Разбор финала олимпиады Innopolis Open по робототехнике

Март 2023

1 Задачка от Арсения

Постановка задачи: Надо составить два t разрядных числа без лидирующих нулей, причем первое число a должно состоять из цифр x_1, x_2, x_3, x_4, а второе число b должно состоять из цифр y_1, y_2, y_3, y_4, а также должно выполняться одно из условий: либо $a \cdot k = b$, либо $b \cdot k = a$.

Подзадача 1

Переберем циклом все возможные t разрядные числа и проверим выполнение условие.

```
t, k = map(int, input().split())
x = list(map(int, input().split()))
y = list(map(int, input().split()))

tmp = 0
if t == 1:
    tmp = 1

for a in range(10**(t-1) - tmp, 10**t):
    flagA = True
    a2 = a
    if a2 == 0 and a2 not in x:
        flagA = False
    while a2 > 0:
        if a2 % 10 not in x:
            flagA = False
        a2 //= 10

    if not flagA:
        continue

    for b in range(10**(t-1) - tmp, 10 ** t):
        flagB = True
        b2 = b
        if b2 == 0 and b2 not in y:
            flagB = False
        while b2 > 0:
            if b2 % 10 not in y:
                flagB = False
            b2 //= 10

        if not flagB:
```

```
            continue

        if a * k == b or a == b * k:
            print("Yes")
            print(a, b)
            exit(0)
print("No")
```

Подзадача 2

Данная подзадача имеет два подхода:

1. Переберем циклом число a, и если оно удовлетворяет условиям, то проверим удовлетворяет ли условиям число $a \cdot k$ или $\frac{a}{k}$(если a делится на k)
2. Рекурсивно будем составлять первое число из цифр x_1, x_2, x_3, x_4, потом повторим тот же самый процесс с цифрами y_1, y_2, y_3, y_4, и осталось проверить, что выполняется одно из условий: либо $a \cdot k = b$, либо $b \cdot k = a$.

Подзадача 3

На полный балл требуется независимо найти все возможные числа из первого и второго набора (рекурсивно). Все числа из первого набора закинуть в множество (set), затем, перебирая число из второго набора, проверять есть ли число $a \cdot k$ или $\frac{a}{k}$ (если a делится на k) в множестве всех чисел из первого набора.

```
numsX = set()
numsY = set()
x = []
y = []
t = 0
k = 1
low = 0

def recX(num, pos):
  global numsX
  global x
  global t
  global low

  if pos == t:
    if t == 1 or num >= low:
      numsX.add(num)
    return

  recX(num*10 + x[0], pos + 1)
  recX(num*10 + x[1], pos + 1)
  recX(num*10 + x[2], pos + 1)
  recX(num*10 + x[3], pos + 1)
  return

def recY(num, pos):
  global numsX
  global y
  global t
```

```
  global low
  global k

  if pos == t:
    if (t == 1 or num >= low):
      if num*k in numsX:
        print("Yes")
        print(num*k, num)
        exit(0)
      if num % k == 0 and num//k in numsX:
        print("Yes")
        print(num//k, num)
        exit(0)
    return


  recY(num*10 + y[0], pos + 1)
  recY(num*10 + y[1], pos + 1)
  recY(num*10 + y[2], pos + 1)
  recY(num*10 + y[3], pos + 1)
  return

def main():
  global numsY
  global numsX
  global y
  global x
  global t
  global k
  global low

  t, k = map(int, input().split())
  x = list(map(int, input().split()))
  y = list(map(int, input().split()))

  low = 10**(t - 1)
  recX(0, 0)
  recY(0, 0)
  print("No")

main()
```

2 Намёк

Постановка задачи: Дан массив A, состоящий из n элементов. Необходимо найти наименьший по длине подотрезок данного массива, сумма которого будет строго больше заданного числа q, или сказать, что это невозможно.

Подзадача 1

Для того, чтобы получить баллы за первую подгруппу, где $n \leq 4$, достаточно вручную разобрать все возможные случаи и с помощью условных операторов выводить нужный результат. Сложность: O(1)

Подзадача 2

Во второй подгруппе $n \leq 100$. Так как все $a_i \geq 0$, то если сумма всех элементов меньше или равна q, то выводим -1. Иначе запишем в текущий ответ n. Далее перебираем все пары элементов массива, посчитаем сумму между ними и, если эта сумма больше q, то проверяем можно ли улучшить ответ. В конце выводим то, что осталось записанным в текущем ответе. Сложность: O(n^3)

```
n, q = map(int, input().split())
a = list(map(int, input().split()))
ans = n
if sum(a) <= q:
    print(-1)
    exit(0)

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

Подзадача 3

Для прохождения всех тестов из третьей подгруппы будем решать задачу следующим образом: будем перебирать длину подотрезка по возрастанию и проверять, что найдется хоть один подходящий отрезок такой длины. Если да, то выводим длину и заканчиваем работу программы. Чтобы оптимально считать сумму на подотрезках длинной k, сначала посчитаем сумму первых k элементов, а затем будем двигать наше «окно» вправо на один элемент, то есть вычитать из получившейся суммы первый из посчитанных элементов и добавлять к ней первый из еще не посчитанных. Сложность: O(n^2)

```
n, q = map(int, input().split())
a = list(map(int, input().split()))
if sum(a) <= q:
    print(-1)
else:
    for k in range(1, n + 1):
        ok = False
```

```
        l, r = 0, k - 1
        curSum = 0
        for i in range(l, r):
            curSum += a[i]
        while r < n and ok is False:
            curSum += a[r]
            if curSum > q:
                ok = True
            curSum -= a[l]
            l += 1
            r += 1
        if ok:
            print(k)
            break
```

Подзадача 4

Рассмотрим теперь два решения полной версии задачи.

Решение 1.

Для того чтобы получить полный балл за задачу, необходимо заметить, что с увеличением длины подотрезка увеличивается и сумма на нем. Отсюда можно сделать вывод, что если существует ответ длиной k, то будет существовать и ответ для любого $l > k$. Значит, мы можем улучшить предыдущее решение, воспользовавшись идеей бинарного поиска. Сложность: $O(n \cdot log_2 n)$

```
n, q = map(int, input().split())
a = list(map(int, input().split()))
if sum(a) <= q:
    print(-1)
else:
    ans = n
    L, R = 1, n
    while L <= R:
        k = (L + R) // 2
        ok = False
        l, r = 0, k - 1
        curSum = 0
        for i in range(l, r):
            curSum += a[i]
        while r < n and ok is False:
            curSum += a[r]
            if curSum > q:
                ok = True
            curSum -= a[l]
            l += 1
            r += 1
        if ok:
            ans = k
            R = k - 1
        else:
            L = k + 1
    print(ans)
```

Решение 2.

Второе решение предусматривает использование метода «двух указателей». Пусть l и r — это границы отрезка, в пределах которого считаем сумму элементов $a_i (l \leq i \leq r)$ (хранить сумму будем в переменной sm). Изначально $l = 0, r = 0$. Будем двигать r вправо, пока

сумма элементов меньше или равна q, а когда значение $sm > q$, то будем двигать вправо l. Двигая границы отрезка, будем пробовать улучшать ответ.

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

if sum(a) <= q:
    print(-1)
else:
    l, r = 0, 0
    sm = 0
    ans = n
    while r < n:
        sm += a[r]
        while sm > q:
            ans = min(ans, r - l + 1)
            sm -= a[l]
            l += 1
        r += 1
    print(ans)
```

Стоит отметить, что для прохождения всех тестов, нужно не забыть воспользоваться 64-битным типом данных, так как ответ может получиться достаточно большим.

3 Фу, геометрия

Постановка задачи: Даны две точки на плоскости: A (x_A, y_A) и B (x_B, y_B). Необходимо перенести эти точки в первую четверть плоскости ($x \geq 0$ и $y \geq 0$) с помощью следующей последовательности действий:

- Выбираем одну из двух точек.
- Фиксируем выбранную точку (x_1, y_1) как центр окружности.
- Двигаем другую точку (x_2, y_2) по окружности, которую она образует с первой точкой (x_1, y_1).

Найдите минимальное количество операций, необходимых для переноса двух точек в первую четверть плоскости, и выведите координаты всех точек, которые будут посещены при выполнении этих действий в порядке их посещения.

3.1 Подзадача 1

Если одна из точек в первой четверти, то за одно действие можно перенести вторую точку. Например, отложить от первой точки вектор $\{1, 0\}$, умноженный на расстояние между точками.

3.2 Подзадача 2

Во второй подзадаче точки лежат на прямой, параллельной оси Oy. Если точки в первой четверти, то никаких операций не нужно производить. Если во второй, то от любой точки откладываем вектор $\{1, 0\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор пока обе точки не окажутся в первой четверти.

3.3 Подзадача 3

Третья подзадача аналогична второй, только вектор должен иметь координаты $\{0, 1\}$, умноженный на расстояние между точками.

3.4 Подзадача 4

Данная подзадача решается алгоритмом, изложенным во второй подзадаче. Единственное отличие в том, что надо использовать вещественный тип данных.

3.5 Подзадача 5

Данная подзадача решается алгоритмом, изложенным в третьей подзадаче. Единственное отличие в том, что надо использовать вещественный тип данных.

3.6 Подзадача 6

Осталось рассмотреть случаи:

1. когда одна из точек во второй четверти, а другая в четвертой четверти, то за две операции можно перенести все точки в первую четверть. Для простоты точка $A\ (x_A, y_A)$ во второй четверти, а точка $B\ (x_B, y_B)$ в четвертой. Отложим от точки во второй четверти вектор $\{1, 0\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

2. когда одна из точек во второй четверти, а другая в третьей четверти, то возможны два варианта:

   (a) Отложим от точки во второй четверти вектор $\{1, 0\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

   (b) Отложим от точки в третьей четверти вектор, направленный к точке $(0, 0)$, равный длине расстояния между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

   Промоделируем обе ситуации и выберем с наименьшим количеством операций.

3. Обе точки в третьей четверти, тогда отложим от точки ближайшей к $(0, 0)$ вектор, направленный к точке $(0, 0)$, равный длине расстояния между точками, производим данную операцию до тех пор пока обе точки не окажутся в первой четверти.

4. Первая точка в третьей четверти, а вторая в четвертой. В этом случае возможны два варианта:

   (a) Отложим от точки в третьей четверти вектор $\{1, 0\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

   (b) Отложим от точки в четвертой четверти вектор $\{0, 1\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

   Промоделируем обе ситуации и выберем с наименьшим количеством операций.

5. Обе точки в четвертой четверти. В этом случае выберем точку с наибольшим значением по оси Oy и от нее будем откладывать вектор $\{0, 1\}$, умноженный на расстояние между точками. Производим данную операцию до тех пор, пока обе точки не окажутся в первой четверти.

Красивый подотрезок

Рассмотрим необходимое и достаточное условие, что последовательность $(b_1, b_2, \ldots, b_m)$ является красивой. Добавим в начало и в конец последовательности 0, то есть $b_0 = b_{m+1} = 0$. Построим по этой последовательности новую $(d_0, d_1, \ldots, d_m)$, где $d_i = b_{i+1} - b_i$.

Операция добавляет элемент c к k элементам последовательности $(b_0, b_1, b_2, \ldots, b_m, b_{m+1})$. То есть получаем последовательность:

$$(b_0, b_1, \ldots, b_i + c, b_{i+1} + c, \ldots, b_{i+k-1} + c, b_{i+k}, \ldots, b_m, b_{m+1})$$

Тогда последовательность d изменится следующим образом:

$$(d_0, d_1, \ldots, d_{i-1} + c, d_i, \ldots, d_{i+k-1} - c, \ldots, d_m).$$

Иначе говоря, добавляем c к d_{i-1} и $-c$ к d_{i+k-1}. Все элементы b равны нулю тогда и только, когда все элементы d равны 0. Значит, последовательность b красивая тогда и только тогда, когда можно сделать все элементы d равными 0, выполняя нижеуказанную операцию некоторое число раз:

- Выберем позицию i ($0 \le i \le m-k$) и целое число c, затем прибавим число c к d_i и $-c$ к $d_{i+k}(*)$.

Используя данную операцию, меняются два элемента, имеющих одинаковый остаток деления на k. Следовательно, надо приравнять к нулю каждую группу элементов последовательности, имеющих одинаковый остаток от деления на k. В каждой группе, имеющей остаток r ($0 \le r < k$ и $r \in \mathbb{Z}$), операция $(*)$ может изменять любые два элемента, к одному прибавлять c, а к другому $-c$. Следовательно, $\sum\limits_{i=0}^{\lfloor \frac{k-r}{k} \rfloor} d_{ik+r} = 0$.

Значит, для того чтобы b была красивой последовательностью, надо чтобы $\sum\limits_{i=0}^{\lfloor \frac{k-r}{k} \rfloor} d_{iK+r} = 0$ для каждого r ($0 \le r < k$ и $r \in \mathbb{Z}$).

Для того чтобы быстро проверять данное условие для отрезка (l_i, r_i), надо посчитать префиксные суммы для каждой группы и не забыть добавить краевым элементам b_{l_i-1} и $-b_{r_i+1}$ соответственно для получения суммы на нужном подотрезке.

4 Прогулка

Задачу можно сформулировать так: дан связный граф из n вершин и $n-1$ ребра. Каждому ребру ассоциировано число. Требуется вычислить количество путей, у которых `XOR` чисел всех ребер, принадлежащих этому пути, равен k.

Подзадача 1

Переберем пары вершин и посчитаем `XOR` на пути с помощью DFS или BFS. Асимптотика решения $O(n^3)$.

Подзадача 2 и 3

Будем перебирать вершину и от нее запускать DFS или BFS. В процессе работы этих алгоритмов считаем `XOR`. Если ответ подходит, то прибавляем единицу к переменной ans. Каждый путь будет учтен два раза, поэтому ответом будет $\frac{ans}{2}$.

Подзадача 4 и 5

Из условия понятно, что граф является деревом, так как всего $n-1$ ребро и одна компонента связаности. Также из дополнительных ограничений ясно, что с каждой вершиной дерева связано не более двух других вершин, то есть дерево превращается в «цепочку» или в «бамбук». Теперь задачу можно переформулировать: дан массив размера $n-1$, надо найти количество подотрезков, `XOR` чисел которых равен k.

Для начала научимся находить ответ при $k = 0$. Давайте идти слева направо и считать в массиве $Pref$ XOR на префиксе r, то есть $Pref_i = \bigoplus_{j=1}^{i} a_j$. Количество значений равных $Pref_r$ в массиве $Pref$ будет количеством отрезков с XOR равным 0 и заканчивающихся в r, так как $x \oplus x = 0$ (свойство само обратимости). Для подсчета прошлых значений XOR на префиксе воспользуемся структурой данных *map* в $c++$ или *dict* в *python*.

```
ans = cur = 0
cnt = dict()
Pref = [0] * n
for i in range(n):
  cur ^= array[i]
  Pref[i] = cur
  ans += cnt.get(Pref[i], 0)
  cnt[Pref[i]] = cnt.get(Pref[i], 0) + 1
```

Для решения 5-й подзадачи надо понять, что ответом для отрезков с XOR равным k и заканчивающихся в r будет количество значений равных $Pref_r \oplus k$ в массиве $Pref$. Так как $x \oplus k \oplus x = k \oplus (x \oplus x) = k$.

Подзадача 6 и 7

Подвесим дерево за вершину u и от нее посчитаем XOR на пути к остальным вершинам. XOR чисел на пути от вершины X к Y равен $pref_X \oplus pref_Y$, так как путь от корня u к родителю вершин X и Y будет учтен два раза, поэтому XOR равен 0. Теперь обратимся к идее в 4-й и 5-й подзадачах и получим решение на 100 баллов.

```
n, k = map(int, input().split())

graph = [[] for i in range(n)]
used = [False] * n
ans = 0
cnt = dict()

def dfs(node, tmp):
  global k
  global ans

  used[node] = True

  ans += cnt.get(tmp ^ k, 0)
  cnt[tmp] = cnt.get(tmp, 0) + 1

  for (to, value) in graph[node]:
    if used[to] == False:
      dfs(to, tmp ^ value)

for i in range(n - 1):
  a, b, v = map(int, input().split())
  a -= 1
  b -= 1

  graph[a].append((b, v))
  graph[b].append((a, v))

dfs(0, 0)

print(ans)
```