Задача A. Освещение площади

Автор и разработчик задачи: Николай Ведерников

Чтобы осветить всю площадь, нам нужно осветить каждую из сторон. Чтобы осветить площадь по длине нам нужно $\lceil \frac{n}{k} \rceil$, а по ширине $\lceil \frac{m}{k} \rceil$. Итого нам нужно поставить $\lceil \frac{n}{k} \rceil \cdot \lceil \frac{m}{k} \rceil$ фонарей.

Задача B. Морской бой

Автор задачи: Артем Васильев, разработчик: Полина Шайдурова

Сначала проверим, что нет касаний углами между занятыми кораблями клетками (то есть клетками с символами '#'). Если их нет, то все фигуры на поле — это прямоугольники размера $1 \times a$, которые расположены вертикально или горизонтально. Осталось проверить, что есть ровно четыре «однопалубника», три «двухпалубника», два «трехпалубника» и один «четырехпалубник». Это можно сделать, например, посчитав количество занятых прямоугольников 1×1, 1×2, 1×3 и 1×4 (их должно быть 20, 10, 4 и 1 соответственно).

Задача C. Витрина ковров

Автор задачи: Михаил Иванов, разработчик: Рита Саблина

Будем считать, что $h \leqslant w$. Обозначим максимальную площадь ковров, которые можно уложить на площадку $h \times w$ как $D_{h,w}$.

Решим задачу для случая $h = 1$. Тогда выгодно построить витрину так: на первом слое положить ковер $1 \times w$, на втором слое ковер 1×1 и $1 \times (w-1)$ и так далее — положить ковры в w слоев, отрезая на каждом слое ковер 1×1. Суммарная площадь получится $D_{1,w} = w + (1 + (w-1)) + (1 + (w-2)) + \ldots + 1 = \frac{w^2+3 \cdot w-2}{2}$.

Решим задачу для случая $h \neq 1$. Всегда выгодно начать укладывать витрину с ковра $h \times w$. Далее нетрудно доказать по индукции, что для максимизации суммарной площади ковров на первом слое выгодно отрезать от прямоугольника полоску стороной $1 \times w$, укладывать ее далее, как в случае 1, от оставшегося ковра $(h-1) \times w$ последовательно отрезать полоски со стороной 1, пока не останется ковер размерами 1×1.

При $h = w$ суммарная площадь получится:

$D_{w,w} = w \times w + D_{1,w} + D_{w,w-1} = w \times w + D_{1,w} + D_{1,w-1} + D_{w-1,w-1} = w \times w + D_{1,w} + 2 \cdot D_{1,w-1} + \ldots 2 \cdot D_{1,1}$.

Итоговая формула для квадрата: $D_{w,w} = w^3 + 2 \cdot w^2 - 2 \cdot w$

При $h \neq w$ будем на каждом слое отрезать полоски $1 \times h$, пока не останется ковер $h \times h$. Суммарная площадь получится $D_{w,h} = h \times w + D_{1,h} + h \times (w-1) + D_{1,h} + h \times (w-2) + \ldots + D_{1,h} + D_{h,h}$.

Итоговая формула: $D_{w,h} = \frac{w^2 \cdot h + h^2 \cdot w + 4 \cdot (w \cdot h) - 2 \cdot h - 2 \cdot w}{2}$

Задача D. Перерисованный граф

Автор задачи: Егор Горбачев, разработчик: Иван Волков

Заметим, что для каждой из трех вершин указанная в задаче операция либо увеличивает ее степень на 2 (если исходно она не была соединена ни с одной другой вершиной из тройки), либо уменьшает на 2 (если она была соединена с обеими вершинами из тройки), либо не меняет вовсе (в остальных случаях). В любом случае, четность степени каждой вершины не меняется. Так что если существует вершина, у которой в одном графе четная степень, а в другом — нечетная, то ответ точно «NO».

Покажем, что в противом случае искомая последовательность действий всегда существует. Построить такую последовательность можно разными способами. Например, достаточно для каждого ребра (u, v), присутствующего в одном графе, и отсутствующего в другом, и такого что $u, v \neq 1$, применить операцию к тройке $(1, u, v)$. Докажем, что эта последовательность будет ответом на задачу.

Предположим, что после выполнения всех таких операций найдется ребро, которое присутствует в нашем графе, но отсутствует в том, который хотим получить (или наоборот). Очевидно, такое

ребро должно иметь вид $(1, x)$ (все остальные ребра мы нашей последовательностью точно "исправили"). Но степень вершины x в нашем графе имеет ту же четность, что и в графе, который мы хотим получить. В частности, степени вершины x не могут отличаться на единицу, а значит ребро $(1, x)$ должно либо присутствовать в обоих графах, либо в обоих отсутствовать — пришли к противоречию. Значит, после выполнения всех указанных операций, мы действительно приведем граф к требуемому виду.

Задача E. Переполох в бухгалтерии

Автор задачи: Федор Царев, разработчик: Анастасия Баркина

В этой задаче нужно найти количество чисел, встречающихся во входных данных по крайней мере n раз, и вывести все такие числа. Например, можно сделать так: отсортируем данный массив и пройдем по всем его элементам, сравнивая попарно соседние. Дальше возможны два случая:

1. Когда соседние элементы совпадают, добавим 1 к счетчику чисел в текущей группе одинаковых.
2. Когда соседние элементы различаются, сравним количество чисел в текущей группе одинаковых с n. Если их больше либо равно, чем n, добавим это число в ответ, иначе — пропустим.

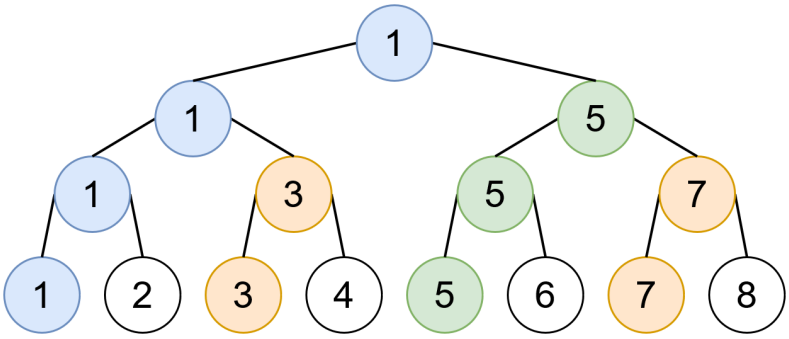
Небольшие технические сложности в том, чтобы правильно обработать первую и последнюю группы одинаковых чисел, и ничего не потерять.

Задача F. Дерево отрезков

Автор задачи: Рита Саблина, разработчик: Григорий Шовкопляс

В данной задаче оригинальный массив состоит из 2^k, следовательно высота построенного дерева отрезков — k.

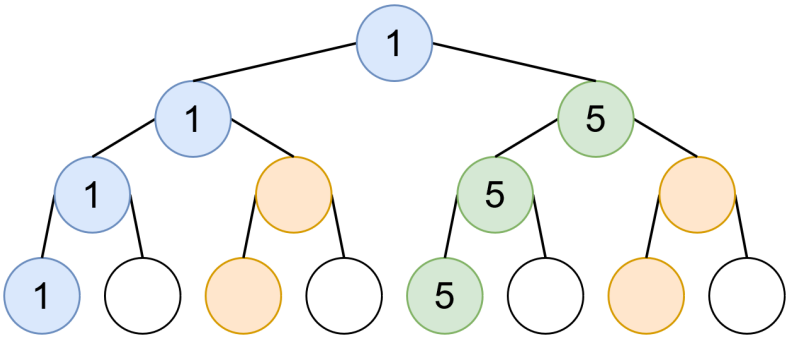
Заметим, что числа в исходном массиве различны, следовательно, если заполнять вершины снизу вверх, то каждому значению будет соответствовать путь от листа до некоторой вершины-предка.



При этом в дереве, содержащем $2^k - 1$ вершин, один такой путь длины k (соответствующий минимальному значению оригинального массива), один путь длины $k - 1$, два пути длины $k - 2$, четыре пути $k - 3$, ..., 2^x путей длины $k - x$, ..., 2^{k-1} путей длины 1. Доказательство этого факта мы оставляем читателю.

Так как все числа различны, посчитав количество вхождений каждого числа из входных данный, мы можем сопоставить каждому числу «уровень» дерева, на котором закончится путь снизу вверх. Далее это значение должно быть перебито меньшим значением.

Далее можно заполнить все дерево отрезков числами из входных данных жадно. Например, можно идти по дереву сверху вниз и заполнять все свободные вершины текущего уровня значениями, число вхождений которых соответствует данному уровню. Например, пусть для дерева на предыдущей картинке мы заполнили нулевой и первый уровень, у нас остались по две тройки и по две семерки, остальных значений по одному вхождению. Значит на втором уровне нужно доставить значения три и семь.



Для того, чтобы понять на какую позицию ставить три, а на какую семь (не всегда выгодно ставить слева направо в возрастающем порядке) можно на каждом уровне составлять массив пар из позиций и значений в родителях, отсортировать его по возрастанию значений в родителе и ставить меньшее значение в вершину с меньшим значением родителя. На каждом уровне стоит проверять, что у нас действительно есть 2^x значений с $k - x$ вхождениями, если на каком-то уровне это не так, восстановить дерево отрезков, а следовательно и оригинальный массив невозможно.

Наконец, нужно проверить, что итоговое дерево — корректное дерево отрезков. Ответ разрешается вывести любой, соответственно, заполнять дерево можно любым способом, например, как на картинках выше, «проталкивать» значение в левого ребенка, а правого оставлять свободным.

Итоговая сложность решения, использующего сортировку на каждом уровне, $O(n \log n)$.

Задача G. Подъем на лифте

Автор и разработчик задачи: Екатерина Ведерникова

Посмотрев на цифры, можно заметить, что при отражении, только цифра 2 переходит в цифру 5 и наоборот. Тогда для решения задачи, надо развернуть номер этажа, заменить цифру 2 на цифру 5 и наоборот. Важно заменять цифры одновременно. В конце не забыть убрать ведущие нули.

Задача H. Юрик и важные дела

Автор задачи: Даниил Орешников, разработчик: Михаил Первеев

Для начала заметим, что удобно обрабатывать запросы с конца, поддерживая индекс интересующего нас дела в текущем порядке их выполнения.

После всех изменений порядка выполнения дел нас интересует дело, которое будет выполнено k-м по счету. Научимся вычислять, на какой позиции в списке находилось данное дело до выполнения последнего изменения порядка выполнения дел.

Пусть последнее изменение затрагивало дела, находящиеся на отрезке $[l, r]$. Если $k \notin [l, r]$, то последняя операция не изменила позицию интересующего нас дела в списке, поэтому ее можно проигнорировать.

В противном случае обозначим изначальную позицию интересующего нас дела в списке как i. Нам известно, что после применения последнего изменения данное дело переместилось с позиции i на позицию k. Это значит, что приоритет, назначенный данному делу, был $(k - l + 1)$-м в порядке возрастания при нумерации с единицы. Также нам известно, что элементам отрезка были назначены приоритеты $p_1, p_2, \ldots, p_{r-l+1}$.

Таким образом, для того, чтобы выяснить, какой приоритет был назначен интересующему нас делу, необходимо найти $(k - l + 1)$-й в порядке возрастания элемент на префиксе перестановки длины $r - l + 1$. Пусть данный элемент равен p_j, тогда $i = l + j - 1$. Теперь присвоим в переменную k данное значение и перейдем к обработке предпоследней операции изменения порядка выполнения дел. После того, как все операции будут обработаны, в переменной k будет находиться ответ, так как до начала выполнения операций дела были упорядочены в порядке возрастания их номеров.

Осталось научиться быстро находить на заданном префиксе перестановки $(k - l + 1)$-й элемент в порядке возрастания.

Для начала выполним следующий препроцессинг. Будем рассматривать все префиксы перестановки в порядке возрастания их длины. Для каждого префикса мы хотим поддерживать массив

длины n, в котором на i-й позиции будет находиться единица, если элемент i содержится на текущем префиксе перестановки, и ноль в противном случае. Для того, чтобы поддерживать данный массив для каждого префикса, будем хранить его в виде персистентного дерева отрезков, которое будет поддерживать отдельную версию для каждого префикса перестановки. Такой препроцессинг можно выполнить за $\mathcal{O}(n \log n)$ времени, затратив на хранение дерева $\mathcal{O}(n \log n)$ памяти.

Теперь для того, чтобы найти на префиксе заданной длины k-й элемент в порядке возрастания, будем работать с соответствующей данной префиксу версией дерева отрезков.

Выполним бинарный поиск по ответу. Пусть бинарный поиск зафиксировал некоторое число x. Для того, чтобы проверить, правда ли, что x меньше, чем k-й элемент в порядке возрастания, необходимо найти количество элементов на данном префиксе, не превосходящих x, и сравнить данное количество с k. Для того, чтобы найти количество элементов, не превосходящих x, необходимо вычислить сумму на отрезке $[1, x]$ в нужной версии дерева отрезков. Таким образом, ответ на запрос работает за $\mathcal{O}(\log^2 n)$. Также можно заменить двоичный поиск по ответу на спуск по дереву отрезков и получить асимптотику $\mathcal{O}(\log n)$.

Итоговая асимптотика: $\mathcal{O}(n \log n + q \log n)$.

Задача I. Крыши

Автор и разработчик задачи: Степан Филиппов

Будем решать задачу рекурсивно, обозначим $f(l, r)$ — ответ на задачу, если рассматривать только колонны с l по r. Тогда в задаче требуется посчитать $f(1, n)$.

Найдем m — позицию самой высокой колонны на отрезке $[l, r]$. Понятно, что на ней необходимо закрепить крышу, при этом наиболее выгодно ее продлить либо максимально влево, накрыв отрезок $[l, m]$, либо максимально вправо, накрыв отрезок $[m, r]$. Для непокрытой части нужно решить задачу рекурсивно. В первом случае суммарная стоимость $c_m + f(m + 1, r)$, во втором — $c_m + f(l, m - 1)$. Посчитаем оба значения и выберем минимум.

Несложно понять, что количество рекурсивных вызовов будет не более n. Например, можно заметить, что каждая колонна может быть самой высокой (обозначали выше m) не более чем для одного отрезка $[l, r]$ среди рекурсивных вызовов.

При использовании структуры данных, позволяющей эффективно находить максимальный элемент на отрезке, например дерево отрезков, сложность решения получается $\mathcal{O}(n \log n)$.

Задача J. Побег слайма

Автор задачи: Рита Саблина, разработчик: Никита Голиков

Построим следующий граф: вершинам будут соответствовать все возможные положения слайма на столе, ребрам — трансформации. Таким образом, в задаче требуется найти кратчайший путь от состояния «квадрат 2×2» в верхнем-левом углу до аналогичного состояния в нижнем-правом углу. Для этого можно применить обход в ширину.

Будем представлять текущее состояние слайма как пару из его верхней левой клетки и сдвига остальных клеток относительно верхней левой. Тогда есть всего 13 возможных сдвигов для остальных клеток слайма.

Предподсчитаем все возможные «хорошие» состояния слайма: переберем все подмножества размера 3 из 13 возможных сдвигов, и для каждого проверим, что вместе с верхним левым углом оно образует 4-связную фигуру. Оказывается, что существует всего 19 «хороших» состояний. Теперь будем представлять каждое множество сдвигов как число от 0 до 18.

Предподсчитаем все возможные переходы. Переход мы будем представлять как тройку из сдвига верхнего левого угла слайма, изначального и конечного «хороших» состояний. Для каждого из 19 состояний переберем все клетки слайма и попробуем заменить их на все соседние по 8-связности. Если мы перешли в «хорошее» состояние, то скажем, что такой переход возможен. Оказывается, что существует всего 104 возможных перехода.

Предподсчитав все возможные переходы, мы готовы реализовать обход в ширину. Заранее проверим для каждого верхнего левого угла на столе и «хорошего» состояния, что все клетки для данного состояния не попадают в дырки. Осталось реализовать стандартный обход в ширину.

Таким образом, после предподсчета решение работает за время $\mathcal{O}(nmT)$, где T равно количеству возможных переходов, что, как описано выше, равно 104. Реализованное таким образом решение с большим запасом укладывается в ограничение по времени. Менее эффективные решения (например, без предподсчета всех возможных состояний и/или переходов) могли не укладываться по времени.

Задача K. Отходы производства

Автор задачи: Федор Царев, разработчик: Даниил Орешников

Заметим следующий факт: если в течение некоторого периода времени заводы простаивают, то, зная количество отходов на производстве в начале этого периода, можно быстро посчитать их количество в любой другой его день. А именно, если в конце дня a на производстве x отходов, и в течение дней с $a+1$-го по b-й не используется новое сырье, в конце b-го дня отходов будет ровно $x \cdot (1-r)^{b-a}$.

Пользуясь этим фактом, задачу можно было решить одним из следующих подходов: либо бинарным поиском, либо методом двух указателей, либо сортировкой событий. Рассмотрим третий способ. Объединим в одну последовательность события дни, в которые заводы работали, и дни, интересующие инспекцию. Для первых создадим событие вида $(d_i, \texttt{+}, w_i)$, а для вторых — $(q_i, \texttt{?}, i)$, после чего отсортируем все события по номеру дня, а при равенстве номера дня сначала расположим событие типа `+`, а затем — событие типа `?`.

Теперь будем обрабатывать события по очереди в том порядке, в котором их отсортировали. Для каждого события будем помнить день d_{prev}, в который произошло предыдущее событие, а также будем поддерживать t — количество отходов на производстве в конце дня d_{prev}. Тогда,

- чтобы обработать событие вида $(d, \texttt{+}, w)$, пересчитаем t как $t \cdot (1-r)^{d-d_{\text{prev}}} + w \cdot r$;
- чтобы обработать событие вида $(q, \texttt{?}, i)$, пересчитаем t как $t \cdot (1-r)^{d-d_{\text{prev}}}$, и запомним t как ответ на i-й запрос.

После остается только поменять d_{prev} на день текущего события, и перейти к обработке следующего. Вычислять степени $1-r$ можно с помощью алгоритма бинарного возведения в степень, который основан на том, что $x^{2k} = (x^k)^2$. Это позволяет считать k-ю степень числа за $\mathcal{O}(\log k)$. Время работы решения, таким образом — $\mathcal{O}((n+m)\log\max(\max(d), \max(q)))$.

Задача L. Места в метро

Автор задачи: Николай Ведерников, разработчик: Владимир Смаглий

Для решения этой задачи необходимо поддерживать два упорядоченных множества отрезков (рекомендуется воспользоваться соответствующей реализацией множества на вашем языке). В первом сете отрезки сравниваются по максимальному расстоянию, которое на нем можно получить (для самого левого и самого правого отрезков это величина равна их длине, так как можно сесть с краю; в остальных случаях $\lfloor \frac{len}{2} \rfloor$), а если дистанции равны, то сравниваются правые границы (для этого нужно воспользоваться соответствующим компаратором или переопределить оператор сравнения). Во втором сете отрезки сравниваются по левой границе.

Чтобы добавить пассажира, нужно достать последний элемент из первого сета и разделить его на две половины, удалив среднее сиденье, или удалить крайнее сиденье, если отрезок самый левый или самый правый.

При освобождении места Y ищем два отрезка во втором сете — ближайший слева и ближайший справа. Далее возможны три случая:

1. Ближайшие отрезки не граничат с Y. Нужно добавить отрезок с Y до Y.
2. Ровно один из отрезков граничит с Y. Нужно присоединить Y к этому отрезку и обновить у него границы.
3. Оба отрезка граничат с Y. Нужно объединить эти два отрезка в один большой.

Все обновления отрезков вносятся одновременно в два сета, для поддержания актуальности и совместимости данных.