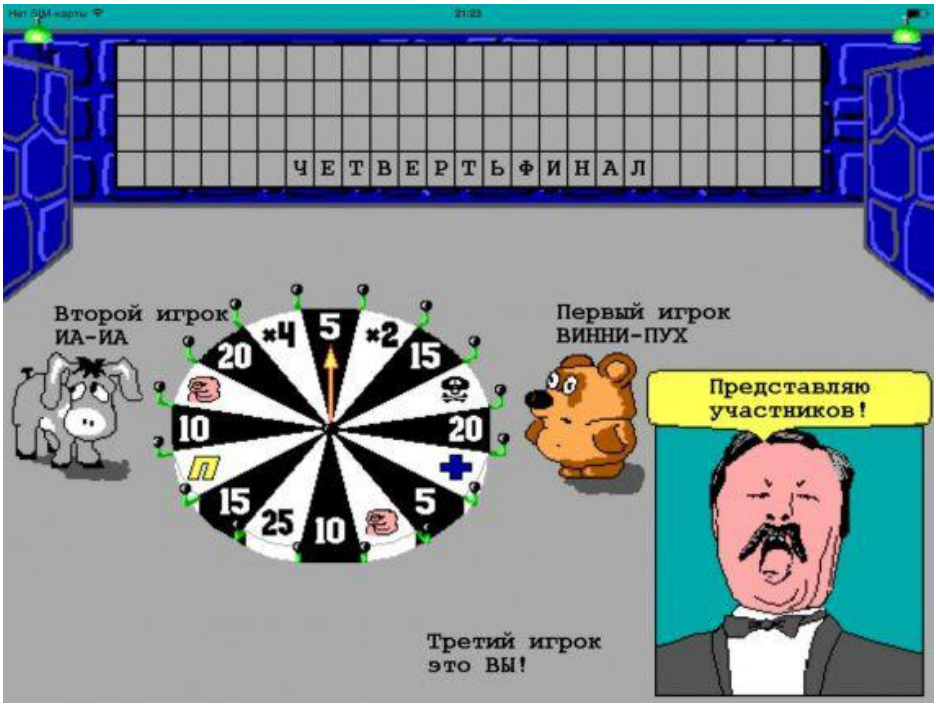
Олимпиада школьников

ТИИМ - Технологии. Интеллект. Информатика. Математика

(Задания заключительного тура по информатике. 2020-2021 учебный год)

14 мая 2021 г.



Задания заключительного тура олимпиады с решениями

Задача 1. Вращайте барабан

Два друга играют в игру: вращают барабан, пока не выпадет сектор "Деньги". Если выпадает сектор "Приз его никто не хочет забирать и игра продолжается: барабан вращает следующий игрок.

Сектора барабана заданы входной строкой, например "ДПДПДДД"

Если на барабане все сектора заняты буквой "П вероятность выигрыша равна 0Если все сектора заняты буквой "Д то первый игрок выигрывает с вероятностью 100

Входные данные:

Строка длиной >=3 символов, состоящая из букв "П"и "Д"

Выходные данные:

Вероятность выигрыша первого игрока в процентах, только число, без символа

Время работы программы: 1 сек

Набор тестовых данных:

Входные данные	Результат работы программы
ДДДДД	100
ДПП	60
ДПД	75
ППППП	0

Решение задачи на языке Python:

Если n - количество секторов, а m - количество секторов с призом, искомая вероятность вычисляется, как сумма бесконечной геометрической прогрессии с первым членом, равным $(n-m)/n$ и знаменателем, равным m^2/n^2

```
def main():
    s = input()
    n = len(s)
    m = len(s.replace('Д',''))
    if m == n:
        res = 0
    else:
        res = round(n/(n+m)*100)
    print(res)

main()
```

Задача 2. Новый шифр Одинокой горы

После того, как пещера Смауга была взломана хоббитами, он решил пересмотреть свои взгляды на безопасность. В новой системе безопасности шифр состоит из 4 цифр, меняется каждую секунду, и вычисляется следующим образом:

1-я цифра - последняя цифра числа Фибоначчи fib(date), где date=год*10000+месяц*100+день

2-я цифра - последняя цифра числа Фибоначчи fib(hours), где hours - часы
3-я цифра - последняя цифра числа Фибоначчи fib(minutes), где minutes- минуты
4-я цифра - последняя цифра числа Фибоначчи fib(seconds), где seconds- секунды
Числа Фибоначчи — элементы числовой последовательности 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ... ,
в которой $F(0) = 0$, $F(1) = 1$, а каждое последующее число равно сумме двух предыдущих чисел
Входные данные:
Текущие дата и время в формате дд.мм.гггг чч:мм:сс
Выходные данные:
код доступа
Время работы программы: 1 сек
Набор тестовых данных:

Входные данные	Результат работы программы
02.01.1921 09:01:01	1411
04.04.2021 10:00:00	3500
09.03.2004 15:30:00	4000
15.01.2021 10:00:00	500
14.06.1999 11:12:13	2943

Решение задачи на языке Python:
Основная идея быстрого вычисления шифра - найти период, с которым повторяются последние цифры чисел Фибоначчи.
После того, как этот период получен, мы можем быстро вычислить последнюю цифру числа Фибоначчи, не вычисляя его непосредственно.

```
def last_fib_digit(n):
    return [0,1,1,2,3,5,8,3,1,4,5,9,4,3,7,0,7,7,4,1,5,
    6,1,7,8,5,3,8,1,9,0,9,9,8,7,5,2,7,9,6,
    5,1,6,7,3,0,3,3,6,9,5,4,9,3,2,5,7,2,9,1][n%60]
```

```
def main():
    (the_date,the_time) = input().split()
    (d,m,y) = map(int,the_date.split('.'))
    the_date_num = 10000*y+100*m+d
    (h,min,sec) =  map(int,the_time.split(':'))

    print (int(str(last_fib_digit(the_date_num))+str(last_fib_digit(
    h))+str(last_fib_digit(min))+str(last_fib_digit(sec))))

main()
```

Задача 3. Небоскребы

Небоскребы(Башни) - японская головоломка, впервые представленная на World Puzzle Championship в 1992 году.
На поле 4*4 нужно расположить 4 небоскреба, так, чтобы выполнялись следующие условия:

- Высота каждого небоскреба - целое число от 1 до 4
- В каждой строке и каждом столбце не может быть двух зданий одной высоты
- В некоторых строках и некоторых столбцах задано количество зданий, которые можно увидеть. Числа расположены по периметру квадрата.
- Высокое здание закрывает более низкие, и их увидеть нельзя
- Головоломка имеет только одно решение

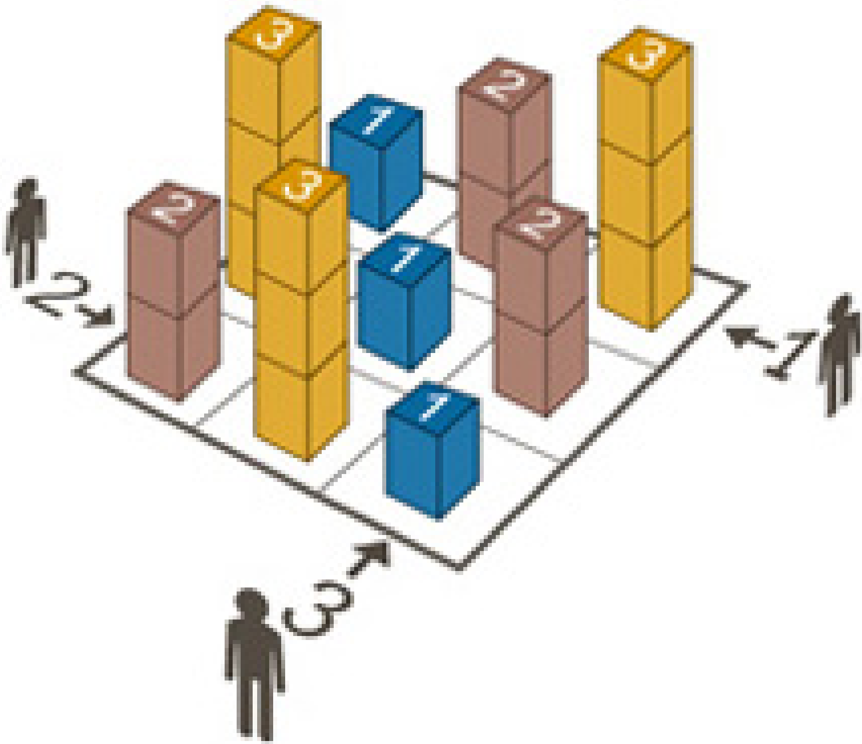
Пример
Рассмотрим строку ниже. Задано, что слева видны 4 небоскреба, а справа 1

4					1

Есть только один способ расположить небоскребы так, чтобы это условие выполнялось

4	1	2	3	4	1

Пример головоломки 4*4



			3		
					2
		2	2		

			3		
	2	3	1	4	
	1	4	2	3	2
	3	1	4	2	
	4	2	3	1	
		2	2		

Входные данные:
Строка из 16 чисел, обозначающих количество видимых зданий Числа расположены по часовой стрелке. Если стоит 0, значит количество видимых зданий для этой строки/столбца не задано

	0	1	2	3	
14					5
13					6
12					7
	11	10	9	8	

Выходные данные:
Решение головоломки - 4 строки по 4 числа
Ограничение времени работы программы:
1 с
Набор тестовых данных:

Входные данные	Результат работы программы
2 2 1 3 2 2 3 1 1 2 2 3 3 2 1 3 4 2 1 3 3 4 2 1 2 1 3 4	1 3 4 2
0 0 1 2 0 2 0 0 0 3 0 0 0 1 0 0 3 4 1 2 4 2 3 1 1 3 2 4	2 1 4 3
1 2 4 2 2 1 3 2 3 1 2 3 3 2 2 1 3 1 2 4 1 4 3 2 2 3 4 1	4 2 1 3

Решение задачи на языке Python:

```
from itertools import permutations, chain

def solve_puzzle (clues):
    size = 4
```

```
    for poss in permutations(permutations(range(1, size+1), size),
    size):
        for i in range(size):
            if len(set(row[i] for row in poss)) < size:
                break
        else:
            cols_top = [[row[i] for row in poss] for i in range(size
    )]
            rows_right = [list(reversed(row)) for row in poss]
            cols_btm = [[row[i] for row in reversed(poss)] for i in
    reversed(range(size))]
            rows_left = list(reversed(poss))
            for i, row in enumerate(chain(cols_top, rows_right,
    cols_btm, rows_left)):
                if not clues[i]:
                    continue
                visible = 0
                for j, v in enumerate(row):
                    visible += v >= max(row[:j+1])
                if visible != clues[i]:
                    break
            else:
                return poss

clues =  list(map(int, input().split()) )
res = solve_puzzle(clues)

for r in res:
    print(' '.join(map(str,r)))
```

Задача 4. Игра в пирамидку

Младший брат Пети, Лева учит цифры и восторгается египетскими пирамидами, а Петя учится писать программы. Их новое развлечение - Лева складывает из карточек с цифрами число, а Петя должен через секунду сказать, сколько кубов потребуется, чтобы собрать пирамидку такого объема.

Пирамида собирается следующим образом: в основании лежит куб с длиной ребра, равной n, объемом n^3, следом идет куб объемом $(n-1)^3$ и так далее. Самый верхний - куб с ребром 1

Входные данные:

Объем пирамиды(сумма объемов всех кубов) -набор цифр, не более 15

Выходные данные:

количество кубов, использованных для пирамидки, если такое построение возможно, -1, если невозможно

Время работы программы: 1 сек

Набор тестовых данных:

Входные данные	Результат работы программы
1071225	45
9	2
16	-1
36	3
20864367469009	2
135440716410000	4824
3408015058561	1921

Решение задачи на языке Python:

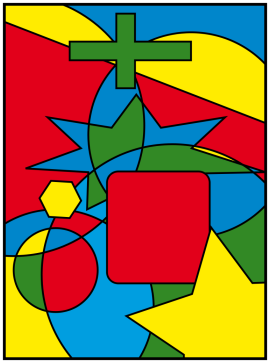
Для решения задачи можно использовать известную формулу суммы первых n кубов:

$$1^3 + 2^3 + 3^3 + \ldots + n^3 = \left[\frac{n\,(n+1)}{2}\right]^2$$

```
def solve_faster(m):
    n = ( m**0.5*2 + 0.25 )**0.5 - 0.5
    if n.is_integer() :
        return int(n)
    else:
        return -1

m = int(input())
print(solve_faster(m))
```

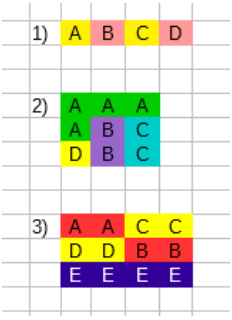
Задача 5. Помогите Даше раскрасить карту



Даша выучила теорему о четырех красках и практикуется в раскрашивании различных карт.

Теорема о четырёх красках — теорема, которая утверждает, что всякую расположенную на плоскости или на сфере карту можно раскрасить не более чем четырьмя разными цветами (красками) так, чтобы любые две области с общим участком границы были раскрашены в разные цвета. При этом области должны быть односвязными, а под общим участком границы понимается часть линии, то есть стыки нескольких областей в одной точке не считаются общей границей для них.

Карты, которые раскрашивает Даша, заданы в виде массива символов. Каждой области соответствует свой символ и раскраска может выглядеть следующим образом:



Входные данные

Набор строк, представляющих из себя карту. Метки областей регистрозависимы, т.е. A и a означают разные области

Выходные данные

Количество цветов, необходимое для раскраски

Время работы программы: 1 сек
Набор тестовых данных:

Входные данные	Результат работы программы
abcd eeee klmn	3
AAAA AAAA AAAA AAAA	1
KKKK Kkkk Kkkk KKKK	2

~~ ~!!!!!!!!!!!!!!!!!!~ ~##################!~ ~#&&&&&&&&&&&&&&&&&#!~ ~#&((((((((((((((&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&($$$$$$$$$$$$(&#!~ ~#&()$$$$$$$$$$(&#!~ ~#%(((((((((((((&#!~ ~#%(((&&&&&&&&&&&&&#!~ ~#%%%%%%%%#######!~ ~######!!!!!!!!!!!!!~ ~~~~~~~~~~~~~~~~~~~~	3
ABCD ABCD ABCD ABCD	2

Решение задачи на языке Python:

```
from itertools import product

def color(testmap):
    m = {r:i for i,r in enumerate(set(testmap.replace('\n','')))}
```

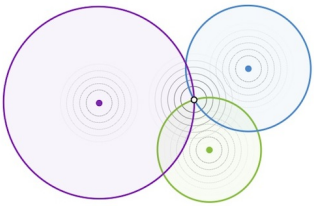
```
    g = testmap.strip().split('\n')
    h,w = len(g),len(g[0])
    adj = set()
    for y,x in product(range(h),range(w)):
        if y+1<h and g[y][x] != g[y+1][x]: adj.add((g[y][x],g[y+1][x
    ]))
        if x+1<w and g[y][x] != g[y][x+1]: adj.add((g[y][x],g[y][x
    +1]))
    for s in range(4):
        for p in product(range(s),repeat=len(m)):
            if all(p[m[a]] != p[m[b]] for a,b in adj):
                return s
    return 4

test = ''

while True:
    try:
        test += input()+"\n"
    except:
        break

print(color(test))
```

Задача 6. Оборона территории



Горожане построили для обороны поселения башни. На каждой башне расположен наблюдатель с известным радиусом обзора

Расположение башен задано их координатами на плоскости

Найти площадь территории, обозреваемой наблюдателями

Входные данные:

Список башен вида

x_1 y_1 r_1

x_2 y_2 r_2

...

x_n y_n r_n

где x_k- первая координата башни, y_k - вторая координата башни, r_k - радиус обзора башни

Максимальное расстояние между башнями для любого из наборов тестовых данных<= 1500 м

10<=Радиус обзора каждой башни <=500 м

1<=Количество башен <=20

Выходные данные:

Площадь обозреваемой территории, округленная до тысяч

Время работы программы:

Время выполнения программы на сервере ограничено 10 секундами

Набор тестовых данных:

Входные данные	Результат работы программы
0 0 50 -100 -100 50 0 -100 50 -100 0 50	31000
0 0 500 0 100 400 0 200 300 0 300 200	785000
0 100 200 100 100 200	165000
-500 -500 400 -500 500 400 500 -500 400 500 500 400	2011000
0 0 100 0 200 100 0 400 100 0 600 100 0 800 100 200 0 100 200 200 100 200 400 100 200 600 100 200 800 100 400 0 100 400 200 100 400 400 100 400 600 100 400 800 100	471000

Решение задачи на языке Python с использованием метода Монте-Карло:

Автор решения - призер заключительного тура олимпиады по информатике, Алексей Ингеройнен, 9 класс, г. Кострома

```
import sys
import random


def main():
    check = sys.stdin.read()
    O = [list(map(int, i.strip().split())) for i in check.split('\n')]
    xmin = ymin =  10**9
    xmax = ymax = - 10**9
    z = []
    for _ in O:
        if _ == []:
            continue
        z.append(_)
        if _[0]-_[2]<xmin: xmin = _[0]-_[2]
        if _[1]-_[2]<ymin: ymin = _[1]-_[2]
        if _[0]+_[2]>xmax: xmax = _[0]+_[2]
        if _[1]+_[2]>ymax: ymax = _[1]+_[2]
    n = len(z)
    dx = xmax - xmin
    dy = ymax - ymin
    insie_points = 0
    d = 1180000
    for i in range(d):
        x = xmin + int(dx * random.random())
        y = ymin + int(dy * random.random())
        inside = False
        for j in range(n):
            if (x-z[j][0])**2 + (y-z[j][1])**2 <= z[j][2]**2:
                inside = True
                break
        if inside:
            insie_points += 1

    print(round(dx * dy / d * insie_points / 1000) * 1000)
    return -1

main()
```