V Областная олимпиада школьников по информатике
2020-2021 учебный год
9-10 классы
Отборочный тур

Разбор задач

Задача 1. Дни недели

Первую подзадачу можно решить с помощью нескольких условных операторов, поэтому она по силам даже для участников, только-только начавших изучать программирование.

Вторую подзадачу можно решить, создав массив с индексами от 0 до 6, содержащий названия дней. Для получения индексов искомых элементов применим операцию взятия остатка от деления на 7. Пример решения на языке Python:

```
wd = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
d = wd.index(input())
d1 = int(input())
d2 = int(input())
print(wd[(d - d1) % 7])
print(wd[(d + d2) % 7])
```

На других языках решение будет чуть сложнее, так как в большинстве других языков остаток от деления отрицательного числа на положительное считается не по «математическим» правилам, и может получиться отрицательным. Чтобы остаток был неотрицательным, добавим к нему семёрку. Но, если перед этим остаток был равен нулю, то теперь он станет равным 7 – а это выход за границу массива. Поэтому от результата нужно ещё раз взять остаток от деления на 7. Получается вот такая (на первый взгляд, странная) формула: ((d - d1) % 7 + 7) % 7. Пример решения на C++:

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  string wd[] = {"Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"};
  string dayName;
  int d1, d2;
  cin >> dayName >> d1 >> d2;
  int d = find(wd, wd+7, dayName) - wd;
  cout << wd[((d - d1) % 7 + 7) % 7] << '\n' << wd[(d + d2) % 7];
}
```

Задача 2. Выборы

Ответы для первых двух наборов N и M можно аккуратно сосчитать вручную, перебрав все варианты голосования. Для третьего набора это, в принципе, тоже можно сделать, но вариантов будет уже больше, и лучше применить один из следующих способов.

Для четвёртого набора N и M можно написать программу, выполняющую перебор вариантов. Заметим, что итог голосования корректен тогда и только тогда, когда за каждое предложение подано от 0 до M голосов, а всего подано голосов 2M.

Перебор можно написать так. Пусть рекурсивная функция принимает два аргумента: i – номер текущего предложения, sum – количество уже распределенных голосов. В цикле пробуем за i-е предложение отдать от 0 до $min(m, 2m\text{-}sum)$ голосов, и каждый раз рекурсивно вызываем эту же функцию с параметром i+1, чтобы она перебрала распределение оставшихся голосов для оставшихся предложений. Пример решения на языке C++:

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

long long ans = 0;
int n, m;

void bt(int i=1, int sum=0) {
  if (i > n) {
    if (sum == 2 * m)
      ans++;
    return;
  }
  for (int j = 0; j <= std::min(m, 2 * m - sum); j++)
    bt (i + 1, sum + j);
}

int main() {
  std::cin >> n >> m;
  bt();
  std::cout << ans << '\n';
}
```

Ответ для четвёртого набора N и M данная программа вычислит не более чем за несколько секунд (в зависимости от скорости компьютера). Однако, ответ для пятого набора этой программой не получить, так как ждать придётся слишком долго.

Чтобы найти последний ответ, для начала применим известный подход из комбинаторики. Пусть у нас есть $2M$ одинаковых шаров (где каждый шар обозначает один голос) и N-1 перегородка. Каждый итог выборов можно описать так: вначале идёт от 0 до M шаров, затем перегородка, снова от 0 до M шаров и перегородка, и так далее. Число шаров до первой перегородки – сколько голосов отдали за первое

предложение, между первой и второй – за второе, и так далее. Пример для N=4, M = 5 (шары показаны звёздочками):

|*||***

В данном примере три голоса отдано за первое предложение, четыре – за второе, ноль – за третье, три – за четвертое. Нам нужно посчитать количество таких допустимых вариантов.

Воспользуемся методом динамического программирования. Пусть *f(i, j)* – количество способов распределить первые i голосов, поставив j перегородок, причём последняя перегородка стоит ровно после *i*-го голоса. Тогда ответ на задачу равен *f(2m, n)*. *Примечание*: дополнительную *n*-ю перегородку мы ввели для удобства, она стоит в самом конце.

Чтобы вычислить *f(i, j)* через предыдущие значения *f*, нужно перебрать все допустимые позиции предыдущей перегородки *p* и сложить все *f(p, j-1)*. Пример решения на языке Python:

```
from functools import lru_cache

n, m = map(int, input().split())

@lru_cache(maxsize=11000)
def f(i, j):
  global n
  global m
  if j == 1:
    if i <= m:
      return 1
    else:
      return 0
  ans = 0
  for p in range(i, -1, -1):
    if i - p > m:
      break
    ans += f(p, j - 1)
  return ans

print(f(2 * m, n))
```

Правильные ответы: 10 15 44 77585586 460420211251.

Задача 3. Факториал

Для решения первой подзадачи можно на самом деле вычислять факториалы чисел 1, 2… и проверять, сколько в них нулей. Для этого нужно использовать либо язык Python (где нет жестких ограничений на длину целых чисел), либо Java с типом BigInteger, либо самостоятельно реализовать простую длинную арифметику.

Более красивый способ состоит в следующем. Количество нулей на конце факториала определяется количеством множителей 5 в нём (так как множителей 2 заведомо больше). Берём по очереди числа, начиная с единицы. Смотрим, сколько раз очередное число нацело разделится на пять, и это количество прибавляем к текущему количеству нулей. Как только количество нулей стало равно искомому, ответ получен. Если оно стало сразу больше искомого, решений нет. Пример решения первой подзадачи на языке Python:

```
k = int(input())
z = 0
i = 0
while z < k:
    i += 1
    j = i
    fives = 0
    while j % 5 == 0:
      fives += 1
      j //= 5
    z += fives
if z >= k:
  if z == k:
    print(i)
  else:
    print(-1)
```

Чтобы решить вторую подзадачу, рассмотрим вначале, как быстро найти количество нулей на конце $n!$. Каждое число от 1 до n, делящееся на 5, добавляет ноль к концу n!. Также по дополнительному нулю добавляет каждое число от 1 до n, делящееся на 5^2, 5^3, и так далее. Таким образом, число нулей равно $n \ div \ 5+n \ div \ 5^2+n \ div \ 5^3+\ldots$, где под операцией *div* понимается целочисленное деление. Слагаемых будет немного, так как степени пятёрки быстро растут, а брать их больше n не имеет смысла.

Теперь воспользуемся методом двоичного поиска. В качестве левой границы можно взять 1, правой – 5k. На каждом шаге берём среднее значение из этого интервала, считаем число нулей и сдвигаем либо левую границу к центру, либо правую. Если на каком-то шаге получилось ровно k нулей, то стоит продолжить поиск в левой части – ведь может существовать ответ и с меньшим значением, а нам по условию задачи нужно минимальное. Пример решения на Python:

```
k = int(input())
left = 1
right = 5 * k
ans = -1
while left <= right:
    mid = (left + right) // 2
    z = 0
```

```
    p = 5
    while p <= mid:
        z += mid // p
        p *= 5
    if z == k:
        ans = mid
    if z >= k:
        right = mid - 1
    else:
        left = mid + 1
print(ans)
```

Пример ещё одного решения на C++ с немного другой реализацией двоичного поиска:

```
#include <cstdio>

long long factZeros(long long n) {
    long long sum = 0;
    while (n >= 5) {
        n /= 5;
        sum += n;
    }
    return sum;
}

int main() {
    int nZeros;
    scanf("%d", &nZeros);
    long long left = 4; // < nZeros
    long long right = 5LL * nZeros; // >= nZeros
    while (left + 1 < right) {
        long long mid = (left + right) / 2;
        if (factZeros(mid) < nZeros) {
            left = mid;
        } else {
            right = mid;
        }
    }
    if (factZeros(right) == nZeros) {
        printf("%lld", right);
    } else {
        printf("-1");
    }
    return 0;
}
```

Задача 4. Сумма дробей.

Нам нужно найти такие знаменатели p и q, что:

$$1/a+1/b+1/c = 1/p+1/q.$$

Заметим, что p и q одновременно не могут быть слишком большими – иначе правая сумма будет меньше левой. Если взять максимальные по условию значения $a=b=c=1000$, то сумма слева

равна 0.003. Если взять p=q=667, то $1/p+1/q < 0.003$. Таким образом, хотя бы один из них должен быть меньше 667. Пусть это будет p.

Тогда решение выглядит так. Перебираем возможные значения p, выражаем q из формулы выше. Если получилось целое значение больше нуля, то ответ найден. Пример решения на C++:

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

int main() {
   long long a, b, c;
   std::cin >> a >> b >> c;
   for (long long p = 1; p < 667; p++) {
     long long d = (b*c+a*c+a*b)*p - a*b*c;
     if (d > 0 && a*b*c*p % d == 0) {
        long long q = a*b*c*p / d;
        std::cout << p << " " << q << '\n';
        return 0;
     }
   }
   std::cout << "-1\n";
}
```