Региональный этап Всероссийской олимпиады школьников по информатике 2017 года.

Разбор задач

Разбор подготовил Андрей Станкевич. В скобках у каждой задачи указан автор оригинальной идеи.

Задача 1 «Кампус» (Андрей Станкевич)

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 $p = (n \operatorname{div} k) \cdot x + (n - (n \operatorname{div} k)) \cdot y$, здесь как $a \operatorname{div} b$ 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 $a_i \operatorname{div} y$. 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 $a_i \operatorname{div} b$ полных блоков, а в блоке, в котором она находится, ниже нее находится $\min((a_i \bmod b) \operatorname{div} y, k - 1)$ 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» (Андрей Станкевич)

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» (Андрей Станкевич)

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» (Станислав Наумов)

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 $s = 2$.

Задача 5 «Автоматизированное управление доставкой» (Андрей Станкевич, Андрей Лопатин)

Заметим сначала, что пакет, отправляемый в муниципальный почтовый центр, может иметь любой вес от a до $a + k - 1$ кг. Вес $a + i$ может получиться, например, после приёма

($a - 1$) пакета по 1 кг и пакета в ($i + 1$) кг.

Проверим теперь, можно ли добиться, чтобы вес контейнера при перевозке его в региональный сортировочный центр, был равен b кг. Рассмотрим максимальное количество пакетов, которое могло быть перевезено в муниципальный почтовый центр перед отправкой контейнера. Это количество $c = b$ div a. Минимальный суммарный вес этих пакетов равен $c \cdot a$, а максимальный – $c \cdot (a + k - 1)$, все промежуточные значения достижимы. Значит если $c \cdot a \le b \le c \cdot (a + k - 1)$, то ответ равен b.

В противном случае доставка c пакетов, даже если они все имеют максимальный возможный вес $a + k - 1$, не приводит к отправке контейнера. Значит для его отправки необходим $c + 1$ пакет. Минимальный вес ($c + 1$) пакета равен $(c + 1) \cdot a$. Значит в этом случае ответ равен $(c + 1) \cdot a$.

Приведем фрагмент программы на C++, вычисляющей ответ.

```
cnt = y / x;
lo = cnt * x;
hi = cnt * (x + k - 1);
if (lo <= y && y <= hi)
    ans = y;
else
    ans = x * (cnt + 1);
```

Задача 6 «Большой линейный коллайдер» (Андрей Лопатин, Андрей Станкевич)

Построим по описанию частиц скобочную последовательность: сопоставим e^+ частице открывающую скобку, а e^- частице закрывающую скобку. Частицы взаимно уничтожаются тогда и только тогда, когда они соответствуют парным скобкам.

Найдем для каждой частицы момент, когда она будет уничтожена. Для этого пройдем по построенной скобочной последовательности слева направо, и будем

поддерживать стек открытых, но пока не закрытых скобок. Если встречаем открывающуюся скобку, поместим ее в стек. Если встречаем закрывающуюся скобку, то посмотрим, пуст ли стек. Если стек пуст, то у этой скобки нет парной и соответствующая частица не будет уничтожена. Если стек не пуст, то скобка на вершине стека является парной к рассматриваемой. Они уничтожатся, когда окажутся в середине отрезка между ними, если их начальные координаты равны x_i и x_j, то это произойдет в точке $(x_i + x_j) / 2$ через время $|x_j – x_i| / 2$.

Теперь отсортируем совместно все моменты времени, когда происходят уничтожения частиц, и когда ученые хотят узнать, сколько частиц еще находятся в коллайдере, при этом запросы о количестве должны при равенстве, в соответствии с условием, идти после уничтожений. Теперь рассматриваем эти моменты по очереди. Исходно в коллайдере $c = n$ частиц. Если происходит уничтожение пары частиц, c уменьшается на 2. Если встречаем запрос о количестве частиц, выводим текущее значение c.

Чтобы избежать использования в решении вещественных чисел, можно исходно умножить все координаты и времена в запросах на 2.

Задача 7 «Силовые поля» (Георгий Корнеев)

Пусть требуется выбрать k полей, и минимальная ширина выбранного поля равна x_i. Рассмотрим все поля, ширина которых хотя бы x_i. Среди них оптимально выбрать k максимальных по y_j. На базе этого наблюдения получим следующее решение задачи.

Отсортируем все силовые поля по убыванию x_i. Рассмотрим первые k полей, поместим их в структуру данных Q, позволяющую добавлять элементы, получать и удалять минимальный элемент. В качестве ключа будем использовать значение высоты поля y_j.

Обновим ответ произведением текущего x_i и минимального значения y_j. Теперь перейдем к следующему силовому полю. Рассмотрим его y_j. Если оно меньше минимального значения в Q, то нет смысла брать это силовое поле, его использование будет заведомо хуже уже рассмотренных вариантов. Иначе удалим из Q поле с минимальным y_j, добавим в него наше поле и обновим ответ произведением текущего x_i и минимального значения y_j. Рассмотрев таким образом все поля, получим оптимальный ответ.

В качестве структуры Q можно использовать std::set из C++, приоритетную очередь или дерево отрезков.

Оценка на время работы решения: $O(n \log n)$.

Задача 8. «Повышение квалификации» (Илья Пересадин)

Переформулируем задачу математически. Задано подвешенное дерево, вершины которого пронумерованы таким образом, что родитель любой вершины имеет номер меньше, чем эта вершина. Есть несколько требований вида (p_i, k_i), где p_i – вершина, а k_i – целое число. Требуется выбрать отрезок минимальной длины $[L, R]$, чтобы для каждой вершины p_i существовала вершина с номером в выбранном отрезке, для которой p_i является предком k_i уровня.

Рассмотрим сначала недостаточно эффективное решение, которое тем не менее приблизит нас к пониманию полного решения задачи. Для каждого требования (p_i, k_i) составим список L_i номеров вершин, которые являются потомками уровня k_i вершины p_i. Необходимо выбрать несколько вершин так, чтобы из каждого множества L_i была выбрана хотя бы одна вершина, а разность между максимальным и минимальным номерами выбранных вершин была как можно меньше. Зафиксируем выбранную вершину с минимальным номером, обозначим этот номер как x. Заметим, что для каждого списка L_i оптимально выбрать вершину с минимальным номером y_i, таким что $y_i \geq x$. Перебрав все варианты выбора вершины x, например, из списка L_1, найдя для каждого из вариантов максимум из вершин y_i, и выбрав минимум из всех вариантов выбора x, мы получим ответ на задачу.

Если построить списки L_i в явном виде, перебрать все варианты вершины x и искать каждый раз вершины y_i просмотром всех элементов каждого из списков, описанное решение будет иметь время работы $O(n^3)$. Такое решение подходит для подзадачи 1.

Заметим, что построение списков L_i занимает $O(n^2)$, кроме того их суммарный размер равен $O(n^2)$. Отсортируем вершины каждого из списков по номеру, и заметим увеличении значения x значения y_i не уменьшаются. Таким образом в каждом списке указатель на y_i, и можно, перемещая указатели только вперед по спискам, обработать все варианты выбора x за суммарное время $O(n^2)$. Время работы такого решения $O(n^2 \log n)$ из-за сортировки списков, а в случае применения корзинной или цифровой сортировки время улучшается до $O(n^2)$. Оба варианта подходят для решения подзадачи 2.

Наконец, перейдем к описанию полного решения. Обойдем дерево в глубину, присвоив каждой вершине вспомогательные значения в порядке входа в эти вершины. Объединим вершины на фиксированной глубине h в список $V[h]$ в порядке входа. Заметим, что для каждого i список L_i состоит из вершин, идущих подряд в списке $V[h]$, то есть соответствует отрезку в этом списке. Заметим, что на каждом уровне такие отрезки либо вложены один в другой, либо не пересекаются.

Если в одном из таких отрезков A содержится другой отрезок B, то ограничения на отрезок A можно убрать: если в ответе будет вершина из отрезка B, то она будет также принадлежать отрезку A. Таким образом, на каждой глубине оставим набор непересекающихся отрезков, в каждом из которых должна быть выбрана хотя бы одна вершина.

Ключевая идея оптимизации решения до времени работы $O(n \log n)$ заключается в следующем. Рассмотрим m_i – максимальный номер вершины в множестве L_i. Обозначим как m_{min} минимальное значение среди m_i. Заметим, что нельзя выбирать значение x больше, чем m_{min}, поскольку в этом случае будут списки L_i, из которых мы не сможем выбрать ни одной вершины. И наоборот, если у нас выбрано значение $x \leq m_{min}$, то из каждого из списков можно выбрать вершину с номером не меньше x так, чтобы все ограничения были удовлетворены.

Поскольку на каждой глубине отрезки, соответствующие оставленным спискам L_i, не пересекаются, суммарное количество вершин в них не больше, чем n. Отсортируем вершины во всех списках L_i.

Рассмотрим значения y_i, выбранные для некоторого значения x. Пусть y_{max} равен максимуму среди них. Для фиксированного x нужно рассмотреть в качестве кандидата на ответ отрезок с x до y_{max} – это минимальный отрезок, для которого выполнены все ограничения. Для того, чтобы пересчитать y_i при увеличении x на один, достаточно посмотреть, была ли выбрана вершина, для которой выполнялось равенство $y_i = x$. Если такого значения y_i не было, то, ни один из y_i пересчитывать не придется. Иначе, нужно заменить этот y_i на следующую вершину в списке L_i – поскольку вершины в нем отсортированы по номерам, это и будет новая минимальная вершина с номером не меньше x на этой глубине, удовлетворяющая ограничению i.

Таким образом, перебирая x от 1 до m_{min}, и пересчитывая множество y_i, мы рассмотрим все отрезки, которые выполняют все ограничения, и у которых нельзя сдвинуть правую границу влево. Ответом будет самый короткий из всех таких отрезков.

Время работы решения – $O(n \log n)$, удаление лишних списков и сортировка списков L_i требует $O(n \log n)$ времени, а остальная часть решения выполняется за $O(n)$.