Разбор задачи «Редактор»

Решение этой задачи напоминает известный алгоритм цифровой сортировки.
Заметим, что если некоторый набор слов начинается с одной и той же буквы, то их всегда выгодно набирать подряд. Действительно: предположим, что мы нашли оптимальное решение, в котором данное условие не выполняется. Предположим, к примеру, что слова, начинающиеся с буквы “a” набирались не подряд. Тогда порядок набора выглядит так:

Блок	Описание	Количество нажатий
1	Различные слова	X1
2	Слова, начинающиеся с буквы “a”	X2
3	Различные слова, не начинающиеся с буквы “a”	X3
4	Слова, начинающиеся с буквы “a”	X4
5	Различные слова	X5

Но тогда, переставим блоки следующим образом:

Блок	Описание	Количество нажатий
1	Различные слова	X1
2	Слова, начинающиеся с буквы “a”	X2
4	Слова, начинающиеся с буквы “a”	Не более X4 - 1
3	Различные слова, не начинающиеся с буквы “a”	X3
5	Различные слова	Не более X5

Тем самым суммарное количество нажатий уменьшилось по крайней мере на 1. А это противоречит утверждению о том, что выбранный порядок нажатий был оптимальным.
Значит, сначала следует набрать все слова, которые начинаются на ту же букву, что и первое слово. Остальные слова следует разбить на группы, в которых все слова начинаются с одной и той же буквы и набирать их по группам.
Заметим теперь, что порядок набора слов в каждой группе не влияет на количество нажатий, требуемое для набора остальных групп, поскольку при смене первой буквы слова, для набора следующего в любом случае придется стереть все буквы предыдущего голосовыми командами (а всегда выгодно сказать “повторить последнее слово”, поскольку в этом случае не приходится набирать пробел). Значит порядок набора слов в каждой группе можно определять независимо.
Рассмотрим набор слов, начинающихся с одной буквы. Отбросив первую букву, получим задачу, эквивалентную исходной с тем лишь исключением, что вместо пробела слова теперь разделяет пробел и первая буква слов нашей группы. Значит ее решение аналогично.
Осталось найти порядок слов, при котором слова, имеющие совпадающими первые *i* букв не будут разделены словом, у которого одна из первых *i* букв отличается от букв данных слов. Ясно что таким является, например, лексикографический (алфавитный) порядок.
Единственная проблема в том, что заданное во входном файле первым слово необходимо оставить первым. Можно предложить, например, такое ее решение: разделим слова на две группы – те, которые начинаются с той же буквы, что и первое слово и все остальные. Как было показано выше, слова в этих группах можно упорядочивать независимо. Упорядочим слова во второй группе лексикографически, а в первой группе виртуально удалим у всех слов первую букву и повторим операцию для них. Зная порядок, в котором следует набирать слова, нетрудно подсчитать количество нажатий, которое для этого потребуется.

Разбор задачи «Марсианские факториалы»

Для решения этой задачи достаточно заметить, что для того, чтобы число A заканчивалось на X нулей в системе счисления с основанием K необходимо и достаточно, чтобы оно делилось без остатка на K^X. Действительно: $A = A_P K^P + A_{P-1} K^{P-1} + ... + A_0 K^0$, если оно заканчивается на X нулей, то $A_i = 0$ для $i = 0,1,..., X-1$. Значит, $A = A_P K^P + A_{P-1} K^{P-1} + ... + A_X K^X = \left(A_P K^{P-X} + A_{P-1} K^{P-X-1} + ... + A_X K^0\right) K^X$, делится на K^X. Обратно, если $A = B \cdot K^X$, то представляя $B = B_Q K^Q + B_{Q-1} K^{Q-1} + ... + B_0 K^0$, получаем, что в разложении A младшие X цифр будут нулями.

Значит задача свелась к тому, чтобы определить, на какую максимальную степень K делится число $N!$. Поскольку $N!$ может быть очень велико, непосредственное его вычисление с целью такой проверки невозможно. Разложим число K на простые множители. Тогда если $K = P_1^{a_1} P_2^{a_2} \cdots P_S^{a_S}$, то если $N!$ делится на соответственно $P_i^{b_i}$, то $N!$ делится на K^Z, где $Z = \min_{i=1..S} \left\{ \left\lfloor \frac{b_i}{a_i} \right\rfloor \right\}$. Единственная оставшаяся проблема – как для простого числа P найти максимальную его степень, на которую делится $N!$.

Для этого применим следующие соображения: количество чисел, кратных P и не превышающих N равно $\left\lfloor \frac{N}{P} \right\rfloor$. Каждое из этих чисел даст по одному простому множителю P в $N!$. Но кроме того, $\left\lfloor \frac{N}{P^2} \right\rfloor$ чисел дадут еще по одному P, $\left\lfloor \frac{N}{P^3} \right\rfloor$ - еще по одному и т. д. Значит, $b_i = \left\lfloor \frac{N}{P_i} \right\rfloor + \left\lfloor \frac{N}{P_i^2} \right\rfloor + \left\lfloor \frac{N}{P_i^3} \right\rfloor + ...$ Заметим, что суммирование можно остановить, когда очередное слагаемое равно 0.

Разбор задачи «Фонтан»

Это самая сложная задача из предлагавшихся на олимпиаде. Идея ее решения следующая: предположим, мы хотим построить фонтан радиуса *R*. Тогда чтобы проверить, что мы можем это сделать, проделаем следующую операцию: увеличим радиус всех колонн на R, а каждую сторону холла уменьшим на 2R, чтобы его центр остался на том же месте. Тогда, чтобы проверить, что мы можем построить фонтан, достаточно проверить, что после этой операции круги колонн полностью покрывают прямоугольник холла. Если это так, то построить фонтан данного радиуса, очевидно, не удастся.

Чтобы проверить, что множество кругов покрывает заданный прямоугольник можно использовать следующий алгоритм: достаточно проверить, что точки пересечения любых двух окружностей, лежащие внутри прямоугольника, точки пересечения окружностей с прямоугольником и углы прямоугольника покрываются каким-либо другим кругом (область, не покрываемая кругами, всегда содержит одну из таких точек).

Для нахождения точек пересечения двух окружностей необходимо решить следующую систему уравнений относительно *x* и *y*:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = r_2^2 \end{cases}$$

После замены $x - x_1 = \bar{x}$, $y - y_1 = \bar{y}$ и обозначения $x_1 - x_2 = \Delta x$, $y_1 - y_2 = \Delta y$, система принимает вид:

$$\begin{cases} \bar{x}^2 + \bar{y}^2 = r_1^2 \\ \bar{x}^2 + 2\bar{x}\Delta x + \Delta x^2 + \bar{y}^2 + 2\bar{y}\Delta y + \Delta y^2 = r_2^2 \end{cases}$$

После вычитания первого уравнения из второго, получаем линейную зависимость между $\bar{x}$ и $\bar{y}$:

$$\begin{cases} \bar{x}^2 + \bar{y}^2 = r_1^2 \\ 2\bar{x}\Delta x + \Delta x^2 + 2\bar{y}\Delta y + \Delta y^2 = r_2^2 - r_1^2 \end{cases}$$

Отсюда несложно получить квадратное уравнение для $\bar{x}$ или $\bar{y}$.

Пересечение прямой с окружностью выполняется аналогично – там линейная зависимость сразу задается принадлежностью к прямой.

После этого для проверки того, что точка лежит внутри круга достаточно проверить, что расстояние от точки до его центра меньше его радиуса.

Однако этот метод сопряжен с техническими трудностями, связанными с проблемами, которые возникают при разрешении линейной зависимости относительно одной из переменных – при этом возникает опасность деления на 0, что требует рассмотрения двух различных случаев. Имеется еще один способ проверки, что набор кругов с заданной точностью покрывает прямоугольник, правда существенно использующий факт наличия требуемой точности вычислений. Предположим, что мы хотим проверить, что некоторый прямоугольник покрывается заданным множеством кругов. Если все четыре его вершины покрываются одним кругом, то, очевидно, прямоугольник полностью покрывается кругами. В противном случае разобьем прямоугольник на четыре одинаковых прямоугольника и рекурсивно проверим, что они покрываются кругами. Если в процессе рекурсии сторона прямоугольника стала меньше некоторой заданной величины, разумно подозреваем, что этот прямоугольник полностью кругами не покрывается (он, вероятно, лежит около точки пересечения кругов, но покрываемой другими).

Разбор задачи «Детский праздник»

Для решения этой задачи можно применить жадный алгоритм – будем добавлять шарики по очереди и давать шарик для надувания тому помощнику, который закончит этот процесс раньше всех. Действительно – предположим, что на каком-то этапе быстрее всех надует шарик помощник X, но выгодно дать шарик другому помощнику – Y. Тогда, поскольку все шарики одинаковые, X больше вообще не следует надувать шариков. Но тогда, отобрав шарик у Y и передав его X, получим, что максимум времени окончания работы среди этих двух помощников уменьшится, а значит общее время работы не увеличится.

Разбор задачи «Симпатичные узоры»

Для решения этой задачи применим динамическое программирование. Сначала заметим, что общее количество узоров не превышает $2^{M \cdot N} \le 2^{30}$, что означает, что ответ на задачу поместится в типы данных `longint` в Паскале и `long` в C. Будем также, не ограничивая общности, считать, что $M \le N$, а значит $M \le 5$.

Рассмотрим узор размера $M \times K$, где K может меняться в пределах от 1 до N, а M зафиксировано. Назовем профилем узора двоичное число, кодирующее его самый правый столбец, где черному квадратику отвечает 1, а белому – 0.

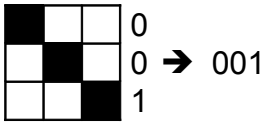

Рисунок 3

На рисунке 3 приведен пример узора 3×3 с профилем 001. Заметим, что колиество различных профилей не зависит от K и равно 2^M. Значит, поскольку $M \le 5$, то количество различных профилей не превышает 32.

Теперь для каждого K и каждого профиля Z посчитаем количество $A[K][Z]$ симпатичных узоров размера $M \times K$, имеющих профиль Z. Поскольку в узоре $M \times 1$ одноцветных квадратов 2×2 встретиться не может, для всех профилей $A[1][Z]=1$. Далее, при переходе от K к $K+1$ следует для данного профиля Z сложить те значения $A[K][Y]$, где в Y `xor` Z не встречается подрят двух нулей (это отвечает нарушению симпатичности).

Наконец, ответ на задачу получается при сложении по всем Z значений $A[N][Z]$.

Разбор задачи «Волшебная последовательность»

Это – несложная задача на творческий поиск. После попыток найти последовательности для различных небольших N, нетрудно заметить, что последовательность $\underbrace{1,1,\ldots,1}_{N-2},2,N$ удовлетворяет условию задачи.

Разбор задачи «Похожие матрицы»

Будем итеративно улучшать матрицу, следя за тем, чтобы количество чисел, не кратных *p* уменьшалось и матрица не становилась непохожей на исходную.

Будем обозначать ячейку на пересечении *i*-й строки и *j*-го столбца (i, j). Скажем, что ячейка (i, j) идет раньше (k, l) в матрице, если $i < k$ или $i = k, j < l$ (лексикографический порядок). Скажем, что ячейка (i, j), удовлетворяющая некоторому свойству – первая, удовлетворяющая этому свойству, если для всех остальных ячеек (k, l), удовлетворяющих этому свойству, (i, j) идет раньше (k, l) в матрице. Несложно показать, что наше определение корректно – т.е. если некоторому свойству удовлетворяет хотя бы одна ячейка, то найдется первая ячейка в матрице, удовлетворяющая этому свойству.

Найдем в матрице первую ячейку, число в которой не кратно *p*. Пусть это ячейка (i, j). Поскольку сумма чисел в этой строке кратна *p*, то в ней найдется еще одно число, не кратное *p*, скажем в ячейке (i, k). Уменьшим наше число на 1, а второе число увеличим на 1.

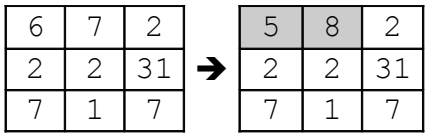

Рисунок 3

Сумма чисел в строке не изменилась, правда испортились суммы в столбцах *j* и *k*. Рассмотрим столбец *k*. Поскольку в нем было число, не кратное *p*, а сумма чисел в нем кратна *p*, то в нем найдется еще одно число, не кратное *p*, скажем в ячейке (l, k). Уменьшим его на единицу и повторим рассуждения для строки *l*. Рано или поздно мы снова попадем в столбец *j*, тогда увеличив соответствующее число на 1, мы получим, что профиль матрицы не изменится, первое не кратное *p*

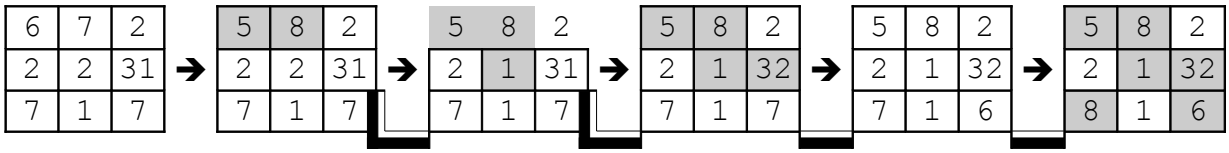

Рисунок 4

число уменьшится на 1, причем ни одно число не будет отличаться от своего исходного значения более чем на *p*, поскольку все изменения за один шаг делаются ровно на 1, значит чтобы измениться более чем на *p*, числу попутно нужно принять значение кратное *p*, а после этого оно уже не будет меняться.

Осталось показать, что в если повторять эту операцию, рано или поздно мы получим матрицу, кратную *p*. Но действительно, если первое не кратное *p* число в матрице имело остаток *x* от деления на *p*, то в течении *x* итераций оно будет оставаться первым не кратным *p* числом в матрице, после *x* итераций станет кратным *p*. Значит не более чем через $m \cdot n \cdot p$ итераций матрица станет кратна *p*.

Разбор задачи «Кубики»

Эта задача сводится к нахождению максимального паросочетания в двудольном графе. Назовем кубики и буквы имени сестры вершинами нашего графа и соединим кубик ребром с буквой, если эта буква написана на этом кубике. Заметим что граф действительно двудольный, а выбор кубиков для выкладывания имени эквивалентен построению паросочетания. Поскольку количество ребер в паросочетании не превышает количества вершин в меньшей доле, то искомое паросочетание действительно максимально.

Для нахождения максимального паросочетания разработаны стандартные алгоритмы. Наиболее популярный алгоритм, использующий удлиняющие чередующиеся цепи описан, например в книге Ахо А.В., Хопкрофт Дж.Э., Ульман Дж.Д. «Структуры данных и алгоритмы», нахождение максимального паросочетания через поиск максимального потока в сети с помощью алгоритма Форда-Фалкерсона описано, например, в книге Кормен Т., Лейзерстон Ч., Ривест Р. «Алгоритмы: построение и анализ», теоретическое исследование проблемы максимального паросочетания можно найти, например, в книге Романовский И.В. «Дискретный анализ».