Разбор задачи «Разбиение на слагаемые»

Автор задачи: Алексей Толстиков, разработчик: Егор Чунаев

Если число можно разбить на k различных слагаемых, то их сумма хотя бы $1+2+\ldots+k = \frac{k(k+1)}{2}$. Действительно, минимальное уникальное число не меньше одного, второе — двух и так далее.

Научимся строить пример, когда $n \geqslant \frac{k(k+1)}{2}$. Выпишем числа от 1 до k и добавим к максимальному числу то, что осталось распределить, то есть $n - \frac{k(k+1)}{2}$. Все числа останутся различными и их сумма будет равна n, что и требовалось.

Разбор задачи «Нетреугольники»

Автор задачи: Григорий Резников, разработчик: Азат Исмагилов

Самое простое решение работало за $O(n^3 \cdot q)$ и набирало 10 баллов. Достаточно было поддерживать рёбра (например в матрице смежности), и после каждого запроса проверять все n^3 троек.

Дальше можно было заметить, максимальная стоимость нетреугольника — на самом деле максимальная стоимость трёх вершин, где первые две не соединены ребром. Поэтому третью вершину всегда выгодно взять наибольшей возможной, при этом она не должна совпадать с первыми двумя. Поэтому такую вершину можно найти за 3 действия среди трёх вершин с наибольшей стоимостью, так что теперь мы можем перебирать только пары вершин. Такое решение работало за $O(n^2 \cdot q)$ и набирало 20 баллов.

Можно было сохранить все подходящие пары вершин (и максимальные нетреугольники для них) в какой-нибудь структуре данных. Такие решения набирали от 20 до 55 баллов в зависимости от реализации.

В подгруппе на 20 баллов можно было заметить следующий факт: ответ не уменьшается (так как рёбра только удаляются). При этом для каждой вершины v, нас интересует только максимальная по стоимости вершина u, с которой v не соединена ребром, поэтому для каждой вершины мы можем перебрать все остальные в порядке невозрастания стоимостей, и найти первую подходящую. После того, как мы нашли ответ для исходной задачи, надо было лишь увеличивать его после каждого добавления. Для этого достаточно было считать ответ для удалённого ребра. Такое решение работало за $O(n \log n + m + q)$ (с учётом сортировки вершин по стоимости).

Чтобы получить 100 баллов, надо было совместить 2 предыдущих решения. Рассмотрим все ребра, и поделим их на 2 типа: рёбра, которые участвуют в запросах, и остальные. Рёбер первого типа не больше q. Для рёбер второго типа ответ не изменится, поэтому можно сразу посчитать для них ответ за $O(n + m + q)$. Рёбра первого типа можно поддерживать в какой-нибудь структуре данных, например, set. Всего удалений, добавлений и запросов минимума будет $O(q)$, поэтому итоговая асимптотика будет $O(n \log n + m + q \log q)$.

Разбор задачи «Найти путь»

Автор задачи: Иван Сафонов, разработчик: Константин Амеличев

Для начала заметим, что каждой вершине можно поставить в пару n вершин. Значит, первая вершина из пары будет иметь номер $v = \lceil \frac{k}{n} \rceil$. Теперь для нее нужно найти парную вершину, удовлетворяя условие на лексикографический порядок. Поскольку первая вершина зафиксирована, то теперь можно искать $((k-1) \mod n)$-й лексикографически путь из v (в 0-индексации).

Предположим, что вершина v не зависела от запроса (как в подгруппе с $k \leqslant n$). Тогда мы могли с самого начала подвесить дерево за вершину v, тем самым превратив дерево в корневое. После этого мы можем отсортировать исходящие ребра из каждой вершины по номеру вершины, в которое это ребро ведет.

Как теперь выглядит отсортированный список путей? Первый путь идет из корня в корень. Дальше сколько-то путей можно получить, если добавить в путь минимальное из ребер, после чего как-то продолжить путь в поддереве. После них — второе из минимальных ребер, и так далее. Список путей можно получить обходом в глубину (нам достаточно знать только последнюю вершину в пути):

```
dfs(v):
    paths.push(v)
    for (to : edges[v]):
```

```
dfs(to)
```

Теперь можно насчитать для каждой вершины v массив $count_{to}$ — количество вершин, в которых можно закончить путь, если мы прошли по ребру $v \to to$. Теперь с помощью префиксных сумм мы для каждого ребра насчитываем, каким должно быть k, чтобы из вершины v первый переход делался по этому ребру.

Вернемся к общей задаче. Пусть вершина v находилась в произвольном месте дерева. Заранее подвесим дерево за какую-то вершину и выпишем обход дерева относительно нее. Заметим, что для любой вершины v обход ее поддерева будет подотрезком общего обхода. Значит, если искомая вершина u лежала в поддереве вершины v, то ее можно найти за $O(1)$ обращением к нужному индексу обхода.

Но бывает так, что путь из v сначала идет на сколько-то ребер наверх, и только потом начинает идти вниз. Сделаем двоичные подъемы, чтобы узнать, насколько вверх нам надо переместиться. Будем считать, что нас всегда интересует задача $find(v, k)$ — найти k-й путь из вершины v. На текущий момент мы можем посчитать $find(v, k)$, если ответ лежит в поддереве вершины v.

Пусть $dp[v][x]$ — подотрезок значений k, при которых мы поднимаемся наверх на 2^x ребер. Как посчитать такую динамику? Для $x = 0$ она у нас уже посчитана ранее, осталось только понять, как объединить значения.

Заметим, что сначала k должно лежать в промежутке $dp[v][x-1]$, чтобы мы поднялись до вершины $parent[v][x-1]$. После этого из k вычитается то количество меньших последовательностей, которые мы отсекли, и нам надо, чтобы новое k попало в интервал $dp[parent[v][x-1]][x-1]$. Решив данные ограничения на k, мы получим новый промежуток для $dp[v][x]$.

Для ответа на запрос переберем степень двойки, на которую мы хотим подняться. После этого, если $k \in dp[v][x]$, то мы можем продолжить с $find(parent[v][x], k')$. В какой-то момент мы закончим подниматься наверх, нам останется сделать спуск вниз с помощью выписанного обхода.

Что надо не забыть:

- При склеивании $dp[v][x-1], dp[parent[v][x-1]][x-1]$ может так оказаться, что $dp[parent[v][x-1]][x-1]$ уже просуммировал размер поддерева, содержащего v. Чтобы это проверить, надо сравнить номера ребер вверх и вниз из вершины $parent[v][x-1]$. Если сын имел меньший номер, то к k надо прибавить размер поддерева.

- Аналогичная ситуация возникает при подъеме наверх при ответе на запрос.

- Когда мы нашли самую верхнюю вершину на пути и хотим взять элемент в ее поддереве, наша операция имеет вид $find(v, k)$. k может содержать в себе размер наддерева, если индекс предка был маленьким. Так что надо с помощью префиксных сумм проверить этот случай и, если k было велико, вычесть из него размер наддерева.

Разбор задачи «Точки на плоскости»

Автор задачи: Максим Ахмедов, разработчик: Игорь Маркелов

Первая, третья и пятая подзадачи решаются наивной реализацией ответов на запросы, нужно проверять, лежит ли точка в полуплоскости/квадрате со сторонами параллельными осям координат/произвольном квадрате. $O(n \cdot q)$

Для решения второй подзадачи построим выпуклую оболочку данного множества точек. Тогда нам нужно проверять, лежит ли хотя бы одна из вершин оболочки в полуплоскости. С помощью бинарного поиска найдём самую далёкую точку оболочки от прямой-направляющей полуплоскости, лежащую в нужной полуплоскости и проверим её. $O(n \log n + q \log n)$

Четвёртая подзадача — классическая задача на двумерную сканирующую прямую и дерево Фенвика. $O(n \log n + q \log n)$

В шестой и седьмой подзадачах нужно заметить, что любой квадрат на плоскости можно представить в виде 4 прямоугольных треугольников с катетами параллельными осям координат и 1 квадрата со сторонами, так же параллельными осям координат, для каждого квадрата выделим данные треугольники. Теперь мы свели исходную задачу к следующей: есть набор прямоугольных

областей на плоскости (заданных границами по x: $[lx, rx]$ и по y: $[ly, ry]$) и полуплоскость для каждой такой области. Применим метод разделяй и властвуй($d\&c$) по координате x, заметим, что на каждом уровне рекурсии у нас O(n) точек и O(q) запросов. Для множества запросов, отрезок $[lx, rx]$ которых целиком покрывает текущий отрезок x в рекурсии применим divide and conquer по координате y. Заметим, что для $d\&c$ по y у нас так же, как и для $d\&c$ по x, линейное число точек и запросов на каждом уровне рекурсии. Теперь, когда мы рассматриваем подпрямоугольник по x и по y, будем решать задачу для всех запросов, целиком покрывающих этот прямоугольник. Для этого воспользуемся решением второй подзадачи. $O((n+q)\log^3(n+q))$, если строить выпуклую оболочку за $O(n \log n)$ и отвечать на каждый запрос с помощью бинарного поиска, или $O((n+q)\log^2(n+q))$, если строить оболочку за $O(n)$, предварительно отсортировав точки и запросы и заменив бинарный поиск на метод двух указателей.

Решение для восьмой группы снова основывается на том, что мы умеем отвечать на на прямоугольной области, когда нас интересует только одна полуплоскость. Воспользуемся квадродеревом — будем делить области пополам (на квадраты), пока не можем ответить на запрос. В силу того, что запрос является квадратом, на каждом уровне деления будет O(1) областей с неотвеченными запросами. Если обрабатывать все запросы одновременно и воспользоваться методом двух указателей как в предыдущей группе, то можно получить асимптотику $O(n \log max\_coor)$.