Всероссийская олимпиада школьников по информатике
Вологодская область, 2023-24 учебный год
II (муниципальный) этап
9 - 11 классы
Методические рекомендации по разбору задач

Задача 1. Ромбический порядок

В подзадаче 1 можно было, например, вручную заполнить все 6 ромбов и сохранить их в матрице прямо в тексте программы.

В подзадаче 2 можно было, например, написать программу для последовательного заполнения матрицы числами в нужном порядке.

Перейдём к полному решению. Заметим, что |x|+|y| даёт нам номер ромба, на котором лежит искомое число. Также заметим, что в каждом следующем ромбе на 4 клетки больше, чем в предыдущем. Найдём количество чисел во всех внутренних ромбах и центральной клетке. Оно равно 1 + 4 * (1 + 2 + ... + |x| + |y| - 1). По формуле арифметической прогрессии получаем 1 + 4 * (1 + |x| + |y| - 1) / 2 * (|x| + |y| - 1). После сокращения получаем: 1 + 2 * (|x| + |y|) * (|x| + |y| - 1).

Осталось добавить количество клеток на последнем ромбе. Для этого рассмотрим, в какой четверти лежит клетка.

- Если $x \geq 0$, $y \geq 0$, то нужно добавить к ответу 1 + |x|.
- Если $x \geq 0$, $y < 0$, то нужно добавить к ответу 1 + (|x| + |y|) + |y|, так как одну сторону мы проходим полностью, и затем ещё опускаемся на |y| клеток.
- Если $x < 0$, $y \leq 0$, то нужно добавить к ответу 1 + 2 * (|x| + |y|) + |x|, так как две стороны мы проходим полностью, и затем ещё двигаемся влево на |x| клеток.
- Если $x < 0$, $y > 0$, то нужно добавить к ответу 1 + 3 * (|x| + |y|) + |y|, так как три стороны мы проходим полностью, и затем ещё двигаемся вверх на |y| клеток.

  Заметим, что данное решение даёт неверный ответ при x=0 и y=0, этот случай можно рассмотреть отдельным условием.

  Пример решения на языке Python:

```
x = int(input())
y = int(input())
if x == 0 and y == 0:
```

```
    ans = 1
else:
    ans = 1 + 2 * (abs(x) + abs(y)) * (abs(x) + abs(y) - 1)
    if x >= 0 and y >= 0:
        ans += 1 + abs(x)
    elif x >= 0 and y < 0:
        ans += 1 + abs(x) + abs(y) + abs(y)
    elif x < 0 and y <= 0:
        ans += 1 + 2 * (abs(x) + abs(y)) + abs(x)
    else:
        ans += 1 + 3 * (abs(x) + abs(y)) + abs(y)
print(ans)
```

Задача 2. Словарный запас

В первой подзадаче можно было вручную вычислить ответы для всех возможных вариантов, не забыв также случай, что при N>3K ответ равен нулю.

Во второй подзадаче можно было, например, написать программу для перебора всех допустимых слов.

Полное решение можно получить как с помощью комбинаторики, так и методом динамического программирования.

Комбинаторное решение выглядит так. Переберём количество первой буквы p от 0 до min(K, N). Количество способов разместить их среди первых N позиций равно C(N, p) – число сочетаний из N по p. Теперь переберём количество способов разместить вторую букву – q от 0 до min(K, N-p). Количество способов разместить их среди оставшихся позиций равно C(N-p, q). Осталось ещё N-p-q позиций под третью букву. Если N-p-q ≤ K, то добавляем к ответу C(N, p) * C(N-p, q). Пример решения на языке Python:

```
from math import factorial

def C(n, k):
    return factorial(n) // (factorial(k) * factorial(n - k))

n = int(input())
k = int(input())
ans = 0
for p in range(min(k, n) + 1):
    c1 = C(n, p)
    for q in range(min(k, n - p) + 1):
        if n - p - q <= k:
            c2 = C(n - p, q)
            ans += c1 * c2
print(ans)
```

Также можно решить эту задачу методом динамического программирования. Пусть dp[i][j][q] – количество слов из i букв первого вида, j – второго и q – третьего. У нас есть три варианта, какая буква стоит последней – получаем рекуррентную формулу: dp[i][j][q] = dp[i-1][j][q] + dp[i][j-1][q] + dp[i][j][q-1]. Пример решения на Python с использованием декоратора lru_cache для «ленивого» динамического программирования:

```
from functools import lru_cache

n = int(input())
k = int(input())

@lru_cache(maxsize=30000)
def dp(i, j, q):
    if i == j == q == 0:
        return 1
    ans = 0
    if i > 0:
            ans += dp(i - 1, j, q)
    if j > 0:
        ans += dp(i, j - 1, q)
    if q > 0:
        ans += dp(i, j, q - 1)
    return ans

ans = 0
for i in range(k + 1):
   for j in range(k + 1):
      q = n - i - j
      if q >= 0 and q <= k:
         ans += dp(i, j, q)
print(ans)
```

Задача 3. Командировки

В первой подзадаче можно создать массив дней. Для каждой поездки проходим в цикле от d_1 до d_2 и заносим номер города в элемент массива дней с соответствующим индексом. Если же там уже занесён другой город, то отмечаем это в отдельном массиве дубликатов. Затем остаётся только вывести количество дубликатов. Пример такого решения на Python:

```
n = int(input())
days = [0] * 1001
dup = [0] * 1001
for _ in range(n):
    d1, d2, c = map(int, input().split())
    for i in range(d1, d2 + 1):
        if days[i] != c and days[i] != 0:
```

```
            dup[i] = 1
        days[i] = c
print(sum(dup))
```

Полное решение выглядит так. Заметим, что поскольку интервалы отсортированы, массивы нам не нужны. В переменной last будем хранить последний день, на который пришлось два (или более) разных города. Это нужно, чтобы не учесть какой-нибудь день более одного раза. Вначале last = 0.

Будем поддерживать текущий интервал <left, right, city> и каждый раз, прочитав очередной интервал, сравнивать его с текущим.

Основной цикл выглядит так. Читаем очередной интервал <d_1, d_2, c>. Сразу присвоим d_1 = max(d_1, last+1). Если теперь стало d_1 > d_2, то continue (пропускаем остаток этой итерации цикла).

Если d1 > right, то заменяем текущий интервал на новый: left = d1, right = d2, city = c, continue.

Если city = c (то есть, города совпали), то удлиняем текущий интервал: right = max(right, d2), continue.

Если ни один continue выше не сработал, то текущий и очередной интервалы пересекаются, и в них разные города. Тогда сделаем следующее.

Добавим к ответу длину пересечения: ans += min(d2, right) - d1 + 1.

Обновим last: last = min(d2, right).

Обновим город в текущем интервале: если d2 > right, то city = c.

Поставим левую границу на следующий день после окончания пересечения: left = min(d_2, right) + 1.

Установим правую границу: right = max(right, d_2).

Если получилось right < left, то присвоим left = right = 0 (поскольку от обоих пересекающихся отрезков ничего не осталось).

Пример такого решения на Python:

```
n = int(input())
left = right = city = last = 0
ans = 0
for _ in range(n):
    d1, d2, c = map(int, input().split())
    d1 = max(d1, last + 1)
    if d1 > d2:
        continue
    if d1 > right:
        left = d1
```

```
            right = d2
            city = c
            continue
        if c == city:
            right = max(right, d2)
            continue
        ans += min(d2, right) - d1 + 1
        last = min(d2, right)
        if d2 > right:
            city = c
        left = min(d2, right) + 1
        right = max(d2, right)
        if right < left:
            left = right = 0
print(ans)
```

Задача 4. Забавные числа

В подзадаче 1 любая возможная сумма состоит из забавных слагаемых, нужно только сосчитать количество возможных сумм – оно равно n div 2 + 1.

В подзадаче 2 можно перебирать первое число в сумме, с помощью вычитания получать второе число и проверять, что каждое из них – забавное.

В полном решении можно перебирать только забавные числа. Для этого перебираем длину числа, какие две цифры встречаются в этом числе, и для каждой пары перебираем, какая цифра стоит в каждой позиции числа. Затем проверяем, является ли забавным результат вычитания. Здесь нужно не забыть учесть случай, что первая цифра не может быть нулём (кроме собственно числа 0).

Числа, в которых только одна различная цифра, можно обрабатывать отдельно (но при этом при переборе чисел их двух цифр проверять, что цифр действительно две, а не одна – иначе сосчитаются дубликаты).

Пример такого решения на Python:

```
def funny(x):
    if x == 0:
        return True
    c = [0] * 10
    while x > 0:
        c[x % 10] = 1
        x //= 10
    return sum(c) <= 2

def difdigits(x):
    if x == 0:
        return 1
    c = [0] * 10
```

```
    while x > 0:
        c[x % 10] = 1
        x //= 10
    return sum(c)

n = int(input())
answer = 0
if funny(n):
    answer = 1

# only one digit in numbers
for d in range(1, 10):
    a = 0
    while True:
        a = a * 10 + d
        if n - a < a:
            break
        if funny(n - a):
            answer += 1

def backtrack(a, d1, d2):
    global n, answer
    if n - a < a:
        return
    if difdigits(a) == 2 and funny(n - a):
        answer += 1
    backtrack(a * 10 + d1, d1, d2)
    backtrack(a * 10 + d2, d1, d2)

# two digits in number
for d1 in range(0, 9):
    for d2 in range(d1 + 1, 10):
        if d1 != 0:
            backtrack(d1, d1, d2)
        backtrack(d2, d1, d2)

print(answer)
```

Задача 5. Симметричные последовательности

В первой подзадаче можно перебрать все скобочные последовательности длины 2n, проверяя каждую на правильность и симметричность.

Во второй подзадаче достаточно заметить, что можно перебирать лишь левую половину последовательности, а правую получать отражением левой.

Полное решение использует метод динамического программирования. Назовём балансом скобочной последовательности разность между количеством открытых и закрытых скобок. Пусть f(n, b) – количество скобочных последовательностей длины 2n, таких что на любом их префиксе баланс не

опускается ниже -b (где b ≥ 0), а баланс всей последовательности равен нулю. Другими словами, b – это тот запас баланса, на который разрешено уходить в минус. Ответом на задачу будет f(n, 0).

Выразим f(n, b) через значения f с меньшими аргументами.

Если b = 0, то первая скобка обязана быть '(', а последняя – ')'. При этом для внутренней части выражения возникнет запас баланса, и можно разрешить внутри этих скобок иметь баланс на единицу меньше. То есть, f(n, b) = f(n - 1, b + 1).

Если b > 0, то у нас есть запас баланса, и начальную скобку можно поставить любого типа: f(n, b) = f(n - 1, b + 1) + f(n - 1, b - 1).

Пример решения на Python с использованием декоратора lru_cache для «ленивого» динамического программирования:

```
from functools import lru_cache

@lru_cache(maxsize=10000)
def f(n, b=0):
    if n == 0:
        return 1
    if b == 0:
        return f(n - 1, b + 1)
    else:
        return f(n - 1, b - 1) + f(n - 1, b + 1)

n = int(input())
print(f(n))
```