Разбор задачи «Автодополнение»

Добавим все слова из словаря в бор. В боре помимо обычных переходов go будем также хранить обратные переходы по буквам $back$ (означающие удаление буквы) и не более трех переходов, описывающих автодополнение — каждая вершина слова w имеет переход в вершину, обозначающую конец w. Так как популярность слова соответствует его индексу в словаре, хранить достаточно только первых три перехода, а остальные игнорировать.

После этого также добавим в бор требуемое сообщение s, однако уже без переходов-автодополнений. Пометим все вершины этого слова s как «хорошие» — при дописывании буквы в конец слова, оказаться мы можем только в них.

Осталось посчитать динамику на построенном боре. Пусть $dp[k][v]$ — количество способов оказаться в вершине бора под номером v, сделав при этом ровно k ходов. Тогда из этого состояния мы можем перейти в состояния $dp[k+1][back[v][c]]$ (удаление буквы), $dp[k+1][go[v][c]]$ (дописывание буквы, надо также не забыть проверить, что $go[v][c]$ — «хорошая» вершина и $dp[k+1][autocompletion[v][index]]$ (соглашение с автодополнением) для некоторых букв c (если такие состояние существуют, то есть существуют такие вершины в боре).

Ответом на задачу будет $\sum_{i \leqslant k} dp[i][finish]$, где $finish$ — номер вершины бора, соответствующей концу слова s. Асимптотика решения — $O(26 \cdot \sum(|w_i|) \cdot k)$.

Разбор задачи «Локи и Шахматы»

В качестве решения данной задачи предполагалось использовать какую-нибудь структуру данных. Например, дерево отрезков. Поступим следующим образом: для всех строк и столбцов заведем деревья отрезков. Каждый раз, когда нужно сдвинуть пешку в каком-либо направлении, будем находить первую пустую клетку в этом направлении. Это можно сделать так: бинарный поиск по длине отрезка, затем находить сумму в дереве отрезков. Если сумма равна длине отрезка, то сдвигать левую границу, иначе правую. Так же это можно сделать за один спуск. После того, как свободная клетка найдена (если такой нет, то делать ничего не нужно), остается присвоить ноль на предыдущую позицию пешки и единицу на только что найденную свободную клетку.

Разбор задачи «Побег с Асгарда»

Переберём, на какой из палуб будет ехать первая группа. Тогда все остальные должны разбиться на две палубы, что бы всем хватило места. Пусть суммарный размер всех групп кроме первой это D. Тогда если на первой палубе летит x человек, то на второй летит $D - x$. Если мы поймём, какие количества человек могут быть на первой палубе, то нам надо проверить, что хотя бы для одного из них, все остальные люди поместятся на вторую палубу. Что бы понять эти количества, можно воспользоваться задачей о рюкзаке, где в качестве предметов будут группы, а в качестве рюкзака — первая палуба.

Разбор задачи «Игра с числами»

Ответ -1 только если $a = b = 0$. Иначе заметим, что нам выгодно всегда брать либо максимальное число(оно же b), кроме возможно последнего шага, где может быть надо взять число чуть меньше, потому что мы можем взять любое число от 1 до b, либо всегда минимальное(оно же a), кроме возможно последнего шага. Исходя из этих замечаний несложно посчитать ответ на задачу.

Разбор задачи «Защитники Асгарда»

Рассмотрим вершину. Посчитаем количество инверсий, которое получится в результате между парами вершин, лежащими в поддеревьях различных сыновей этой вершины. Заметим, что оно не зависит от порядка вершин в поддеревьях детей, а зависит только от порядка обхода детей в текущей вершине. Найдем для каждой пары детей количество инверсий, если один из них идет раньше другого. И затем за $2^p \cdot p$, где p — количество детей, динамическим программированием по подмножествам найдем оптимальный порядок обхода. Осталось научиться вычислять количество инверсий, которое образуется между парами детей. Для этого функция обхода дерева будет возвращать сбалансированное дерево поиска, содержащее индексы всех вершин в поддереве. А внутри сливать деревья поиска от детей, каждый раз приливая меньшее к большему, и в конце добавлять

туда свой индекс. Тогда, чтобы вычислить количество инверсий между парами детей, переберем все пары детей, для каждой пары переберем все вершины ребенка, размер которого меньше, и для каждой вершины сделаем запросы к дереву поиска второго ребенка, чтобы узнать, сколько там вершин с индексом больше/меньше, чем эта. Оценим асимптотику. Посмотрим на случай, когда мы делаем запрос к дереву поиска с фиксированным индексом. Это значит, что после поднятия в родителя той вершины, в которой произошел запрос, размер поддерева, в котором лежит этот индекс, увеличится хотя бы в 2 раза. Значит, количество запросов с одним индексом будет не больше $log(n) \cdot 7$, каждый запрос выполняется за $O(log(n))$, итоговая асимптотика получается $O(n \cdot 2^p \cdot p + n \cdot log^2(n) \cdot p)$, где p — максимальное количество детей.

Разбор задачи «Тренировки Тора»

Будем отвечать на запросы в обратном порядке. Для начала, будем рассматривать рамку как не более чем 4 полоски $1 \times x$ или $y \times 1$. Так как отвечаем мы на запросы в обратном порядке, полоски не добавляются, а удаляются. Что происходит при удалении полоски?

При удалении полоски некоторые непораженные области объединяются. А именно, если у нас есть полоска $(x1, y1), (x2, y2)$, то объединиться могут только клетки в области $(x1-1, y1-1), (x2+1, y2+1)$ (так как $(x1, y1), (x2, y2)$ — полоска, то эта область размера не больше, чем $3 \times x$ или $y \times 3$. Используя СНМ (систему непересекающихся множеств) мы можем объединить все непораженные области за $O(\max(n, m))$.

Итоговая асимптотика: $O(q \cdot \max(n, m))$.

Разбор задачи «Красивое число»

Будем вычитать из данного числа наибольшее красивое число, меньшее его, пока не получим 0. После каждого вычитания полученное число будет меньше 111..1 (количество единиц равно длине числа). Следовательно, количество чисел в ответе не превзойдет длины исходного числа.

Разбор задачи «Испытание»

Заметим, что после применения операции побитового «ИЛИ» элемента массива f_i с числом x в f_i могли измениться только те биты, на месте которых в x стоят единицы (то есть нулевые биты в f_i, стоящие на позициях, соответствующих единицам в числе x, стали единицами). После применения операции побитового «И» с числом x в f_i могли поменяться только те биты, на месте которых в x стоят нули (единичные биты в f_i становятся нулями). То есть при операции «ИЛИ» имеет смысл рассматривать только единичные биты из x, а при операции «И» — на нулевые биты из x , потому что только такие биты «затрагивают» какие-то биты в f_i при применении той или иной операции.

Теперь заметим, что если операция «затрагивает» какие-то биты, то после ее применения эти биты во всех числах станут одинаковыми и далее будут изменяться синхронно, и последующие изменения какого-то из этих битов не будут влиять на ответ, так как изменение всех единиц, стоящих на k-ом месте в каждом числе, на ноль будет означать лишь вычитание числа 2^k из каждого числа, что не влияет на количество неубывающих отрезков (а изменение всех нулей на единицу, соответственно, прибавление некоторой одной и той же степени двойки к каждому числу). Из всего вышеизложенного можно сделать вывод, что если бит с номером k в каждом числе из массива был «затронут» какой-то операцией, мы не будем обращать внимание на последующие «затрагивания» этого бита. Соответственно, будем поддерживать некую переменную *changed*, хранящую в своем двоичном представлении единицы на позициях тех битов, которые хоть раз были «затронуты» какой-то операцией. И если «затрагиваемые» текущей операцией биты являются подмаской числа *changed*, ответ на текущем запросе будет равен ответу на предыдущем. Если же «затрагиваемые» биты не являются подмаской *changed*, значит какие-то биты в числах из массива еще не достигли «синхронного» состояния, и ответ должен быть пересчитан. Ответ пересчитывается за длину массива. Ответ в худшем случае будет пересчитан столько раз, сколько битов содержится в двоичном представлении чисел в массиве (в этом случае каждая операция «затрагивает» только один новый бит). Поэтому асимптотика — $O(\log_2(MAX) \times n)$.

Разбор задачи «Гонки на колесницах»

Научимся для начала решать задачу доехать за минимальную стоимость из одного перекрестка

до другого. Для этого заведем $m \cdot 4$ вершин. Вершина — конец ребра на котором стоит колесница и направление, в котором она смотрит (по направлению на вторую вершину ребра, или в противоположном направлении). Проведем ребра между состояниями, между которыми можно перейти. А именно, из состояния, соответствующего колеснице, стоящей в начале ребра и смотрящей на другую вершину ребра, проведем ребро стоимости длинна ребра умноженная на коэффициент в состояние, в котором колесница стоит на второй вершине ребра и смотрит в ту же сторону. Так же, для каждой вершины отсортируем состояния, стоящие на этой вершине, по углу направления и проведем между соседними в таком порядке состояниями ребра стоимости угол между направлениями умноженный на коэффициент. Теперь запустим алгоритм Дейкстры из состояний, стоящих на начальной вершины, а за ответ возьмем минимальное из расстояний до состояний, стоящих на финальной вершине.

Чтобы научится решать исходную задачу, построим k слоев. Между слоями проведем ребра из состояний, стоящих на очередном контрольном пункте в соответствующие состояния следующего слоя.

Обратите внимание, что если решение явно строит граф размера $m \cdot 4 \cdot k$, оно может не уложится в ограничения по памяти. Количество вершин на слое можно соптимизировать до $m \cdot 2$, а так же можно хранить расстояние только внутри последних двух слоев. Потому что никакое ребро не ведет из более позднего слоя в более ранний.

Разбор задачи «Гармонический ряд»

Пусть $n = r - l + 1$. Для начала рассмотрим решение за $\mathcal{O}(n + \log p)$ времени и $\mathcal{O}(n)$ памяти. Посчитаем $R = (l \cdot (l+1) \cdot (l+2) \dots (r-1) \cdot r)^{-1}$ при помощи бинарного возведения в степень за $\mathcal{O}(\log p)$. Теперь предподсчитаем следующие величины: $p_i = l \cdot (l+1) \cdot \dots \cdot (i-1) \cdot i$ и $s_i = i \cdot (i+1) \cdot (i+2) \cdot \dots \cdot (r-1) \cdot r$. Заметим, что тогда $x^{-1} = R \cdot p_{x-1} \cdot s_{x+1}$ и мы можем посчитать нужную сумму за $\mathcal{O}(n)$.

Для уменьшения потребления памяти разобьем отрезок от l до r на блоки размера $\sqrt{n}$, на каждом посчитаем произведение всех чисел в нем и посчитаем префиксные и суффиксные произведения этих величин. Будем решать для каждого блока отдельно. Посчитаем префиксные и суффиксные произведения в каждом блоке и домножим их на префиксные и суффиксные произведения предыдущего и следующего блока. Таким образом наше решение будет работать за $\mathcal{O}(n + \log p)$ времени и $\mathcal{O}(\sqrt{n})$ памяти.