Разбор задач

В разработке задач принимали участие Елена Андреева, Алексей Дацковский, Денис Кириенко, Сергей Князевский, Семён Кухаренко, Дмитрий Михалин, Александр Понкратов, Арсений Порхунов, Владимир Рагулин.

Задача 1. Речные прогулки

Автор задачи: Денис Кириенко.

Набрать 60 баллов можно при помощи перебора по ответу. Переберём все пристани с номерами от 2 до $n-1$, и для каждой из них посчитаем разность между продолжительностью пути вверх и вниз.

Пусть рассматриваемая пристань имеет номер x, тогда продолжительность пути до пристани 1 равна $a(x-1)$, а вниз — $b(n-x)$. Нужно найти такую пристань x, для которой модуль разности этих величин будет наименьшим.

Такое решение имеет сложность $O(n)$. Пример такого решения.

```
n = int(input())
a = int(input())
b = int(input())

def ans(x):
    return abs((x - 1) * a - (n - x) * b)

d = 2
for i in range(3, n):
    if ans(i) < ans(d):
        d = i
print(d)
```

Для решения на полный балл заметим, что если некоторая пристань x будет ответом, то значения $a(x-1)$ и $b(n-x)$ будут близки. Приравняем их, откуда получим решение уравнения $x = (bn+a)/(a+b)$. Это число было бы ответом, если задача решалась в действительных числах, когда началом маршрутов может быть любая точка. Но мы рассматриваем только целочисленные значения ответа, поэтому ответом может быть одно из двух целых чисел: указанное значение x, округлённое вниз и вверх. Выберем из этих значений такое, для которого модуль разности времени пути до верхней и нижней пристани будет наименьшим, учтя, что ответ не может равняться 1 и n. Такое решение имеет сложность $O(1)$.

```
n = int(input())
a = int(input())
b = int(input())

def ans(pos):
    return abs((pos - 1) * a - (n - pos) * b)

d = max(2, (b * n + a) // (a + b))
if d + 1 < n and ans(d + 1) < ans(d):
    d += 1

print(d)
```

Также полный балл можно было набрать при помощи двоичного или троичного поиска по ответу.

Задача 2. Треугольники

Автор задачи: Владимир Рагулин.

Подготовка задачи: Александр Понкратов.

Внутри каждого квадрата 1×1 можно выбрать 8 маленьких треугольников, как в первом примере, то есть число таких треугольников равно $8nm$.

Внутри прямоугольника 1×2 есть 2 больших треугольника площади 1. Прямоугольник 1×2 можно выбрать $n(m-1)$ способами, поэтому таких треугольников будет $2n(m-1)$.

Аналогично, существует $2(n-1)m$ больших треугольников, расположенных внутри какого-то прямоугольника размером 2×1.

Наконец, внутри квадрата 2×2 можно выбрать 4 треугольника площади 2, таких треугольников будет $4(n-1)(m-1)$.

Нужно вывести сумму этих величин.

```
n = int(input())
m = int(input())
print(n * m * 8 + (n - 1) * m * 2 + n * (m - 1) * 2 + (n - 1) * (m - 1) * 4)
```

Задача 3. Порядок во всём

Автор задачи: Сергей Князевский.

Подготовка задачи: Владимир Ильин.

Несложно понять, что на каждом шаге нужно получить как можно меньшее число. То есть задача сводится к необходимости реализовать один шаг: даны два числа A и B, необходимо из числа B дописыванием цифр в конец получить наименьшее число C, которое не меньше A.

Отметим, что задачу удобнее решать с использованием строковых типов данных, а не числовых. Если длина числа B больше, чем длина числа A, то $B > A$ и дописывать ничего не нужно.

Если длина числа B не больше длины числа A, то рассмотрим префикс A' числа A, длина которого равна длине числа B. Например, если $A = 1357$, а число B — двузначное, то $A' = 13$. Поскольку числа A' и B имеют одинаковую длину, то их можно сравнивать в лексикографическом порядке, как строки. Рассмотрим разные варианты сравнения чисел A' и B.

Если $A' = B$, то число B является префиксом A, тогда мы можем дописать в конец B цифры так, что получится число A, то есть $C = A$. Например, при $A = 1357$ и $B = 13$ значение $C = 1357$.

Если $A' > B$ (префикс A больше B), то при дописывании в конец числа B новых цифр мы можем получить большее число, только когда длина числа C станет больше длины числа A. Тогда необходимо дописать минимальное число нулей, чтобы длина числа C стала на 1 больше длины числа A. Например, при $A = 1357$ и $B = 12$ значение $C = 12000$.

Наконец, если $A' < B$, то нужно дописать нули так, чтобы длины чисел A и C стали равны. Например, при $A = 1357$ и $B = 14$ значение $C = 1400$.

Пример решения на языке Python.

```
n = int(input())
A = input()
for i in range(n - 1):
    B = input()
    if len(A) >= len(B):
        Ap = A[:len(B)]
        if Ap == B:
            # Первый случай, дополним число B до A
            C = A
        elif Ap > B:
            # Второй случай, дополним B нулями с увеличением длины числа
            C = B + "0" * (len(A) + 1 - len(B))
        else:
```

```
            # Третий случай, дополним B нулями до длины числа A
            C = B + "0" * (len(A) - len(B))
    else:
        # Длина числа B больше длины A, поэтому ничего добавлять не надо
        C = B
    A = C
print(A)
```

Заметим, что длина числа может увеличиваться на 1 на каждом шаге. Если взять пример, в котором исходные числа убывают, то каждое следующее полученное число будет на 1 длиннее предыдущего числа. Например, если входные числа таковы:
9999
9998
9997
9996
9995
...
то получатся следующие числа:
9999
99980
999700
9996000
99950000
...

Несложно построить тест, на котором такое решение имеет сложность $O(n^2)$. Такие решения набирают 60 баллов.

Для того, чтобы набрать 100 баллов, необходимо заметить, что длины чисел увеличиваются за счёт добавления нулей в конец, и сложность $O(n^2)$ возникает из-за того, что мы создаём строки увеличивающейся длины, добавляя нули в конец чисел. Большую часть ответа в этом случае составляет суффикс нулевой длины, поэтому вместо ответа в виде длинной строки будем хранить его префикс и количество нулей, которое нужно дописать в конец, в переменной zero_suff_len. Такое решение имеет сложность $O(n)$ и набирает 100 баллов.

```
n = int(input())
A = input()
zero_suff_len = 0
for i in range(n - 1):
    B = input()
    if len(A) + zero_suff_len >= len(B):
        # Случай когда длина A <= длина B, но с учётом дополнительного
        # нулевого суффикса. Для построения префикса A такой же длины,
        # как B, будем добавлять в конец A нули из суффикса
        while len(A) < len(B):
            A += "0"
            zero_suff_len -= 1
        Ap = A[:len(B)]
        # Последующий разбор случаев дублирует неэффективное решение
        if Ap == B:
            C = A
        elif Ap > B:
            C = B
            zero_suff_len = zero_suff_len + len(A) + 1 - len(B)
        else:
            C = B
```

```
            zero_suff_len = zero_suff_len + len(A) - len(B)
    else:
        # Длина B больше, поэтому C = B и нужно сбросить zero_suff_len
        C = B
        zero_suff_len = 0
    A = C
print(A + "0" * zero_suff_len)
```

Задача 4. Тройка

Автор задачи: Денис Кириенко.
Подготовка задачи: Арсений Порхунов.

В этой задаче можно было придумать решения, работающие в некоторых частных случаях. Например, если все поездки были совершены на метро, то каждая поездка стоит 57 рублей, поэтому программа, выводящая число 57 * n, наберёт 15 баллов.

Отметим «жадное» решение, в котором используется только одна карта «Тройка», а все списания соответствуют правилам тарификации. То есть результат работы этого решения будет таким же, когда пассажир каждый раз при оплате проезда использует одну и ту же карту «Тройка». Для реализации этого алгоритма необходимо запоминать, сколько поездок было совершено по данному билету, сколько из этих поездок было поездок на метро и время совершения первой поездки. Такое решение набирает 30 баллов. Пример решения.

```
n = int(input())
subway = [1] * (n + 1)
time = [-1000] * (n + 1)
day = 0
for i in range(1, n + 1):
    trans, tm = input().split()
    tm = tm.split(":")
    time[i] = int(tm[0]) * 60 + int(tm[1]) + day * 24 * 60
    subway[i] = int(trans == "M")
    if i > 0 and time[i] <= time[i-1]:
        day += 1
        time[i] += 24 * 60
ans = [10**9] * (n + 1)
ans[0] = 0
ticket_start = -10**9
ticket_count = 0
ticket_count_subway = 0
for i in  range(1, n + 1):
    if time[i] - ticket_start > 90 or ticket_count_subway + subway[i] > 1:
        ans[i] = ans[i - 1] + 57
        ticket_start = time[i]
        ticket_count = 1
        ticket_count_subway = subway[i]
    else:
        if ticket_count == 1:
            ans[i] = ans[i - 1] + 28
        else:
            ans[i] = ans[i - 1]
        ticket_count += 1
        ticket_count_subway += subway[i]
print(ans[-1])
```

В этом решении начальная часть заключается в считывании данных, результатом являются два списка time, в котором хранится время совершения поездки в минутах от условного нуля, с учётом перехода на следующие сутки, и subway, в котором хранится признак того, была ли эта поездка совершена на метро (число 0 или 1). Дальше опустим эту часть программы.

Рассмотренное «жадное» решение не работает, например, на третьем примере из условия, где даны 4 поездки: 22:00, 23:00, 23:50, 00:30. Это решение разобьёт их на группы (22:00, 23:00) и (23:50, 00:30), что потребует двух тарифов «90 минут», а если разбить решения на группы (22:00) и (23:00, 23:50, 00:30), то получится тариф «единый» и «90 минут». Чтобы правильно разбивать решения на группы, необходимо использовать идею динамического программирования. Пусть ans[i] — ответ для первых i поездок, то есть минимальная стоимость оплаты первых i поездок. Переберём все поездки, для каждой поездки рассмотрим два случая тарификации.

1. Новая поездка оплачивается по тарифу «единый». Тогда ans[i] = ans[i−1] + 57.
2. Новая поездка оплачивается по тарифу «90 минут». Переберём первую поездку по этому тарифу j. Должны выполняться условия time[i] − time[j] <= 90 и сумма чисел subway[i], ..., subway[j] не больше 1. Тогда ответ равен ans[i] = ans[j−1]+85.

Из возможных способов получения ans[i] нужно выбрать наименьшее. Такое решение набирает 50 баллов. В частности, оно работает правильно, когда все поездки совершены на наземном транспорте.

```
ans = [10**9] * (n + 1)
ans[0] = 0
for i in  range(1, n + 1):
    ans[i] = 57 + ans[i - 1]
    count_subway = subway[i]
    j = i - 1
    while j >= 1 and time[i] - time[j] <= 90:
        count_subway += subway[j]
        if count_subway > 1:
            break
        ans[i] = min(ans[i], ans[j - 1] + 85)
        j -= 1
print(ans[-1])
```

Это решение предполагает, что никакие два использованных тарифа не «пересекаются» по времени, то есть не бывает ситуации, когда одна поездка оплачивается по одному билету, другая поездка — по другому билету, а потом снова поездка по первом билету. Но во втором примере из условия показано, что, например, в случае четырёх поездок В, М, М, В выгодно использовать тариф «90 минут» для наземного транспорта и одной поездки на метро, и отдельный тариф «единый» для другой поездки на метро, то есть использованные билеты будут пересекаться по времени. Можно показать, что пересечения возможны, только если внутри одного тарифа «90 минут» использовать отдельные билеты для совершения поездок на метро, чтобы соблюсти условия одного тарифа «90 минут» для поездок на наземном транспорте. А пересечения тарифов «90 минут» невозможны, то есть можно построить лучшее решение, в котором использованные тарифы «90 минут» не будут пересекаться.

Чтобы учесть это в решении, введём новый вид тарифа «90 минут+», который допускает любое число поездок на любом транспорте в течение 90 минут. Правила тарификации этого тарифа будут такими: 85 рублей плюс 57 рублей за вторую и каждую последующую поездку на метро. Правильное решение получится с использованием динамического программирования, как в предыдущем решении, с тарификацией последней группы поездок с номерами от j до i по правилам тарифа «90 минут+».

```
ans = [10**9] * (n + 1)
ans[0] = 0
```

```
for i in  range(1, n + 1):
    ans[i] = 57 + ans[i - 1]
    count_subway = subway[i]
    j = i - 1
    while j >= 1 and time[i] - time[j] <= 90:
        count_subway += subway[j]
        price = 85 + max(0, count_subway - 1) * 57
        ans[i] = min(ans[i], ans[j - 1] + price)
        j -= 1
print(ans[-1])
```

Задача 5. Все на съезд!

Автор задачи: Елена Андреева.

Подготовка задачи: Семён Кухаренко.

При $n = 1$ достаточно поставить желаемые секции единственного слушателя в разные дни.

При $n = 2$ также можно полностью удовлетворить обоих участников. Для этого разнесём в разные дни секции первого участника, а потом ещё не распределённые секции второго поставим так, чтобы он мог посетить все три.

При $n = 3$ всегда существует расписание, позволяющее двум участникам посетить все три желаемые лекции, а третьему — две из трёх, при этом полностью удовлетворить всех трёх не всегда возможно (это показано в примере из условия). Оптимальное расписание можно найти следующим жадным алгоритмом: распределим секции по очереди начиная с той, в которой заинтересовано больше всего участников. Каждый раз будем ставить секцию в такой день, где она принесёт больше всего пользы (то есть где на неё попадёт больше всего заинтересованных участников, учитывая уже распределённые секции). Перебором случаев (различных распределений участников по секциям) можно показать, что при всех возможных комбинациях желаемых секций этот алгоритм находит оптимальное решение.

Полное решение задачи заключается в переборе всех вариантов расписания. Такое решение может набирать разное количество баллов в зависимости от сделанных оптимизаций.

При $n \leqslant 100$ можно перебрать все возможные варианты расписания, а потом для каждого из участников посчитать, сколько из интересующих его секций он сможет посетить при данном расписании. Перебор расписаний можно организовать следующим образом: будем для каждой секции перебирать, в какой из дней она будет проведена, при этом для каждого дня запомним, сколько секций мы туда уже поставили, и не будем ставить новые секции в дни, в которые уже назначены четыре секции. При этом мы переберём всего $C_{12}^4 \cdot C_8^4 \cdot C_4^4 = 34650$ расписаний, для каждого из них за $O(n)$ посчитаем суммарное число посещений. Можно также заметить, что порядок следования дней не влияет на ответ. Следовательно, количество перебираемых расписаний можно уменьшить в 6 раз, если предположить, например, что первая секция всегда стоит в первый день, а секция с минимальным номером, стоящая не в первый день, стоит во второй день.

Это решение можно улучшить. Заметим, что количество различных пожеланий участников (т.е. троек интересных секций) невелико (всего $C_{12}^3 = \frac{12 \cdot 11 \cdot 10}{3 \cdot 2} = 220$). Для каждой возможной тройки секций a, b, c сохраним wishes[a][b][c] — количество человек, желающих её посетить, и при проверке расписания вместо подсчёта числа посещённых секций для каждого человека будем считать это число для тройки и умножать результат на значение wishes[a][b][c]. Тогда ответ мы найдём приблизительно за $\frac{34650}{6} \cdot 220 = 1270500$ действий.

Ниже приведён код решения на языке `Python` с применением всех оптимизаций. В этом решении используется нумерация с нуля как для секций, так и для дней.

```
wishes = [[[0 for _ in range(12)] for _ in range(12)] for _ in range(12)]

ans = -1            # макс. число посещений
res = [-1] * 12     # оптимальное расписание
```

```
plan = [-1] * 12  # перебираемое расписание
cnt = [0] * 3     # число уже поставленных секций в каждый день

def calc_vis_for_part(a, b, c): # считаем, сколько секций из тройки можно посетить
    if plan[a] == plan[b] and plan[b] == plan[c]:
        return 1
    if plan[a] == plan[b] or plan[b] == plan[c] or plan[a] == plan[c]:
        return 2
    return 3

def check():  # считаем суммарное число посещений для plan
    cnt = 0
    for a in range(12):
        for b in range(a + 1, 12):
            for c in range(b + 1, 12):
                cnt += calc_vis_for_part(a, b, c) * wishes[a][b][c]
    return cnt

# Рекурсивная функция перебора расписания
def gen(i):  # i - номер секции, которую будем распределять
    global ans, res, cnt, plan
    if i == 12:   # расписание сформировано, выполняем проверку
        nw = check()
        if nw > ans:
            ans = nw
            res = plan.copy()
        return
    for t in range(3):
        # проверяем, что мы не можем назначить секцию в третий день
        # если во второй день ещё не назначена ни одна секция
        if t == 2 and cnt[1] == 0:
            break
        if cnt[t] < 4:   # если в день t есть свободные места
            cnt[t] += 1  # назначаем секцию i в день t
            plan[i] = t
            gen(i + 1)   # вызываем рекурсивно алгоритм перебора
            plan[i] = -1
            cnt[t] -= 1

n = int(input())
for _ in range(n):
    a, b, c = map(int, input().split())
    # вычитаем 1 из номера секции, чтобы перейти в ноль-нумерацию
    # и упорядочиваем пожелания участника, чтобы a < b < c
    a, b, c = sorted([a - 1, b - 1, c - 1])
    wishes[a][b][c] += 1  # учитываем пожелания участника

# первую секцию поставим в первый день
# везде используется ноль-нумерация
plan[0] = 0
cnt[0] = 1
```

```
gen(1)

# выводим расписание
f = [[] for _ in range(3)]
for i in range(12):
    f[res[i]].append(i + 1)

for i in range(3):
    for j in range(4):
        print(f[i][j], end=' ')
    print()
```

В приведённом выше решении используется рекурсивный алгоритм перебора расписания. Но поскольку количество дней секций невелико и фиксировано, то вместо рекурсивного перебора можно использовать вложенные циклы. Например, будем считать, что первая секция стоит в первый день, номера оставшихся трёх секций первого дня переберём тремя вложенными циклами. Из оставшихся секций выберем секцию с минимальным номером и поставим её во второй день, другие три секции второго дня переберём вложенными циклами. Такое решение, вероятно, будет понятней для начинающих.

```
n = int(input())

# В словаре count считаем количество участников
# выбравших данную тройку секций
count = dict()
for i in range(n):
  part = tuple(sorted(map(int, input().split())))
  count[part] = count.get(part, 0) + 1

best_count = 0
best_ans = []

# d11, d12, d13, d14 - номера секций первого дня
d11 = 1
for d12 in range(2, 13):
  for d13 in range(d12 + 1, 13):
    for d14 in range(d13 + 1, 13):
      # day1 - список секций дня 1
      day1 = [d11, d12, d13, d14]
      # day23 - список нераспределённых секций
      day23 = [i for i in range(2, 13) if i not in day1]
      # 0, i1, i2, i3 - индексы элементов из списка day23
      # которые будут расставлены в день 2
      for i1 in range(1, len(day23)):
        for i2 in range(i1 + 1, len(day23)):
          for i3 in range(i2 + 1, len(day23)):
            # day2 - список секций дня 2
            day2 = [day23[0], day23[i1], day23[i2], day23[i3]]
            # day3 - список секций дня 3
            day3 = [i for i in range(2, 13) if i not in (day1 + day2)]
            curr_count = 0
            # Перебираем все пожелания и считаем количество выполненных
            for part in count:
              for day in (day1, day2, day3):
```

```
                if part[0] in day or part[1] in day or part[2] in day:
                    curr_count += count[part]
            if curr_count > best_count:
                best_count = curr_count
                best_ans = (day1, day2, day3)
for day in best_ans:
    print(*day)
```