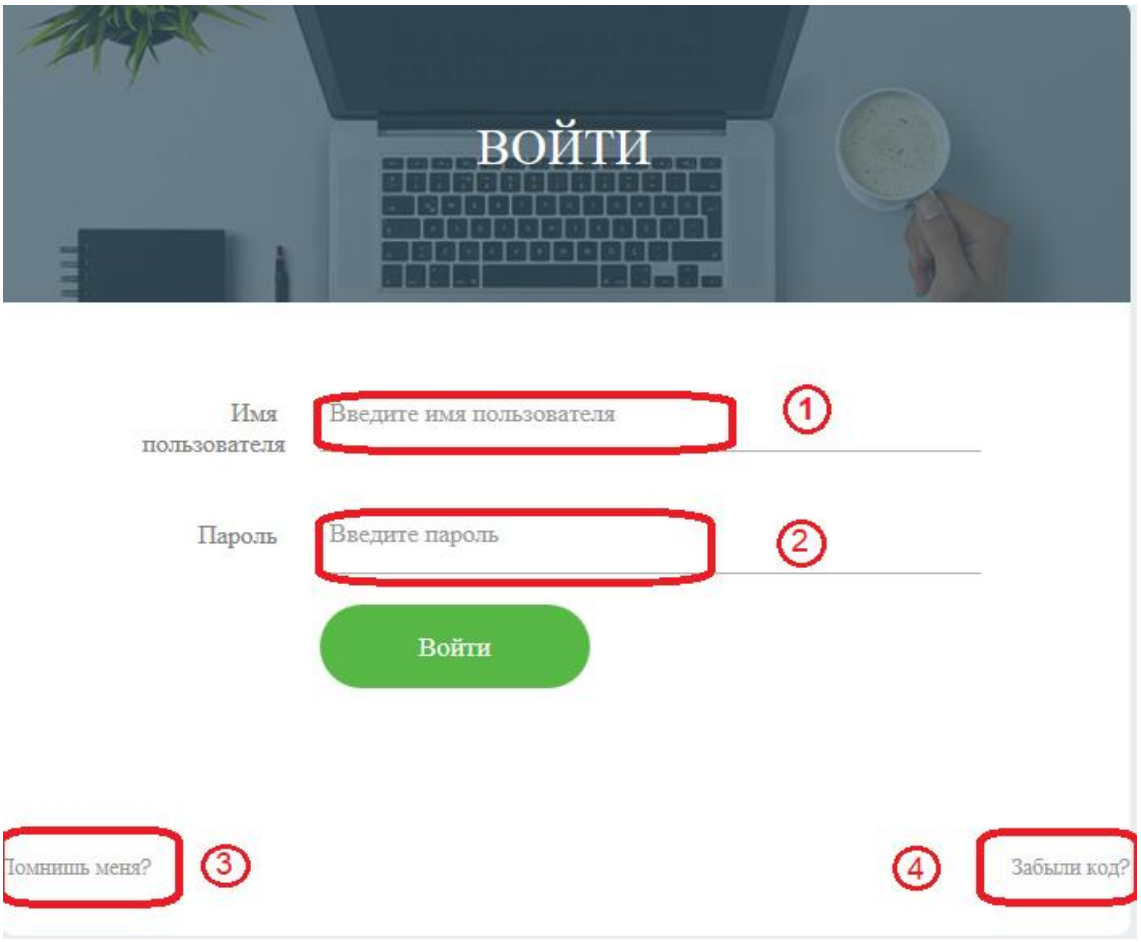


Рисунок 5.1-1 Внешний вид web-страницы

Известно, что логин состоит только из букв, пароль состоит только из цифр. Длина логина и пароля должны совпадать.

Нажав на ссылку «Помнишь меня?», страница выдает сообщение:

`zjmkhgw`

Нажав на ссылку «Забыли код?», страница выдает сообщение:

```
alfredojustmybingvxchwqkpz
01234567890123456789012345
```

Можно заметить, что в первой строке содержатся все символы английского алфавита без повторений (26 символов). Каждому символу соответствует свой номер из нижней строки: `a`–0, `l`–1, `f`–2, … `z`–5.

Можно предположить, что это не случайность, и не просто так. Поэтому можно попробовать взять сообщение *zjmkhgw*, использовать его как логин, а в качестве пароля подобрать цифры из второй подсказки «Забыли код?»:

z–5, j–7, m–1, k–3, h–0, g–6, w–1.

Вводим следующие данные:

логин – *zjmkhgw*,

пароль – *5713061*

В результате на странице отобразится код: *Dragonfly* (рисунок 5.1-2).

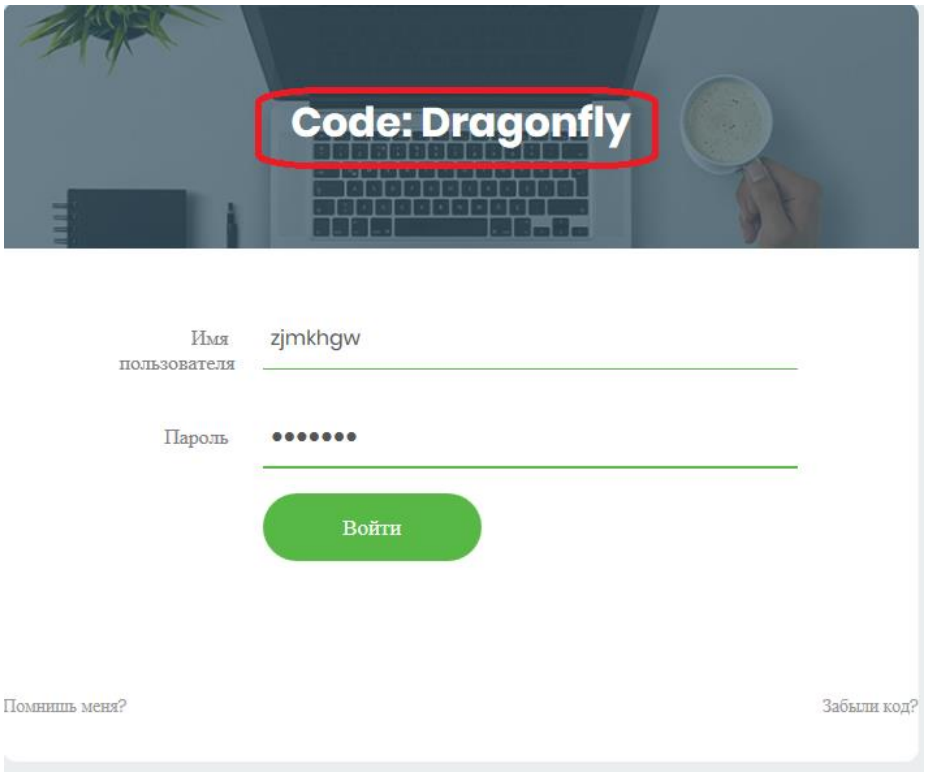


Рисунок 5.1-2 – Содержимое web-страницы после правильно введенных логина и пароля

Способ 2.

Проанализировав код страницы можно увидеть, что к странице подключены следующие JavaScript-файлы:

```
<script src="script/proof.js"></script>
<script src="script/script.js"></script>
<script src="js/script.js"></script>
```

Остальные файлы относятся к платформе для корректного отображения объектов.

Рассмотрим содержимое файла `js/script.js`. В этом файле устанавливаются обработчики событий click на объекты web-страницы:

- при нажатии на ссылку «Помнишь меня?» выводится сообщение с результатом выполнения функции `forgetCode()`;
- при нажатии на ссылку «Забыли код?» выводится сообщение с выражением `getBaseString()+'\n012345678901234567890123456789012345';`
- при нажатии на кнопку «Войти» вызывается функция `myfunc()`, которая описана в этом же файле.

Интерес вызывает именно функция `myfunc()`. Рассмотрим её подробнее (листинг 5.1-1).

Листинг 5.1-1 – Содержимое функции myfunc()

```
1  function myfunc() {
2      let code = '';
3      if (username_form.value && passwd_form.value)
4          code = getCode(username_form.value, passwd_form.value);
5      if (code && code.length > 0)
6          code_label.innerText = code;
7      else
8          code_label.innerText = 'ВОЙТИ';
9  }
```

В строке 3 проверяется на непустые значения полей логин и пароль, после чего вызывается функция `getCode()`. Результат функции отображается на web-странице.

Проанализируем функцию `getCode()`. Она реализована в файле `script/script.js` (листинг 5.1-2).

Листинг 5.1-2 – Содержимое функции getCode()

```
1.  function getCode(username, password) {
2.      if (username.length !== password.length && username.length > 0 &&
    password.length > 0) {
3.          alert('Длина имени пользователя и пароля не совпадают!');
4.          return String('');
5.      }
6.      const str = getBaseString();
7.      const nums = '0123456789';
8.      let res = Bar();
9.      let check = Foo();
10.     let code = '';
11.     let pos1 = -1;
12.     let pos2 = -1;
13.     let letter = '';
14.     for (let i = 0; i < username.length; i++) {
15.         pos1 = str.indexOf(username[i]);
16.         pos2 = nums.indexOf(password[i]);
17.         if (pos1 === -1 || pos2 === -1) {
18.             code += '-';
19.         }
20.         else {
21.             letter = str[(pos1 + pos2) % str.length];
22.             if (letter === Foo(i))
23.                 code += letter.toUpperCase();
24.             else
25.                 code += letter;
26.         }
27.     }
28.     if (code.toLowerCase() === check.toLowerCase()) {
29.         return res;
30.     }
31.     else {
32.         return code;
33.     }
34. }
```

В самой функции интерес представляет лишь последняя конструкция if (строки 28-33). В этих строках и формируется результат. Если условие в строке 28 верное, то результатом выполнения функции является переменная `res`, иначе переменная `code`. Значение переменной `res` получается из функции `Bar()` (строка 8).

Проанализируем функцию `Bar()`, которая реализована в файле `script/proof.js` (листинг 5.1-3).

Листинг 5.1-3 – Содержимое функции Bar()

```
1.  function Bar() {
2.      const arr = [67, 110, 98, 98, 54, 27, 62, 107, 89, 94, 101, 99,
    90, 95, 107];
3.      let res = '';
4.      for (let i = 0; i < arr.length; i++) {
5.          res = res + String.fromCharCode(arr[i] + i);
6.      }
7.      return res;
8.  };
```

Функция преобразовывает массив чисел в символы в соответствии с ASCII-таблицей. Можно запустить в отладчике эту функцию и посмотреть результат ее выполнения (ответом

будет строка “*Code: Dragonfly*”), а можно в функции `getCode()` в строке 32 вместо строки `return code;`
записать
`return Bar();` ИЛИ `return res;`

Тогда при любых значениях логина и пароля с одинаковой длиной получится следующий результат (рисунок 5.1-3).

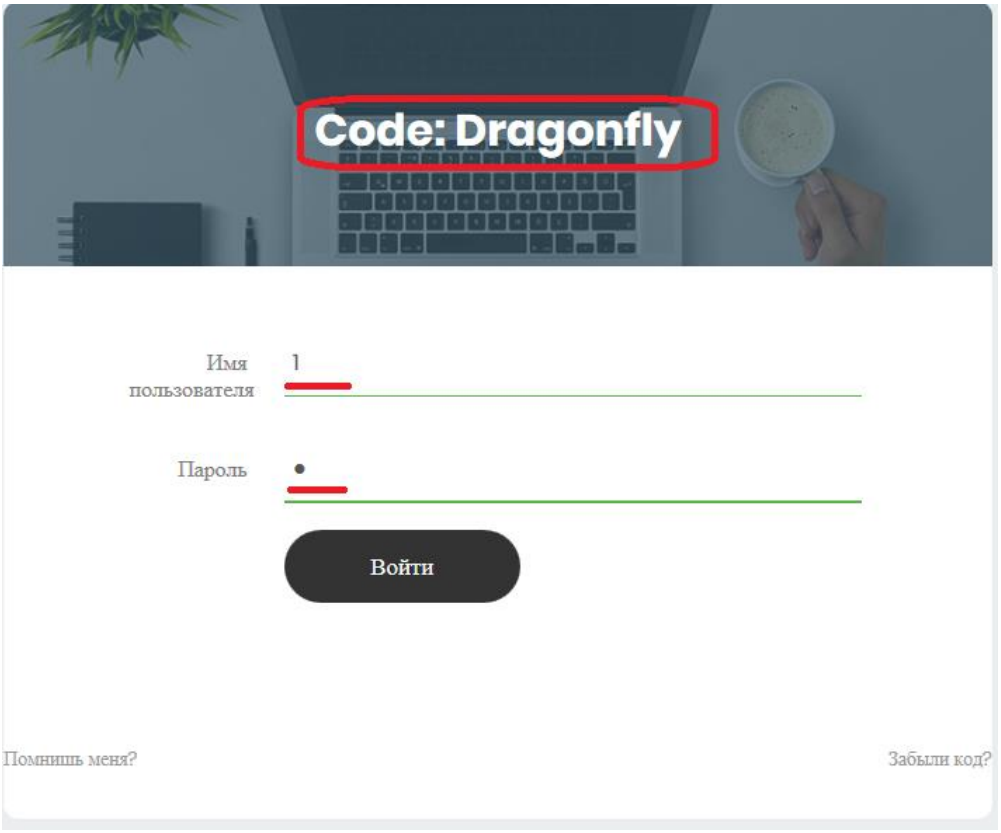


Рисунок 5.1-3 – Содержимое web-страницы с измененной функцией getCode()

Ответ: Dragonfly.

Вариант 2

Пользователь хранит на сервере секретное слово, доступ к которому можно получить, авторизовавшись через web-сайт. Сервер выдаст секретное слово только в том случае, если ему будет отправлена верная зашифрованная последовательность, сформированная из логина и пароля. Чтобы не забыть логин и пароль, пользователь оставил себе подсказки на сайте.

Также известно, что:

1. Логин и пароль имеют одинаковую длину.
2. Логин состоит только из латинских букв, пароль состоит только из цифр.

Определите секретное слово.

К задаче прилагается:
папка с содержимым web-страницы.

Решение

Задача предполагает 2 способа решения:
1) аналитический;
2) анализ исходного кода (reverse-engineering).

Способ 1.

На открывшейся web-странице есть следующие активные поля (рисунок 5.2-1):

–логин (1);

–пароль (2);

– ссылка «Помнишь меня?» (3);

–ссылка «Забыли код?» (4).

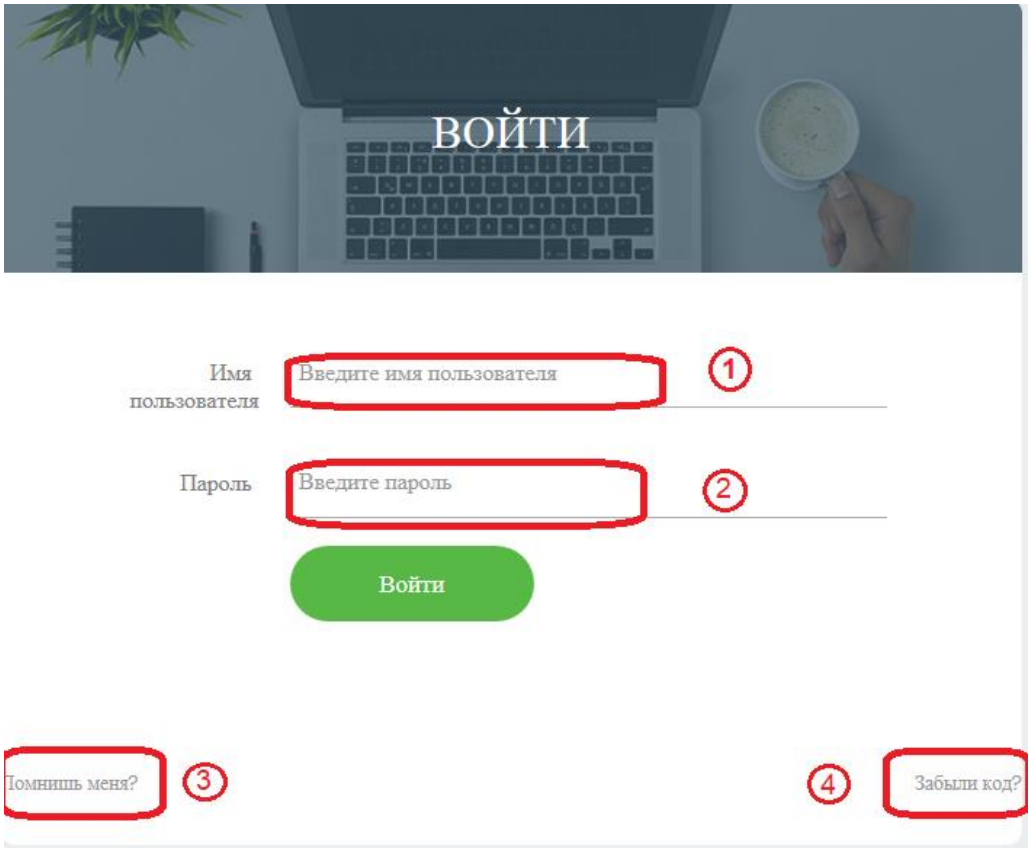


Рисунок 5.2-1 Внешний вид web-страницы

Известно, что логин состоит только из букв, пароль состоит только из цифр. Длина логина и пароля должны совпадать.

Нажав на ссылку «Помнишь меня?», страница выдает сообщение:

mhjyzko

Нажав на ссылку «Забыли код?», страница выдает сообщение:

thequickbrownfxjmpsvlazydg
01234567890123456789012345

Можно заметить, что в первой строке содержатся все символы английского алфавита без повторений (26 символов). Каждому символу соответствует свой номер из нижней строки: t–0, h–1, e–2, … q–5.

Можно предположить, что это не случайность, и не просто так. Поэтому можно попробовать взять сообщение *mhjyzko*, использовать его как логин, а в качестве пароля подобрать цифры из второй подсказки «Забыли код?»:

m–6, h–1, j–5, y–3, z–2, k–7, o–0.

Вводим следующие данные:

логин – *mhjyzko,*

пароль – *6153270*

В результате на странице отобразится код: *ROCKET* (рисунок 5.2-2).

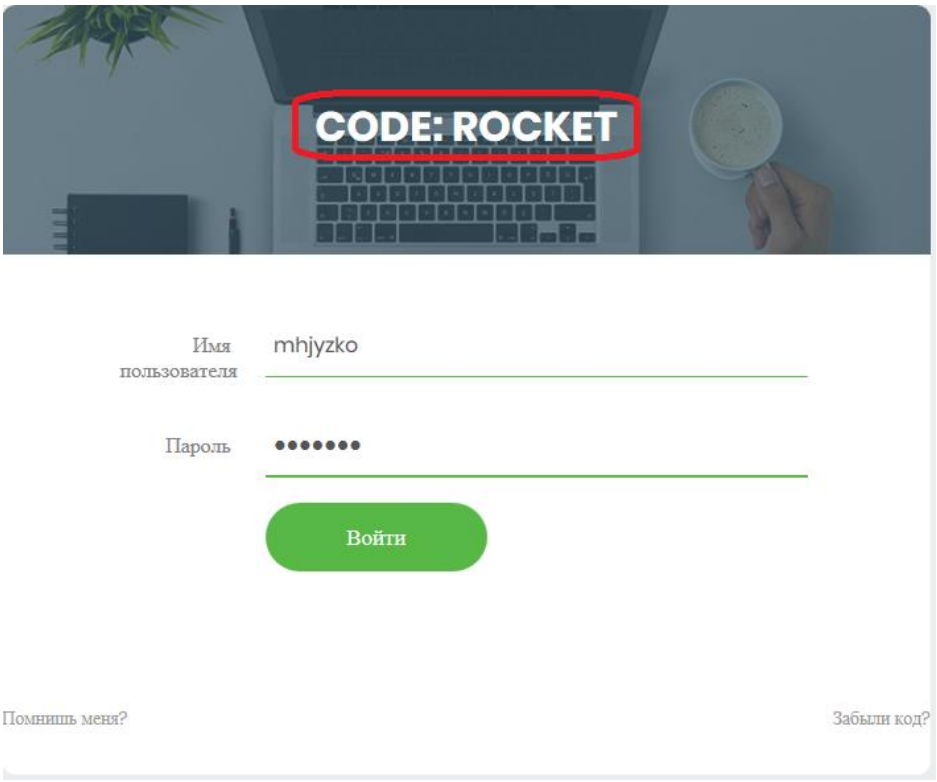


Рисунок 5.2-2 – Содержимое web-страницы после правильно введенных логина и пароля

Способ 2.

Проанализировав код страницы можно увидеть, что к странице подключены следующие JavaScript-файлы:

```
<script src="script/proof.js"></script>
<script src="script/script.js"></script>
<script src="js/script.js"></script>
```

Остальные файлы относятся к платформе для корректного отображения объектов.

Рассмотрим содержимое файла `js/script.js`. В этом файле устанавливаются обработчики событий click на объекты web-страницы:

- при нажатии на ссылку «Помнишь меня?» выводится сообщение с результатом выполнения функции `forgetCode()`;
- при нажатии на ссылку «Забыли код?» выводится сообщение с выражением `getBaseString()+'\n012345678901234567890123456789012345'`;
- при нажатии на кнопку «Войти» вызывается функция `myfunc()`, которая описана в этом же файле.

Интерес вызывает именно функция `myfunc()`. Рассмотрим её подробнее (листинг 5.2-1).

Листинг 5.2-1 – Содержимое функции myfunc()

```
1  function myfunc() {
2      let code = '';
3      if (username_form.value && passwd_form.value)
4          code = getCode(username_form.value, passwd_form.value);
5      if (code && code.length > 0)
6          code_label.innerText = code;
7      else
8          code_label.innerText = 'ВОЙТИ';
9  }
```

В строке 3 проверяется на непустые значения полей логин и пароль, после чего вызывается функция `getCode()`. Результат функции отображается на web-странице.

Проанализируем функцию `getCode()`. Она реализована в файле `script/script.js` (листинг 5.2-2).

Листинг 5.2-2 – Содержимое функции getCode()

```
1.  function getCode(username, password) {
2.      if (username.length !== password.length && username.length > 0 &&
    password.length > 0) {
3.          alert('Длина имени пользователя и пароля не совпадают!');
4.          return String('');
5.      }
6.      const str = getBaseString();
7.      const nums = '0123456789';
8.      let res = Bar();
9.      let check = Foo();
10.     let code = '';
11.     let pos1 = -1;
12.     let pos2 = -1;
13.     let letter = '';
14.     for (let i = 0; i < username.length; i++) {
15.         pos1 = str.indexOf(username[i]);
16.         pos2 = nums.indexOf(password[i]);
17.         if (pos1 === -1 || pos2 === -1) {
18.             code += '-';
19.         }
20.         else {
21.             letter = str[(pos1 + pos2) % str.length];
22.             if (letter === Foo(i))
23.                 code += letter.toUpperCase();
24.             else
25.                 code += letter;
26.         }
27.     }
28.     if (code.toLowerCase() === check.toLowerCase()) {
29.         return res;
30.     }
31.     else {
32.         return code;
33.     }
34. }
```

В самой функции интерес представляет лишь последняя конструкция if (строки 28-33). В этих строках и формируется результат. Если условие в строке 28 верное, то результатом выполнения функции является переменная `res`, иначе переменная `code`. Значение переменной `res` получается из функции `Bar()` (строка 8).

Проанализируем функцию `Bar()`, которая реализована в файле `script/proof.js` (листинг 5.2-3).

Листинг 5.2-3 – Содержимое функции Bar()

```
1.  function Bar() {
2.      const arr = [67, 110, 98, 98, 54, 27, 76, 104, 91, 98, 91, 105];
3.      let res = '';
4.      for (let i = 0; i < arr.length; i++) {
5.          res = res + String.fromCharCode(arr[i] + i);
6.      }
7.      return res;
8.  };
```

Функция преобразовывает массив чисел в символы в соответствии с ASCII-таблицей. Можно запустить в отладчике эту функцию и посмотреть результат ее выполнения (ответом будет строка “*CODE: ROCKET*”), а можно в функции `getCode()` в строке 32 вместо строки
`return code;`
записать
`return Bar();` ИЛИ `return res;`

Тогда при любых значениях логина и пароля с одинаковой длиной получится следующий результат (рисунок 5.2-3).

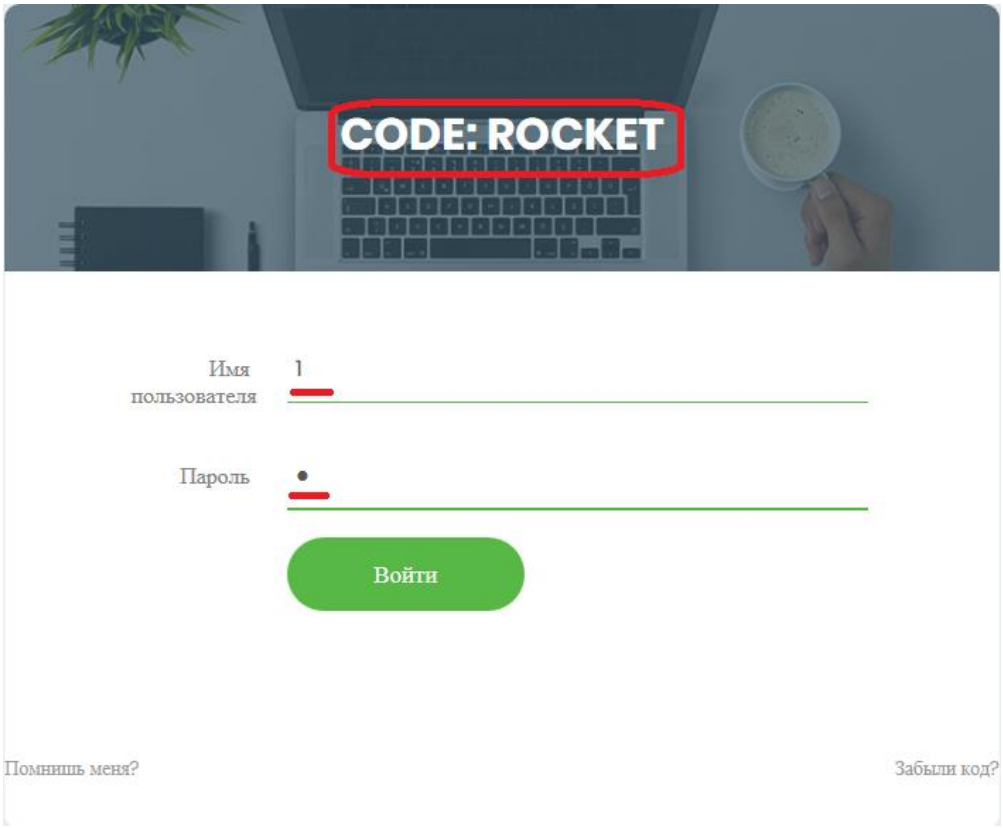


Рисунок 5.2-3 – Содержимое web-страницы с измененной функцией getCode()

Ответ: ROCKET.