Заключительный этап VIII открытой олимпиады по программированию

Разбор задач первого дня

Задача «Пляжный волейбол»

Автор идеи: Елена Андреева
Разработчик и автор разбора: Антон Полднев

Формальная постановка

Дана перестановка из чисел от 1 до N. В конце очередного шага минимальное из первых двух чисел перемещается в конец.

Нужно для каждого из Q запросов с номером шага определить, какие два числа будут в начале массива на этом шаге.

1 группа: N, $K_1 \leq 2000$, 1 запрос

K_1 раз проделаем над исходным массивом описанную на предыдущем слайде процедуру: с помощью цикла переместим минимальное из первых двух чисел в конец.

Сложность — $O(N \cdot K_1)$, **30 баллов.**

2 группа: $N, K_i \leq 10^5, Q \leq 10$

Как и в первой группе, реализуем моделирование: для каждого из Q запросов смоделируем соответствующее число шагов, выполняя каждый шаг не за линейное время, а за константу.

Как выполнять один шаг за O(1)? Для начала, отсортируем первые два числа по возрастанию — теперь нужно лишь научиться за константное время перемещать первое число массива в конец.

2 группа: $N, K_i \le 10^5, Q \le 10$

Способ 1. Встроенный дек (deque).

Способ 2. Изначально заведём массив размера N+K, исходный массив сохраним в первые N позиций. Чтобы переместить нулевой элемент в конец, сделаем просто a[N] = a[0] (индексация с 0) и запомним, что наш массив теперь начинается с 1, а не с 0. Далее a [start+N] = a[start], start += 1

Сложность — O(Q·(K+N)), **60 баллов.**

Оптимизируем: N, $K_i \leq 10^5$, $Q \leq 10^5$

В решении на 60 баллов мы моделировали последовательность игр для каждого запроса отдельно. Зачем?

Смоделируем последовательность игр один раз до максимального K_i и запомним силы игравших на каждом шаге команд.

Теперь, используя запомненные значения, на каждый запрос отвечаем за $O(1)$.

Сложность — $O(K+Q)$, **80 баллов.**

Полное решение: $N, Q \leq 10^5, K_i \leq 10^{18}$

1 3 2 4
3 2 4 1
3 4 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2

Заметим, что, как только самая сильная командаа окажется в начале очереди, остальные будут ходить по циклу и проигрывать. Пусть первый шаг, на котором сильнейшая команда стоит в начале очереди, имеет номер Z.

Ответ для всех $K < Z$ можно найти как в предыдущем решении.

Как находить ответ за запрос $K \geq Z$?

Полное решение: $N, Q \le 10^5, K_i \le 10^{18}$

1 3 2 4
3 2 4 1
3 4 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2
4 1 2 3
4 2 3 1
4 3 1 2

Как находить ответ за запрос $K \ge Z$?

Все команды, кроме сильнейшей, при $K \ge Z$ будут ходить по циклу. На K-м шаге с сильнейшей командой будет играть команда, которая на Z-м шаге стояла на позиции $(K-Z) \% (N-1) + 1$, если нумеровать позиции с нуля.

Сложность — O(N+Q).

$K_i \le 10^9$ — **90 баллов.**

Не забыли про long long (int64) — **100 баллов.**

Задача «Путешествие»

Автор идеи и разбора: Михаил Пядеркин
Разработчик: Павел Кунявский

Формальная постановка

Дан граф, вершины которого имеют некоторые веса. Необходимо найти простой путь максимального суммарного веса, состоящий из не более чем 4 вершин.

Добавляя фиктивные вершины веса 0, можно свести задачу к поиску пути ровно из 4 вершин

Будем обозначать число вершин графа буквой V, а число ребер графа буквой E

Простейшее решение

Будем перебирать все четверки вершин (a,b,c,d), и если пары (a, b), (b, c) и (c, d) соединены ребром, то пробуем обновить ответ:

```
for a=1..V:
    for b=1..V:
        for c=1..V:
            for d=1..V:
            if ((a, b), (b, c), (c, d) соединены ребром и вершины a, b, c, d различны) then обновить текущий ответ
```

$O(V^4)$ и 20 баллов

Почувствуйте разницу!

```
for a=1..V:
    for b - смежная (соединенная ребром) с a:
        for c - смежная с b:
            for d - смежная с c:
                if (вершины a, b, c, d различны) then
                    обновить текущий ответ
```

Такое решение работает за $O(E^2)$, так как каждая пара ребер (a, b)-(c,d) будет рассмотрена не более 4 раз.

40 баллов.

Ключевая идея

Давайте переберем «центральное» ребро (b,c):

После этого можно перебрать вершину a. В качестве вершины d логично выбрать самую «красивую» вершину, смежную с c и не совпадающую ни с одной из множества {a, b, c}.

Таким образом, среди соседей вершины c нас интересуют лишь 3 самые красивые вершины.

O(VE) и 70 баллов

А что, собственно, осталось?

На предыдущем слайде мы выяснили, что при фиксированном ребре (b,c) нас интересуют лишь 3 самых красивых соседа вершины c.

Однако, в силу симметрии, то же верно и для вершины b.

Таким образом, необходимо заранее для каждой вершины найти 3 самых красивых соседа, затем перебрать ребро (b, c), а затем перебрать a и d среди самых красивых у вершин b и c.

O(V+E), 100 баллов

Задача «Шустрая черепашка»

Автор идеи: Глеб Евстропов
Разработчик и автор разбора: Максим Ахмедов

Формальная постановка

В задаче требуется ответить на некоторое количество запросов следующего вида: «может ли черепашка добраться из клетки a_i в клетку c_i, если клетка b_i заблокирована».

В зависимости от группы тестов, количество запросов достигает 1000, 1 000 000 и 20 000 000.

Также меняется размер поля от 100x100 до 150x150.

Простая симуляция: *30 баллов*

Будем на каждый запрос блокировать соответствующую клетку и запускать любой алгоритм поиска пути на табличке (обход в ширину/глубину).

Запросов — 1000, каждый запрос обрабатываем за $100 \cdot 100$.

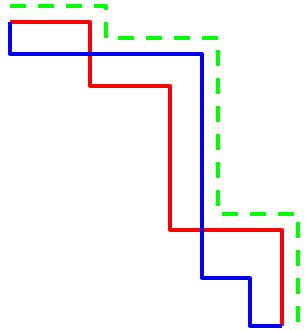
Подобное решение с запасом укладывается в ограничение по времени.

Верхний и нижний путь

Будем отвечать на все запросы, заканчивающиеся в фиксированной клетке c.

Посчитаем из каждой клетки «самый верхний» и «самый нижний путь», идущий в c. Это понятие корректно: если есть два пути, среди которых непонятно, кто выше, то можно взять огибающий путь выше их обоих.

Утверждение: если есть какой-то путь, не проходящий через b, то один из этих двух путей подходит.



60 баллов: считаем пути

Пойдём из клетки c во все клетки вверх и влево. Про каждую посещенную клетки мы знаем, что из неё можно добраться до c.

Теперь, используя эти пометки, мы для каждого запроса (a, b) можем пойти только по верхнему и только по нижнему пути из a.

```
| while i != c.i and j != c.j:
| | if flag[i][j + 1]:
| | | j += 1
| | else:
| | | i += 1
```

← *программа, идущая по “верхнему” пути*

Если оба этих пути содержат клетку b то NO, иначе YES.

$O(N^4)$ предподсчёт, $O(N)$ на запрос. Лучше писать с $O(N^2)$ памяти.

100 баллов #1: считаем количества

Обозначим за P(x, y) количество путей из клетки x в клетку y.

Путей из a в c, проходящих через клетку b — $P(a, b) \cdot P(b, c)$.

Всего путей — P(a, c).

Нас спрашивают — равны ли эти два числа? Числа — большие, но мы будем считать их по некоторому большому простому модулю. Вероятность “случайного” совпадения **крайне мала**.

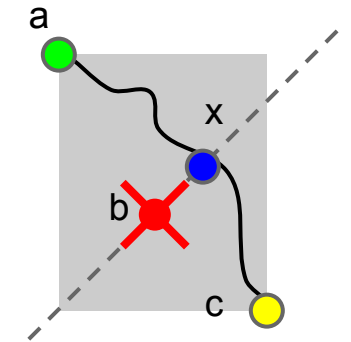
Посчитаем эти количества динамическим программированием за $O(N^4)$.

Считаем оптимально, используем только $O(N^2)$ памяти для динамики и $3 \cdot Q$ памяти, чтобы сохранить посчитанные значения.

Сложность — $O(N^4 + Q)$.

100 баллов #2: разрезы и битовое сжатие

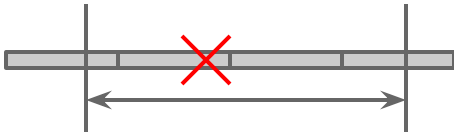
Что означает, что есть путь “в обход” клетки b? Рассмотрим диагональный разрез доски, проходящий через клетку b.



Нам надо, чтобы существовала клетка x на той же диагонали, которая достижима из a и из которой достижима c.

Воспользуемся битовым сжатием: будем хранить информацию о достижимости из вершины в массиве из пяти 32-битных чисел.

Обозначим from[a][i] — маска достижимых вершин на i-й диагонали из клетки a, to[c][i] — маска вершин на i-й диагонали, из которых достижима c.



Тогда мы должны рассмотреть (from[a][i] **AND** to[c][i]), где **AND** — попарное битовое “И” двух массивов, и проверить, есть ли на нём хотя бы одна единица на некотором отрезке кроме одной клетки b. Это условие проверяется за O(N / 32) битовых операций, что даёт нам сложность — $O(N^4 / 32 + QN / 32)$. На практике это решение даже быстрее предыдущего!

Задача «„Чапаев“ на дереве»

Автор идеи, разработчик и автор разбора
Максим Ахмедов

Формальная постановка

В задаче фактически ведётся речь о следующей игре.

- У нас имеется корневое дерево.
- Каждая вершина покрашена либо в белый (шашка есть), либо в черный цвет (шашки нет)
- За один ход можно взять любую белую вершину и покрасить весь путь от неё до корня в чёрный цвет.
- Определить, кто выигрывает при некоторых определёных начальных расположениях.

20 баллов: $O(2^N \cdot N^2)$, простейший анализ

Так как $N \le 20$, можно определить для всех 2^N возможных состояний выигрышность/проигрышность.

Каждое состояние — маска чёрных вершин. Можно воспользоваться динамическим программированием, можно написать перебор с запоминанием.

```
win[msk] = OR(!win[msk | path[x]])
```

где path[x] — это маска вершин, лежащих на пути от x до корня дерева, а OR берётся по всем вершинам x, куда ещё можно сходить, т. е. для которых соответствующий бит в msk — нулевой.

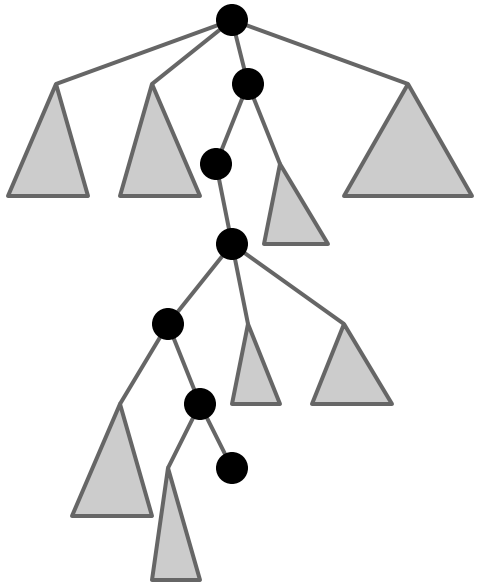
Сложность решения — $2^N \cdot N \cdot N$, на практике — гораздо меньше.

Важное соображение

Для дальнейшего продвижения необходимо заметить важный факт.

Рассмотрим дерево, в котором все вершины белые. После произвольного хода образуется несколько поддеревьев, подвешенных за образовавшиеся чёрные вершины.

Утверждение: дальнейшая игра будет происходить независимо в каждом из этих поддеревьев, причём можно считать, что каждое из этих поддеревьев «отвалилось» и стало самостоятельным.



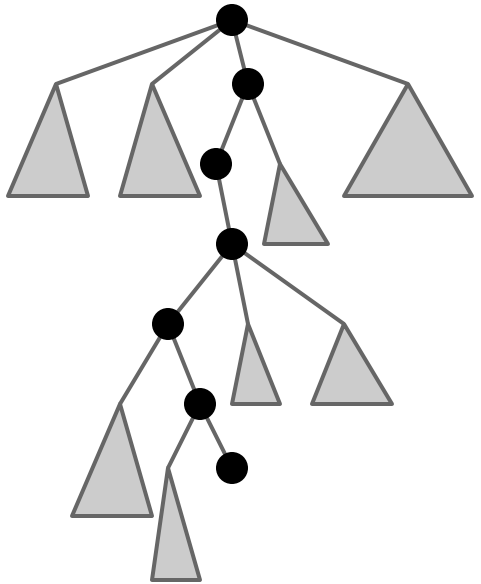
Важное соображение

Действительно, после этого любой ход затрагивает лишь вершины из того подерева, куда мы попали.

Такая особенность означает, что после каждого хода исходная игра распадается в **прямую сумму игр**, происходящих на некоторых поддеревьях исходного дерева.

В частности, исходная игра по условию представляет собой прямую сумму игр на поддеревьях некоторой вершины Root.

Это означает, что для анализа игры можно использовать теорию Шпрага — Гранди.

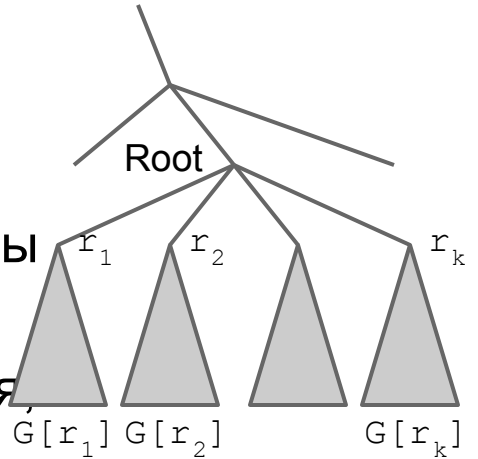


Использование функции Гранди

Обозначим за `G[v]` функцию Гранди игры на поддереве вершины `v`.

Тогда функция Гранди для начального положения, задаваемого вершиной Root — $G[r_1]$ **xor** $G[r_2]$ **xor** … **xor** $G[r_k]$, где r_1, r_2, …, r_k — дети вершины Root (так как игра ведется по ним независимо).

Если величина выше — не ноль, то игра выигрышная, иначе — проигрышная.



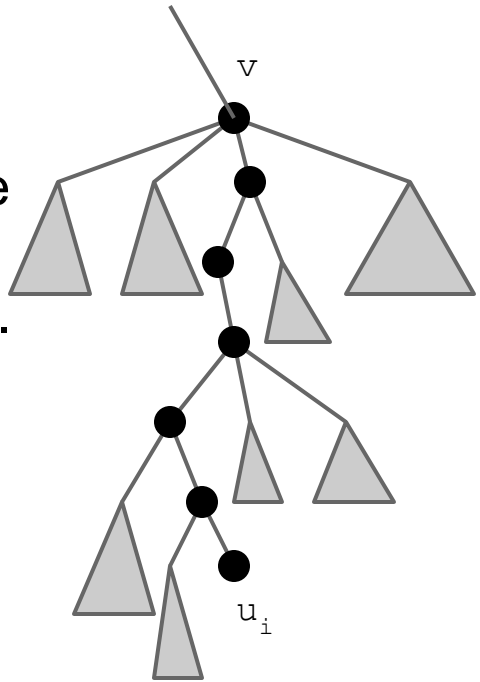
Вычисление функции Гранди

Как вычислить функцию Гранди от вершины v — $G[v]$?

$G[v] = \mathbf{MEX}(S(u_1,v), S(u_2,v), \ldots, S(u_m,v))$, где $u_1,u_2,\ldots,u_m$ — все вершины в поддереве v, а $S(u_i,v)$ — xor всех поддеревьев, “подцепленных” за путь $u_i \to v$.

$S(u_i,v)$ можно посчитать, воспользовавшись формулой:

$S(u_i,v) = \mathbf{xor}(G[x]$ if parent$[x]$ on $u_i \to v)$, где x пробегает поддерево v, при этом включаются только те вершины x, которые “подцеплены” за путь $u_i \to v$.



40 баллов: O(N^3), считаем в лоб

Будем считать фукнцию Гранди, как только что было описано.

```
перебираем v в порядке DFS: // функция DFS(v)
| перебираем u в поддереве v: // функция DFS2(v, u)
| | xor_val = 0;
| | перебираем x в поддереве u: // функция DFS3(v, u, x)
| | | if (parent[x] on u —> v):
| | | | xor_val = xor_val xor G[x]
| | val[u] = xor_val
| G[v] = MEX(val[])
```

Как нетрудно видеть, полученное решение работает за O(N^3).

Мы считаем одно и то же много раз!

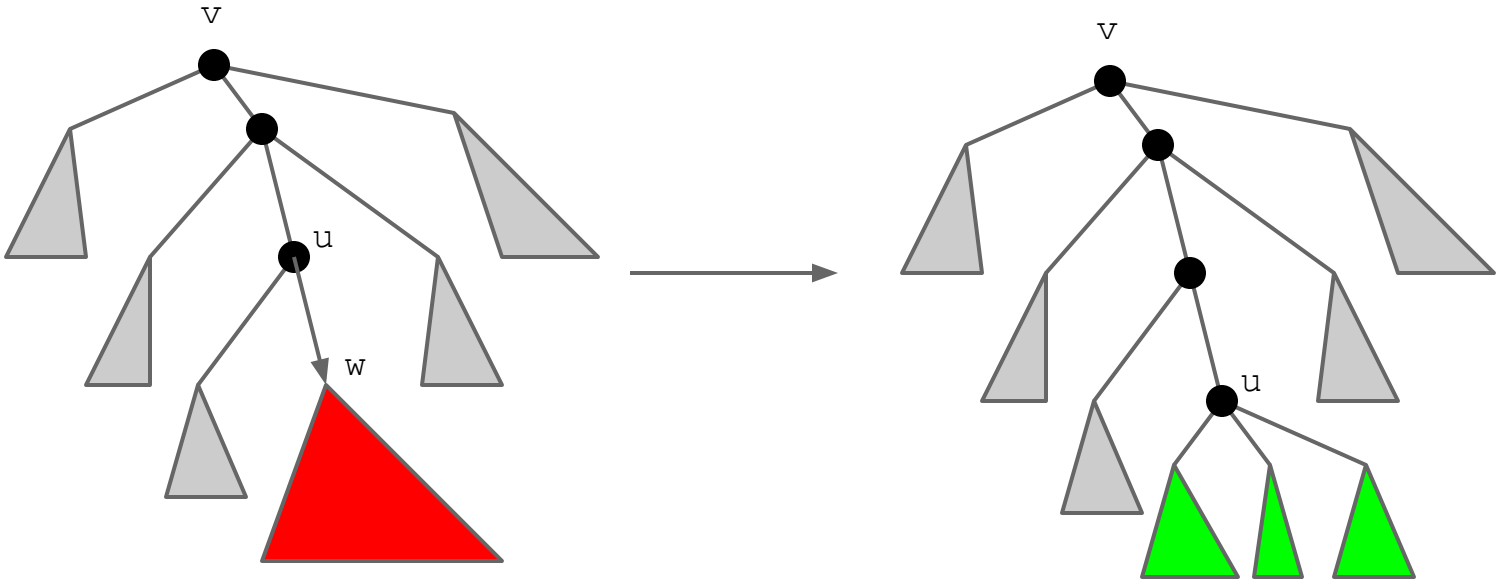
Соптимизируем предыдущее решение. Заметим, что подсчёт xor всех поддеревьев, подцепленных за путь (третий цикл) работает неоптимально.

Идея: когда мы сдвигаемся из `u` в дочернюю вершину `w` множество зацепленных за путь поддеревьев меняется несильно, а мы пересчитываем их все заново.

Давайте это исправим.

Исследуем изменение

При спуске из вершины u в вершину w значение меняется на `G[w]` **`xor`** `childrenG[w]`, где `childrenG[w]` — xor-сумма гранди-значений всех поддеревьев вершины `w`



60 баллов: O(N^2)

Теперь ясно, как следует изменить рекурсивную процедуру `DFS2(v, u)`, чтобы она работала за `O(N)` для одного значения `v`.

```
DFS2(v, u, xor_val):
| val[u] = xor_val
| for w - сын u:
| | DFS2(v, w, xor_val xor G[w] xor childrenG[w])
```

Теперь наше решение единожды запускает обход в глубину для каждой вершины `v`, а значит работает за O(N^2).

We need to go deeper...

Что нам надо?

Давайте поставим цель. Мы хотим обойти дерево в глубину, при этом в каждой вершине v:

1) получить в каком-нибудь виде множество всех ходов из вершины `v` — обозначим его `M[v]`.

2) взять MEX от этого множества.

Под ходом мы понимаем гранди-значение, в которое ведёт ход.

Рассуждаем: как можно получить в вершине множество ходов?

Во-первых, есть ход в корень `v`, дающий гранди-значение `childrenG[v]`.

Анализируем множество ходов

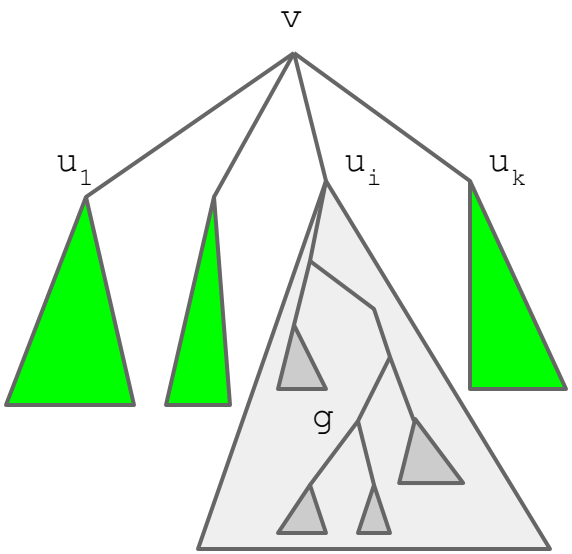
Во-вторых, есть множество ходов в дочерние поддеревья. Рассмотрим поддерево дочерней вершины u_i.

Легко пересчитать из уже посчитанного множества $M[u_i]$, какие нам добавятся ходы в поддерево u_i.

Утверждение: ход в гранди-значение g превратится в ход

g **xor** $(childrenG[v]$ **xor** $G[u_i])$

(иными словами, сверху прицепятся все поддеревья кроме самого u_i).



Какая нам нужна структура данных?

Подытожим: $M[v] = \text{sum}(c_i \text{ xor } M[u_i]) + \{childrenG[v]\}$

$G[v] = \text{MEX}(M[v])$.

Значит нам нужна структура данных, которая:

1) содержит множество неотрицательных целых чисел

2) позволяет быстро проксорить всё множество с какой-то константой

3) позволяет быстро объединять множества

4) позволяет быстро считать MEX по множеству.

Такая структура данных существует! И это…

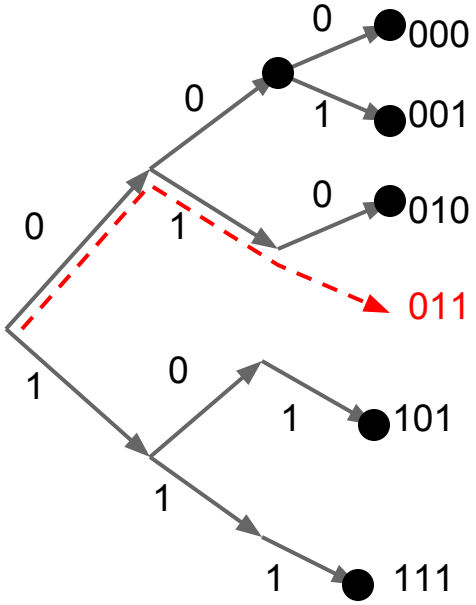
Битовый бор!

Рассмотрим бор, построенный на двоичной записи содержащихся в множестве чисел.

Гранди-функции в нашем дереве не превзойдут N, значит глубина бора не превосходит logN.

Как быстро посчитать MEX на битовом боре?

Поставим флаг во все вершины бора, поддерево которых полное. Тогда с помощью этих пометок легко найти MEX за один спуск — идём налево, если левое поддерево не полное, иначе направо.



Отложенные операции рулят!

Что значит проксорить все числа в боре с какой-то константой *c*? Это означает, что надо в некоторых слоях (соответствующих 1 в двоичной записи *c*) поменять у всех вершин левого и правого сына местами.

Всего таких вершин может быть много (например, в последнем слое их линейное количество).

Нам на помощь приходят… отложенные операции!

В каждой вершине храним число-маску, в соответствии с которой надо изменить поддерево. Приходя в вершину меняем местами, если надо, сыновей, раздаём маску детям и двигаемся дальше.

Приливаем меньшее к большему

С объединением дела обстоят труднее. Тут нам поможет идея приливания меньшего множества к большему.

Утверждение: если сливать множества, соответствующие поддеревьям, каждый раз добавляя по очереди все элементы меньшего множества к большему, количество операций вставки в бор будет O(N log N).

Доказательство: проследим за каким-нибудь числом. По ходу своей жизни оно периодически ксорится с чем-то и меняет своё «место жительства», когда мы приливаем бор, содержащий её, в какой-то больший бор.

Заметим, что после каждого переливания размер множества, содержащего число, увеличивается как минимум в два раза. Значит каждое число переместится не более O(log N) раз.

Значит, в итоге произойдёт не более O(N log N) переливаний.

100 баллов: O(N $\log^2$ N)!

Таким образом, суммарно мы произведём O(N log N) операций вставки в бор, каждая из которых работает за время O(log N).

Не забываем при добавлении числа в бор пересчитывать флаги.

Операция взятия MEX работает за O(log N) для каждой вершины.

Это решение, наконец, набирает 100 баллов!

Дальнейшие улучшения

Если хранить бор именно такой глубины, как надо (т. е. в поддереве размера s он будет иметь глубину log s), то тогда размер бора будет пропорционален размеру поддерева, а значит итоговая сложность всего решения будет вообще O(N log N).

Полезная оптимизация: выделять вершины не с помощью оператора **new**, а на статически заведённой памяти.

Альтернативный вариант использования бора: можно хранить числа не от старших бит к младшим, а от младших к старшим. Тогда можно сразу честно инвертировать детей во всех нужных вершинах, так как это будет работать за O(min(c, size)), где c — число, с которым мы ксорим, а size — размер поддерева.