Задача A. Производство роботов

Автор задачи: Константин Бац, разработчик: Мария Жогова

Ограничения первой группы позволяли явно перебрать пары элементов. Для прохождения второй группы нужно было рассмотреть два случая: $a_i \bmod 100 < 50$ и $a_i \bmod 100 \geqslant 50$. Заметим, что в первом случае объединять производство в пары не имеет смысла, так как для таких роботов $\left\lfloor \frac{a_i+a_j}{100} \right\rfloor = \lfloor \frac{a_i}{100} \rfloor + \lfloor \frac{a_j}{100} \rfloor$. Во втором случае имеет смысл объединить в пары производство максимальное количество машин, так как каждое объединение будет дополнительно сохранять одну единицу ресурсов. Поскольку все a_i равны, то объединять пары можно любым способом. Ясно, что можно получить не более $\lfloor \frac{n}{2} \rfloor$ пар.

В третьей подгруппе, аналогично, можно было разделить все затраты на производство на две группы: $a_i < 50$ и $a_i = 50$. Ясно, что производство из первой группы не имеет смысла объединять ни с производством из первой группы, ни с производством из второй группы — если $a_i < 50$, то $a_i + a_j \leqslant a_i + 50 < 100$, и это не сохранит ни одной единицы ресурсов. С другой стороны, имеет смысл объединить в пары как можно больше производств из второй группы, и каждое объединение даст одну единицу ресурсов. Таким образом, оптимальное распределение будет выглядеть так: $a_i < 50$ не объединяются ни с чем, $a_i = 50$ объединяются между собой.

Для полного решения необходимо заметить, что при формировании пар достаточно рассматривать не сами числа, а их остатки от деления на 100, после чего объединять в пары остатки, дающие в сумме хотя бы 100 (только в таком случае нам удастся сохранить дополнительную единицу ресурсов).

Давайте заменим все a_i на $b_i = a_i \bmod 100$, и отсортируем эти остатки по неубыванию. Посмотрим, с чем можно объединить b_1:

- если $b_1 + b_n < 100$, то его вообще ни с чем не имеет смысла объединять, так как $b_1 + b_i$ будет меньше 100 для любого i;
- иначе, их имеет смысл объединить — действительно, если бы в оптимальном ответе они были заняты в других парах ($b_1 + b_i \geqslant 100$ и $b_j + b_n \geqslant 100$), то и пары $b_1 + b_n$ и $b_i + b_j$ давали бы не худший ответ.

Таким образом, достаточно двигаться двумя указателями l и r навстречу друг другу от самых маленьких ($l = 1$) и самых больших ($r = n$) остатков. Если $b_l + b_r \geqslant 100$, объединяем их в пару и сдвигаем указатели друг к другу — теперь можно повторить аналогичное рассуждение. Иначе, если b_l не с чем объединять, сдвигаем только $l \to l + 1$. Время работы такого решения — $\mathcal{O}(n \log n)$ из-за сортировки.

Также заметим, что из-за того, что все остатки лежат от 0 до 99, можно обойтись без сортировки за $\mathcal{O}(n \log n)$, а просто посчитать количество a_i с каждым отдельным остатком по модулю 100 (сортировка подсчетом). Это бы дало время работы решения $\mathcal{O}(n)$, хотя и не было необходимо для получения полного балла.

Задача B. Журнал квестов

Авторы задачи: Константин Бац, Даниил Орешников, Владимир Рябчун, разработчик: Владимир Рябчун

Для решения первой подгруппы достаточно было рассмотреть несколько случаев соотношений приоритетов взятых квестов. Первые два квеста никогда не будут вычеркнуты, а для оставшихся двух достаточно просимулировать описанный в условии процесс.

В третьей и пятой подгруппах было достаточно просимулировать описанный процесс полностью для каждого нового квеста. За время $\mathcal{O}(n^2)$ (в третьей) или $\mathcal{O}(n)$ (в пятой) можно было для каждого нового квеста найти два минимальных значения приоритета в очереди, и либо добавить квест в конец, либо за еще $\mathcal{O}(n)$ времени пройтись по всей очереди целиком и выписать в новую очередь все важные квесты. Суммарное время работы такого решения (при линейном поиске минимумов) равно $\mathcal{O}(n^2)$.

Для решения оставшихся подгрупп было достаточно заметить, что если квест в какой-то момент не является неважным, то он таким и останется до момента выполнения. Действительно, если перед ним не было двух квестов с меньшим приоритетом, они и не появятся, так как новые квесты добавляются только в конец очереди.

Тогда можно отдельно хранить часть очереди, состоящую из гарантированно «важных» квестов, которая никогда не будет меняться, и следующий за ней хвост очереди, начинающийся с первого неважного квеста. Если добавленный новый квест вызывает очистку журнала, достаточно пройтись по второй части очереди, и удалить неважные квесты, а все оставшиеся перенести в первую часть. При такой реализации решение будет работать за $\mathcal{O}(n \cdot \texttt{check})$, где `check` — время поиска двух минимумов и проверки на неважность одного квеста, так как каждый квест будет обработан не более трех раз: при добавлении, при переносе в первую часть и при удалении.

Ограничения второй подгруппы гарантируют, что квесты добавляются в журнал в порядке увеличения приоритета, то есть в порядке уменьшения важности. Это гарантирует, что pm_1 и pm_2 будут равны приоритетам двух последних взятых квестов, а тогда для каждого нового квеста можно за $\mathcal{O}(1)$ проверить, вызывает ли он очищение журнала. Пользуясь рассмотренным выше фактом, получаем решение за $\mathcal{O}(n)$.

Четвертая подгруппа тоже позволяет выполнять `check` за $\mathcal{O}(1)$, например, если хранить количество квестов в журнале с каждым значением приоритета.

Для полного решения достаточно было рядом с каждой из двух частей очереди хранить кучу (`heap`) из всех приоритетов. Эта структура данных позволяет за время $\mathcal{O}(\log n)$ получать минимальный элемент, и с помощью нее можно реализовать поиск двух минимальных элементов за ту же асимптотику. В языке C++ можно было воспользоваться для этого структурой `std::set` и брать `*s.begin()` и `*(++s.begin())`.

Если такое решение не проходило по времени на последней группе тестов, можно было реализовать очередь, поддерживающую запросы минимумов за $\mathcal{O}(1)$ на двух «стеках с минимумом». Время работы полного решения, таким образом — $\mathcal{O}(n \log n)$ или $\mathcal{O}(n)$.

Задача C. Установка модулей GAIA

Автор задачи и разработчик: Даниил Орешников

Задача может быть вкратце описана следующим образом: для каждого i от 1 до n выбрать либо p_i, либо $(q \circ p)_i$, чтобы все выбранные числа были различны, и чтобы числа ни из какой пары (a_t, b_t) не были выбраны для i и $i+1$.

Первая подгруппа решается полным перебором всех случаев. Достаточно просто посмотреть на то, какие комбинации могут получиться при тех или иных выборах, и для каждого проверить все необходимые условия. Вторая подгруппа аналогична, однако для ее решения уже не хватит только условных операторов, потребуется написать полный перебор всех вариантов выбора с проверкой условий за время $\mathcal{O}(2^n \cdot n)$.

Третья подгруппа гарантирует, что перестановка p не двигает элементы, то есть для каждого i можно выбрать либо i, либо q_i. Можно также заметить, что пятая группа полностью аналогична — для каждого i можно выбрать либо p_i, либо i. Не теряя общности, расмотрим третью погруппу. Можно было заметить, что множество тех i, для которых выбрано **не** i, должно сохраняться при применении к нему перестановки q, значит должно состоять из одного или более *циклов* перестановки q. На таких циклах в качестве вершин можно построить граф, показывающий, можно или нет эти циклы выбрать вместе, после чего написать перебор с отсечением по времени.

Ограничения четвертой группы гарантируют только одно условие на ненахождение каких-то двух модулей рядом. Если в перестановке p они уже не находятся рядом, достаточно просто выбрать ее. Иначе, надо рассмотреть три случая — когда вместо $q \circ p$ выбрано для a_1, для b_1 и для них обоих, и проверить, подходит ли хотя бы один.

Полное решение можно было получить, сведя задачу к `2-SAT`. Заведем переменную t_i, которая принимает значение `true`, когда выбрана p_i, и `false`, когда выбрана $(q \circ p)_i$. Место, на котором окажется модуль x при выборе p, обозначим p_x^{-1}, и аналогично $(qp)_x^{-1}$ для $q \circ p$.

Теперь для каждой пары (a_i, b_i) переберем все возможные способы их расположения при выборе p или $q \circ p$ (их всего четыре). Например, $x = p_{a_i}^{-1}$ и $y = (qp)_{b_i}^{-1}$. Если $|x - y| = 1$, то добавим импликации $t_x \to \neg t_y$ и $t_y \to \neg t_x$, так как эти два выбора не могут быть сделаны вместе. Так мы учтем все условия на соседние модули.

Теперь еще учтем условия вида $t_{p_i^{-1}} \to \neg t_{(qp)_i^{-1}}$ и аналогичное ему обратное, $t_{(qp)_i^{-1}} \to \neg t_{p_i^{-1}}$, что позволит гарантировать, что все конечное назначение тоже будет перестановкой (все модули будут различными, каждый — в своем слоте).

Всего мы получили $\mathcal{O}(n+m)$ условий, на которых мы можем запустить решатель `2-SAT` через выделение компонент сильной связности за время $\mathcal{O}(n+m)$.

Задача D. Подземная лаборатория

Автор задачи и разработчик: Константин Бац

Переформулируем задачу: дано взвешенное дерево с вершинами-комнатами и ребрами-трубами. Корень дерева находится в самой нижней комнате, длина каждого ребра — разность между глубинами соответствующих вершин. Также в каждой вершине дерева изначально было некоторое количество льда, которое моментально тает от пришедшей сверху воды и начинает стекать вниз по трубе.

Ограничения первой группы тестов позволяли явно просимулировать то, как меняется уровень воды в комнатах и на основании этого ответить на запросы. Для каждой комнаты будем хранить четыре значения — флаг «растаял ли лед», текущий уровень воды, и состояние воды в трубе, ведущей из комнаты вниз. Состояние воды в трубе описывается тем, до какого нижнего уровня она уже опустилась, и, если вода в комнате уже закончилась, то на каком уровне находится самая верхняя оставшаяся единица воды в трубе.

Будем за один шаг увеличивать время на 1, и обрабатывать комнаты в порядке увеличения глубины:

- если в комнате i растаял лед и там больше 0 воды, изменим уровень воды на -1, увеличим нижний уровень воды в трубе на $+1$;
- если в трубе верхний уровень больше 0, то увеличим оба уровня на $+1$ (весь «отрезок» воды переместился на 1 вниз);
- если нижняя граница воды в трубе меньше нуля (это значит, что вода уже дотекла до следующей комнаты b_i), отметим, что в комнате b_i растаял лед (если он до этого был замерзшим, выставим уровень воды равным a_i) и увеличим уровень воды в комнате b_i на $+1$.

Стимуляцию стоит проводить до тех пор, пока вода не прекратит движение. Для каждой комнаты можно в любой удобной структуре данных хранить «график» количества воды в ней. Если посчитать на нем суффиксные суммы, на запросы можно будет отвечать за $\mathcal{O}(1)$. Время работы такого решения — $\mathcal{O}(T \cdot n + m)$, где T — время, за которое вода из комнат дотечет до самой глубокой комнаты и вытечет из нее.

Для решения следующих подгрупп следует обратить внимание на то, как изменяется уровень воды в комнатах, которые находятся под землей. Для начала можно (например, по индукции по глубине) доказать, что лед в комнате на глубине d_i тает ровно в момент времени d_i, потому что вода «непрерывно» движется вниз со скоростью 1.

Более того, вода из всех труб, ведущих в конкретную комнату, начнет поступать в нее одновременно, после чего скорость роста воды в ней будет равна $|\texttt{children}(i)| - 1$. Опять же, по индукции, можно доказать, что каждая комната является непустой в течение непрерывного промежутка времени. Поэтому:

1. после того, как последние единицы воды вытекут из комнаты, в ней больше никогда не появится вода;
2. скорость поступления воды в комнату будет монотонно уменьшаться с тем, как она будет заканчиваться в комнатах над ней, пока не достигнет нуля;
3. когда вода закончится в последней трубе, ведущей в комнату, уровень воды в комнате начнет уменьшаться со скоростью 1, пока она в ней не закончится.

Во второй группе в качестве дерева был дан бамбук. Это позволяет заметить, что вся вода, которая попадает в комнату, сразу же уходит ниже, и в комнате не скапливается (скорость роста воды равна 0). Пользуясь этим, можно было для каждой комнаты в порядке увеличения глубины подсчитать h_i^{stop} — время, в которое вода прекратит поступать из комнаты сверху. Как уже было сказано, лед в комнате i тает в момент d_i, поэтому с момента d_i минут по h_i^{stop} минут уровень воды будет в точности равен a_i, после этого в течении a_i минут он будет уменьшаться со скоростью 1. Это позволяет отвечать на запросы за время $\mathcal{O}(1)$.

В третьей группе было дано полное двоичное дерево, и все трубы были равной длинны. Это ограничение гарантировало, что для всех вершин «на одном уровне» уровень воды будет меняться одинаково. И для каждой вершины изменение уровня воды будет проходить в два этапа: сперва уровень будет расти со скоростью 1, а затем, когда вода перестанет течь из верхних комнат, уровень воды будет уменьшаться со скоростью 1. Для ответов на запросы можно было вывести формулу, соответствующую такому процессу.

В подзадачах четыре и пять можно было в той или иной мере неэффективно применить идею полного решения.

Как уже было сказано выше, уровень воды в комнатах сперва монотонно возрастает, а потом монотонно убывает. Ясно, что изменение уровня воды напрямую зависит от скорости поступления воды в комнату. Если в комнату поступает вода из v труб, то за одну минуту уровень воды повышается на $v - 1$. Поэтому максимальный уровень воды достигается, когда все трубы, кроме одной, перестают давать воду.

Для полного решения следовало для каждой комнаты создать массив событий, описывающих изменение скорости роста. Обрабатывая комнаты в порядке возрастания их глубины, несложно определить, в какие моменты будет меняться скорость: если комната i станет полностью пустой в момент времени t, то в момент времени $t + d_{b_i} - d_i$ скорость роста уровня воды в комнате b_i уменьшится на 1. Все такие события, плюс событие «в момент времени d_i скорость роста становится равна $|\texttt{children}(i)|$», можно отсортировать по возрастанию момента времени, после чего посчитать на них префиксные суммы скоростей с учетом длин интервалов времени (это и будут уровни воды в моменты изменения скорости).

Тогда для ответа на запрос достаточно с помощью бинпоиска по «растущей» части массива событий найти первый момент времени, в который уровень достигает x, и вычесть его из последнего момента, когда уровень равен x. Последний момент найти еще проще — он случается, когда все трубы над комнатой опустели, и уровень воды в комнате снижается. Зная момент последнего события и уровень воды на этот момент, можно найти интересующий нас конец интервала. Такое решение будет работать за $\mathcal{O}((m + n) \cdot \log n)$.