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 $n \leq 50$, 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.