Всероссийская олимпиада школьников по информатике
Вологодская область
II (муниципальный) этап
2018-2019 учебный год
9 - 11 классы
Методические рекомендации по разбору задач

Задача 1. Подготовка к ЕГЭ (100 баллов)

Первую подзадачу можно решить простым перебором всех пар. Заметим, что произведение может быть порядка 10^{18}, поэтому необходимо использовать 64-битный целый тип.

Чтобы понять, что перед нами вторая подзадача, сосчитаем количество делителей *S*. Если их два (само *S* и единица), то *S* – простое число, и это подзадача №2. Для её решения сосчитаем количество входных чисел, которые делятся на *S* – пусть их количество равно *cnt*. Любая пара из двух таких чисел делится на *S*. Также на *S* делится любая пара, в которой только одно из чисел делится на *S*. В итоге ответ равен:

$$cnt \cdot (cnt - 1) / 2 + cnt \cdot (N - cnt)$$

Если S имеет 4 различных делителя, то это произведение двух простых чисел, и перед нами третья подзадача (как раз то, что встречалось в одном из вариантов ЕГЭ). Сосчитаем следующие величины:

cnt_{12} – количество входных чисел, которые делятся на оба простых множителя.

cnt_1 – количество входных чисел, которые делятся на первый простой множитель, но не делятся на второй.

cnt_2 – количество входных чисел, которые делятся на второй простой множитель, но не делятся на первый.

Искомое количество включает в себя:

- пары чисел, каждое из которых делится на оба простых множителя.

- пары, где одно число делится на оба простых множителя, а другое таким свойством не обладает.

- пары, где первое число делится на первый простой множитель (но не делится на второй), а второе число – наоборот

В итоге получается ответ:

$$cnt_{12} * (cnt_{12} - 1) / 2 + cnt_{12} * (N - cnt_{12}) + cnt_1 * cnt_2$$

Для решения четвёртой подзадачи воспользуемся тем фактом, что если (a · b) mod S = 0, то ((a mod S) · (b mod S)) mod S = 0. Найдём для каждого числа остаток от деления на *S* и для каждого остатка запомним в массиве, сколько раз он встретился. Всего будет S различных остатков (от 0 до *S*-1). Теперь двумя вложенным циклами пройдём по этому массиву и сосчитаем ответ.

Пример решения на C++:

```
#include <stdio.h>
#include <vector>

int main() {
  int n, s, a;
  scanf("%d %d", &n, &s);
  std::vector<long long> c(s);
  for (int i = 0; i < n; i++) {
    scanf("%d", &a);
    c[a % s]++;
  }
  long long ans = 0;
  for (long long i = 0; i < s; i++) {
    if (i * i % s == 0)
      ans += c[i] * (c[i] - 1) / 2;
    for (long long j = i + 1; j < s; j++)
      if (i * j % s == 0)
          ans += c[i] * c[j];
  }
  printf("%lld\n", ans);
}
```

Заметим, что данное решение позволяет проходит четыре первые подзадачи и набирает 80 баллов.

Пятую и шестую подзадачи можно решить похожим образом. Вместо того чтобы считать, сколько раз встречались остатки от деления входных чисел на *S*, мы сосчитаем, сколько раз встречались наибольшие общие делители от каждого входного числа и *S*. Действительно, ведь если *a·b* делится на *S*, то и *НОД(a,S)·НОД(b,S)* делится на *S*. И наоборот, если *a·b* не делится на *S*, то и *НОД(a,S)·НОД(b,S)* не делится на *S*.

Оценим, сколько различных НОД может получиться. Количество делителей натурального числа грубо можно оценить как корень кубический из него, умноженный на небольшую константу. Более точную оценку можно выполнить экспериментально: при *S* до 10^9 количество делителей *S* не превосходит 2000.

Пример решения на C++:

```
#include <stdio.h>
#include <map>
```

```
long long gcd(long long a, long long b) {
  while (b != 0) {
    int c = a % b;
    a = b;
    b = c;
  }
  return a;
}

int main() {
  int n, s;
  scanf("%d %d", &n, &s);
  std::map<long long, long long> count;
  for (int i = 0; i < n; i++) {
    int a;
    scanf("%d", &a);
    count[gcd(a, s)]++;
  }
  long long ans = 0;
  for (auto r1 : count) {
    if (r1.first * r1.first % s == 0) {
      ans += r1.second * (r1.second - 1) / 2;
    }
    for (auto r2 : count) {
      if (r1.first < r2.first && r1.first * r2.first % s == 0) {
        ans += r1.second * r2.second;
      }
    }
  }
  printf("%lld\n", ans);
}
```

В заключение отметим, почему в условии отдельно выделена подзадача 4, где $S \leq 10^6$. Дело в том, что существует алгоритм, который решает четвёртую подзадачу, но не решает пятую, и кто-то из участников мог до него догадаться. Однако, этот способ сложнее, чем рассмотренное выше решение, поэтому здесь его описывать не будем.

Задача 2. Числовая последовательность (100 баллов)

Первую подзадачу можно решить, просто дописывая в конец строки (или массива) цифры чисел N, N+1, …, до тех пор, пока не наберётся нужная длина (максимальная из запрошенных). Дальше просто выведем ответы.

Для решения второй подзадачи будем пропускать большие группы чисел. Например, пусть исходное число равно 567. Посмотрим, какая добавится длина, если пропустить все оставшиеся трёхзначные числа до 999 включительно. Если мы не превысим требуемую длину, то пропустим всю эту группу чисел и придём к аналогичной задаче для N=1000. Если же требуемая длина будет

превышена, то сосчитаем, сколько максимум чисел данной разрядности можно пропустить. Например, если N = 567, и нам нужна цифра K=8, то мы можем пропустить (K div 3 = 2) числа и перейти к задаче с N=569 и K=2. После этого останется только аккуратно извлечь нужную цифру.

Пример решения на C++:

```
#include <iostream>

void solve(unsigned long long n, unsigned long long k) {
  unsigned long long p10 = 1, len = 0;
  while (p10 - 1 < n) {
    p10 *= 10; ++len;
  }
  while (len * (p10 - n) < k) {
    k -= len * (p10 - n);
    n = p10; p10 *= 10; ++len;
  }
  unsigned long long skip = (k - 1) / len;
  k -= skip * len;
  n += skip;
  for (unsigned long long i = 0; i < len - k; i++) {
     n /= 10;
  }
  std::cout << n % 10 << std::endl;
}

int main() {
  unsigned long long n;
  int m;
  std::cin >> n >> m;
  for (; m > 0; m--) {
    unsigned long long k;
    std::cin >> k;
    solve(n, k);
  }
}
```

Задача 3. Согласные строки (100 баллов)

Первую подзадачу можно решить «в лоб» – тремя вложенными циклами. Перебираем начало и конец подстроки, после чего третьим циклом считаем количество гласных и согласных букв в ней.

Вторая подзадача решается немногим сложнее: нужно просто избавиться от третьего цикла. Для этого заведём два счётчика – для гласных и согласных букв – и будем их увеличивать прямо во втором цикле. Данное простое решение набирает 75 баллов.

Пример решения на C++:

```
#include <iostream>

const int NMAX = 1000000;
```

```
char s[NMAX + 1];

int main() {
  std::cin >> s;
  long long ans = 0;
  for (int i = 0; s[i] != 0; i++) {
    int vowels = 0, cons = 0;
    for (int j = i; s[j] != 0; j++) {
      if (s[j]=='a'||s[j]=='o'||s[j]=='u'||s[j]=='i'||s[j]=='e')
         vowels++;
      else
        cons++;
      if (cons > vowels) ans++;
    }
  }
  std::cout << ans << std::endl;
}
```

Третья подзадача заметно сложнее второй. Вначале упростим немного задачу – заменим согласные буквы на единицы, гласные – на ноли. Нужно найти число подстрок, где единиц больше, чем нолей.

В цикле пройдёмся по всем позициям исходной строки. Для каждой позиции *i* определим, сколько согласных подстрок кончаются в этой позиции, и просуммируем все эти числа.

Чтобы уметь это быстро делать, создадим следующую структуру данных. Представим себе массив *cnt* с индексами от *–n* до *n* (где n – длина строки *s*). Элементы этого массива хранят количество подстрок, кончающихся в текущей позиции, у которых разность между числом нулей и единиц равна соответственно -n, -n+1, …, 0, …, n-1, n.

К примеру, пусть *s* = “0010”, и текущая позиция *i*=3. Тогда *cnt*[−1] = 1, cnt[0] = 1, cnt[1] = 1, а остальные элементы равны нулю.

Как пересчитывать значения в массиве при переходе к следующей позиции *i*? Если *s*[*i*] = ’1’, то нужно сделать сдвиг всего массива вправо и увеличить *cnt*[1] на единицу (так как добавилась строка из одного символа). Аналогично, если *s*[*i*]=’0’, то нужно сделать сдвиг всего массива влево и увеличить *cnt*[-1] на единицу.

Заметим, что вместо сдвига целого массива достаточно сдвигать положение нулевого индекса: если нужен сдвиг влево, то нулевая точка смещается вправо, и наоборот. Таким образом, сдвиги легко делаются за время O(1).

Осталось научиться эффективно находить количество подстрок, которые кончаются в позиции *i*, и у которых единиц больше чем нулей. Это количество можно найти как сумму всех элементов массива *cnt* с индексами от 1 до *n*.

Однако, каждый раз искать сумму в цикле мы не можем себе позволить – такое решение не пройдёт по времени. Для ускорения

можно поддерживать в отдельной переменной сумму всех элементов массива с индексами от 1 до *n* и аккуратно обновлять её при переходе к следующей позиции (можно также воспользоваться какой-либо широко известной структурой данных для быстрого нахождения суммы на подмассиве - sqrt-декомпозиция, дерево отрезков, дерево Фенвика).

Пример решения на C++:

```
#include <iostream>
#include <vector>
#include <cstring>

char s[1000000 + 1];

int main() {
  std::cin >> s;
  int n = strlen(s);
  long long ans = 0, sum = 0;
  int z = 2 * n; // zero position
  std::vector<int> c(4 * n);
  for (int i = 0; i < n; i++) {
   if (s[i]=='a' || s[i]=='o' || s[i]=='u' || s[i]=='i' || s[i]=='e'){
      c[z]++;
      z++;
      sum -= c[z];
    } else {
      c[z]++;
      sum += c[z];
      z--;
    }
    ans += sum;
  }
  std::cout << ans << std::endl;
}
```

Задача 4. Инверсии (100 баллов)

Первую подзадачу можно решить различными способами – вплоть до полного перебора всех перестановок.

Для решения второй подзадачи можно воспользоваться следующим свойством алгоритма сортировки «пузырьком»: при сортировке по убыванию каждая замена увеличивает количество инверсий на одну. Отсюда алгоритм: берём перестановку 1, 2, …, n и сортируем её «пузырьком» по убыванию, останавливаемся при достижении нужного числа инверсий. Возможны и другие верные способы решения.

Третью подзадачу можно решить из следующих соображений. Пусть K – нужное число инверсий, N – оставшееся число элементов. Если $K \geq N-1$, то выводим самый большой элемент в ответ. Этот

элемент даст (N-1) инверсию с остальными (которые будут выведены после него), то есть мы пришли к задаче с $N'=N-1$, $K'=K-(N-1)$.

Если $K < N-1$, то выведем в ответ самый маленький элемент – тогда N уменьшится на 1, а число инверсий не изменится. В итоге получается очень простое и короткое решение.

Пример решения на C++:

```
#include <iostream>

int main() {
  int n;
  long long k;
  std::cin >> n >> k;
  int l = 1, r = n;
  while (l <= r) {
    if (k >= r - l){
      std::cout << r << ' ';
      k -= r - l;
      r--;
    } else {
      std::cout << l << ' ';
      l++;
    }
  }
}
```

Задача 5. Игра с числами (100 баллов)

Первую подзадачу можно решить, например, перебором всех возможных вариантов ходов игроков.

Для решения второй подзадачи можно воспользоваться методом динамического программирования. Заметим, что ответ полностью определяется тем, сколько у нас чисел при делении на 3 даёт остаток 0, 1 и 2. Создадим четырёхмерную логическую матрицу w, где $w[p][c0][c1][c2]$ = true, если игрок с номером p выиграет, когда на доске записано $c0$ чисел с остатком 0, $c1$ – с остатком 1 и $c2$ – с остатком 2. Аккуратно заполним эту матрицу от меньших индексов к большим и получим ответ (пример такого решения смотрите в файле sol_dp.cpp в архиве олимпиады).

Третью подзадачу можно решить, проведя небольшое исследование: возьмём решение для первой либо второй подзадачи и, пользуясь им, попробуем найти закономерность – как зависит ответ от M и N. Несложно заметить, что ответ зависит только от чётности ($N – M$), и находится по простой формуле $(N – M) \bmod 2 + 1$

Пример решения на C++:

```
#include <iostream>

int main() {
  int k, m, n;
  for (std::cin >> k; k > 0; k--) {
    std::cin >> m >> n;
      std::cout << (n - m) % 2 + 1 << std::endl;
  }
}
```

В заключение обоснуем вышеприведённую формулу. По сути, она говорит о том, что всегда выиграет тот игрок, которому достанется последний ход. Пусть в начале игры у нас есть три группы чисел: *c0* чисел с остатком 0, *c1* – с остатком 1, *c2* – с остатком 2. При этом размер любых двух групп отличается не более чем на 1 (так как изначально взяты все числа подряд от *M* до *N*). Рассмотрим 4 случая.

Случай 1: *c0* чётно, $c1 \neq c2$. Тогда первым ходом первый игрок сравнивает группы 1 и 2. Теперь если второй игрок стирает число из группы 1, то первый – из группы 2, и наоборот. Если же второй игрок стирает из группы 0, то и первый – тоже из 0 (она гарантированно не будет пустой). Ясно, что в итоге первый игрок выигрывает.

Случай 2: *c0* нечётно, $c1 = c2$. Первым ходом первый игрок стирает число из группы 0, дальше всё аналогично предыдущему случаю.

Случай 3: *c0* чётно, $c1 = c2$. Выиграет второй игрок. Его стратегия – вначале стирать всё из группы 0. Тогда после некоторого чётного числа шагов будет ожидаться следующий ход первого игрока, при этом $(c1 + c2)$ будет чётно. Тогда после хода первого игрока одно число среди *c1* и *c2* станет нечётным, другое – чётным. Далее стратегия второго игрока – каждый раз вычёркивать число из группы нечётного размера. В конечном итоге останется одна группа с двумя элементами, сумма которых на 3 не делится.

Случай 4: *c0* нечётно, $c1 \neq c2$. Выиграет второй игрок. Первыми своими ходами он сотрёт все числа из нулевой группы. После некоторого чётного числа шагов будет ожидаться ход первого игрока, при этом $(c1 + c2)$ будет чётно – дальше всё аналогично предыдущему случаю.

В материалах олимпиады можно найти также примеры и других решений задач.