VI Областная олимпиада школьников по информатике
2021-2022 учебный год
9-10 классы
Основной тур

Разбор задач

Задача 1. Отгадывание числа

Проще всего угадывать слева направо (результат от порядка не зависит). Чтобы угадать первую цифру, нужно 8 попыток. Пояснение: поскольку эта цифра – точно не 0, то имеется 9 вариантов. Сделав 8 попыток, мы либо угадаем в одной из них, либо ответом будет цифра, которая не называлась.

Для второй цифры также достаточно 8 попыток, так как у нас снова 9 вариантов (добавился ноль, но ушла одна цифра, совпадающая с первой – помним, что все цифры в числе различны).

Для третьей цифры нужно 7 попыток, для четвертой – 6, и так далее. Видим, что, начиная со второй цифры, у нас наблюдается арифметическая прогрессия: 8 + 7 + …, всего (N – 1) член. Последний член прогрессии равен 8 + (N - 2) * -1 = 10 - N. Сумма прогрессии равна (18 - N) * (N - 1) / 2. И нужно ещё прибавить 8 для отгадывания первой цифры. Итого получается ответ:

8 + (18 - N) * (N - 1) / 2

Можно построить формулу проще. Добавим к количеству попыток для первой цифры единицу, а потом из ответа её вычтем. Тогда получится прогрессия с первым членом 9, последним членом (10 - N) и их количеством N. Сумма прогрессии минус единица равна:

(19 – N) * N / 2 - 1

Возможная ошибка при построении формулы – делить раньше умножения, так как у нас деление целочисленное. Например, формула (19 – N) / 2 * N - 1 является неправильной, и такое решение получит лишь частичный балл.

К верной формуле можно также прийти, отгадывая цифры с конца, но рассуждения будут сложнее. Последнюю цифру мы отгадаем за 9 вопросов, предпоследнюю – за 8, и так далее. Вторую цифру отгадаем за (11 - N) вопросов. Что касается первой цифры, то здесь есть нюанс – был ли среди отгаданных цифр ноль.

На первый взгляд может показаться, что выгоднее, если б он был – тогда первую цифру мы отгадываем за (10 – N) вопросов. Тогда

получается арифметическая прогрессия из N элементов с первым членом 9 и последним (10 - N). Её сумма равна (19 – N) * N / 2.

Но, если при отгадывании каждой цифры в первом вопросе всегда спрашивать про ноль, то это будет означать, что какую-то цифру мы отгадаем всего за 1 вопрос. Поэтому выгоднее, чтобы ноль не встретился. Тогда на отгадывание первой цифры потребуется на 1 вопрос меньше – то есть, (9 - N). Результат также будет на 1 меньше:

(19 – N) * N / 2 - 1

Здесь может возникнуть вопрос – а верный ли ответ даёт формула для N = 10? Ведь (9 - N) при этом становится отрицательным.

Заметим, что при N = 10 у нас особый случай: когда мы угадаем 8 цифр, не встретив нуля, то оставшиеся две цифры сразу станут известны. При этом потребуется 9+8+…+2 = 44 попытки. А если мы встретили нуль (пусть в третьей цифре), то эту цифру мы угадаем за 1 попытку, и плюс ещё нужна одна попытка на оставшиеся две цифры – итого 9+8…+3+1+1 = 44. Формула выше также даёт ответ 44, совпадающий с правильным.

Задача 2. Булевы функции

Подзадача 1. Поскольку f(0, 1) = f(1,0), то у нас есть всего три возможных комбинации в последнем столбце таблицы истинности. Это даёт 2^3 = 8 функций.

Подзадача 2. Количество способов поставить ноль, один, два и три нуля в столбце из восьми ячеек равно C(8,0)+C(8,1)+C(8,2)+C(8,3) = 1+8+28+56=93. Здесь C(n, m) – число сочетаний из n по m.

Подзадача 3. Заполним таблицу истинности. В последнем столбце запишем одинаковые переменные в тех клетках, где ответы должны совпадать:

x	y	z	f(x,y,z)
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	d
1	0	1	c
1	1	0	b
1	1	1	a

Как видим, всего у нас 2^4=16 вариантов.

Подзадача 4. Ответом будет C(10, 5) = 252.

Все правильные ответы: **8 93 16 252**

Задача 3. Странный элемент

Для решения первой подзадачи можно вначале найти, чему равны минимум и максимум в массиве. Затем для каждого элемента циклом влево и вправо ищем ближайшие минимум и максимум, запоминаем наилучший вариант.

Полное решение выглядит так. Вначале найдём значения максимума и минимума. Далее для каждого элемента найдём ближайший к нему минимум и максимум слева – это можно сделать за один проход слева направо. Затем для каждого элемента найдём ближайший к нему минимум и максимум справа – это можно сделать за один проход справа налево. Из двух расстояний до минимума выберем наименьшее, для максимума – аналогично. Осталось за ещё один проход по массиву найти ответ. Пример решения на Python:

```
n = int(input())
a = list(map(int, input().split()))

dist2min = [n] * n
dist2max = [n] * n
minVal = min(a)
maxVal = max(a)
minPos = a.index(minVal)
maxPos = a.index(maxVal)
for i in range(n):
    if a[i] == minVal:
        minPos = i
    if a[i] == maxVal:
        maxPos = i
    dist2min[i] = min(dist2min[i], abs(i - minPos))
    dist2max[i] = min(dist2max[i], abs(i - maxPos))

minPos = len(a) - 1 - a[::-1].index(minVal)
maxPos = len(a) - 1 - a[::-1].index(maxVal)
for i in range(n - 1, -1, -1):
    if a[i] == minVal:
        minPos = i
    if a[i] == maxVal:
        maxPos = i
    dist2min[i] = min(dist2min[i], abs(i - minPos))
    dist2max[i] = min(dist2max[i], abs(i - maxPos))

ans = 0
ansVal = abs(a[0] * (dist2max[0] - dist2min[0]))
for i in range(1, n):
   val = abs(a[i] * (dist2max[i] - dist2min[i]))
   if val > ansVal:
       ansVal = val
       ans = i
print(ans + 1, ansVal)
```

Задача 4. Слоновый конь.

Вначале заметим, что на каждом шаге коня одна из координат меняется на ±3, а другая – на ±1.

Для решения подзадачи 1 можно, например, реализовать перебор либо рассмотреть какую-то часть случаев вручную.

Подзадачу 2 можно решить, реализовав алгоритм поиска в ширину (также известный как волновой алгоритм).

Полное решение задачи выглядит так. Кратчайшее расстояние между двумя точками – это прямая. Направим коня так, чтобы он шёл из точки (0, 0) в точку (x, y) вдоль прямой линии. Для этого используем следующий подход. Пусть x_k, y_k – текущие координаты коня. Если $x_k < x$, то x_k надо увеличить, иначе – уменьшить. Аналогично поступаем для y_k. К которой из координат будем прибавлять ±1, а к которой ±3, зависит от того, по какой координате конь ближе к точке назначения, а по какой – дальше. В результате получился алгоритм, очень похожий на алгоритм Брезенхема для рисования линий на двухмерном растре.

Двигаться таким способом конь будет до тех пор, пока не подойдёт к точке назначения достаточно близко. Величину близости можно взять с запасом – например, 20 клеток по каждой координате. Оставшуюся часть пути можно найти поиском в ширину, как во второй подзадаче. Пример решения на C++:

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

const int limit = 20, Delta = 50;

int main(){
  int xfinish, yfinish;
  cin >> xfinish >> yfinish;
  if ((xfinish + yfinish) % 2 != 0) {
    cout << "No solution\n";
    return 0;
  }
  int x = 0, y = 0;
  while (abs(x-xfinish) > limit || abs(y-yfinish) > limit) {
    cout << x << ' ' << y << '\n';
    if (abs(x - xfinish) > abs (y - yfinish)) {
      if (x < xfinish) x += 3; else x -= 3;
      if (y < yfinish) y += 1; else y -= 1;
    } else {
      if (x < xfinish) x += 1; else x -= 1;
      if (y < yfinish) y += 3; else y -= 3;
    }
  }
```

```
  int deltax = -xfinish + Delta;
  int deltay = -yfinish + Delta;
  int startx = x, starty = y;
  xfinish += deltax;
  yfinish += deltay;
  vector<vector<int> > a(2*Delta, vector<int>(2*Delta, -1));
  int n = (int) a.size();
  int m = (int) a[0].size();
  a[yfinish][xfinish] = 0;
  queue<pair<int, int> > q;
  q.push({yfinish, xfinish});
  while (a[starty + deltay][startx + deltax] == -1) {
    pair<int, int> p = q.front();
    int y = p.first, x = p.second;
    q.pop();
    for (int dy = -3; dy <= 3; dy += 2) {
      for (int dx = -3; dx <= 3; dx += 2) {
        if (abs(dx)+abs(dy)==4 && x+dx<m
        && x+dx>=0 && y+dy<n && y+dy>=0 && a[y+dy][x+dx]<0) {
          a[y + dy][x + dx] = a[y][x] + 1;
          q.push({y + dy, x + dx});
        }
      }
    }
  }
  y = starty + deltay;
  x = startx + deltax;
  cout << startx << ' ' << starty << '\n';
  while(a[y][x] > 0) {
    for (int dy = -3; dy <= 3; dy += 2) {
      for (int dx = -3; dx <= 3; dx += 2) {
        if (abs(dx)+abs(dy)==4 && x+dx<m && x+dx>=0 &&
        y+dy<n && y+dy>=0 && a[y+dy][x+dx] == a[y][x] - 1) {
          y += dy;
          x += dx;
          cout << x - deltax << ' ' << y - deltay << '\n';
        }
      }
    }
  }
}
```

Задача 5. Очень непростые числа.

Подзадачу 1 можно решить «в лоб».

Подзадачу 2 можно решить, используя те же идеи, что в подзадаче 3, но реализовав их не столь эффективно – например, искать делители числа циклом не до корня из числа, а до него самого.

Полное решение выглядит так. Возьмём все делители числа *n*, кроме единицы, пусть их всего *x* штук. Нам надо понять, делится ли число 1*2*…*n на число *x***x***…**x* (*n div* 3 множителей). Рассмотрим различные значения *x*.

x	Делится либо нет
1	**Делится**.
2	**Делится**, так как каждый второй множитель в факториале чётен.
3	**Делится**, поскольку каждый третий множитель в факториале делится на 3.
4	**Делится**, поскольку каждый второй множитель в факториале чётен, и плюс каждый четвёртый делится на 4 – уже будет n div 2 + n div 4, а это больше, чем n div 3 * 2.
5	**Не делится**: столько пятёрок в факториале не набирается.
6	**Делится**, так как 6 = 2*3, а для 2 и 3 ответ получен выше.
7	**Не делится** (раз уж пятерок не хватило, то семерок тем более).
8	**Не делится**, кроме n=256. Двоек в факториале не хватает (нам надо n div 3 * 3 двоек – то есть, почти столько же, сколько множителей в факториале). Однако, есть одно исключение – это число 256. Его можно было подобрать экспериментально.
9	**Не делится** (в факториале нет такого количества троек).
10	**Не делится** (10 = 2*5, а для 5 смотрите выше).
11	**Не делится** (аналогично 5 и 7).
12	**Делится**, так как 12 = 3*4, а для 3 и 4 делится.
>12	**Не делится**: любое число больше 12 либо простое, либо имеет делителем один из рассмотренных выше x, для которого ответ – «не делится».

Итак, число является ОНЧ, если количество его делителей больше 4, но при этом не равно 6, 9 и 12, а само число не равно 256.

Решение задачи выглядит так. Для каждого числа от *a* до *b* находим количество делителей, и проверяем его на ОНЧ согласно критерию выше.

Находить делители чисел можно двумя способами. Во-первых, циклом до корня квадратного из числа. Если число *n* делится на какое-то число *d*, то оно также делится на *n/d* – то есть, к числу делителей сразу добавляем 2 (исключение, когда *d* = *n/d* – тогда добавляем 1).

Еще более эффективный способ состоит в том, чтобы модифицировать решето Эратосфена. Если в обычном решете мы ходим с шагом 2, 3, 5, 7…, то здесь будем ходить с шагом 1, 2, 3, 4....

Когда мы идём с шагом s, то проходим по всем числам, которые делятся на s, и прибавляем единицу к количеству их делителей.

Можно ускорить решето, если брать размер шага только до корня из *n*. При этом, когда мы идём с шагом *i* и дошли до некоторого числа *j*, надо увеличивать счётчик для *j* дважды, если *j* div *i* > $\sqrt{n}$. Пример такого решения на языке Python 3 (сдавать нужно на компиляторе PyPy 3):

```
a = int(input())
b = int(input())
d = [1] * (b + 1)
i = 2
while i * i <= b:
    for j in range(i + i, b + 1, i):
        d[j] += 1
        p = j // i
        if p * p > b:
            d[j] += 1
    i += 1
ans = 0
for n in range(a, b + 1):
    if not(d[n] < 5 or d[n] == 6 or d[n] == 12 or d[n] == 8
and n == 256):
        ans += 1
print(ans)
```