Разбор задачи «Выход участников»

Автор задачи: Алексей Михненко
Разработчик задачи: Артём Агафонов

Будем симулировать процесс. Переберём участников в порядке возрастания номеров. Если цвет очередного участника совпадает с цветом предыдущего вышедшего, то поместим его в очередь. Иначе отметим, что данный участник выходит следующим, а затем выпустим участника, ожидающего в очереди, если она не пуста. Наконец, когда мы переберём всех участников, выпустим всех участников, которые ожидают в очереди. Получаем линейную симуляцию данного процесса, а значит данное решение будет работать за $O(qn)$.

Заметим, что если при такой симуляции очередь пуста и цвет футболки участника отличается от цвета предыдущего вышедшего, то данный участник выйдет под своим номером. Заметим, что все участники, имеющие номер меньше данного, не влияют на расстановку его и всех последующих, а значит мы можем считать, что на таких участниках процесс начинается заново с поправкой на их номер.

Пусть мы начинаем процесс с участника номер i, как найти следующего участника, с которого можно считать, что процесс начался заново? Поставим $+1$ на позиции участников с цветом футболки a_i и -1 на все остальные. Тогда наименьший индекс j такой, что сумма от i до j включительно равна 0, будет искомым. Для каждой позиции i проведём ориентированное ребро в $p_i = j$. Такие ребра образуют ориентированный лес.

Поймём, как с помощью данного леса можно отвечать на запрос. Начиная с первого участника, будем постепенно переходить по рёбрам пока следующий участник имеет номер не больший, чем y. Пусть мы остановились на i, тогда по свойству выше можем считать, что процесс начинается с участника i. Заметим, что при расстановке участников на отрезке с i по $p_i - 1$ на позициях i, $i + 2$, $i + 4$, …, $p_i - 1$ будут стоять участники с цветом футболки a_i, а на позициях $i + 1$, $i + 3$, …, $p_i - 2$ все остальные. Тогда, чтобы узнать позицию участника y, нужно выяснить сколько участников с цветом a_i стоят до него. Для этого можно воспользоваться бинарным поиском по массиву позиций участников с футболками цвета a_i. Такой массив можно легко поддерживать при запросах изменения.

Далее, задача разбивается на две части: необходимо динамически поддерживать данный лес при смене цветов двух соседних участников и быстро находить ребро, накрывающее позицию y. Ограничения в некоторых подгруппах позволяли упростить реализацию данных частей решения. Далее, будет рассмотрено решение на полный балл, использующее link-cut. Другой вариант – использование корневой декомпозиции, которая при аккуратной реализации тоже позволяла набрать полный балл в данной задаче.

Вторая часть, состоящая в поиске накрывающего ребра, реализуется операцией expose от вершины 1, что позволяет выделить путь, в котором будут лежать все участники, на которых процесс будет начинаться заново. Найти интересующего нас участника можно с помощью спуска в полученное splay-дерево.

Если мы выясним, у каких участников и как поменяются значения p_i, то данные изменения можно поддержать в дереве с помощью операций link и cut. Заметим, что для цветов, не равных a_x и a_{x+1}, оба участника вносили вклады, равные -1 в соответствующие суммы. Значит их перестановка не повлияет на суммы. Если цвета a_x и a_{x+1} совпадают, то никакого изменения не происходит. Рассмотрим случай $a_x \neq a_{x+1}$. Во-первых, изменятся рёбра исходящие из x и $x + 1$. Заметим, что для цвета a_x произошло изменение вкладов позиций x и $x + 1$ с $+1$, -1 на -1, $+1$. Теперь, при наборе суммы из какой-то позиции i цвета a_x алгоритм может получить 0 на позиции x. Это означает, что ранее сумма до позиции x была равна 2. Зная это мы можем найти позицию, в которой префиксная сумма для цвета a_x оказалось на 2 ниже, чем в x. Например, это можно реализовать спуском в дерево отрезков для цвета a_x, которое в листе i хранит сумму на префиксе до i-го участника с футболкой цвета a_x (заметим, что хранить такие суммы достаточно, так как между такими участниками префиксные суммы линейно убывают). Данная позиция и будет той, в которую попало бы ребро, которое теперь мы должны перенаправить в x. В силу способа проведения рёбер такое ребро будет одно. Аналогично, можно разобрать случай для цвета a_{x+1} – из него мы тоже получим

не более одного кандидата, для которого изменится значение в массиве p. Чтобы найти позицию, в которую будет смотреть новое ребро, необходимо сделать ещё один запрос в соответствующее дерево отрезков, чтобы найти первое вхождение после позиции i префиксной суммы на 1, меньшей, чем префиксная сумма до позиции i.

Таким образом, получим решение, работающее за $O(q \log(n))$. Также, на полный балл можно было реализовать решения, работающие за $O(q \log^2(n))$, в которых splay-дерево в link-cut заменяется на декартово дерево, или за $O(q\sqrt{n})$, в котором весь массив разбивается на блоки размера $\sqrt{n}$ и для каждого элемента поддерживается сжатый переход, который указывает на первый элемент, лежащий вне данного блока, в который можно перейти, двигаясь по ребрам вверх. Тогда при запросе изменения нам потребуется обновить данный переход для элементов из не более чем 4 блоков, а поиск накрывающего ребра будет выполняться проходом по таким сжатым переходам до тех пор, пока не будет достигнут блок, в который попал запрос.

Разбор задачи «Перестановки и запросы»

Автор задачи: Алексей Михненко
Разработчик задачи: Алексей Васильев

Представим перестановку, как набор точек (i, p_i). Тогда операция первого типа переводит все точки (x, y) в $(n - x + 1, y)$. Операция второго типа переводит (x, y) в $(x, n - y + 1)$. Операция третьего типа переводит (x, y) в (y, x).

Заметим, что любым количеством этих операций мы можем получить не больше восьми различных перестановок, которые мы опишем следующим образом. Пусть f_1, f_2, f_3 — три переменные, которые принимают значение 0 или 1.

Пусть p — наша изначальная перестановка, дальше по порядку применяем к ней следующие модификации:

1. Если $f_1 = 1$, то делаем операцию третьего типа (то есть переводим (x, y) в (y, x)).
2. Если $f_2 = 1$, то делаем операцию первого типа (то есть переводим (x, y) в $(n - x + 1, y)$).
3. Если $f_3 = 1$, то делаем операцию второго типа (то есть переводим (x, y) в $(x, n - y + 1)$).

Изначально предподсчитаем стоимость восьми перестановок, которые соответствуют каждому возможному набору переменных f_1, f_2, f_3.

Теперь будем поддерживать текущие переменные f_1, f_2, f_3, которые изначально равны нулю.

- Запрос первого типа: заменяем f_2 на $1 - f_2$.
- Запрос второго типа: заменяем f_3 на $1 - f_3$.
- Запрос третьего типа: заменяем f_1 на $1 - f_1$ и меняем местами f_2 и f_3.

Таким образом мы после каждого запроса поддерживаем актуальные переменные f_1, f_2, f_3 и выводим предподсчитанную стоимость для таких флагов. Время работы $O(n \log n + q)$, где log от бинарного возведения в степень для подсчета стоимости.

Разбор задачи «Задачи от Саши»

Автор и разработчик задачи: Герман Перов

Переформулировка условия

Необходимо найти количество способов разбить вершины на $k+1$ множество, в котором наименьший общий предок i-го (для $1 \leqslant i \leqslant k$) множества равняется x_i (множество под номером $k+1$ содержит вершины, соответствующие жильцам, которых Саша не позовет решать задачи).

Решение для $k = 1$

Заметим, что Саша может звать жильцов только из поддерева вершина x_1.

- если Саша позовет жильца из вершины x_1, то любого жильца из поддерева можно как звать, так и не звать. Таких вариантов $2^{sz(x_1)-1}$
- если Саша не позовет жильца из вершины x_1, то необходимо позвать хотя бы по одному жильцу хотя бы из двух поддеревьев. Чуть удобнее вычислить данное значение как $cnt_{all} - cnt_1 - cnt_0$, где cnt_{all} — общее количество способов позвать жильцов из поддеревьев (то есть $2^{sz(x_1)-1}$), cnt_1 — количество способов позвать хотя бы одного жильца только из одного поддерева (то есть $\sum\limits_{(x_1,u)\in E}(2^{sz(u)}-1)$) и cnt_0 — количество способов не звать жильцов вообще (то есть 1).

Решение для $p_i = i - 1$

- каждая x_i обязана войти в i-е множество;
- вершина v, в которой не будут решать задачу, может войти в множество j ($j \leqslant k$) тогда и только тогда, когда x_j является предком v. Чтобы найти количество способов выбрать множество для вершины, достаточно посчитать, сколько вершин над ней находятся в массиве x. Также есть опция взять данную вершину в множество $k + 1$.

Достаточно посчитать произведение числа способов выбрать множество для v по всем вершинам.

Решение за $O(nk)$

Переформулируем задачу так, что будем не определять вершины в множества, а красить их. Необходимо, чтобы наименьший общий предок вершин цвета i равнялся x_i (также будет фиктивный цвет $k + 1$, для которого такое условие не требуется).

Будем считать $dp[v][c]$ — количество способов корректно раскрасить поддерево v, при этом имея c дополнительных цветов (которые в будущем потребуются для покрытия вершин из x не из поддерева v)

- если вершина v не лежит в x, то просто $dp[v][c] = c \cdot \prod\limits_{(v,u)\in E} dp[u][c]$. Это следует из того, что для получения c дополнительных цветов достаточно взять соответствующие раскраски для поддеревьев и выбрать сам цвет для v
- если вершина v лежит в x, то необходимо покрыть данную вершину. Будем обрабатывать детей по одному и поддерживать три значения: $cnt[c][0]$, $cnt[c][1]$ и $cnt[c][2]$, где $cnt[c][i]$ — количество способов получить в результате c свободных цветов, при этом покрасив в i поддеревьях детей хотя бы одну вершину в цвет вершины v (состояния при $i \geqslant 2$ для нас неотличимы, поэтому считаем, что $cnt[c][2]$ хранит суммарное число способов для всех $i \geqslant 2$). Тогда добавление очередного ребенка u выражается в
  - если в u будет использован свободный цвет (покрашен в цвет v) и должно остаться c цветов, то $newcnt[c][min(i + 1, 2)]$ надо увеличить на $cnt[c][i] \cdot (dp[u][c + 1] - dp[u][c])$
  - если в u не будет использован свободный цвет, то $newcnt[c][i]$ надо увеличить на $cnt[c][i] \cdot dp[u][c]$

  Тогда $dp[v][c] = (cnt[c][0] + cnt[c][1] + cnt[c][2]) + cnt[c][2]$. Первые три слагаемых соответствуют варианту "v покрасили в свой цвет, поддеревья можно красить произвольно". Второе слагаемое соответствует варианту "v покрасили в свободный цвет, надо раскрасить вершины хотя бы в двух поддеревьях".

Ответ на задачу будет лежать в $dp[root][1]$ (единица соответствует тому, что у нас остался свободный цвет для $k + 1$).

Получили динамику, вычисляемую за $O(nk)$.

Решение за $O(n + k^2 + k^{1.5}\sqrt{n})$

Назовем вершины из x плохими, а вершины не из x хорошими.

Предположим, что у хорошей вершины v есть хороший ребенок w.

Получим формулу

$$dp[v][c] = c \cdot \prod_{(v,u)\in E} dp[u][c] = c \cdot \left(\prod_{w\neq u\ (v,u)\in E} dp[u][c] \right) \cdot dp[w][c]$$

$$dp[v][c] = c \cdot \left(\prod_{w\neq u\ (v,u)\in E} dp[u][c] \right) \cdot c \left(\prod_{(w,t)\in E} dp[t][c] \right)$$

Заметим, что можно отдельно сгруппировать c в некоторой степени и произведение состояний динамик при фиксированном c. Это соответствует тому, что если есть ребро (v, w) между двумя хорошими вершинами, то можно удалить нижнюю из вершин и поддеревья w подвесить к v (при этом надо не забыть учесть, что можно красить и саму вершину w; для этого предлагается в вершинах поддерживать вес — скольким вершинам соответствует данная).

После такой обработки дерева не будет двух хороших вершин, связанных ребром. Значит, в итоговом дереве будет $2k$ вершин плюс сколько-то (возможно, n) хороших листьев. При этом заметим, что суммарный вес хороших листьев будет не может быть больше n.

Таким образом, получили решение за $O(n + k^2 + leaves \cdot k)$, где $leaves$ — количество хороших листьев после вышеописанной обработки дерева. Осталось научиться эффективно обрабатывать вершину, к которой подвешено множество хороших листьев какого-то веса.

Посчитаем массив $cnt[c][i]$ на хороших листьях. Для удобства будем считать, что к вершине подвешено l листьев весов $w_1, w_2, \ldots, w_l$. Пока что мы умеем вычислять $cnt[c][i]$ для всех c за $O(k \cdot l)$ и хотим научиться быстрее.

- $cnt[c][0] = c^{\sum w_j}$. Соответствует тому, что просто в каждом поддереве красим вершины в c свободных цветов.
- $cnt[c][2] = (c+1)^{\sum w_j} - cnt[c][0] - cnt[c][1]$. Следует из того, что $cnt[c][0]$, $cnt[c][1]$ и $cnt[c][2]$ покрывают всевозможные варианты покрасить поддерева детей, оставив c свободных цветов. Таких вариантов, в свою очередь, $(c+1)^{\sum w_j}$
- осталось разобраться с подсчетом $cnt[c][1]$. Заметим, что при добавлении очередного w, получается $newcnt[c][1] = cnt[c][0] \cdot ((c+1)^w - c^w) + cnt[c][1] \cdot c^w = newcnt[c][0] \cdot \left(\left(\frac{c+1}{c}\right)^w - 1\right) + cnt[c][1] \cdot c^w$. Отсюда следует равенство $\frac{newcnt[c][1]}{newcnt[c][0]} = \frac{cnt[c][1]}{cnt[c][0]} + \left(\frac{c+1}{c}\right)^w - 1$

  Значит, можно поддерживать значение $\frac{cnt[c][1]}{cnt[c][0]}$ и обновлять его при очередном весе. Благодаря такому трюку можно добавлять не один вес, а сразу несколько одинаковых весов.

Имея массив $w_1, \ldots, w_l$ посчитаем, сколько раз какой вес встречался и будем скармливать в алгоритм не только веса, но и их количества. Получим подсчет $cnt[c][i]$ для всех c за $O(k \cdot distinct(w_1, \ldots, w_l)) \leqslant O\left(k \cdot \sqrt{\sum w_j}\right)$ для фиксированной вершины.

Мы получили оценку $O\left(k \cdot \sqrt{\sum w_j}\right)$ для обработки хороших листьев одной плохой вершины. Теперь оценим общее время обработки всех листьев.

Пусть W_i — суммарный вес хороших листьев, подвешенных к вершине x_i. Так как всего вершин не больше n, имеем ограничение $\sum_{i=1}^{k} W_i \leqslant n$, а время работы равняется $\sum_{i=1}^{k} k\sqrt{W_i}$. Худшим случаем будет $W_i = \frac{n}{k}$ для всех i, и общее время работы обработки листьев составит $O\left(\sum_{i=1}^{k} k\sqrt{\frac{n}{k}}\right) = O\left(k^{1.5}\sqrt{n}\right)$

Таким образом, общее время работы всего алгоритма составляет $O\left(n + k^2 + k^{1.5}\sqrt{n}\right)$

Разбор задачи «Волнение перед олимпиадой»

Автор и разработчик задачи: Алексей Михненко

Предположим, что Александр поговорил с первыми k участниками, и текущий ответ равен ans. Пусть P — сумма уровней волнения всех участников в текущий момент времени, с которыми Александр уже поговорил. В этой сумме мы учитываем участников, которые уже зашли в зал соревнования, с коэффициентом $a_i + t \cdot b_i$, как будто они продолжают стоять в очереди.

Будем следить за величиной $Q = \text{ans} - P$. Заметим, что если Александр поговорит с участником с уровнем волнения x, то ans и P увеличатся на x, поэтому эта разность не изменится. Таким образом, в рамках фиксированного значения t величина Q неизменна.

Теперь рассмотрим, что происходит с величиной Q при увеличении t на 1. Если B — сумма значений b_i всех участников, с которыми Александр уже поговорил, то P увеличится на B, то есть Q уменьшится на B. Теперь рассмотрим момент времени, когда Александр закончит говорить с участниками. Так как изначально $Q = 0$, в этот момент времени $Q = -\sum B_t$ по всем прошедшим моментам времени. Тогда $\text{ans} = Q + P$. То есть, если мы зафиксировали, в какой момент времени Александр закончил, а также количество участников, с которыми он успел поговорить, ответ зависит только от $\sum B_t$.

При этом фиксировать момент времени и количество участников нет смысла, так как Александр всегда может подождать, пока все участники, с которыми он уже поговорил, выйдут в зал соревнования. То есть достаточно фиксировать только количество участников, с которыми он в итоге поговорил. Если зафиксировать эту величину (назовём её k), Александру выгодно каждый раз находиться в минимально возможном значении B_t для фиксированного момента времени t. Если на план Александра нет ограничений, то это минимальное значение B_i, где i от t до k. Если же ограничения на план есть, то на i также накладывается дополнительное ограничение сверху, так как Александр не может поговорить с некоторыми участниками раньше своего плана.

Для фиксированного значения k задачу можно решить за $\mathcal{O}(n)$, если явно найти эти ограничения на i для каждого t и просуммировать минимумы на соответствующих отрезках. Это позволяет решать задачу за $\mathcal{O}(n^2)$.

Чтобы улучшить время работы данного решения, надо заметить, что Александр между фиксированными точками своего плана, а также после последней фиксированной точки, будет находиться только на суффиксных минимальных элементах без учёта последних $n - k$ участников. Таким образом, если перебирать количество людей, с которыми он в итоге поговорит, от последней фиксированной точки до n, можно явно поддерживать стек позиций, в которых он будет находиться. Параллельно с этим можно поддерживать величину Q, с помощью которой уже вычислять ans. Данный подход позволяет решать задачу за $\mathcal{O}(n)$.