V Областная олимпиада школьников по информатике на приз Губернатора
Вологодская область, 20 декабря 2020
9-10 класс, заключительный этап

Разбор задач

Задача A – Полоска
Если *K* нечётное, то ответ равен K/2+1. Умножим его на K%2, чтобы для чётных K ответом был ноль: (K/2+1)*(K % 2)
Если *K* чётное, то ответ равен N-K/2+1. Умножим его на 1-K%2, чтобы для нечётных K ответом был ноль: (N-K/2+1)*(1-K%2)
Теперь сложим обе формулы и получим окончательный ответ:

(K/2+1)*(K%2)+(N-K/2+1)*(1-K%2)

Задача B – Увеличиватель чисел
Первые два ответа можно при желании посчитать вручную. Третий ответ также при желании можно найти без программирования – например, с помощью электронных таблиц. Пусть f[i] – ответ для числа i. Тогда f[1] = 1, а остальные ответы находятся так:

- если i делится на 5, то f[i] = 0
- иначе, если i чётное, то f[i] = f[i/2] + f[i-1]
- иначе f[i] = f[i-1].

Вбив соответствующие формулы в электронную таблицу, мы получим ответ.
По сути, это метод динамического программирования: решения подзадач выражаются через решения ещё более мелких подзадач.
Ответ на четвёртый вопрос можно получить, написав программу, выполняющую вычисления согласно формулам выше. Пример такой программы на языке Python 3:

```
n = int(input())
d = [0] * (n + 1)
d[1] = 1
for i in range(2, n + 1):
    if i % 5 != 0:
        d[i] += d[i - 1]
        if i % 2 == 0:
            d[i] += d[i // 2]
print(d[n])
```

Для нахождения пятого ответа (а заодно и всех предыдущих) можно использовать "ленивое" динамическое программирование (известное также рекурсия с запоминанием или мемоизация). Напишем рекурсивную функцию, которая будет вычислять ответ по формулам выше. Вычисленные значения будем запоминать в словаре (ассоциативном массиве). Если функция вызывается с параметром, для которого ответ уже есть в словаре, то сразу его возвращаем, а иначе считаем его через рекурсивные вызовы. Пример решения на C++:

```
#include <bits/stdc++.h>
using namespace std;

map<long long, long long> ans;

long long f(long long n) {
  if (n % 5 == 0) return 0;
  auto it = ans.find(n);
  if (it != ans.end())
    return it->second;
  long long s = 0;
  if (n % 2 == 0)
```

```
    s = f(n / 2);
  s += f(n - 1);
  ans[n] = s;
  return s;
}

int main() {
  ans[1] = 1;
  cout << f(8) << ' ' << f(18) << ' ' << f(79) << ' ' << f(83886079) << ' ' <<
f(128849018879ll) << '\n';
}
```

Может возникнуть вопрос, почему такое решение работает на порядки быстрее, чем прошлый вариант? Дело в том, что для вычисления f(n) вовсе не требуется вычислять все промежуточные значения f(1), f(2), … f(n-1). До большинства из них мы просто не доберёмся: ведь при вычитании 1 мы очень быстро останавливаемся, дойдя до числа, делящегося на 5, а при делении на 2 мы резко уменьшаем аргумент функции, пропустив всё значения от n/2+1 до n-1.
Заметим ещё, что на языке Python 3 с использованием декоратора lru_cache можно написать решение ещё короче:

```
from functools import lru_cache

@lru_cache(maxsize=100)
def f(n):
  if n == 1: return 1
  if n % 5 == 0: return 0
  s = 0
  if n % 2 == 0:
    s = f(n // 2)
  s += f(n - 1)
  return s

print(f(8), f(18), f(79), f(83886079), f(128849018879))
```

Задача C – Неквадратные суммы
Ответ можно получить следующим алгоритмом. Будем добавлять к ответу числа 1, 4, 9, 16, 25, и так далее. Но при добавлении очередного числа проверяем, будет ли выполняться свойство, что сумма его и любых предыдущих не является точным квадратом. Если нет, то это число пропускаем.
Такое решение, реализованное авторами, при n=20 работает около 10 секунд (на процессоре Intel Core i5-3317U 1.7GHz). Чтобы получить полный балл, можно заранее на своём компьютере вычислить последние три ответа и вставить в программу уже готовые значения (а при желании можно так сделать и вообще со всеми ответами). Пример решения на C++:

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

int main() {
  int n;
  std::cin >> n;
  std::vector<int> a, s;
  a.push_back(1);
  s.push_back(1);
  int i = 1;
  while ((int)a.size() < std::min(n, 17)) {
    i++;
    int sq = i * i;
    bool ok = true;
    for (int j : s) {
      int sum = j + sq;
```

```
      int r = sqrt((double) sum) + 0.1;
      if (r * r == sum) {
        ok = false;
        break;
      }
    }
    if (ok) {
      a.push_back(sq);
      int len = (int) s.size();
      s.push_back(sq);
      for (int i = 0; i < len; i++){
         s.push_back(s[i] + sq);
      }
    }
  }
  for (int x : a) {
    std::cout << x << ' ';
  }
  if (n >= 18)
    std::cout << 64352484 << ' ';
  if (n >= 19)
    std::cout << 161391616 << ' ';
  if (n == 20)
    std::cout << 976375009;
}
```

Интересное решение этой задачи получилось у участника Руслана Редькина (попробуйте догадаться, что за "магическая константа" `31622776`):

```
n = int(input())
x = 31622776
for i in range(n):
    print(x ** 2)
    x -= 1
```

Ещё пример интересного решения участника Дмитрия Кумзерикова:

```
n = int(input())

cnt = 0

for i in range(1,n+1):
    if 3**(i*2) < pow(10,15):
        cnt += 1
        print(3**(i*2))


if cnt < n:
    for i in range(1,1 + n-cnt):
        print(10 ** (2*i))
```

Ещё один пример решения от участника Дениса Струментова:

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

using namespace std;
int main()
{
    int N;
    cin >> N;
    for (long long i = 0; i < N; i++) {
        cout << (65536*i+1)*(65536*i+1) << endl;
    }
```

```
    return 0;
}
```

И совсем неожиданное решение, которое придумали школьники Люлин Кирилл и Орлов Артемий уже после олимпиады:

```
from random import *

n = int(input())
for i in range(n):
    print(randrange(1, 10000000) ** 2, end=" ")
```

Как видим, решить задачу можно было множеством разных способов.

Задача D – Взвешивания

Если сумма всех гирь меньше массы груза, то решений нет.

В противном случае заметим, что для построения ответа нам достаточно перевести массу груза в симметричную троичную систему счисления. В такой системе используются три цифры: 0, 1 и -1. Соответственно, 1 в разряде *i* будет означать, что гиря 3^i кладётся на левую чашу, -1 – на правую, 0 – данную гирю не используем.

Для перевода в симметричную систему счисления можно использовать обычный алгоритм деления с остатком на основание системы счисления – число 3. Однако, вместо остатка 2 мы хотим получать -1. Это несложно сделать: если при делении на 3 получили остаток 2, то увеличим на 1 целую часть от деления, а остаток уменьшим на 3, и он станет равным -1. Например, при делении 5 на 3 таким путём получаем целую часть 2 и остаток -1.

Пример решения на языке Python 3:

```
k = int(input())
n = int(input())
s = 0
for i in range(k + 1):
    s += 3**i
if s < n:
    print(-1)
else:
    left = []
    right = []
    pow3 = 1
    while n > 0:
        r = n % 3
        if r == 0:
            n //= 3
        elif r == 1:
            left.append(pow3)
            n //= 3
        else:
            right.append(pow3)
            n = n // 3 + 1
        pow3 *= 3
    if len(left) == 0: left.append(0)
    if len(right) == 0: right.append(0)
    print(*reversed(left))
    print(*reversed(right))
```

Задача E – Радости и огорчения

Будем запоминать для каждой пары a[i], b[i] следующие числа в трёх отдельных массивах:
- ans[i] - ответ для дня i,

- same_a[i] - на сколько шагов влево повторяется значение a[i],
- same_b[i] - на сколько шагов влево повторяется значение b[i].

Алгоритм выглядит так. Для поиска очередного ответа идём влево, используя массивы same_a и same_b, чтобы пропускать участки с одинаковыми значениями a и b соответственно.

- Если встречаем пару, где оба числа отличаются от a[i], b[i], то это будет ответ.
- Если встречаем пару, где оба числа совпадают с a[i], b[i], то берём ответ, найденный ранее для этой позиции (то есть, применяем динамическое программирование).

Пример решения на языке Python (для полного балла нужно сдавать это решение на PyPy):

```
n = int(input())
ans = [-1] * n
a = [0] * n
b = [0] * n
same_a = [0] * n
same_b = [0] * n
a[0], b[0] = map(int, input().split())
print(0)
for i in range(1, n):
    a[i], b[i] = map(int, input().split())
    if a[i] == a[i - 1]:
        same_a[i] = same_a[i - 1] + 1
    if b[i] == b[i - 1]:
        same_b[i] = same_b[i - 1] + 1
    j = i - 1
    while j >= 0:
        if a[i] == a[j] and b[i] == b[j]:
            ans[i] = ans[j]
            break
        if a[i] != a[j] and b[i] != b[j]:
            ans[i] = j
            break
        if a[j] == a[i]:
            j -= max(same_a[j], 1)
        else:
            j -= max(same_b[j], 1)
    print(ans[i] + 1)
```

Почему это работает быстро? Рассмотрим очередную пару a[i],b[i]. При поиске ответа мы идём влево, проскакивая участки с одинаковыми значениями a=a[i] либо b=b[i].

Проблема может возникнуть, если у нас будет длинный интервал с чередованием большого количества таких участков. Действительно, в этом случае мы можем сделать много шагов влево. Но на самом деле, это не проблема, поскольку такой интервал "заточен" лишь на конкретную пару значений a[i],b[i]. Ни для какой другой последующей пары он уже не вызовет проблем, потому что следующие такие пары не могут пройти сквозь предыдущую такую же пару - они от неё просто возьмут ответ.

Таким образом, сложность решения – O(n).

На самом деле, если учитывать, что количество радостей и огорчений – числа всего лишь до 1000, то можно и не "проскакивать" целые отрезки с одинаковыми a и b – всё равно в среднем мы сделаем в среднем около 500 итераций по ним, пока либо не встретим пару, которая уже была, либо которая полностью отличается от текущей. А при ограничении времени 3 секунды это позволяет такому решению набирать 100 баллов, если оно написано на компилируемом языке (например, C++). Если бы значения в парах могли быть до 10^6, то подобное решение, вероятно, на 100 баллов бы не проходило.

Пример такого решения участника Голубева Максима:

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

```
#define ll long long
using namespace std;

int main(){
    ll n, x, y;
    cin >> n;
    vector <pair<ll, ll> > days(n);
    for (ll i = 0; i < n; i++){
        cin >> x >> y;
        days[i] = {x, y};
    }
    vector <ll> ans(n, 0);
    for (ll i = 1; i < n; i++){
        for (ll j = i - 1; j >= 0; j--){
            if (days[i].first != days[j].first && days[i].second != days[j].second){
                ans[i] = j + 1;
                break;
            }
            if (days[i].first == days[j].first && days[i].second == days[j].second){
                ans[i] = ans[j];
                break;
            }
        }
    }
    for (ll i = 0; i < n; i++)
        cout << ans[i] << endl;
    return 0;
}
```

Ещё до того, как было придумано вышеописанное решение, авторы придумали более сложное решение этой задачи с использованием двух бинарных поисков. Это решение также набирает 100 баллов.

Будем поддерживать два массива и матрицу из векторов a, b и c, где:

$a[f]$ – вектор с номерами дней, в которые было f радостей,

$b[s]$ – вектор с номерами дней, в которые было s огорчений,

$c[f][s]$ – вектор с номерами дней, в которые было одновременно f радостей и s огорчений.

Для текущего дня i ответ можно подобрать двоичным поиском. Пусть mid – предполагаемый номер дня – кандидата в ответ для дня i. Найдём, сколько в интервале от mid до i было дней с таким же количеством радостей, огорчений, а также радостей и огорчений одновременно. Это тоже можно сделать с помощью двоичного поиска по массивам a, b и c соответственно. Пусть у нас получилось nA, nB и nC дней.

Далее используем формулу включений-исключений. Если $nA+nB-nC = i - mid + 1$, то это значит, что на интервале от mid до i не было ни одного дня, в котором отличались оба числа. Значит, ответ лежит левее mid. В противном случае, ответ либо равен mid, либо лежит правее.

Получается вычислительная сложность $O(n \cdot \log^2(n))$. Пример решения:

```
#include <bits/stdc++.h>
using namespace std;

const int M = 1000;
vector<int> a[M + 1], b[M + 1], c[M + 1][M + 1];

int main() {
  int n;
  cin >> n;
  for (int i = 1; i <= n; i++) {
    int f, s;
    cin >> f >> s;
    a[f].push_back(i);
    b[s].push_back(i);
    c[f][s].push_back(i);
    int left = 0, right = i - 1, ans = 0;
    while (left <= right) {
```

```
      int mid = (left + right) / 2;
      int nA = a[f].end() -
          lower_bound(a[f].begin(), a[f].end(), mid);
      int nB = b[s].end() -
          lower_bound(b[s].begin(), b[s].end(), mid);
      int nC = c[f][s].end() -
          lower_bound(c[f][s].begin(), c[f][s].end(), mid);
      if (nA + nB - nC == i - mid + 1) {
        right = mid - 1;
      } else {
        ans = mid;
        left = mid + 1;
      }
    }
    cout << ans << '\n';
  }
}
```