Всероссийская олимпиада школьников по информатике
Вологодская область, 2022-23 учебный год
II (муниципальный) этап
7 - 8 классы
Методические рекомендации по разбору задач

Задача 1. Занимательные строки (100 баллов)

Вопрос 1. В каждой следующей строке символов вдвое больше, чем в предыдущей, плюс 1. Можно просто сосчитать первые 6 членов этой последовательности. Удобно делать это в электронной таблице, но можно и с помощью калькулятора. Ответ равен 63.

Вопрос 2**.** Можно продолжить вычисление этой последовательности и получить ответ 10.

Вопрос 3**.** Во второй строке одна буква ‘B’, в каждой следующей их вдвое больше – в третьей их 2^1, в четвертой – 2^2, …, в пятнадцатой – $2^{13} = 8192$.

Вопрос 4**.** Проще всего получить ответ, написав небольшую программу.

Также ответ можно получить в текстовом редакторе: начав со строки ‘A’ и используя операции копирования и вставки, можно быстро добраться до девятой строки (выделяем, копируем и дважды вставляем предыдущую строку, дописываем новый символ, повторяем). К сожалению, не все редакторы показывают текущую позицию курсора (но обычно её показывают редакторы кода, встроенные в среды программирования).

Ответ: CDEFAA

Вопрос 5**.** Данный вопрос рассчитан на «продвинутых» участников, которые по какой-то причине решают за свой 7 или 8 класс, а не за девятый. Для вычисления ответа нужно написать программу. При этом решение совсем «в лоб» с тремя вложенными циклами будет работать слишком долго. Чтобы избавиться от третьего вложенного цикла, можно накапливать количество встретившихся букв ‘A’ в отдельной переменной. Пример решения на C++:

```
#include<bits/stdc++.h>

using namespace std;

int main() {
   string s = "";
```

```
    for (int i = 0; i < 15; i++) {
        s = s + s;
        s.push_back('A' + i);
    }
    long long cnt = 0;
    for (int i = 0; i < s.length(); i++) {
        int cA = 0;
        for (int j = i; j < s.length(); j++) {
            if (s[j] == 'A')
                cA++;
            if (2*cA > j - i + 1)
                cnt++;
        }
    }
    cout << cnt;
}
```

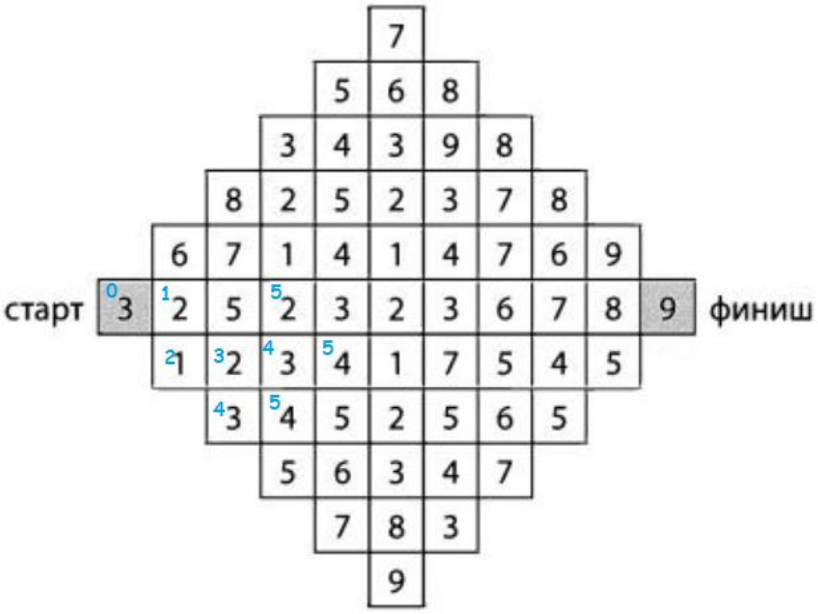
Ответ программы: 310058443

Все ответы к задаче: 63 10 8192 CDEFAA 310058443

Задача 2. Лабиринт (100 баллов)

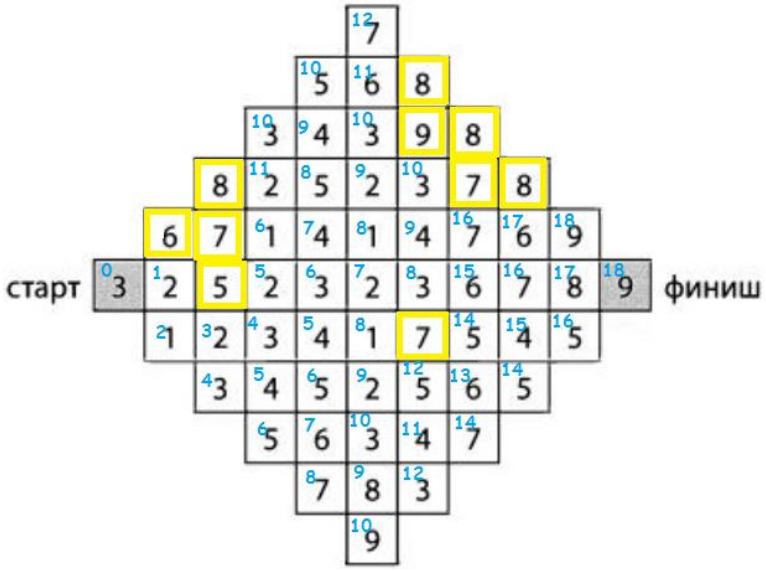
Вопрос 1. Можно скопировать рисунок лабиринта в графический редактор (например, Paint) и отмечать, какие клетки мы можем посетить за один, два, три и так далее шагов.

Чтобы не запутаться, можно делать это разными цветами (за 1 ход – один цвет, за 2 ходов – другой, и т.д.) либо писать в клетке число шагов, например:



Ответ = 9.

Вопрос 2**.** Получить ответ можно, продолжив действия из предыдущего пункта и посмотрев в конце, какие клетки не посещены.

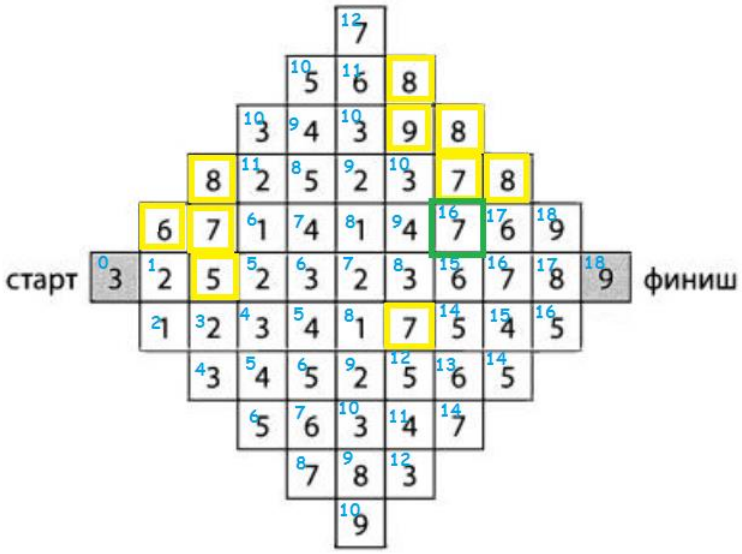


Ответ равен 10.

Вопрос 3**.** Число в финишной клетке – это кратчайшее число шагов, чтобы дойти до неё. А клеток на этом пути на одну больше – то есть, 19. Поскольку лабиринт небольшой, то ответ можно найти и «методом задумчивого взгляда».

Ответ: 19.

Вопрос 4**.** Можно заметить, что у нас есть «более прямой» путь, почти доходящий до финиша, но ему мешает одна клетка (выделена зелёным).



Если в ней поменять число на 5, то из клетки слева можно будет сходить вправо, затем вниз, и затем по прямой дойти до финиша. Число клеток на таком пути равно 15. Более короткого пути в данном лабиринте не получить (проверено программой, реализующей перебор всех вариантов). Ответы на все вопросы задачи: 9 10 19 15

Задача 3. Счастливые билеты (100 баллов)

Заметим, что в левой (и правой) половинке номера все цифры должны быть уникальны. Всего имеется $C(10, N/2)$ способов выбрать цифры для левой половины, эти же цифры будут и в правой (здесь C – это количество сочетаний). Также в левой и в правой половине цифры можно произвольно переставлять – это $(N/2)!$ перестановок в каждой половине. Получается $C(10, N/2) * (N/2)! * (N/2)!$

Немного упростим: $10! / ((N/2)! * (10 – N/2)!) * (N/2)! * (N/2)! =$

$$10! * (N/2)! / (10 – N/2)!$$

Пример решения на языке Python:

```
from math import factorial
n = int(input())
print(factorial(10) * factorial(n//2) // factorial(10-n//2))
```

Задача 4. Максимальная выручка (100 баллов)

Очевидно, что цена торта будет совпадать с максимальной ценой одного из покупателей. Пусть мы рассматриваем покупателя с номером i и ценой P_i. По этой цене купит торт он, а также все следующие за ним, поэтому мы сможем сразу сосчитать выручку как $P_i * (n – i + 1)$. Примечание: если нумеровать покупателей с нуля, то единицу добавлять не нужно. Таким образом мы можем выбрать максимальную выручку, а также цену, на которой она достигается.

Заметим, что в этой задаче даже нет необходимости в использовании массивов – можно обновлять ответ по мере чтения входных данных. Пример решения на Python:

```
n = int(input())
best_price = 0
best_revenue = 0
for i in range(n):
    p = int(input())
    if p * (n - i) > best_revenue:
        best_revenue = p * (n - i)
        best_price = p
print(best_price)
```

Задача 5. Ремонт мотора (100 баллов)

В первой подзадаче можно рассмотреть различные случаи с помощью условного оператора.

Во второй задаче можно реализовать перебор (в Питоне для этого удобно использовать стандартную функцию permutations, в C++ – функцию next_permutation).

Полное решение задачи – следующий жадный алгоритм. Отсортируем оба массива. Для каждого следующего элемента *a[i]* нужно взять самый маленький ещё не взятый элемент массива *b*, который больше или равен *a[i]*.

Для эффективной реализации данной идеи можно использовать метод двух указателей. Под словом "указатель" здесь понимается индекс массива. Пусть индекс *i* движется по массиву *a*, индекс *j* — по массиву *b*. Для каждого следующего *i* будем увеличивать *j* до тех пор, пока не получим $b[j] \geq a[i]$, при этом к ответу прибавляем единицу. Поскольку индекс *j* никогда не уменьшается, то внутренний цикл суммарно делает не более *m* шагов, и вычислительная сложность алгоритма составляет $O(n+m)$.

Обоснование правильности. Заметим, что после сортировки всегда выгодно брать элемент *a[1]* и искать к нему пару из *b*. Действительно — если в оптимальном решении нет *a[1]*, то любой другой взятый элемент из *a* можно заменить на *a[1]*, при этом решение останется оптимальным. Аналогично получаем, что на следующем шаге выгодно искать пару для *a[2]*, затем для *a[3]*, и так далее. Осталось заметить, что при поиске в массиве *b* пары для очередного *a[i]* выгодно брать элемент из *b* как можно меньше, чтобы бо́льшие значения оставить в запасе для следующих шагов.

Пример решения на Python:

```
n = int(input())
a = [0] * n
for i in range(n):
   a[i] = int(input())
m = int(input())
b = [0] * m
for i in range(m):
   b[i] = int(input())
a.sort()
b.sort()
ans = 0
j = 0
for i in range(n):
   while j < m and b[j] < a[i]:
```

```
        j += 1
    if j == m:
        break
    ans += 1
    j += 1
print(ans)
```