Муниципальный этап всероссийской олимпиады школьников по информатике 2022/2023 года в Камчатском крае.

I. Рекомендации по оцениванию

Решением задачи является программа, написанная на одном из доступных на олимпиаде языков программирования. Для проверки и оценивания решений жюри использует автоматическую тестирующую систему. На проверку отправляется исходный текст программы. При отправке решения на проверку участник указывает, с использованием какого языка программирования и компилятора выполнено решение. Разные решения, отправленные на проверку, могут использовать разные языки программирования и/или компиляторы.

Программа запускается на тестах. Для каждого теста, на котором был выполнен запуск, устанавливается результат выполнения на этом тесте. Верный ответ на тест, выданный при соблюдении указанных в условии задачи ограничений, соответствует результату OK.

Когда программа запускается, ей указанным в условии задачи способом передаются входные данные. Для ввода данных используется стандартный поток ввода.

В условии каждой задачи приведены примеры входных и выходных данных для этой задачи. Решение участника запускается на тестах из примеров, приведенных в условии задачи, результат работы на этих тестах сообщается участнику.

Каждая задача оценивается максимум в 100 баллов. Каждый пройденный тест (за исключением тестов из условия) оценивается в 5 баллов. Оценка за задачу вычисляется по формуле: (кол-во пройденных тестов) × 5.

II. Краткие рекомендации по решению задач, примеры решений

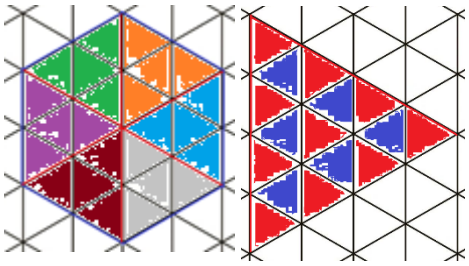
В состав пакета для каждой задачи входят решения (в электронном виде) на языках программирования Python и Си++.

Ниже представлены краткие рекомендации к решению задач.

Задача 1. Шестиугольник
Задача на вывод формулы.

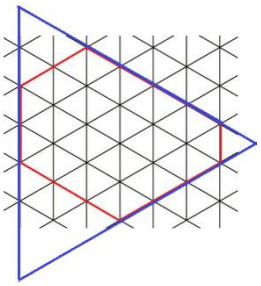
Разберем частный случай.

1) Все стороны шестиугольника равны.



Тогда его можно разбить на 6 равных частей, так, как показано на рисунке. Каждую часть можно разбить по слоям от центра к стороне на красные и синие и посчитать: красных будет 1 + 2 + 3 + … + a = a × (a + 1) / 2 (арифметическая прогрессия), а синих будет 1 + 2 + 3 + … + (a – 1) = a × (a - 1) / 2. Всего их будет a × (a + 1) / 2 + a × (a - 1) / 2 = a × a в каждом треугольнике, а в шестиугольнике 6 × a × a

2) Полное решение.



Как мы уже знаем, площадь правильного треугольника со стороной a равна a × a единичных треугольников. Достроим наш шестиугольник до треугольника (продолжив его не соседние стороны). Длина стороны получившегося треугольника будет равна сумме длин трёх любых последовательных сторон шестиугольника, а площадь интересующей нас центральной части выразится через разность площади большого треугольника и трёх маленьких, чьи стороны (а значит и площади) нам известны.

Пример программы (Python):

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
e = int(input())
f = int(input())
ans = (a + b + c) ** 2 - a ** 2 - c ** 2 - e ** 2
print(ans)
```

C++:

```
#include <iostream>
using namespace std;
int main()
{
   long long a,b,c,d,e,f;
   cin>>a>>b>>c>>d>>e>>f;
   cout<<(a+b+f)*(a+b+f)-b*b-f*f-d*d;
    return 0;
}
```

Задача 2. Длинная улица.
Задача на целочисленное деление, проверку условий.

Разберем частный случай.

1) $a \le 10^5$,

Используем идею динамического программирования. Будем хранить в массиве DP время, за которое можно добраться от дома Тимофея до дома Арсения. Количество элементов массива должно быть равно количеству домов на улице до остановки автобуса за домом Арсения. База динамики DP[0] = 0, остальные значения очень большие (например, 10^{18}). При первом проходе от меньших чисел к большим значение элемента массива будет равно значению предыдущего элемента массива, увеличенному на 1 (Тимофей идёт пешком). Но если индекс массива делится на 10 (есть остановка автобуса), то значение элемента массива равно его индексу, уменьшенному в 10 раз. (Тимофей доехал на автобусе). При втором проходе от больших чисел к меньшим мы проверяем, не быстрее ли добраться до текущего дома, вернувшись назад (то есть, при DP[i] > DP[i + 1] + 1, и если да, то меняем значение элемента массива.

Пример программы:

```
n = int(input())
k = n + (10 - n % 10)
DP = [10 ** 18 for i in range(k + 1)]
DP[0] = 0
for i in range(1, k + 1):
    if i % 10 == 0:
        DP[i] = i // 10
    else:
        DP[i] = DP[i - 1] + 1
for i in range(k - 1, 0, -1):
    if DP[i] > DP[i + 1] + 1:
        DP[i] = DP[i + 1] + 1
print(DP[n])
print(DP)
```

Понятно, что при больших n формирование ответа будет занимать продолжительное время.

2) Полное решение.

Ясно, что Тимофею быстрее добраться на автобусе до одной из двух остановок, между которыми расположен дом Арсения. Если Тимофей выйдет на остановке до дома друга, то затратит на поездку `n // 10` минут, и ещё `n % 10` минут пройдёт пешком (этот же случай поддерживает вариант, когда дом Арсения точно напротив остановки). Если Тимофей выйдет на остановке после дома друга, то затратит на поездку `n // 10 + 1` минут, и ещё `(10 - n % 10)` минут пройдёт пешком в обратном направлении. Выберем из этих двух результатов наименьший.

Пример программы (Python):

```
n = int(input())
ans1 = n // 10 + n % 10
ans2 = n // 10 + 1 + (10 - n % 10)
ans = min(ans1, ans2)
print(ans)
```

C++:

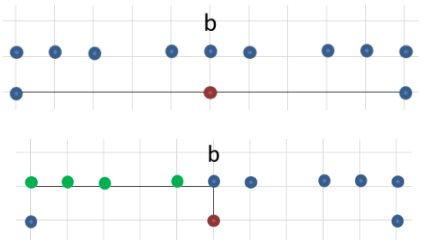
```
#include <iostream>
using namespace std;
int main()
{
    long long n;
    cin>>n;
    cout<<min(n/10+n%10,n/10+1+10-n%10);
```

```
    return 0;
}
```

Задача 3. Робот-гардеробщик.
Задача на сокращение перебора.

Разберем частные случаи.

1) ***a*** = 1.



Для достижения двух крючков (и пути назад), находящихся на одной с роботом горизонтальной линии, тому потребуется преодолеть расстояние 2 × b.

Для достижения всех крючков выше и левее места гардеробщика (выделены зелёным, их `b // 2`), тому придётся `b // 2` раз подняться на 1 и сместиться влево на (1 + 2 + 3 + … + `b // 2`). Такое же расстояние займёт обратный путь. Упростив выражение в скобках (это арифметическая прогрессия), получим `b` + (`b // 2`) × (`b // 2 + 1`).

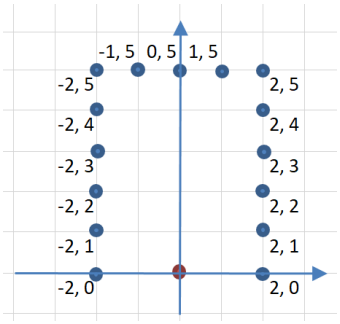
Такое же расстояние робот пройдёт для достижения всех крючков выше и правее своего места. Ещё 2 единицы расстояния до верхней клетки.

Всего `4 * b + 2 * (b // 2) * (b // 2 + 1) + 2`.

Пример программы:

```
a = int(input())
b = int(input())
ans = 4 * b + 2 * (b // 2) * (b // 2 + 1) + 2
print(ans)
```

2) ***a, b*** ≤ 10^5



Введём систему координат на плоскости и поместим робота в точку (0, 0). Переберём координаты всех крючков найдём сумму модулей их координат. Сумма этих сумм и будет ответом на вопрос задачи. Поскольку положение крючков имеет вертикальную ось симметрии, перебор можно уменьшить.

Пример программы:

```
a = int(input())
b = int(input())
ans = 0
for i in range(0, a + 1):
    ans += 4 * (b // 2 + i)
for i in range(1, b // 2):
    ans += 4 * (a + i)
ans += 2 * a
print(ans)
```

3) Полное решение.

Поскольку при переборе всех крючков при вычислении расстояний опять возникает арифметическая прогрессия, можно воспользоваться формулой суммы её членов и вообще отказаться от перебора. После упрощений формула будет выглядеть так:

```
ans = 4 * a * b + 2 * a * a + b * b // 2 + b
```

Пример программы (Python):

```
a = int(input())
b = int(input())
ans = 4 * a * b + 2 * a * a + b * b // 2 + b
print(ans)
```

C++:

```
#include <iostream>
using namespace std;
int main()
{
    long long a,b;
    long long s;
    cin>>a>>b;
    s=4*a*b+2*a*a+b*b/2+b;
    cout<<s;
    return 0;
}
```

Задача 4. Новогодняя гирлянда
Задача на массивы или строки.

1) Разберем частный случай ***y*** = ***x***.

В этом случае должны гореть все лампочки. Узнаем количество символов 0 в строке, это и будет ответом

```
n = int(input())
s = input()
ans = s.count(str(0))
print(ans)
```

2) Полное решение.

Подсчитаем количество единичек среди первых x символов строки. Исправим последние нули среди этих символов на единицы, если нужно. Далее будем двигать это «окно» вправо до конца строки на одну позицию, подсчитывая количество единиц и при недостатке (это может произойти, только если слева «ушла» единица, а справа пришёл ноль) заменяя новый ноль на единицу. Количество замен и будет ответом на вопрос задачи.

Пример программы (Python):

```
n = int(input())
L = list(input()[:-1])
x, y = map(int, input().split())

ans = 0
count = 0
for i in range(x):
    if L[i] == '1':
        count += 1
i = x - 1
while count < y:
    if L[i] == '0':
        L[i] = '1'
        count += 1
        ans += 1
    i -= 1
for i in range(x, n):
    if L[i - x] == '1' and L[i] == '0':
        count -= 1
        if count < y:
            L[i] = '1'
            count += 1
            ans += 1
    elif L[i - x] == '0' and L[i] == '1':
```

```
        count += 1
print(ans)
```

C++:

```
#include <iostream>
using namespace std;
int main()
{
    int n,x,y,k=0,k1=0,i;
    string s;
    cin>>n;
    cin>>s;
    cin>>x>>y;
    for (i=0;i<x;i++)
        if (s[i]=='1') k1++;
    for(i=x-1;k1<y;i--)
        if (s[i]=='0')
    {
        s[i]='1';
        k1++;
        k++;
    }
    for(i=x;i<n;i++)
    {
        if(s[i-x]=='1'&&s[i]=='0')
        {
            k1--;
            if (k1<y)
            {
                s[i]='1';
                k1++;
                k++;
            }
        }
        else
            if (s[i-x]=='0'&&s[i]=='1') k1++;
    }
    cout<<k;
    return 0;
}
```

Задача 5. Тимофей вычёркивает числа.

Задача на идею.

1) Разберем частный случай $n \leq 10^5$.

Будем моделировать процесс вычёркивания. Если число не будет удалено на очередном ходу, то оно приблизится к началу списка на `n // k` позиций.

Пример программы:

```
n = int(input())
k = int(input())
ans = 0
while n % k:
    n -= n // k
    ans += 1
print(ans + 1)
```

2) Полное решение.

Предыдущий подход хорошо работает при малых значениях k, например, при $k = 2$, каждый раз будет вычёркивается половина всех чисел и случай, когда n станет делиться на

k наступит очень быстро. А вот при ***k*** = 1000000 и ***n*** = 6999999 число итераций будет очень большим, при этом вначале каждый раз будет удаляться по 6 чисел.

Сократим этот перебор, узнав сколько раз придётся удалить ровно по 6 чисел. Это число равно 999999 // 6 + 1 (а в общем случае `n % k` делённое на `n // k` с округлением вверх), после этого будет удаляться уже по 5 чисел либо само число `n` будет удалено. При этом новая позиция числа ***n*** станет ближе к началу списка не менее, чем в `n // k` раз и её несложно определить.

Пример программы (Python):

```
n = int(input())
k = int(input())
ans = 1
while n % k:
    p = (n % k + n // k - 1) // (n // k)
    n -= p * (n // k)
    ans += p
print(ans)
```

C++:

```
#include <iostream>
using namespace std;
int main()
{
    long long n,k,ans=1,p;
    cin>>n>>k;
    while (n%k)
    {
        p=(n%k+n/k-1)/(n/k);
        n-=p*(n/k);
        ans+=p;
    }
    cout<<ans;
    return 0;
}
```