Разбор задач

Задача 1 «Кампус»

Для решения задачи сначала вычислим суммарное количество комнат в одном подъезде. Число этажей, номер которых кратен k, равно целой части от деления n на k. Значит общее число комнат в подъезде $p = (n \text{ div } k) \cdot x + (n - (n \text{ div } k)) \cdot y$, здесь как $a \text{ div } b$ обозначена целая часть от деления a на b. Заменив в каждом запросе числа a_i на числа $(a_i - 1) \bmod p$, будем считать, что все комнаты находятся в первом подъезде и нумерация комнат начинается с 0.

Если бы все этажи имели по y комнат, то этаж, на котором находится комната, был бы равен $a_i \text{ div } y$. Это решение подходит для третьей подзадачи.

Для получения номера этажа в общем случае необходимо действовать следующим образом. Разобьем все этажи на блоки по k подряд идущих (последний блок может содержать менее k этажей). Каждый такой блок содержит $b = x + (k - 1) \cdot y$ комнат. Значит ниже искомой комнаты находится $a_i \text{ div } b$ полных блоков, а в блоке, в котором она находится, ниже нее находится $\min((a_i \bmod b) \text{ div } y, k - 1)$ этажей. Суммируя эти значения, получаем номер этажа, на котором находится комната.

Приведем фрагмент код на языке C++, вычисляющий номер этажа для одной комнаты.

```
long long p = (n / k) * x + (n - n / k) * y;
a = (a - 1) % p;
long long b = x + (k - 1) * y;
long long res = a / b * k + min((a % b) / y, k - 1) + 1;
```

Отметим, что для получения полного балла по этой задаче надо не забыть использовать 64-битный тип данных.

Задача 2 «Калькулятор»

Первое решение этой задачи основано на идее динамического программирования. Заметим, что все три описанные в условии операции являются монотонными: для большего значения n результат не меньше.

Рассмотрим значения dp[i][j][k] – минимальное число, которое можно получить из числа n, нажав на кнопку A i раз, на кнопку B j раз и на кнопку C k раз. Тогда dp[0][0][0] = n, а ответ на задачу находится в значении dp[a][b][c].

Для всех значений i, j, k значение dp[i][j][k] позволяет получить числа, которыми можно потенциально улучшить значения dp[i + 1[j][k], dp[i][j + 1][k] и dp[i][j][k + 1], рассмотрев результат действия операций A, B и C, соответственно. Приведем фрагмент кода на C++ для заполнения массива dp.

```
dp[0][0][0] = n;
for (int i = 0; i <= a; i++) {
  for (int j = 0; j <= b; j++) {
    for (int k = 0; k <= c; k++) {
      if (i < a)
        dp[i + 1][j][k] = min(dp[i + 1][j][k], dp[i][j][k] / 2);
      if (j < b)
        dp[i][j + 1][k] = min(dp[i][j + 1][k], (dp[i][j][k] + 1) / 2);
      if (k < c)
        dp[i][j][k + 1] = min(dp[i][j][k + 1], (dp[i][j][k] - 1) / 2);
    }
  }
}
```

Второй подход заключается в том, чтобы применить жадный алгоритм.

Проанализируем действия пользователя с конца. Посмотрим, из какого максимального числа можно получить число X за одно нажатие кнопки. Если была нажата кнопка A, то перед этим максимальное возможное значение $2X + 1$, кнопка B — значение $2X$, кнопка C — значение $2X + 2$. Пусть теперь конечное значение X, посмотрим, из какого максимального числа оно могло быть получено. Для этого заметим, что оптимально в конце нажимать кнопку C, перед ней кнопку A, а в начале кнопку B. Разворачивая действия пользователя обратно, получаем, что если всегда оптимально сначала нажимать кнопку B, затем кнопку A, а в конце кнопку C. Применив такую последовательность операций к исходному числу, получим минимальный возможный результат.

При решении подзадач 2 и 3 можно использовать описанные идеи не полностью, а, например, только что C оптимально нажимать после A, или B оптимально нажимать до A.

Для решения подзадачи 1 можно использовать полный перебор вариантов.

Задача 3 «Размещение данных»

Переводя условие на математический язык, получаем следующую формулировку: задан неориентированный граф, требуется пометить минимальное число вершин k, такое что при удалении любого ребра существует путь от каждой вершины до одной из помеченных. Также требуется определить число способов пометить таким образом k вершин.

Для решения подзадачи 1 достаточно сделать полный перебор всех множеств и для каждого из них проверить, подходит ли оно.

В подзадаче 2 граф во вводе является деревом. Докажем, что оптимальный способ пометить вершины ровно один: пометить все листья дерева – вершины степени 1. Действительно: не пометить лист нельзя, после удаления ребра, соединяющего этот лист с остальным деревом, из него нельзя будет достичь никакой другой вершины. С другой стороны, после удаления любого ребра в обеих получившихся компонентах остается хотя бы один из листьев исходного дерева, поэтому отметить все листья достаточно.

Теперь рассмотрим полную версию задачи. Напомним, что мостом называется ребро, удаление которого приводит к тому, что граф теряет связность. Заметим, что при удалении любого ребра, не являющегося мостом, граф остается связным, поэтому если отмечена хотя бы одна вершина, она будет достижима из любой другой. Значит интерес представляют только мосты.

Удалим все мосты и сожмем в одну вершину каждую компоненту связности. После этого вернем мосты. Получившийся граф будет деревом, для которого решением является пометить все листья. Поскольку листья дерева соответствуют компонентам связности после удаления мостов, то получается, что в каждом из них для решения задачи можно выбрать одну любую вершину.

Таким образом, k равно количеству компонент связности, получающихся после удаления мостов, таких, что в них ведет в исходном графе ровно один мост, а число способов выбрать k вершин равно произведению размеров этих компонент.

Для решения подзадачи 3 достаточно провести изложенные рассуждения и найти мосты перебором всех ребер графа и проверкой на связность после удаления. Для решения подзадачи 4 необходимо применить эффективный алгоритм поиска мостов в графе, который можно найти, например, в [21].

Задача 4 «Полезные ископаемые»

Для полного решения этой задачи необходимо аккуратно применить лемму Холла. Рассмотрим двудольный граф с долями X и Y. Пусть A – множеств вершин из доли X, обозначим как $N(A)$ множество соседей вершин из множества A. Лемма Холла утверждает следующее: в графе существует насыщающее долю X паросочетание тогда и только тогда, когда для любого A множество $N(A)$ содержит не меньше вершин, чем A. Доказательство леммы можно найти, например, в [23].

Применим двоичный поиск по числу принятых на планету партий, а также по числу роботов последней, неполной, партии. Пусть зафиксированы все партии, которые

будут приняты на планету, а также количество роботов из следующей партии. Необходимо научиться проверять, можно ли таким образом распределить роботов по клеткам, чтобы в каждой клетке оказалось не более q роботов.

Рассмотрим сначала задачу с $q = 1$. Построим двудольный граф, где вершины одной доли – это роботы, а вершины другой доли – клетки. Соединим робота и клетку ребром, если он может добраться до этой клетки. Заметим, что распределение роботов по клеткам соответствует насыщающему первую долю паросочетанию в получившемся графе. Используя алгоритмы поиска максимального паросочетания, можно проверить, существует ли в графе такое паросочетание. Этот подход позволяет решить подзадачи 1–3.

Для $q > 1$ во второй доле каждой клетке соответствует не одна, а q вершин. Задача по прежнему сводится к проверке существования насыщающего первую долю паросочетания, но количество вершин слишком велико, и алгоритмы поиска максимального паросочетания не позволяют решить даже 4 подзадачу.

Будем проверять наличие паросочетания, используя лемму Холла. Рассмотрим подмножество роботов. Заметим, что имеет смысл рассматривать только целые группы, а также если рассматривается группа с мобильностью m, то осмысленно включить в подмножество и все группы на той же базе с мобильностью не превышающей m, так как это увеличивает размер A, но не меняет размер $N(A)$. Таким образом, если условие леммы Холла выполняется для такого выбора подмножеств, то оно тем более выполняется и для остальных подмножеств.

Итак, отсортируем на каждой базе принятые партии роботов по неубыванию мобильности. Переберем для каждой базы несколько первых партий роботов и для них построим множество достижимых клеток. Если количество клеток в этом множестве, умноженное на q, меньше общего числа роботов в выбранных партиях, то мы нашли контрпример к лемме Холла и распределить роботов по полю нельзя. Если для всех множеств контрпример не найден, то искомое распределение возможно.

Осталось понять, как вычислить суммарное количество клеток, достижимых роботами. Для каждой базы это множество представляет собой квадрат, достижимой самой мобильной группой с этой базы. Для подсчета воспользуемся формулой включения-исключения: переберем все подмножества баз, для каждого подмножества вычислим пересечение всех искомых квадратов и учтем его с знаком «+», если выбрано нечетное число баз, либо «–», если выбрано четное число баз.

Оценим время работы алгоритма. Пусть на i-ю базу отправлено g_i групп роботов. Тогда необходимо перебрать $g_1 \cdot g_2 \cdot \ldots \cdot g_s$ вариантов выбора групп роботов, а для каждого варианта 2^s вариантов в формуле включения-исключения. Произведение при

фиксированной сумме максимально для равных сомножителей, значит итоговая оценка на время работы $O((2t/s)^s)$.

Для решения подзадач с $s = 1$ можно использовать жадный алгоритм, отдавая каждому роботу наиболее удаленную из свободных клеток, до которых он может добраться. Этот алгоритм можно также обобщить для решения подзадач с $s = 2$.