Всероссийская олимпиада школьников по информатике 2017/2018
Региональный этап

Разбор задач

Автор разбора: Андрей Станкевич

Задача 1. Улучшение успеваемости

Автор задачи: Андрей Станкевич
Сложность задачи: доступна широкому кругу участников
Основные темы: целочисленная арифметика, формула, двоичный поиск

Пусть ученик получит x оценок в 5 баллов. Тогда общее число оценок будет равно $(a + b + c + x)$, а средняя оценка будет равна $(2a + 3b + 4c + 5x) / (a + b + c + x)$. По условию задачи должно выполняться условие

$$(2a + 3b + 4c + 5x) / (a + b + c + x) \geq 3.5.$$

Возможны два подхода для поиска минимального x, удовлетворяющего этому неравенству.

Подход 1. Вывод формулы.

Умножив обе части неравенства на положительную величину $2(a + b + c + x)$, получим неравенство

$$4a + 6b + 8c + 10x \geq 7a + 7b + 7c + 7x,$$

перенося x в левую часть неравенства, остальные слагаемые — в правую, получаем неравенство

$$3x \geq 3a + b - c.$$

Таким образом, минимальное подходящее значение x равно $(3a + b - c) / 3$, округленному вверх. Необходимо также учесть, что x не может быть меньше 0. Для вычисления соответствующего значения, например, в языке C++, можно воспользоваться выражением

```
x = max((3 * a + b - c + 2) / 3, 0);
```

Подход 2. Двоичный поиск.

Заметим, что каждая дополнительная оценка в 5 баллов увеличивает среднюю оценку, следовательно для поиска минимального значения x, при котором среднее значение не меньше 3.5 можно применить двоичный поиск.

Для того, чтобы избежать переполнения целочисленного типа при вычислениях в обоих подходах следует использовать 64-битный тип данных.

Решения, которые используют линейный поиск вместо двоичного, подходят для решения подзадач 1, 2, 3 и 4. Для решения подзадач 1 и 2 можно использовать более простые формулы. Для решения подзадачи 3 можно вычислять каждый раз значение средней оценки, суммируя все оценки по одной, вместо использования формулы $(2a + 3b + 4c + 5x) / (a + b + c + x)$.

Задача 2. Квадраты и кубы

Автор задачи: Андрей Станкевич
Сложность задачи: доступна широкому кругу участников
Основные темы: линейный поиск, операции с вещественными числами

Переберем значение y. Заметим, что поскольку $a \leq y^3 \leq b \leq 10^{18}$, достаточно перебирать значение y до 10^6. Убедимся, что $a \leq y^3 \leq b$ и найдем количество подходящих значений x.

Для фиксированного y значение x^2 должно лежать в диапазоне от $L = \max(y^3 - k, a)$ до $R = \min(y^3 + k, b)$: $L \le x^2 \le R$. Извлекая квадратный корень из всех частей неравенства, получаем ограничение на x: $\sqrt{L} \le x \le \sqrt{R}$.

Нас интересует количество целых значений в интервале от $\sqrt{L}$ до $\sqrt{R}$. Для его определения округлим вверх значение $\sqrt{L}$, пусть $l = \lceil \sqrt{L} \rceil$, и округлим вниз значение $\sqrt{R}$, пусть $r = \lfloor \sqrt{R} \rfloor$. Количество подходящих значений x для данного y, таким образом, равно $(r - l + 1)$.

Перебрав значения y и просуммировав подходящие значения x, получим ответ.

Для поиска квадратного корня из числа можно воспользоваться либо функцией для извлечения корня из вещественного числа `sqrt`, с последующим округлением, либо двоичным поиском.

Для решения подзадач 1 и 2, где $k = 0$, можно заметить, что искомые пары $x^2 = y^3$ являются шестыми степенями некоторого числа z, следовательно достаточно перебрать z до 10^3 и посчитать подходящие варианты, для которых $a \le z^6 \le b$. При этом для подзадачи 1 можно, вместо этого, перебирать значения от a до b, проверяя, что оно является шестой степенью числа, либо просто заметить, что $z^6 \le 1000$ выполнено только для $z = 1$, $z = 2$ и $z = 3$.

В подзадаче 3 можно перебрать все потенциальные пары чисел от a до b и проверить, являются ли они x^2 и y^3 для некоторых x и y.

В подзадаче 4 можно перебрать все варианты значений x и y.

В подзадаче 5 можно перебирать y как в полном решении, но проверять все значения от L до R по отдельности на то, является ли оно квадратом.

В подзадаче 6, перебирая y как в полном решении, можно использовать для поиска подходящих значений x метод двух указателей.

Задача 3. Лифт

Автор задачи: Георгий Корнеев
Сложность задачи: средняя, многие подзадачи доступны широкому кругу участников
Основные темы: моделирование, структуры данных, обработка событий

Основная трудность при решении этой задачи — правильная организация хранения данных в программе.

Будем использовать алгоритм обработки событий. Создадим события для каждого факта «сотрудник подошёл к лифту», для каждого такого события запомним его время и номер сотрудника. Будем хранить события в структуре данных, поддерживающей добавление и извлечение элемента с минимальным ключом, в качестве ключа будем использовать время события. Подойдет, например, структура данных из стандартной библиотеки C++ std::set, либо самостоятельно реализованная структура данных «приоритетная очередь».

Кроме событий «сотрудник подошёл к лифту» нам потребуются события «лифт прибыл на интересный этаж». Интересным при движении вверх будет только этаж, на котором сделан активный вызов, а при движении вниз — каждый этаж ниже текущего положения лифта, на котором находится хотя бы один ожидающий лифта сотрудник, а также первый этаж. Для каждого этажа будем хранить множество ожидающих лифта сотрудников.

Лифт может находиться в трех состояниях:

- на первом этаже в ожидании вызова,
- движется вверх к активному вызову,
- движется вниз на первый этаж.

Будем рассматривать события в порядке возрастания их времени, при необходимости добавляя новые события. Рассмотрим действия при обработке очередного события для каждого

состояния лифта. Для любого состояния лифта, когда сотрудник подходит к лифту, будем добавлять этого сотрудника в множество ожидающих лифт на соответствующем этаже.

Если лифт находится на первом этаже и ожидает вызова, то единственный возможный тип события — сотрудник подошел к лифту. Пусть это происходит на этаже s. При обработке этого события данный вызов становится активным, а лифт переходит в состояние «движется вверх к активному вызову». Если текущее время — максимум из времени события и времени, когда лифт последний раз освободился на первом этаже, — равно t, то добавим событие «лифт прибывает на этаж s» с временем $t + s$.

Если лифт движется вверх к активному вызову, то при обработке события «лифт прибыл на интересный этаж» необходимо сделать следующее: добавим в очередь события «лифт прибыл на интересный этаж» для всех этажей, где находятся сотрудники, ожидающие лифт, которые находятся ниже текущего положения лифта. Если время текущего события t, а этаж, на котором находится лифт — s, то время для события «лифт прибыл на интересный этаж a» равно $t + s – a$. Добавим также событие для первого этажа. Затем переместим всех находящихся на текущем этаже сотрудников из множества ожидающих лифт на этом этаже в множество перемещающихся на лифте, и перейдем к обработке следующего события.

Если лифт движется вниз, то обработка события «сотрудник подошёл к лифту на этаже a», на котором в этот момент нет других сотрудников, происходит следующим образом. Если лифт в этот момент находится не ниже этажа a, то необходимо добавить в очередь событие «лифт прибыл на интересный этаж a». При обработке события «лифт прибыл на интересный этаж a», если $a > 1$, то все ожидающие лифт сотрудники на этом этаже перемещаются с этажа в лифт, а если $a = 1$, то все сотрудники из лифта помечаются как обработанные, а время их прибытия на первый этаж равно времени рассматриваемого события.

В заключение описания алгоритма отметим, что при сравнении событий с одинаковым временем события «лифт прибыл на интересный этаж» должны идти после событий «сотрудник подошёл к лифту», а события типа «сотрудник подошёл к лифту» следует обрабатывать в порядке возрастания номера сотрудника.

Для решения подзадачи 1 можно применить простую формулу: ответ для первого и единственного сотрудника равен $t_1 + 2(a_1 – 1)$.

Для решения остальных подзадач необходимо с различной эффективностью промоделировать описанный алгоритм обработки событий.

В подзадаче 2 возможно пошаговое моделирование каждой секунды описанного в условии процесса. При этом можно каждую моделируемую секунду произвольным разумным образом обрабатывать входные данные, например, просматривая всех сотрудников и проверяя, не находится ли он на этаже, где находится лифт.

В подзадаче 3 событий мало, поэтому каждый раз для обработки события можно выбирать очередное событие проходом по всем имеющимся событиям, использование быстрых структур данных не требуется.

В подзадаче 4, как и в подзадаче 2, возможно пошаговое моделирование процесса, но необходимо аккуратное хранение информации о сотрудниках, чтобы суммарное время работы было $O(T + n)$ или $O(T + n \log n)$, где T — максимальное время, в которое может произойти событие (порядка 10^8 в этой подзадаче).

Наконец для решения последней пятой подзадачи необходима эффективная реализация моделирования, описанного в разборе, время работы $O(n \log n)$.

Задача 4. Мониторинг труб

Автор задачи: Николай Будин

Сложность задачи: высокая, первые подзадачи доступны широкому кругу участников

Основные темы: деревья, алгоритмы на графах, динамическое программирование

Переформулировав задачу математически, получим следующее условие: задано корневое дерево, на каждом ребре которого написана буква. Требуется покрыть все ребра дерева словами из заданного множества, причем слово покрывает путь вниз по дереву, буквы на котором при прочтении вдоль пути образуют это слово.

У этой задачи есть два принципиально разных решения: первое использует алгоритм поиска остовного дерева минимального веса в ориентированном графе, а второе основано на методе динамического программирования на поддеревьях. Рассмотрим оба решения.

Решение 1. Построим вспомогательный ориентированный граф. Вершины графа будут совпадать с узлами заданного во входном файле дерева, а ориентированные взвешенные ребра построим следующим образом. Если, начав от вершины u можно пройти вниз по дереву по буквам слова s_i и попасть в вершину v, то добавим в граф ребро uv с весом w_i. Также для каждой вершины u добавим ребро веса 0 в ее родителя в исходном дереве p_u.

Легко убедиться, что минимальная стоимость проверки всех труб равна стоимости минимального ориентированного остова в получившемся графе, с корнем в корне исходного дерева. Для поиска минимального остова можно использовать алгоритм Чу Йонджина и Лю Цзенхонга, известный в русскоязычной литературе как «алгоритм двух китайцев». Его время работы в самой простой реализации составляет $O(n^3)$, что достаточно для этой задачи.

Отметим, что построение графа в подзадачах 1, 3 и 4 можно выполнить наивно, попытавшись отложить каждое слово от каждой вершины, получив сложность $O(n^2m)$. В остальных подзадачах требуется использовать структуру данных «бор». А именно, сложим в бор все заданные слова и при обходе дерева от каждой вершины будем помнить, в какой вершине бора мы находимся. При таком подходе время работы этой части решения составляет $O(\sum \text{len}(s_i) + n^2)$.

Решение 2. Как и в предыдущем предыдущем решении, найдем все пути, которые можно покрыть каким-либо словом.

Теперь воспользуемся методом динамического программирования. Рассмотрим какой-нибудь ответ на задачу. Это множество путей, которое покрывает все ребра дерева. Возьмем некоторое поддерево и рассмотрим подмножество путей из ответа, нижние вершины которых лежат в этом поддереве. Заметим, что такое подмножество путей покрывает все ребра поддерева и также некоторый путь выше по дереву до корня поддерева. Значит, за состояние алгоритма можно принять пару вершин (u, v), одна из которых находится выше другой: нижняя обозначает корень поддерева, которое покрыто целиком, а верхняя обозначает вершину в которой заканчивается покрытый путь из корня. Обозначим как $dp[u][v]$ минимальную стоимость покрыть поддерево вершины v так, чтобы одновременно покрыть путь от u до v.

Будем считать значения одновременно для всех состояний с фиксированным v. Для листа $dp[u][v]$ равно минимальной стоимости слова, которым можно покрыть путь от u до v, если такое есть. Для внутренней вершины найдем сначала значения для детей. Для каждого ребенка

Итоговое время работы $O(n^2)$.

Для решения подзадачи 1 заметим, что каждое ребро необходимо проверять независимо, а значит можно просто для каждой буквы выбрать минимальную стоимость, для которой слово совпадает с этой буквой, и просуммировать соответствующие стоимости для всех труб.

В подзадаче 2 возможны различные подходы, основанные на динамическом программировании на прямой, также можно, построив граф как указано в разборе, вместо минимального остовного дерева найти кратчайший путь от корня до листа.

В подзадаче 3 помимо описанных в разборе решений возможно решение перебором или динамическим программированием с состоянием «множество покрытых ребер».

Наконец, учитывая технические сложности восстановления ответа в обоих подходах к решению, решения, которые только находят минимальную стоимость покрытия проходят все подзадачи, кроме подзадачи 6.

Задача 5. Удаление чисел

Автор задачи: Андрей Станкевич
Сложность задачи: доступна широкому кругу участников
Основные темы: моделирование, целочисленная арифметика, идея

Посмотрим, какой номер позиции, на которой будет стоять число с позиции t после одной операции удаления. Заметим, что если t делится на k, то это число будет удалено на этой операции. В противном случае будет удалено t div k чисел, меньших k, следовательно число с позиции t после операции удаления переместится на позицию $(t - t \text{ div } k)$, где div обозначает целочисленное деление.

Проведем моделирование процесса, считая количество выполненных операций удаления. Будем поддерживать только номер позиции, который имеет число n. Если на очередном шаге удаления номер позиции делится на k, то ответ найден. В противном случае продолжаем моделирование. Заметим, что любое число в итоге будет удалено, за исключением чисел от 1 до $k - 1$, но по условию $k < n$, поэтому такой случай рассматривать не требуется.

Оценим максимальное число операций удаления. После одной операции удаления n умножается на величину $(1 - 1/k) \geq 0.99$. Следовательно количество операций в худшем случае составляет порядка $\log_{1/0.99} 10^{18} \approx 4200$.

Для решения подзадач 1 и 2 с $k = 2$ можно внимательнее изучить, как происходит удаление чисел в этом случае. На первом шаге будут удалены числа, кратные 2, на втором шаге числа, кратные 2 после целочисленного деления на 2, и так далее. Число n будет удалено на шаге $t + 1$, где t — количество единиц на конце двоичной записи числа n.

В подзадачах 1 и 3 можно непосредственно промоделировать описанный в условии процесс. При этом можно каждый раз сдвигать оставшиеся элементы, заполняя освободившиеся позиции.

В подзадаче 4 необходимо моделировать процесс более эффективно, например, храня элементы в списке или отмечая удаленные элементы, в массиве, но не сдвигая оставшиеся элементы на освободившиеся позиции.

Задача 6. Старая книга

Автор задачи: Георгий Корнеев
Сложность задачи: доступна широкому кругу участников
Основные темы: двоичный поиск, два указателя, формула

Начнем с описания решения первых двух подзадач, где $k = 0$. Выясним, какое минимальное количество страниц могло быть в книге. Для этого будем последовательно суммировать натуральные числа, пока сумма $1 + 2 + 3 + \ldots + n$ не окажется больше или равна s. Возможны два случая.

Случай 1. $1 + 2 + 3 + \ldots + n = s$. В этом случае в книге может вообще не быть иллюстраций, все страницы содержат текст и на них написан их номер. Ответ равен 0.

Случай 2. $1 + 2 + 3 + \ldots + n = t > s$. Тогда заметим, что $t - s < n$, поэтому существует страница с номером t. Значит, если на этой странице находится иллюстрация, а остальные страницы содержат текст. то сумма номеров оставшихся страниц равна s. Таким образом ответ равен 1.

Перейдем теперь к решению задачи для случая $k > 0$. Пусть в книге $(k + z)$ иллюстраций, будем перебирать значение z, начиная с 0. Первые k иллюстраций по условию находятся на первых k страницах, посмотрим, можно ли разместить остальные z так, чтобы сумма номеров страниц с текстом оказалась равна s. Пусть в книге n страниц, тогда минимальная возможная сумма номеров страниц с текстом получается, если все z оставшихся иллюстраций находятся

на последних z страницах, а максимальная – если эти иллюстрации занимают страницы с $k + 1$ по $k + z$. Таким образом, сумма номеров страниц с текстом t удовлетворяет условию:

$$(k + 1) + (k + 2) + \ldots + (n - z) \le t \le (k + z + 1) + (k + z + 2) + \ldots + n$$

Заметим, что любое t в указанном диапазоне может быть получено как сумма номеров страниц с текстом. Действительно, начав с ситуации, когда все страницы с иллюстрациями имеют максимальные возможные номера, будем постепенно перемещать иллюстрации к началу, каждый раз сумма будет возрастать на 1. Процесс завершится, когда все иллюстрации располагаются на первых страницах, а сумма номеров страниц с текстом будем максимальной возможной.

Найдем (линейным или двоичным поиском) максимальное n, для которого $(k + 1) + (k + 2) + \ldots + (n - z) \le s$. Если для этого n также выполнено второе неравенство $s \le (k + z + 1) + (k + z + 2) + \ldots + n$, то для данного числа страниц и числа иллюстраций существует распределение иллюстраций по страницам, при котором сумма номеров страниц с текстом равна s. Если же второе неравенство не выполнено, то ни для какого n нельзя одновременно выполнить оба неравенства (для меньших n второе неравенство тем более не будет выполнено, а для больших n нарушится первое неравенство). Поэтому для данного z требуемого распределения иллюстраций по страницам не существует.

Заметим, что максимальное значение z, которое потребуется перебрать имеет порядок $O(\sqrt{s})$, поэтому выполняя действия при фиксированном z за O(1) или O(log s), мы получаем достаточно эффективное по времени решение. Искомое значение n можно искать двоичным поиском за O(log s), либо, учитывая, что при увеличении z значение n также увеличивается, использовать метод двух указателей, параллельно увеличивая z и n.

В подзадачах 1 и 3 маленькие ограничения на k и s позволяют выполнять все действия с меньшей эффективностью.

Задача 7. Степень дерева

Автор задачи: Дмитрий Саютин
Сложность задачи: средняя, многие подзадачи доступны широкому кругу участников
Основные темы: динамическое программирование, обработка деревьев, обход в глубину

Самый длинный путь в графе называется его диаметром. Красота салюта, таким образом, равна числу вершин в диаметре заданного во входных данных дерева.

В первых двух подзадачах $m = 1$ и требуется просто найти диаметр явно заданного дерева. В первой подзадаче это можно сделать наивно. Запустим обход в глубину от каждого листа, выберем самый длинный путь, который в нем начинается, выберем самый длинный из рассмотренных путей.

Во второй подзадаче требуется более эффективное решение. Мы рассмотрим два способа поиска диаметра, один проще в реализации (хотя его корректность требует отдельного доказательства, которое мы оставим как упражнение), зато второй допускает обобщение, которое потребуется в полном решении.

Способ 1. Запустим обход в глубину от любой вершины, например от корня дерева, и найдем наиболее удаленную от него вершину u. Найдем теперь наиболее удаленную от u вершину v, запустив еще один обход в глубину, на этот раз от вершины u. Путь между u и v является диаметром дерева. Доказательство этого факта оставим как упражнение, основная идея доказательства — рассмотреть диаметр дерева x—y и убедиться, что если u находится не ближе к корню, чем y, то путь x—u не короче, чем путь x—y.

Способ 2. Используем динамическое программирование. Обозначим как down[u] максимальный путь вниз по дереву, начинающийся в вершине u, а как down2[u] — самый

длинный из путей, начинающихся в вершине u, идущих вниз по дереву, и использующих в качестве первого ребра ребро, отличное от того, которое использует самый длинный путь.

Тогда down[u] равно max(1 + down[v]), где максимум берется по всем v — детям вершины u, а down2[u] равно второму максимуму среди этих величин. Если у вершины нет детей, от для нее положим down[u] = down2[u] = 0, аналогично положим down2[u] = 0 для вершин, у которых только один ребенок.

Ответ на задачу — число вершин в диаметре дерева — равен максимальному значению down[u] + down2[u] + 1 по всем вершинам u.

Перейдем теперь к полному решению для $m > 1$.

Диаметр любого дерева устроен следующим образом: он составлен из двух путей, ведущих от некоторой вершины x до листа, либо представляет собой путь от корня до листа.

Рассмотрим первый случай, когда диаметр составлен из двух путей. Заметим, что оптимально составить путь следующим образом: в том экземпляре дерева T, в котором находится вершина x, необходимо взять диаметр, а копиях T, которые подвешены к листьям, являющихся концами диаметра, необходимо взять максимальные по длине пути, начинающиеся в корне. Аналогично следует поступить и в копиях T, подвешенных к концам этих путей, и так далее. Таким образом, если количество вершин в диаметре дерева T равно d, а максимальный по длине путь от корня до листа содержит p вершин, то ответ равен $d + 2(m - 1)p$.

Если предположить, что диаметр представляет собой путь от корня до листа, то его оптимальная длина равна mp, заметим, что это значение не может превышать $d + 2(m - 1)p$ при $m > 1$, поэтому этот вариант не может быть лучше.

Диаметр дерева мы научились находить при решении второй подзадачи, а значение p равно down[r] для корня дерева r.

При решении подзадач 3 и 4 можно воспользоваться наивным способом поиска диаметра дерева, подзадача 3 также допускает решение с помощью других алгоритмов динамического программирования.

Задача 8. Обработка больших данных

Автор задачи: Павел Кунявский

Сложность задачи: высокая, ориентирована на опытных участников, некоторые подзадач имеют среднюю сложность

Основные темы: динамическое программирование, структуры данных, оптимизации динамического программирования, операции с множествами

Структура корректных отрезков, описанная в условии, напоминает устройство структуры данных дерево отрезков. Начнем строить дерево отрезков на числах от 0 до $2^n - 1$, но не будем разбивать те вершины, которые соответствуют отрезкам, целиком находящимся внутри заданных во входных данных отрезков с одинаковыми значениями, сделав их листьями. Ясно, что нет смысла вызывать STORE от более мелких отрезков.

Пусть вершина u соответствует отрезку [L, R]. Докажем индукцией по $R - L$, что для заполнения всех значений в ячейках памяти отрезка [L, R] и только их, можно в качестве первой операции выполнить операцию STORE для всего отрезка [L, R]. Если вершина u соответствует листу, то утверждение очевидно. Пусть у вершины u есть два ребенка v и w, которые соответствуют отрезкам [L, M] и [$M + 1$, R]. Тогда, если одна и та же операция STORE используется для присваивания значений в обоих отрезках для v и w, то она должна быть первой и должна быть выполнена для отрезка [L, R]. В противном случае операции заполнения отрезков [L, M] и [$M + 1$, R] проводятся независимо, тогда по индукционному предположению первая операция для отрезка [L, M] — STORE для всего этого отрезка. Выполним вместо нее

операцию STORE для отрезка $[L, R]$ с таким же значением, а затем остальные операции отрезка $[L, M]$ и все операции отрезка $[M + 1, R]$. Заметим, что результат выполнения будет тем же, поскольку операции внутри отрезка $[M + 1, R]$ полностью заменят присвоенные этой операцией значения, а для отрезка $[L, M]$ эти операции неотличимы. Таким образом утверждение доказано.

Рассмотрим теперь решение с использованием динамического программирования. Обозначим как $opt[u][c]$ минимальное количество операций STORE, которое необходимо выполнить для заполнения отрезка $[L, R]$, соответствующего вершине u дерева отрезков, требуемыми значениями, если первым действием мы выполняем операцию STORE для отрезка $[L, R]$ и значения c.

Если у вершины u нет детей, то $opt[u][c] = 1$, если в итоговом заполнении памяти на позициях, соответствующих отрезку вершины u должно стоять значение c, и $opt[u][c] = 2$ в противном случае.

Пусть теперь у вершины u два ребенка v и w, тогда $opt[u][c]$ выбирается среди следующих значений: $opt[v][c] + opt[w][c] - 1$ (единица вычитается, потому что мы выполняем одну операцию заполнения отрезка значением c вместо двух в отрезках детей), минимуму величины $opt[v][c] + opt[w][d]$ по всем возможным значениям d и минимуму величины $opt[v][d] + opt[w][c]$ по всем возможным значениям d.

Время работы такого решения равно $O(nm^2 \log n)$, поскольку число вершин в рассматриваемом фрагменте дерева отрезков есть $O(n \log n)$, в каждой вершине хранится m значений и пересчет осуществляется за $O(m)$.

В таком варианте решение проходит только подзадачи 1, 2 и 3. Избавимся от одного из множителей m, чтобы пройти подзадачу 4. Заметим, что $\min(opt[v][c] + opt[w][d])$ по всем возможным значениям d равен $opt[v][c] + \min(opt[w][d])$, второе слагаемое не зависит от c и может быть посчитано один раз. Аналогично для $\min(opt[v][d] + opt[w][c])$. Получаем решение за $O(nm \log n)$, которое проходит подзадачи 1, 2, 3 и 4.

Чтобы решить последние две подзадачи необходимо сделать еще одно важное наблюдение. Величины $opt[u][c]$ для одного u и разных c различаются не более чем на 1.

Действительно, пусть $opt[u][d] = t$. Значит если первым действием мы заполняем весь отрезок, соответствующий вершине u, значениями d, то все значения могут быть получены за t операций. Но тогда если первым действием заполнить весь отрезок значениями c, а затем применить последовательность операций из оптимального решения для d, мы получим решение за $t + 1$ операцию. Значит, $opt[u][c] \le opt[u][d] + 1$, что и требовалось.

Следовательно, вместо хранения значений $opt[u][c]$ для каждой вершины и каждого возможного значения, достаточно хранить $best[u]$, равное минимуму $opt[u][c]$ по всем значениям c и множество значений $bestv[u] = \{c_1, c_2, \ldots, c_s\}$, для которых $opt[u][c_i]$ минимально и равно $best[u]$. Для остальных значений d значение $opt[u][d]$ равно $best[u] + 1$.

Если вершина u является листом, который должен быть заполнен значениями c, то $best[u] = 1$, $bestv[u] = \{c\}$.

Иначе пусть v и w — дети u. Если есть значение c, которое входит одновременно в $bestv[v]$ и $bestv[w]$, то $best[u] = best[v] + best[w] - 1$, $bestv[u] = bestv[v] \cap bestv[w]$. Если же $bestv[v]$ и $bestv[w]$ не пересекаются, то $best[u] = best[v] + best[w]$, $bestv[u] = bestv[v] \cup bestv[w]$.

Заметим, что каждое значение попадает в $bestv$ от листа, и далее может попасть только в $bestv$ вершин на пути от листа до корня дерева отрезков, которых $O(k)$. Пересечение и объединение множеств осуществляется за время, пропорциональное их размеру, таким образом, время работы решения равно $O(kn \log n)$.

В заключение отметим, что для решения подзадачи 1 можно использовать любые, в том числе переборные решения, а в подзадачах 2 и 5 можно не использовать дерево отрезков, а считать значение динамического программирования для всех различных корректных отрезков.