Задача 1. «Кастинг»

Рассмотрим решение данной задачи отдельно для случаев поиска наибольшего и наименьшего количества актеров. В первом случае подходящий актер должен обладать всеми тремя свойствами в отдельности. Наибольшее количество актеров, обладающих всеми тремя свойствами, не превосходит каждое из чисел a, b, c. Несложно построить пример распределения свойств, при котором это количество в точности равно $\min\{a, b, c\}$. Например, пусть первым свойством обладают первые a актеров, вторым свойством – первые b актеров, а третьим – первые c актеров. Тогда всеми тремя свойствами обладают только первые $\min\{a, b, c\}$ актеров.

При решении данной задачи в случае поиска наименьшего количества актеров рассмотрим два возможных способа рассуждений.

Первый способ заключается в следующем. Расставим актеров по кругу и будем последовательно назначать им свойства следующим образом: первым a актерам присвоим первое свойство, следующим за ними b актерам – второе, и следующим за ними c актерам – третье. При этом может оказаться, что некоторым актерам будет назначено два или три свойства. Поскольку движение осуществляется по кругу, то для того, чтобы какому-то актеру назначить три свойства, необходимо сначала сделать два полных круга, то есть назначить каждому актеру по два свойства. Количество актеров, которые получат три свойства, будет равно количеству актеров, которых обошли, делая третий круг. Если всего назначено $(a + b + c)$ свойств, а длина круга равна n, то за два круга будет назначено $2n$ свойств, а на третьем круге – оставшиеся $(a + b + c - 2n)$ свойств. При этом, если $(a + b + c - 2n) < 0$, то ни один актер не получит все три свойства. В этом случае ответ будет равен 0.

Теперь осталось только доказать, что если $(a + b + c - 2n) > 0$, то полученный вариант действительно будет соответствовать наименьшему количеству актеров. Для этого воспользуемся формулой включения-исключения для трех подмножеств A, B, C, соответствующих актерам, обладающим каждым из трех свойств:

$$|A| + |B| + |C| - |AB| - |BC| - |CA| + |ABC| = n.$$

Отсюда $|ABC| = n - a - b - c + |AB| + |BC| + |CA|$. Чтобы минимизировать $|ABC|$, нам нужно минимизировать $|AB| + |BC| + |CA|$. Минимальное значение $|AB|$ равно $(a + b - n)$, минимальное значение $|BC|$ равно $(b + c - n)$, минимальное значение $|CA|$ равно $(c + a - n)$. Подставляя эти значения в выражение для вычисления $|ABC|$, получаем требуемое соотношение:

$$|ABC| = n - a - b - c + (a + b - n) + (b + c - n) + (c + a - n) = a + b + c - 2n.$$

Второй способ рассуждений для случая поиска наименьшего количества актеров заключается в следующем. Сначала решим аналогичную задачу для двух свойств. Пусть a актеров обладают первым свойством, а b актеров – вторым. Тогда всего у нас имеется $(a + b)$ свойств на n актеров, следовательно, как минимум $(a + b - n)$ актеров обладают обоими свойствами. Пример получить несложно: выстроим актеров в ряд и наделим первым свойством первых a актеров, а вторым свойством – последних b актеров. При этом если $(a + b) \le n$, то актеров с обоими свойствами не будет вовсе. Поэтому ответ в задаче для двух свойств будет равен $\max\{a + b - n, 0\}$.

Теперь применим полученный результат к таким двум свойствам: 1) высокие и голубоглазые; 2) блондины. В первой группе не более $(a + b - n)$ актеров, во второй – c актеров. С учетом доказанного выше факта, всеми тремя свойствами будут обладать не более чем $\max\{\max\{a + b - n, 0\} + c - n, 0\} = \max\{\max\{a + b - n + c - n, c - n\}, 0\} = \max\{a + b + c - 2n, c - n, 0\} = \max\{a + b + c - 2n, 0\}$ актеров.

Последнее равенство верно в силу того, что $(c - n)$ не больше нуля.

Задача 2. «Города»

Одно из возможных решений данной задачи состоит из двух этапов. На первом этапе считываются входные данные и подсчитывается общее количество заданных городов, то есть букв ‘C’ во входном файле. Разделив это количество на 2, можно узнать требуемое количество городов K в каждом государстве.

На втором этапе выбираются города-клетки, относящиеся к первому государству. Это можно сделать разными способами. Самый простой из них заключается в следующем. Рассматриваем строки игрового поля сверху вниз, а элементы в каждой строке – слева направо и подсчитываем количество встречающихся городов. Процесс останавливается, как только встречается K-й город. Все пройденные к этому моменту города-клетки, включая клетку с K-м городом, следует отнести к первому государству, а остальные города-клетки – ко второму. Полученные государства будут связными, так как они состоят из нескольких последовательных полных строк и части еще одной строки, соседней с полными строками.

В простейшем случае, когда на игровом поле всего два города, для решения данной задачи достаточно найти один город и создать первое государство, состоящее только из клетки с этим городом. Второе государство будет состоять из всех остальных городов-клеток.

Задача 3. «A+B=C»

Для решения данной задачи можно применить метод динамического программирования [20]. С этой целью введем следующие обозначения.

Обозначим через *C'* число, состоящее из последних *k* цифр числа *C*.

Пусть s[*k*][*i*][*j*][0] – это количество способов разложения числа *C'* в сумму чисел *A'* и *B'*, каждое из которых состоит из *k* цифр, при этом *A'* начинается с цифры *i* ($0 \le i \le 9$), а *B'* – с цифры *j* ($0 \le j \le 9$). Рассматриваются также суммы, в которых слагаемые начинаются с нуля, например, 25 = 03 + 22.

Далее припишем слева к числу *C'* единицу и обозначим через *s*[*k*][*i*][*j*][1] количество способов разложения нового числа в такую же сумму, о которой шла речь выше (можно сказать, что здесь при сложении «переносится 1 в старший разряд»). Обобщая вышесказанное, *s*[*k*][*i*][*j*][*p*] – это количество способов представления числа *C'* в виде суммы с переносом *p* в старший разряд ($0 \le i \le 10,\ 0 \le j \le 10,\ 0 \le p \le 1,\ 1 \le k \le |C|$ и $|C|$ обозначает длину исходного числа – *n*).

С учетом сказанного не сложно придти к выводу, что ответ на поставленную задачу будет равен сумме всех *s*[|*C*|][*i*][*j*][0] по всем *i* и *j*, отличным от нуля (слагаемые по условию не могут начинаться с нуля). Теперь осталось только вычислить *s*[*k*][*i*][*j*][*p*]. Покажем, как это можно сделать.

Пусть *t* – *k*-я справа цифра числа *C*. Рассмотрим три случая.

1) $(i + j) = (10p + t)$, то есть, $(i + j)$ равно либо *t*, либо $(10 + t)$ в зависимости от значения *p*. В этом случае переноса вправо не происходит, и нужно только просуммировать *s*[*k* – 1][*x*][*y*][0] для всех $0 \le x \le 9$ и $0 \le y \le 9$ кроме тех, для которых $i = x$ или $j = y$, то есть, две соседние цифры равны.

2) $(i + j) = (10p + t - 1)$. В этом случае 1 переносится в правый разряд, и теперь нужно просуммировать *s*[*k* – 1][*x*][*y*][1] (опять же кроме тех, для которых $i = x$ или $j = y$).

3) В остальных случаях *s*[*k* – 1][*x*][*y*][1] = 0, так как нельзя получить цифру *t* из чисел, начинающихся на *i* и *j*.

Алгоритм, реализующий описанное решение, имеет линейную сложность относительно длины числа *C*, но константа будет довольно большой: для каждой длины числа *k* необходимо вычислить $10 \cdot 10 \cdot 2 = 200$ чисел *s*[*k*][*i*][*j*][*p*], а для вычисления каждого из них требуется порядка $10 \cdot 10 = 100$ сложений. Но на самом деле большинство значений *s*[*k*][*i*][*j*][*p*] будет равно нулю (см. третий случай, о котором речь шла выше), и всего потребуется вычислить лишь порядка 40 ненулевых таких значений. В итоге, асимптотическая сложность алгоритма будет порядка $4000 \cdot N$.

Заметим, что искомое количество пар красивых чисел *A* и *B* может быть очень большим, поэтому ответ в задаче предлагается выводить по модулю (10^9+7). Все

вычисления также нужно делать по этому модулю, чтобы в процессе вычислений не произошло переполнение используемого типа.

Задача 4. «Березовая аллея»

В зависимости от общего количества берез V ($V = N + M$) данная задача имеет ряд частичных и полных решений. Рассмотрим их последовательно, «от простого к сложному».

Первое частичное решение, которое имеет асимптотическую сложность $O(V^4)$ и позволяет получить ответ для $1 \le V \le 50$, основано на полном переборе и заключается в следующем. Рассмотрим точки с номерами i_1 и j_1 ($j_1 \ge i_1$) на левой стороне аллеи и точки с номерами i_2 и j_2 ($j_2 \ge i_2$) на правой ее стороне. Для каждой четверки таких точек вычислим периметр «четырёхугольника» с вершинами в этих точках. Сравним эти периметры с длиной ленты, и среди четырёхугольников, чьи периметры не больше длины ленты, найдем тот, на границе которого находится больше всего берез (количество берез на границе равно $(j_1 - i_1 + 1) + (j_2 - i_2 + 1)$).

Заметим, что здесь четырёхугольник может выродиться в треугольник и даже в отрезок. Чтобы не пропустить эти случаи, в неравенствах ($j_1 \ge i_1$, $j_2 \ge i_2$) допускается равенство номеров берез. В дальнейшем будем это учитывать.

Второе частичное решение, которое имеет асимптотическую сложность $O(V^3)$, позволяет получить ответ для $1 \le V \le 500$ и заключается в следующем. Выберем некоторые i_1, j_1, i_2 (обозначения аналогичны предыдущему частичному решению). Заметим, что для этих трех выбранных значений нас на самом деле интересует лишь одно значение j_2 – максимальное из тех значений, для которых периметр четырехугольника не превосходит длины ленты. Рассмотрим теперь точки i_1, $(j_1 + 1)$, i_2. Заметим, что интересующее нас значение j_2' для новой тройки чисел не может быть больше, чем j_2, то есть, с ростом j_1 значение j_2 не возрастает. Поэтому, чтобы найти искомое значение j_2', можно уменьшать его, начиная со значения j_2, на котором мы остановились на предыдущем шаге, пока периметр не станет меньше или равен длине ленты.

Этот прием использует идею «двух указателей»: в данном случае для фиксированных i_1 и i_2 «указатели» j_1 и j_2 пробегают весь диапазон значений по одному разу (j_1 – в порядке возрастания, j_2 – в порядке убывания), то есть для каждой пары (i_1, i_2) время работы алгоритма линейное. Если вместо идеи двух указателей использовать двоичный поиск [20] для нахождения j_2, то можно получить решение, имеющее асимптотическую сложность $V^3 \cdot \log V$.

Наряду с описанными частичными решениями существует несколько вариантов *полных* решений. *Первый вариант* основан на использовании двоичного поиска по ответу совместно с деревом отрезков и имеет асимптотическую сложность $O(V^2 \cdot \log^2 V)$. Опишем, как проверить, можно ли оградить заданной лентой некоторое фиксированное количество берез S.

Зафиксируем березы i_1 и i_2 как в предыдущих решениях, и рассмотрим, как проверить за время порядка $\log V$ существует ли четырехугольник с периметром, не превышающим L, и содержащий в себе S берез. Заметим, что для всех интересующих нас пар (j_1, j_2) значение суммы $(j_1 + j_2)$ одинаково, так как $(j_1 + j_2 - i_1 - i_2 + 2) = S$. Среди них нас интересует та пара, для которой периметр четырехугольника с вершинами i_1, j_1, j_2, i_2 минимален. Но для этой же пары периметр четырехугольника с вершинами 1, j_1, j_2, 1 также минимален среди всех четырехугольников, для которых $(j_1 + j_2)$ равно фиксированной величине $(S + i_1 + i_2 - 2)$. Важно, что эти значения уже не зависят от i_1 и i_2, их можно подсчитать один раз и далее только использовать. Но на самом деле нам нужны не все такие четырехугольники, а лишь те, для которых $j_1 \geq i_1$ и $j_2 \geq i_2$. Чтобы найти среди них четырёхугольник с минимальным периметром за время порядка $\log V$, модифицируем предподсчет следующим образом.

Пусть $a[S][k]$ – периметр четырехугольника с вершинами: 1, k, $(S - k)$, 1, то есть, строка массива $a[S]$ содержит периметры всех четырехугольников, охватывающих ровно S берез и содержащих березы с номерами 1 с каждой стороны аллеи. По каждой строке массива a построим дерево отрезков для функции min, то есть при фиксированном количестве берез S мы сможем вычислять $\min\{a[S][x], \ldots, a[S][y]\}$ с логарифмической асимптотической сложностью, что и требуется. Теперь для нахождения требуемого четырехугольника достаточно ответить на запрос:

$$\min \{a[S + i_1 + i_2 - 2][i_1], \ldots, a[S + i_1 + i_2 - 2][S + i_1 + i_2 - 2 - i_2].$$

Второй вариант полного решения имеет асимптотическую сложность $O(V^2 \cdot \log V)$. Такую сложность можно достичь, если заменить в предыдущем варианте решения дерево отрезков на разреженные таблицы (*Sparse Table*). При этом затраты по памяти возрастут до $V^2 \cdot \log V$. Более подробная информация о разреженных таблицах размещена в онлайн-статье [35].

Второй вариант полного решения можно *улучшить по памяти*, если обратить внимание на следующий факт. Будем перебирать пары (i_1, j_1) в порядке увеличения суммы $(i_1 + j_1)$, а при равных суммах – в порядке возрастания i_1. Для каждой такой суммы с ростом i_1 оба конца отрезка допустимых (для данного i_1) значений j_2 будут уменьшаться. Тогда для хранения этого отрезка можно использовать очередь с поддержкой операций

добавления в конец, удаления из начала и поиска минимум за O(1) [20]. В этом случае нам потребуется лишь *линейная дополнительная память* для хранения очереди, длина которой в каждый момент времени не будет превосходить N.

Задача 5. «Игральные кубики»

Решение данной задачи основано на том факте, что сумма очков на противоположных гранях кубика всегда равна 7. Таким образом, если брошено N кубиков, то сумма очков на их верхней и нижней гранях равна $7N$. Осталось вычислить, какое минимальное и максимальное количество кубиков необходимо бросить, чтобы сумма очков на верхних гранях могла оказаться равной предложенному во входных данных числу K.

Не сложно определить, что максимальное количество кубиков будет равно K – на каждом кубике выпадет одно очко. Минимальное количество кубиков будет равно округленному вверх числу K/6: на всех кубиках, кроме, может быть, одного, выпало 6 очков. Таким образом, максимальная сумма на нижних гранях равна $(7K - K) = 6K$, а минимальная сумма равна $(7\left\lceil \frac{K}{6} \right\rceil - K)$.

Задача 6. «Имена»

Следуя принципу «от простого к сложному» при рассмотрении возможных решений данной задачи, начнем с частичного решения для случая, когда имена матери и отца и соответствующие им строки состоят только из букв `a` и `b`. Пусть в первой строке x букв `b`, а во второй – y. Поскольку для того, чтобы получить лексикографически последнюю общую подпоследовательность, нужно начинать ее с как можно большего количества букв `b`, мы можем взять $\min\{x, y\}$ букв `b` из каждой последовательности. Найдем в обеих последовательностях букву `b` с номером $\min\{x, y\}$. Пусть в первой последовательности после нее идет p букв `a`, а во второй – q букв `a` (буквы `b` нас уже не интересуют). Тогда мы можем записать в искомую общую подпоследовательность $\min\{p, q\}$ букв `a`. Таким образом, искомая подпоследовательность будет состоять из $\min\{x, y\}$ букв `b`, за которыми следуют $\min\{p, q\}$ букв `a`.

Теперь перейдем к решению исходной задачи для последовательностей, состоящих из произвольных строчных латинских букв. Одно из возможных решений можно представить в виде повторяющейся последовательности следующих шагов:

1) найдем самую последнюю по алфавиту букву, которая встречается в обеих последовательностях;

2) запишем эту букву в общую подпоследовательность;

3) найдем самое левое вхождение этой буквы в обе последовательности;

4) удалим эти буквы и всё, что находится левее них в каждой последовательности;

5) для оставшихся последовательностей повторим эти шаги.

Асимптотическая сложность получающегося в этом случае алгоритма зависит от того, насколько оптимально будет реализована эта последовательность шагов. В частности, возможна квадратичная или линейная сложность алгоритма.

Квадратичная сложность получается в случае использования следующего алгоритма (наивная реализация). Будем линейным поиском каждый раз находить максимальную букву, встречающуюся в обеих последовательностях, добавлять ее в ответ и удалять из каждой последовательности эту букву и все буквы левее ее первого вхождения.

Линейная сложность получается в случае использования следующего алгоритма (эффективная реализация). Для каждой строки подсчитаем и запомним в массиве (или в словаре) количество букв `a`, количество букв `b`, и так далее до последней буквы латинского алфавита. Теперь будем просматривать этот массив по буквам от `z` к `a`, находить букву, которая еще встречается в обоих массивах, идти по каждому массиву до первого вхождения этой буквы, уменьшая счетчики для каждой пройденной буквы.

Следует заметить, что классическая задача о поиске наибольшей общей подпоследовательности [20], в которой требуется найти подпоследовательность наибольшей длины, решается с использованием динамического программирования. В предложенном решении используется «жадный» алгоритм, в соответствии с которым на каждом шаге выбирается самая выгодная для нас буква, не думая о том, что будет происходить на следующем шаге.

Задача 7. «Две окружности»

Рассмотрим сначала решение данной задачи для малого количества камушков. Если камешков не больше трёх, то ответом (одним из возможных) для первой окружности будет камушек с номером 1, для второй – камешки с номерами 2 и 3. Если камешков – четыре, то первой окружности будут принадлежать камешки с номерами 1 и 2, а второй окружности – камешки с номерами 3 и 4.

В более общем случае, когда $n \le 50$, в качестве частичного можно использовать следующее решение. Переберем все тройки камушков (это можно сделать тремя вложенными циклами). Для каждой тройки предположим, что все три камушка лежат на одной из искомых окружностей. Для каждого камушка проверим, лежит ли он на этой окружности, а затем для всех камушков, которые не лежат на полученной окружности, проверим, лежат ли они на одной из других окружностей. Если да – решение найдено, если нет – переходим к следующей тройке точек. При реализации этого решения для каждой тройки точек нужно предварительно проверить, что они лежат на одной окружности, то есть, не лежат на одной прямой.

Для того чтобы проверить, что четыре точки лежат на одной окружности, не обязательно находить ее центр и радиус. Более того, можно обойтись исключительно вычислениями с целыми числами, то есть, задача допускает решения *без использования вещественной арифметики.* Рассмотрим два возможных случая взаимного расположения точек A, B, C, D.

Случай 1. Точки A и D лежат по одну сторону от прямой BC, то есть, псевдоскалярные произведения $[AB, AC]$ и $[DB, DC]$ имеют одинаковый знак (их произведение больше нуля). Тогда угол BAC должен быть равен углу BDC, то есть, их косинусы должны быть равны. Косинусы можно выразить через скалярные произведения и длины векторов, при этом мы получим равенство:

$$\frac{(AB, AC)}{|AB| \cdot |AC|} = \frac{(DB, DC)}{|DB| \cdot |DC|}.$$

Поскольку обе части равенства – одного знака, то можно их возвести в квадрат и помножить на знаменатели:

$$(AB, AC)^2 \cdot |DB|^2 \cdot |DC|^2 = (DB, DC)^2 \cdot |AB|^2 \cdot |AC|^2.$$

Полученное выражение может быть вычислено без использования вещественной арифметики, что позволяет решить задачу точно (при этом потребуется реализовать операции с длинными числами).

Случай 2. Точки A и D лежат по разные стороны от прямой BC. В этом случае точки A и B лежат по одну сторону от прямой CD и для них можно повторить рассуждения для случая 1.

При рассмотрении обоих случаев нужно не забывать, что три точки могут оказаться и на одной прямой. В этом случае соответствующее псевдоскалярное произведение будет равно 0. Поэтому при проверке знака псевдоскалярного произведения все неравенства должны быть строгими.

Не сложно показать, что описанное решение имеет асимптотическую сложность $O(n^4)$, и поэтому не для всех заданных в условии задачи значений n позволяет получить правильный ответ. Чтобы его улучшить, покажем, как при переборе обойтись *десятью* окружностями вместо порядка n^3 окружностей.

Пусть задано не меньше пяти камешков. Рассмотрим первые пять из камушков. Среди них хотя бы три принадлежат одной из искомых окружностей. Переберем все возможные тройки камушков из этих пяти (их всего $C_5^3 = 10$). Далее решение в точности повторяет описанное выше частичное решение.

Данное решение уже является полным и позволяет получить правильный ответ для всего диапазона значений n. Асимптотическая сложность этого решения – $O(n)$.

Задача 8. «Столицы»

Начнем решение данной задачи с уточнения, какая тройка городов может быть столицей. Пусть города A, B, C находятся на расстоянии d друг от друга. Если для описания схемы дорог использовать граф, вершинами которого являются города, а ребрами – соединяющие их дороги, то очевидно, что ни одна из вершин, соответствующих городам A, B, C, не лежит на кратчайшем пути между двумя другими. Из этого следует, что существует вершина D, через которую проходят все три кратчайших пути, то есть вершины A, B, C лежат в разных поддеревьях относительно D.

С учетом сказанного можно получить следующую систему уравнений:

$$|AB| = |AD| + |BD| = d,$$
$$|BC| = |BD| + |CD| = d,$$
$$|AC| = |AD| + |CD| = d.$$

Поскольку эта система имеет единственное решение $|AD| = |BD| = |CD| = d/2$, то из этого следует вывод: чтобы три города являлись столицами, они должны лежать в разных поддеревьях относительно некоторого четвертого города и находиться от него на расстоянии $d/2$. Отсюда в частности следует, что d должно быть чётно.

Сказанное выше позволяет свести решение исходной задачи к вычислению количества описанных троек городов. Для этого можно использовать различные алгоритмы, отличающиеся по сложности в зависимости от количества городов n. Рассмотрим последовательно такие алгоритмы, используя принцип «от простого к сложному».

Относительно простое частичное решение, которое имеет асимптотическую сложность $O(n^4)$, заключается в следующем. Переберем все тройки вершин (их всего n^3).

Для каждой тройки вычислим расстояния между вершинами, например, обходом в ширину, и сравним эти расстояния с d. В случае если все они равны d, то тройка городов может быть столичной, и ее надо учесть при подсчете количества кандидатов.

Описанное решение можно улучшить, если с помощью алгоритма Флойда [20] заранее вычислить попарные расстояния между всеми вершинами. Получающееся в этом случае решение будет иметь асимптотическую сложность $O(n^3)$ и использовать объем памяти порядка $O(n^2)$.

Стремление уменьшить объем используемой памяти в описанном выше алгоритме приводит к новому частичному решению, суть которого заключается в следующем. Для каждой вершины найдем с помощью поиска в ширину или в глубину [20] количество вершин в каждом ее поддереве, которые находятся на расстоянии $d/2$ от нее. Затем найдем для каждой вершины суммы произведений всех неупорядоченных троек для всех (различных) поддеревьев каждой вершины.

Полученное таким образом частичное решение также имеет асимптотическую сложность $O(n^3)$, так как сумма произведений троек по всем поддеревьям вычисляется за время $O(n^3)$ суммарно для всех вершин (эта сложность не превосходит сложности ответа, которая равна $O(n^3)$), но при этом использует объем памяти порядка $O(n)$. Этого достаточно, чтобы получить ответ для $n \le 500$. Для больших значений n требуется дальнейшее улучшение этого решения, и это можно сделать следующим образом.

Пусть нам дан набор чисел a_1, …, a_S – количество искомых вершин в каждом поддереве. Требуется вычислить сумму всех возможных произведений вида $a_i a_j a_k$, где числа i, j, k попарно различны. Заметим, что это симметрический многочлен третьей степени, который можно выразить через многочлены:

$$P_1 = a_1 + \ldots + a_k,$$
$$P_2 = a_1^2 + \ldots + a_k^2,$$
$$P_3 = a_1^3 + \ldots + a_k^3.$$

Зная значения P_1, P_2, P_3, не сложно вычислить искомую сумму, которая будет равна $(P_1^3 - 3P_1P_2 + 2P_3)/6$.

Решения, использующие вышеописанную идею, уже имеют асимптотическую сложность $O(n^2)$. Действительно, многочлены P_1, P_2, P_3 можно вычислить за время $O(k)$, где k – степень данной вершины. Следовательно, суммарно эта часть решения будет работать за линейное время, а основной вклад в сложность алгоритма будет давать вычисление значений a_i для каждой вершины. На это потребуется квадратичное время (линейное для каждой вершины).

Все ранее рассмотренные решения исходной задачи являются частичными. Чтобы получить полное решение, покажем, как можно еще быстрее подсчитать количество вершин, находящихся на расстоянии $d/2$ от данной вершины в каждом поддереве.

Назначим произвольную вершину корнем дерева. Подсчитаем сначала для каждой вершины количество потомков на глубине $d/2$ в каждом из ее поддеревьев. Для этого воспользуемся обходом в глубину из корня дерева. Будем хранить количество вершин на каждой глубине, в которые мы вошли, но из которых еще не вышли. Разность этого количества на глубине $(h + d/2)$ в момент входа и выхода из некоторой вершины даст нам количество детей на глубине $d/2$ для этой вершины.

Теперь перейдем к более сложной части – подсчету количества вершин на расстоянии $d/2$ от вершины, путь к которым выходит из этой вершины вверх. Прежде рассмотрим некоторый путь, который начинается из вершины вверх. Самую близкую к корню дерева вершину этого пути назовем перегибом пути.

С учетом сказанного реализуем обход дерева в глубину. В процессе обхода для всех вершин в поддереве будет пересчитываться количество подходящих троек вершин, если выбрать эти вершины в качестве центральных, и возвращаться для каждой высоты число $add[h]$. Это число означает, что к ответам для всех вершин в поддереве, находящихся на высоте h, надо прибавить $add[h]$. Кроме того, обход в глубину будет возвращать список вершин на каждой высоте.

Будем поддерживать следующий инвариант: после выхода из вершины для всех вершин в ее поддереве посчитано количество путей, точка перегиба которых находится в этом поддереве (если учесть массив $add[h]$).

Для перехода от детей к родителю необходимо перебрать все вершины в меньшем (по количеству вершин) поддереве и применить добавки $add[x]$ к вершинам в нем, а также обнулить значения $add[x]$.

Пусть текущая вершина, в которую зашел обход дерева, имеет высоту r. Для того чтобы учесть пути, проходящие через нее как точку перегиба, надо по очереди попарно слить все поддеревья этой вершины. При каждом слиянии пары поддеревьев для каждой вершины на высоте h в меньшем поддереве необходимо увеличить число $add[2r + d – h]$ на единицу в большем поддереве. Также при этом слиянии суммируются попарно значения $add[x]$ для всех высот x. Все эти значения можно пересчитать за время, пропорциональное высоте меньшего из поддеревьев.

При объединении списков вершин поддеревьев, когда добавляется вершина из меньшего множества в большее, надо учесть, что там уже есть запись о том, что к нему

надо прибавить $add[x]$. Чтобы компенсировать эту величину, надо вычесть $add[x]$ из количества путей для этой вершины.

После того, как подсчитано количество вершин на расстоянии $d/2$, осталось только вычислить для каждой вершины количество подходящих троек. Это можно сделать, как описано в приведенном ранее решении.

В заключении следует отметить, что в этом решении каждая вершина участвовала в перекладывании из множества в множество при слиянии не более $\log n$ раз, поскольку размер поддерева после каждого перекладывания увеличивался как минимум в два раза. Кроме того, суммарное время работы всех слияний пропорционально количеству переложенных элементов. Из сказанного следует, что полученное полное решение имеет асимптотическую сложность $O(n \cdot \log n)$ и позволяет получить правильный ответ для всего диапазона изменения значений n.

Список рекомендуемой литературы

1. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 320 с. – (Серия «Основы информационных технологий»)
2. Андреева Е.В., Босова Л.Л., Фалина И.Н. Математические основы информатики. Элективный курс: Учебное пособие. – М.: БИНОМ. Лаборатория Знаний, 2007. – 312 с.
3. Арсак Ж. Программирование игр и головоломок. – М.: Наука, 1990. – 224 с.
4. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.
5. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — Пер. с англ. — М.: Мир, 1979. — 536 с.
6. Бентли Д. Жемчужины творчества программистов: пер. с англ. – М.: Радио и связь, 1990. – 224 с.
7. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. – М.: Мир, 1985.
8. Вирт Н. Алгоритмы и структуры данных. Пер. с англ. М.: Мир, 1989. – 360 с.
9. Гасфилд Дэн. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. – СПб.: Невский Диалект; БХВ Петербург, 2003. – 654 с.
10. Джонстон Г. Учитесь программировать. – М.: Финансы и статистика, 1989. – 336 с.
11. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов. – М.: Наука, 1990. – 384 с.
12. Задачи по программированию /С.М. Окулов, Т.В. Ашихмина, Н.А. Бушмелева и др.; Под ред. С.М. Окулова. – М.: БИНОМ. Лаборатория знаний, 2006. – 820 с.
13. Златопольский Д. М. Программирование: типовые задачи, алгоритмы, методы. – М.: БИНОМ. Лаборатория знаний, 2007. – 223 с.
14. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 1. – М.: Просвещение, 2008. – 220 с. – (Пять колец).
15. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 2. – М.: Просвещение, 2009. – 222 с. – (Пять колец).
16. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 3. – М.: Просвещение, 2011. – 222 с. – (Пять колец).

17. Кирюхин В.М. Информатика. Международные олимпиады. Выпуск 1. – М.: Просвещение, 2009. – 239 с. – (Пять колец).
18. Кирюхин В.М., Окулов С. М. Методика решения задач по информатике. Международные олимпиады. – М.: БИНОМ. Лаборатория знаний, 2007. – 600 с.
19. Кнут Д. Искусство программирования для ЭВМ. Т. 1-3. – М., СПб., Киев: Вильямс, 2000.
20. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 1999. – 960с.
21. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.
22. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 77 с.
23. Майерс Г. Искусство тестирования программ. Пер. с анг. под ред. Б.А. Позина. – М.: Финансы и статистика, 1982. – 176 с.
24. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. – СПб.: БХВ-Петербург, 2003. – 560 с.
25. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний. 2002. – 341 с.
26. Окулов С.М. Дискретная математика. Теория и практика решения задач по информатике: учебное пособие. – М.: БИНОМ. Лаборатория знаний. 2008. – 422 с.
27. Окулов С.М. Алгоритмы обработки строк: учебное пособие. – М.: БИНОМ. Лаборатория знаний, 2009. – 255 с.
28. Окулов С.М. Абстрактные типы данных. – М.: БИНОМ. Лаборатория знаний, 2009. – 250 с.
29. Окулов С.М. Динамическое программирование. – М.: БИНОМ. Лаборатория знаний, 2011. – 260 с.
30. Просветов Г.И. Дискретная математика: задачи и решения: учебное пособие. – М.: БИНОМ. Лаборатория знаний. 2008. – 222 с.
31. Рейнгольд Э. Комбинаторные алгоритмы: теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М.: Мир, 1980. – 476 с.
32. Столяр С.Е., Владыкин А.А.. Информатика. Представление данных и алгоритмы. – СПб.: Невский Диалект; М.: БИНОМ. Лаборатория знаний. 2007. –382 с.
33. Уэзерелл Ч. Этюды для программистов. – М.: Мир, 1982. – 288 с.
34. Шень А. Программирование: теоремы и задачи. – М.:МЦНМО, 1995. – 264 с.
35. Ахмедов М. Задача RMQ - 1. Static RMQ – http://habrahabr.ru/post/114980/