Задача 1. Дан пустой граф G_0 на n вершинах. Есть запросы двух типов:

1. Построить граф G_i, проведя ребро (u, v) в графе G_j;
2. Проверить, в одной ли компоненте связности графа G_i вершины u и v.

Требуется ответить на q запросов:

a. Оффлайн, $\mathcal{O}(n + q \log n)$;

b. Онлайн, $\mathcal{O}(q \log^2 n)$.

Разбор. В обоих пунктах нам понадобится СНМ, но при этом без эвристики сжатия путей: мы оставляем только ранговую эвристику, чтобы гарантированное время работы одной операции было $\mathcal{O}(\log n)$.

a. Будем обрабатывать запросы в оффлайн. Когда приходит запрос на создание графа G_j из графа G_i, проведем в дереве версий ребро из i в j, на котором будет написано, что в этот момент произошло добавления ребра (u, v) графа. Также для каждой вершины дерева версий (то есть для каждого графа) запомним, какие запросы нужно обработать для нее. Теперь обойдем дерево версий dfs-ом, используя СНМ с откатами: когда проходим по ребру вниз, добавляем ребро в СНМ, а когда возвращаемся, просто откатываем модификацию (это просто присвоить одной переменной исходное значение).

b. В этом пункте можно просто использовать в качестве массивов для СНМ персистентные массивы на деревьях отрезков.

Задача 2. Дан массив длины n. Ответьте в онлайне на q запросов вида: «Найти k-ую порядковую статистику на отрезке от l до r».

a. $\mathcal{O}(n \log n + q \log^2 n)$;

b. $\mathcal{O}(n \log n + q \log n)$.

Разбор. Рассмотрим сразу решение пункта **b**. Сожмем все числа. Насчитаем префиксные персистентные ДО подсчета (в точке x ДО хранится количество вхождений значения x на соответствующем префиксе). Теперь очень просто находить k-ую порядковую статистику на префиксе: достаточно просто в соответствующей версии ДО найти первый момент, когда сумма стала $\geq k$.

Чтобы ответить на запрос для отрезка $[l; r]$ необходимо взять версию r ДО и версию $l - 1$, а затем сделать тот же спуск, но одновременно по двум этим ДО: они имеют одинаковую структуру, поэтому достаточно просто смотреть на разность значений в соответствующих друг другу вершинах.

Также заметим, что это решение позволяет решать задачу не только на отрезке массива, но и на пути в дереве.

Задача 3. В предыдущей задаче добавить запрос изменения. $\mathcal{O}((n + q) \log^2 n)$.

Разбор. Воспользуемся все той же идеей с ДО подсчета, но теперь построим еще и внешнее ДО, в вершинах которого будут такие ДО подсчета, соответствующие отрезку, за который отвечает вершина внешнего ДО. Изменение делается очевидно. Для ответа на запрос достаточно просто спускаться одновременно по $\mathcal{O}(\log n)$ ДО.

Задача 4. Есть изначально пустое множество битовых строк длины n. Необходимо в онлайн за время $\mathcal{O}(n^2 \cdot 2^n + qn)$ ответить на q запросов двух видов:

1. Добавить в множество заданную битовую строку.
2. Дана битовая строка. За 1 операцию можно изменить 1 символ. За какое минимальное количество операций можно добиться того, чтобы строка оказалось в нашем множестве?

Разбор. Будем поддерживать величину ans_v — ответ для строки v. Когда приходит запрос добавления новой строки u, делаем $ans_u = 0$ и запускаем bfs из нее. При этом в bfs используется все тот же массив ans, поэтому мы сделаем ровно столько итераций, сколько нужно для обновления лишь изменившихся его значений.

Расстояния в таком графе не превосходят n, а потому от каждой вершины мы совершим $\mathcal{O}(n)$ релаксаций. Каждая релаксация из вершины занимает $\mathcal{O}(n)$ времени, а вершин всего $\mathcal{O}(2^n)$.

Задача 5. Есть n множеств прямых. Выполнить в онлайн q запросов трех типов:

1. Добавить новую прямую в множество.
2. Объединить два множества.
3. Найти лучшее значение в точке x среди прямых заданного множества.

Время $\mathcal{O}(n + q \log q)$. В каждый момент времени вы должны использовать не более $\mathcal{O}(n + q)$ памяти.

Разбор. Воспользуемся СНТ и идеей о том, что мы можем превратить структуру без добавления в структуру с добавлением, поддерживая вместо одной структуры несколько структур размером, равным степеням двойки.

Осталось научиться объединять множества, однако объединение таких множеств соответствует просто сложению двоичных чисел.

Задача 6. Изначально есть число $a_0 = 0$. Ответить в онлайне за время $\mathcal{O}(q \log k)$ на q запросов двух видов:

1. Пусть сейчас есть уже c чисел. Сделать новое число $a_c = a_i + 2^j$ для заданных i, j ($0 \le i < c$, $0 \le j < k$).
2. По заданным i, j определить, какое из чисел a_i, a_j меньше.

Разбор. Будем хранить двоичные числа в персистентных ДО. Операция прибавления степени двойки сводится к поиску ближайшего слева нуля и присвоению на отрезке 0 или 1. Операция сравнения сводится к операции поиска первого отличающегося элемента и запроса в точке. Это можно сделать с помощью хешей.

Обратим внимание на то, что хеши можно сделать и точными: пусть у детей a, b вершины v хеши равны $h(a)$, $h(b)$, соответственно. Рассмотрим пару $(h(a), h(b))$, если мы уже ее когда-то рассматривали, то хеш для $h(v)$ уже известен. А иначе скажем, что этой паре (и, соответственно, $h(v)$) соответствует какой-то новый уникальный хеш (просто прибавить к счетчику хешей 1).

Задача 7. Изначально есть n чисел $a_0 = a_1 = \ldots = a_{n-1} = 0$. Ответить в онлайне за время $\mathcal{O}(q \log q \log k)$ на q запросов трех видов:

1. Сделать $a_i \mathrel{+}= 2^j$ для заданных i, j ($0 \le i < c$, $0 \le j < k$).
2. Сделать $a_i, a_j = a_i + a_j, 0$ (присвоение как в Python) для заданных i, j.
3. По заданным i, j определить, какое из чисел a_i, a_j меньше.

Разбор. К решению предыдущей задачи достаточно просто добавить переливания меньшего к большему: из числа к меньшим количеством единичных битов по одному биту прибавляем к большему числу.

Задача 8. Изначально есть пустое множество строк. Ответить в онлайне за время $\mathcal{O}(q \log q \log^2 k + \sum l_i)$ на q запросов следующих видов:

1. Добавить в множество строку длины l_i.
2. Добавить в множество строку, полученную конкатенацией i-й и j-й лексикографически строк множества.
3. Добавить в множество подстроку с x-й по y-ю позиции ($x \le y \le k$) строки, i-й в лексикографическом порядке.
4. Изменить i-й символ j-й в лексикографическом порядке строки на c.
5. Перевернуть подстроку с x-й по y-ю позиции ($x \le y \le k$) строки, i-й в лексикографическом порядке.
6. Вывести i-й символ ($1 \le i \le k$) строки, j-й в лексикографическом порядке.

Разбор. Заметим, что строку можно хранить в персистентном ДД с хешами. Тогда все такие операции изменения легко реализовать. Важно заметить, что необходимо ограничивать количество элементов в ДД числом k (иначе оно может стать очень большим), для этого достаточно просто выкидывать все дальше k-й позиции.

Осталось поддерживать все эти ДД внутри внешней структуры, позволяющей вставлять новые элементы и находить k-й в порядке сортировки. Например, подходит ДД.

Задача 9. Предложить реализацию вектора, которая будет работать гарантированно за $\mathcal{O}(1)$ на основные операции (получение и изменение значения i-го элемента, добавление в конец).

Разбор. Воспользуемся той же идеей, что и для обычного вектора, но в тот момент, когда будем выделять основную память размера *capacity*, также выделим вспомогательную и память размера $2 \cdot capacity$, а во время каждого push back будем по два элемента из основной памяти перемещать во вспомогательную, а во время модификаций изменять элемент не только в основной памяти, но и во вспомогательной. Легко заметить, что к моменту, когда нам приходится увеличить *capacity*, все элементы из заполненной основной памяти уже будут скопированы во вспомогательную, поэтому можно в качестве новой основной памяти взять вспомогательную, старую основную освободить, а также создать новую вспомогательную.