Задача A. Episodes

Автор задачи и разработчик: Анатолий Максудов

Подзадача 1

Для решения первой подзадачи достаточно внимательно прочитать условие и смоделировать процесс. Обозначим за s_i суммарный размер i-й серии, который еще осталось скачать. По условию изначально $s_i = v_i \cdot t_i$, так как браузер отображает сколько осталось скачивать серии при условии, что скорости меняться не будут.

Первой скачается та серия, у которой t_i меньше. Пусть, не теряя общности, $t_1 \leqslant t_2$. Тогда первая серия скачается ровно за t_1 минут, а вторая за это время успеет скачаться на $t_1 \cdot v_2 \leqslant s_2$. После этого момента времени скорость скачивания второй серии становится равна $v_1 + v_2$, поэтому остаток скачается за $\frac{s_2 - t_1 v_2}{v_1 + v_2}$. Таким образом, в ответе будут два числа — t_1 и $t_1 + \frac{s_2 - t_1 v_2}{v_1 + v_2}$.

Подзадача 2

Сведем эту подзадачу к предыдущей. Обозначим за V сумму всех v_i, эта величина по условию постоянна. Пусть снова, не теряя общности, t_1 минимально. Тогда спустя t_1 минут скорости скачивания второй и третьей серий станут равны $v_2 \frac{V}{v_2 + v_3}$ и $v_3 \frac{V}{v_2 + v_3}$, соответственно, так как отношение $\frac{v_2}{v_3}$ должно быть постоянно, а суммарная скорость должна оставаться равна V.

Получается, мы свели задачу к поиску ответа для двух серий размеров $s_2 - t_1 v_2$ и $s_3 - t_1 v_3$, соответственно, со скоростями скачивания $v_2 \frac{V}{v_2 + v_3}$ и $v_3 \frac{V}{v_2 + v_3}$, соответственно. Ее мы научились решать в предыдущей подзадаче.

Общие наблюдения

Обобщим рассуждения из предыдущих групп. Отсортируем серии по возрастанию t_i, и заметим, что они скачаются ровно в таком порядке. Пусть следующей скачается i-я серия, и нам известны текущие значения s_i, v_i и t_i.

Чтобы после завершения скачивания i-й серии соотношения скоростей скачивания еще не скачанных серий не изменились, а сумма скоростей осталась равна V, скорость скачивания всех не скачанных серий увеличивается в $\frac{V}{V - v_i}$ раз. Во столько же раз уменьшается оставшееся время еще не скачанных серий, то есть $t_j \leftarrow (t_j - t_i) \cdot \frac{V - v_i}{V}$, следовательно, как и было отмечено выше, порядок завершения скачивания серий от этого не изменился. Таким образом мы умеем переходить к той же задаче, но с количеством серий на один меньше.

Подзадачи 3 и 4

Пользуясь рассуждением выше, процесс можно промоделировать за время $\mathcal{O}(n^2)$: просто отсортируем все серии, и в момент скачивания очередной серии пересчитаем t_i и v_i для всех оставшихся. При этом в третьей группе тестов достаточно было написать более наивное моделирование, которое просто итерируется по целым моментам времени, и для каждого обрабатывает все скачавшиеся за последнюю минуту серии. Иными словами, в этой группе не обязательно было замечать, что порядок t_i не изменяется или сортировать серии по возрастанию t_i.

Во всех следующих подгруппах будем подразумевать, что серии уже отсортированы по значению параметра t, и будем считать, что мы перенумеровали серии таким образом, что $t_i \leqslant t_{i+1}$.

Подзадача 5

При $v_i = v_j$ формулы упрощаются довольно понятным образом: в каждый момент времени все скорости равны $w_r = \frac{V}{n - r}$, где r — количество уже скачавшихся серий. Таким образом, i-я серия

1. скачивается t_1 минут со скоростью $w_0 = \frac{V}{n}$;
2. скачивается $t_2' = \frac{s_2 - t_1 w_0}{w_1}$ минут со скоростью w_1;

3. скачивается $t'_3 = \frac{s_3 - t_1 w_0 - t'_2 w_1}{w_2}$ минут со скоростью w_2;
4. и так далее...

Заметим, что

$$t'_2 = \frac{s_2 - t_1 w_0}{w_1} = \frac{t_2 w_0 - t_1 w_0}{\frac{n}{n-1} w_0} = (t_2 - t_1)\frac{n-1}{n}.$$

.

Аналогично,

$$t'_3 = \frac{s_3 - t_1 w_0 - t'_2 w_1}{w_2} = \frac{t_3 w_0 - t_1 w_0 - (t_2 - t_1)\frac{n-1}{n} \cdot \frac{n}{n-1} w_0}{\frac{n}{n-2} w_0} = (t_3 - t_2)\frac{n-2}{n}.$$

Если продолжить то же рассуждение, можно доказать по индукции, что $t'_i = (t_i - t_{i-1})\frac{n-i+1}{n}$, что и будет уже почти являться ответом — останется только просуммировать $\sum\limits_{j \leqslant i} t'_j$, чтобы получить ответ для i-й серии.

Полное решение

Похожие преобразования можно провести и в общем случае. Обозначим за t'_i время, которое i-я серия скачивалась после окончания скачивания $i-1$-й серии. Тогда ответом для i-й серии будет $\sum\limits_{j \leqslant i} t'_j$. Осталось только эти t'_i посчитать.

Так как все скорости домножаются на одни и те же коэффициенты, то $\frac{V}{V - v_i}$ в момент времени, когда серии до i-й уже скачались, равно $\frac{v_i + \ldots + v_n}{v_{i+1} + \ldots + v_n}$. Для каждого промежутка времени рассчитаем его $\texttt{up}_i = \frac{v_i + \ldots + v_n}{v_{i+1} + \ldots + v_n}$. Тогда:

1. $t'_1 = t_1$;
2. $t'_2 = \frac{t_2 v_2 - t'_1 v_2}{v_2 \cdot \texttt{up}_1} = \frac{t_2 - t'_1}{\texttt{up}_1}$;
3. $t'_3 = \frac{t_3 v_3 - t'_1 v_3 - t'_2 (v_3 \cdot \texttt{up}_1)}{v_3 \cdot \texttt{up}_1 \cdot \texttt{up}_2} = \frac{t_3 - t_2}{\texttt{up}_1 \texttt{up}_2}$;
4. и так далее...

Полностью аналогично рассуждениям из предыдущей группы, все лишние множители сокращаются, и получается, что $t'_i = \frac{t_i - t_{i-1}}{\prod\limits_{j<i} \texttt{up}_j}$. Действительно, на это можно еще посмотреть по-другому: если бы «ускорений» не было, и скорости бы не менялись, все время разбивалось бы на периоды длины $t_i - t_{i-1}$. Здесь же каждый из этих периодов просто ускоряется в $\texttt{up}_i$ раз по сравнению с предыдущим.

Тогда для каждого промежутка времени рассчитаем его $\texttt{speed\_up}_i = \prod\limits_{j=1}^{i} \texttt{up}_i$ и вычислим ответы по очереди. Такое решение работает с асимптотикой $\mathcal{O}(n \log n)$, но до полного решения может не хватить точности вычислений.

Повышение точности вычислений

Поскольку $\texttt{up}_i = \frac{v_i + \ldots + v_n}{v_{i+1} + \ldots + v_n}$, то

$$\texttt{speed\_up}_i = \prod_{j=1}^{i} \texttt{up}_i = \frac{V}{V - v_1} \cdot \frac{V - v_1}{V - v_1 - v_2} \cdot \ldots \cdot \frac{v_i + \ldots + v_n}{v_{i+1} + \ldots + v_n} = \frac{V}{v_{i+1} + \ldots + v_n}.$$

То есть все $\texttt{speed\_up}_i$ можно вычислить за $\mathcal{O}(n)$, не жертвуя точностью вычислений. Причем числитель и знаменатель полученной дроби не превосходят 10^{18}, а значит все t'_i можно хранить как пару из числителя и знаменателя, а сложения выполнять в 128-битных целочисленных типах

данных. Подзадача 7 дает баллы в том числе решениям с хранением числителя итоговой дроби в переменной 64-битного типа данных.

При этом решение в `long double` с вычислением $\texttt{speed\_up}_i$ в сокращенной форме также получает 100 баллов.

Задача B. Board Game

Автор задачи и разработчик: Даниил Орешников

Для простоты обозначим j-ю по порядку ($1 \leqslant j \leqslant 3$) опцию i-го игрока за $a_{i,j}$, а соответствующее ей приращение очков за $c_{i,j}$. Так, например, $a_{2,1} = a_1$, а $c_{2,1} = a_2 \ \& \ a_3$.

Подзадача 1

В первой группе достаточно было написать полный перебор всех возможных выборов игроков за $\mathcal{O}(3^n)$. При $n \leqslant 14$ такой перебор укладывается в ограничения по времени и позволяет по всем возможным выборам игроков проверить, что $\sum a_{i,j}$ не превосходит данного во вводе ограничения, и для всех подходящих вариантов выборов обновить ответ через $\sum c_{i,j}$.

Подзадачи 2 – 4

Для решения групп со второй по четвертую на самом деле было достаточно добавить в этот полный перебор мемоизацию, то есть запоминание уже обработанных «состояний». Если рекурсивный перебор задавался функцией `generate(`i`, `a_{total}`, `c_{total}`)`, то можно было завести ассоциативный массив `memory`, который по тройке $(i, a_{\text{total}}, c_{\text{total}})$ хранил бы результат вызова функции `generate` от соответствующих аргументов — максимальное $\sum c_{i,j}$, которое можно набрать, если первыми i выборами было набрано суммарное число ресурсов a_{total} и суммарное число очков c_{total}. Если происходит повторный вызов `generate` с уже встреченными ранее аргументами, можно просто вернуть уже запомненный результат.

Оказывается, что в группах со второй по четвертую количество различных «состояний» в процессе перебора достаточно небольшое (линейное относительно n), поэтому такой перебор укладывался в ограничения по времени. Тем не менее, даже если не догадаться до такой оптимизации, каждую из этих групп можно было решить обычным перебором случаев, заметив, что опции у всех игроков, кроме $\mathcal{O}(1)$, одинаковы.

Подзадача 5

В пятой группе работал жадный алгоритм. Условия группы гарантируют, что $a_i \ \& \ a_{i+1} = a_i$, а в таком случае $a_i \ \& \ a_j = a_{\min(i,j)}$. Это означает, что если каждый игрок выберет минимальную доступную ему опцию, количество ресурсов будет минимальным возможным, а количество очков — максимальным возможным. При этом если каждый выбирает минимальную доступную опцию, то суммарное количество взятых ресурсов не будет превосходить $\sum a_i$, а значит не превысит лимит.

Подзадача 6

Шестая группа была расчитана на решение с помощью классического рюкзака за $\mathcal{O}(n \cdot A)$, где $A = X + \sum a_i$. Каждый игрок имеет выбор между тремя «предметами», вес j-го из которых равен $a_{i,j}$, а стоимость — $c_{i,j}$. Запустим классический алгоритм для решения задачи о рюкзаке, и будем вместо одного обновления делать три — для каждого из возможных предметов i-го игрока. Также надо учесть, что рюкзак, в котором какой-то из игроков не сделал выбор, не является корректным.

Полное решение

Полное решение требовало одной ключевой идеи: вместо того, чтобы делать классический рюкзак «с нуля», изначально выберем за каждого игрока минимальную доступную ему опцию, после чего дадим ему три выбора: ничего не менять, заменить выбор на вторую из опций и заменить

выбор на третью. Не теряя общности, пусть $a_{i,1} = \min(a_{i,1}, a_{i,2}, a_{i,3})$, тогда у i-го игрока будут три предмета: $(0,0)$, $(a_{i,2} - a_{i,1}, c_{i,2} - c_{i,1})$, $(a_{i,3} - a_{i,1}, c_{i,3} - c_{i,1})$.

Заметим, что суммарный вес выбранных «по умолчанию» опций не сильно меньше $\sum a_i$ (отличие между ними строго меньше $a_{n-1} + a_n$), и суммарный вес получившихся предметов не превосходит $a_{n-1} + a_n$, так как если просуммировать $\max\limits_j a_{i,j} - \min\limits_j a_{i,j}$ по всем i, большинство a_i войдут и с плюсом, и с минусом.

Таким образом, остается сделать рюкзак на полученных предметах, при чем его вес не превосходит $a_{n-1} + a_n + X$, что при данных в задаче ограничениях укладывается в ограничения по времени. Опять же, надо следить за тем, чтобы для каждого игрока был выбран ровно один из доступных ему предметов.

Задача C. Helicopter

Автор задачи: Даниил Голов, разработчик: Егор Юлин

Подзадача 1

Поскольку в этой группе тестов могут быть разрешены все направления перемещений, базовое решение с полным перебором всех возможных перемещений по времени не укладывалось, однако была возможность пройти все тесты этой группы, написав полный перебор с какими-то оптимизациями, основанными на описанных ниже наблюдениях. Мы здесь не будем фокусироваться на полном описании такого решения, так как есть достаточно большое количество идей, ведущих в сторону ключевой идеи задачи.

Подзадача 2

Заметим, что общее количество возможных позиций вертолета не очень большое, всего $n \cdot m \cdot A$. При этом в данной группе $A \leqslant 1$, значит любое действие имеет «стоимость» либо 0, либо 1, и тогда для поиска оптимального ответа мы можем запустить `0-1 bfs`.

Переходы из клетки (i, j) и высоты h:

- в $(i, j, 1)$ стоимости 1, если $h = 0$;
- в $(i, j, 0)$ стоимости 0, если $h = 1$;
- перелет в соседнюю клетку стоимости h, если $h \geqslant a_{i,j}$ и $h \geqslant a_{\texttt{next}}$.

Такое решение работает за время $\mathcal{O}(nm)$.

Подзадача 3

В данной подзадаче необходимо было двигаться только вправо. Используя рассуждения, описанные ниже, можно было просто просимулировать весь процесс, каждый раз перемещаясь на минимальной возможной высоте, либо же можно было заметить, что вертолету нет смысла находиться на высоте, не совпадающей с некоторым $a_{i,j}$, а тогда можно написать динамику $\texttt{dp}[j][k]$ — минимальный расход топлива, чтобы оказаться в $(1, j)$ на высоте $a_{1,k}$. Такое решение будет работать за $\mathcal{O}(m^2)$.

Подзадача 4

Если разрешенные направления перемещения — это «`RD`», то можно написать классическую динамику по матрице. Для решения четвертой группы можно в явном виде в состоянии динамики учитывать текущую высоту вертолета: $\texttt{dp}[i][j][h]$ — оптимальный расход топлива до состояния на высоте h сразу после попадания в клетку (i, j). Тогда

$$\texttt{dp}[i][j][h] = \min_{\texttt{prev} \text{ – предыдущая клетка}} \left(\texttt{dp}[\texttt{prev}][h'] + \max(h - h', 0) + h\right).$$

Это соответствует перемещению с высоты h' на высоту h в предыдущей клетке и перелету на высоте h. Если для каждой (i, j, h) перебирать все возможные предыдущие h', то решение не пройдет по времени, однако можно после подсчета $\texttt{dp}[i][j]$ насчитать префиксные и суффиксные минимумы величин $\texttt{dp}[i][j][h]$ и $\texttt{dp}[i][j][h] - h$ и с их помощью делать пересчет динамики за $\mathcal{O}(1)$.

Ключевая идея

Важной идеей для полного решения является тот факт, что всегда выгодно перемещаться на наименьшей возможной высоте. То есть, если мы перемещаемся между (i, j) и $(i + d_i, j + d_j)$ при $|d_i| + |d_j| = 1$, выгодно совершать этот перелет на высоте $h_0 = \max(a_{i,j}, a_{i+d_i,j+d_j})$.

Действительно, пусть в оптимальном ответе перелет совершается на высоте $h > h_0$. Тогда сделаем следующее: в клетке (i, j) опустимся до высоты h_0, совершим перелет на этой высоте, а затем поднимемся до h. В обоих случаях суммарный расход топлива равен h (во втором случае это $h_0 + (h - h_0)$. Такими изменениями можно без увеличения расхода топлива любой путь привести к описанному виду.

Подзадачи 5 и 6

Таким образом, для решения пятой группы, достаточно сделать ту же самую динамику, но для каждой клетки создать состояния, соответствующие высотам $\max(a_{i,j}, a_{i-1,j})$ и $\max(a_{i,j}, a_{i,j-1})$. Тогда пересчет динамики в сумме займет $\mathcal{O}(nm)$ времени.

Несложно заметить, что в шестой группе, хоть ограничение на направления и не дано в явном виде, оптимально всегда двигаться вправо или вниз. Любой другой путь гарантированно приведет к большим заратам топлива. Тогда мы можем, аналогично пятой группе, пересчитывать путь через динамику, при чем нет необходимости в состоянии динамики отслеживать высоту, так как при перемещении в (i, j) всегда надо будет подняться на высоту $a_{i,j}$.

Подзадача 7

Для решения седьмой группы достаточно было добавить один пересчет в динамику для пятой группы. Динамику для $\texttt{s} = \texttt{«RD»}$ мы считали по строкам. Теперь, перед тем как переходить от строки i к строке $i + 1$, пройдемся по строке справа налево и для каждого j обновим $\texttt{dp}[i][j]$ через $\texttt{dp}[i][j + 1]$. Так мы учтем возможные перелеты влево.

Поскольку любой путь выглядит как последовательность перемещений «один раз вниз и затем произвольное число перемещений влево или вправо» (при чем летать в одной строке и влево и вправо, очевидно, невыгодно), а вдоль каждого такого пути мы сделали обновление динамики, в конце в $\texttt{dp}[n][m]$ будет записан оптимальный ответ.

Подзадачи 8 и 9

Решения этих двух подзадач полностью аналогичны решению второй группы. В восьмой группе можно было написать $\texttt{0-}A$ $\texttt{bfs}$ по состояниям (i, j, h), а в девятой — запустить по тем же состояниям алгоритм Дейкстры. Время работы таких решений — $\mathcal{O}(nm \cdot A^2)$ и $\mathcal{O}(nm \cdot A \cdot \log(nm \cdot A))$.

Полное решение

Полное решение использует идею того, что один из оптимальных путей — перелетать всегда на самой низкой доступной высоте. Тогда можно найти оптимальное решение с помощью алгоритма Дейкстры, и есть несколько способов построить граф, на котором алгоритм Дейкстры надо запускать.

Один из способов — в качестве вершин графа использовать стороны клеток. Для каждого ребра поля между клетками (i, j) и $(i + d_i, j + d_j)$ создадим две вершины $v_{i,j,i+d_i,j+d_j}$ и $u_{i,j,i+d_i,j+d_j}$, после чего добавим ребро из $v_{(\ldots)}$ в $u_{(\ldots)}$ веса $\max(a_{i,j}, a_{i+d_i,j+d_j})$, соответствующее перелету. А также для одной клетки посмотрим на все ее $u_{\ldots,i,j}$ и $v_{i,j,\ldots}$ и добавим из каждого u в каждый v ребро с весом, равным расходу топлива при соответствующем изменении высоты.

Альтернативно можно было просто для каждой клетки создать четыре вершины, соответствующие нахождению в этой клетке на высоте $\max(a_{i,j}, a_{i+d_i,j+d_j})$ для всех (d_i, d_j). На таком графе найдем ответ алгоритмом Дейкстры за $\mathcal{O}(nm \cdot \log(nm))$.

Задача D. Bill Restoration

Автор задачи: Ильнур Валеев, разработчик: Даниил Орешников

Для простоты переобозначим за s строку A, в которой надо закрыть k подряд идущих символов, чтобы формулы выглядели более читаемо. Индексы s будем воспринимать в нумерации с нуля.

Подзадача 1

В первой группе достаточно было написать вероятностное решение, либо придумать стратегию как надо закрывать один символ. Фокусироваться на вероятностных решениях не будем, а стратегия, позволяющая восстановить один символ, существует, например, такая: если все символы строки равны, закроем первый, иначе закроем первый символ, который отличается от предыдущего. В таком случае закрытый символ восстанавливается однозначно.

Подзадача 2

Во второй подзадаче можно было попробовать различные эвристические решения. Например, можно было выбирать заранее одно и то же случайное подмножество позиций несколько раз, и для первого множества, в котором самая частая цифра встречается хотя бы на два раза больше любой другой, закрывать одно вхождение самой частой цифры. Также возможны другие похожие по смыслу решения, все из которых мы здесь освещать не будем.

Вспомогательное утверждение

Если мы закрываем k символов строке s, то ее длина не должна быть меньше чем $10^k + k - 1$, чтобы можно было эти k символов однозначно восстановить.

Действительно, пусть $|s|$ — длина строки s. Тогда по каждому варианту последовательности с k закрытыми цифрами нужно восстановить исходную; значит, каждой последовательности с k закрытыми цифрами однозначно надо поставить в соответствие восстановленную последовательность из $|s|$ цифр.

Вариантов строки с k закрытыми символами ровно $(|s|-(k-1)) \cdot 10^{|s|-k}$. Строк длины s всего $10^{|s|}$. Следовательно, необходимо, чтобы выполнялось $(|s| - (k-1)) \cdot 10^{|s|-k} \geqslant 10^{|s|} \iff |s| \geqslant 10^k + k - 1$.

Подзадача 3

В данной подзадаче все цифры от '0' до '7', и в двоичной записи их можно представить как последовательности от «000» до «111». Значит можно было посчитать XOR всех цифр и закрыть позицию номер $P = \bigoplus_{i=0}^{|s|-1} s_i$. Можно отметить, что $0 \leqslant P \leqslant 7$.

Для того, чтобы востановить закрытый символ, посчитаем XOR всех цифр s' и индекса элемента, который был закрыт. Из-за свойств XOR (что $a \oplus a = 0$), полученное число будет в точности равно цифре, которая была закрыта.

Подзадачи 4 и 5

Применим аналогичное решение. Отличие только в том, что нам необходимо восстановить $k > 1$ цифр, а в таком случае нам надо уметь получать k «блоков» информации. Заметим, что длина строки явно больше 8^k, поэтому можно выбирать позицию позицию начала закрываемого отрезка P от 0 до $8^k - 1$, и можно разбить ее на k цифр в восьмеричной системе.

Пусть p_j — j-я цифра P. Посчитаем ее как $\bigoplus_{i \bmod k = j} s_i$, то есть XOR всех цифр, стоящих на позициях с остатком j по модулю k. Посчитав все цифры, мы получим число P — позицию начала закрываемого отрезка.

Аналогично третьей подзадаче происходит и восстановление. Поскольку среди позиций, имеющих фиксированный остаток по модулю k, была закрыта ровно одна, можно тем же алгоритмом восстановить каждую закрытую цифру, зная соответствующую цифру P и все остальные цифры s, стоящие на позициях с тем же остатком.

Полное решение

Для полного решения остается только заметить, что `XOR` — не что иное, как сумма по модулю 8. Абсолютно та же идея работает и с суммой по модулю 10, причем теперь мы естественным образом получим P от 0 до $10^k - 1$. По сути алгоритм кодирования и восстановления почти не поменяется относительно предыдущих групп.

Задача E. Innopolis Data Center

Автор задачи и разработчик: Олег Христенко

Подзадача 1

В первой подзадаче для каждого запроса достаточно за $\mathcal{O}(2^n)$ перебрать множество серверов, которые войдут в систему, после чего посчитать попарные расстояния между ними и обновить ответ, если максимальное расстояние не превосходит d.

Подзадача 2

В случае, когда $d = n-1$, оптимальный ответ достигается, когда все n серверов входят в систему. Соответственно, для тестов этой группы ответы на запросы не зависят от их параметра u_i, и просто равны сумме попарных расстояний между всеми серверами.

Задача поиска суммы длин всех путей в дереве — классическая, и решается с помощью динамического программирования по подеревьям. Подвесим дерево за произвольную вершину и посчитаем для каждой вершины количество вершин в поддереве, тогда ребро (u, v) (где u — родитель v) принадлежит ровно $\texttt{size}(v) \cdot (n - \texttt{size}(v))$ путям. Остается только просуммировать эти величины по всем ребрам дерева.

Подзадачи 3 и 4

Сделаем важное наблюдение, которое также важно и для полного решения. В задаче по сути требуется найти множество вершин с диаметром не более d, содержащее данное u_i и имеющее максимальную сумму попарных расстояний. Следует заметить, что это достигается, если выбранное множество связно, и тогда у него есть определенный «центр» — середина диаметра (вершина при четном d и ребро при нечетном).

В таком случае все искомое множество — это просто некоторый центр c и все вершины, находящиеся на расстоянии не более $k = \left\lfloor \frac{d}{2} \right\rfloor$ от c, и сам c должен находиться от данного в запросе u_i на расстоянии не больше k.

Таким образом, общий план решения:

1. для каждого возможного центра предподсчитать ответ;
2. при ответе на запрос перебрать все возможные центры в k-окрестности u_i и выбрать среди их ответов максимальный.

В третьей и четвертой подзадаче это можно было делать за $\mathcal{O}(n^3)$ или $\mathcal{O}(n^2)$. Решение за $mathcalO(n^2)$ заключается в независимом поиске k-окрестности для каждой вершины, после чего применения для нее решения из второй подзадачи.

Частичные решения

Частичные баллы в последних двух группах можно было набрать различными эвристическими оптимизациями описанных выше решений. В частности, в некоторых тестах d четно и нет необходимости рассматривать ребра в качестве центров, в некоторых тестах d достаточно небольшое, поэтому аккуратно написанные решения за $\mathcal{O}(nd)$ могли эти тесты пройти.

Могут быть также решения, которые проходят только при достаточно больших d, например, пользуясь тем, что при больших d, если перейти от c_1 к соседнему с ним c_2, его k-окрестность не сильно изменится.

Полное решение

Для полного решения было необходимо быстрее пересчитывать ответы, проходя по дереву. Будем поддерживать три величины: количество вершин на расстоянии не более k, сумму расстояний до этих вершин, а также сумму попарных расстояний между этими вершинами.

Для каждой вершины разобьем эту задачу на две: посчитать это для k-окрестности «снизу» по дереву от нее и «сверху». Для подсчета снизу от v достаточно сделать динамику по поддеревьям: объединяя ответы для детей, надо всего лишь для каждого ребенка u сначала исключить вершины на расстоянии ровно k от него, затем объединить полученное поддерево с уже сохраненным для v ответом.

1. Чтобы исключить вершину x, достаточно воспользоваться `heavy-light декомпозицией`, с помощью которой можно делать запрос на пути между u и x, возвращающий сумму длин всех путей от x до остальных вершин в поддереве u (поддерево поддерживаем содержащим только вершины на расстоянии не больше k).

2. Затем, если объединять текущий ответ для v, $(\mathtt{sz}_v, \mathtt{sum}_v, \mathtt{ans}_v)$, с полученным ответом для ребенка $(\mathtt{sz}_u, \mathtt{sum}_u, \mathtt{ans}_u)$, получится:

   - $\mathtt{sz}_v \leftarrow \mathtt{sz}_v + \mathtt{sz}_u$;
   - $\mathtt{sum}_v \leftarrow \mathtt{sum}_v + \mathtt{sum}_u + \mathtt{sz}_u$, так как пути до вершин в поддереве u стали на 1 длиннее;
   - $\mathtt{ans}_v \leftarrow \mathtt{ans}_v + \mathtt{ans}_u + (\mathtt{sz}_v \cdot (\mathtt{sum}_u + \mathtt{sz}_u) + \mathtt{sz}_u \cdot (\mathtt{sum}_v + \mathtt{sz}_v))$, так как надо учесть пути между старыми вершинами, пути между новыми вершинами и пути от старых к новым, которые состоят из пути от старой до v, ребра (v, u) и пути от u к новой.

Для того, чтобы посчитать окрестности «сверху», используется аналогичная идея: возьмем ответ для родителя вершины, удалим все вершины на расстоянии ровно k, а затем обновим ответ. Для эффективного обновления ответа, опять же, будем использовать `hld`.

Чтобы объединить ответы «снизу» и «сверху», воспользуемся похожей на указанную выше формулу для объединения поддеревьев. Наиболее эффективная реализация такого решения работает за $\mathcal{O}(n \log n + qn)$, однако в предпоследней группе могли проходить и менее эффективные.

В качестве менее эффективных реализаций могли быть использованы двумерные структуры (например, двумерное дерево отрезков) или различные эвристические или корневые оптимизации для пересчета значений динамики без использования `hld`.