Разбор задачи «Рабочий процесс»

Опишем полное решение.

Сначала определим момент времени, в который начнется монтаж сетки.

В момент времени 0 со склада выезжает машина с a единицами сетки, которая будет добираться до места в течение d единиц времени. В этот же момент времени рабочие начинают свой первый перерыв. Они будут устраивать перерывы длительностью p до тех пор, пока не приедет машина с сеткой. Когда машина приедет, то пройдет $d \mod p$ единиц времени от начала последнего перерыва. Если это число не 0, то спустя еще $p - d \mod p$ единиц времени рабочие выгрузят сетку из машины и начнут монтаж. Для краткости обозначим это время как $start$.

В момент начала монтажа машина уедет на склад за новой порцией сетки и вернется через $2 \cdot d$ единицы времени. Монтаж сетки (мы пока рассматриваем ситуацию, что всю сетку нужно смонтировать) займет $m \cdot a$ единиц времени. Возможны два варианта: либо машина приезжает раньше и ожидает разгрузки до истечения времени монтажа, либо монтаж завершается раньше, после чего рабочие устраивают перерыв в ожидании машины. Заметим, что эта ситуация будет в точности воспроизводиться и в дальнейшем, кроме последнего завоза сетки, когда, возможно, не нужно будет монтировать ее всю.

Рассмотрим оба варианта.

Пусть $2 \cdot d \leqslant m \cdot a$. Тогда рабочим больше не удастся устроить ни одного перерыва, и вся сетка будет смонтирована за $m \cdot s + start$.

Пусть $2 \cdot d > m \cdot a$. В этом случае после монтажа сетки рабочие отдохнут еще некоторое время. К моменту, когда приедет машина, от последнего начавшегося перерыва пройдет $(2 \cdot d - m \cdot a) \mod p$ единиц времени. Так что работа начнется в момент $2 \cdot d + (p - (2 \cdot d - m \cdot a) \mod p)$, если считать со времени предыдущего приезда машины с сеткой.

Теперь остается собрать все вместе: посчитать, сколько целых порций сетки будет доставлено после старта и сколько времени будет потрачено на монтаж последней порции. Вычисления нужно проводить достаточно аккуратно: хотя все числа по отдельности помещаются в 32-битный целый тип, для записи результата его будет недостаточно.

Подзадачи меньшего размера можно было решить, выполнив симуляцию процесса. В первой подзадаче при этом можно было обойтись 32-битным целым типом для представления всех данных и результатов.

Разбор задачи «Вопросы печати»

В этой задаче нужно было выполнить аккуратную симуляцию описанного процесса.

Опишем некоторые полезные приемы. Пусть фрагменты хранятся в массиве $image[i]$. Также пусть имеется вспомогательный массив $colors[] = \{1, 2, 4, 8\}$, элементы которого отвечают «чистым» цветам: черному, желтому, красному и синему соответственно. Тогда определить значение бита цвета $\#j$ в фрагменте $\#i$ можно так:

```
int bit = ((image[i] & colors[j]) >> j);
```

Здесь & — побитовый оператор «И», $>>$ — оператор побитового сдвига вправо, а $j = 0, 1, 2, 3$.

Конечно, это не единственный способ определения битов цветов, но, пожалуй, он наиболее короткий.

Когда значения битов цветов фрагмента будут определены, легко проверить, можно ли напечатать этот фрагмент. Если это невозможно, сохраним его номер в какой-либо списочной структуре и «выполним» замены тех секций картриджа, количество краски в которых на текущий момент строго меньше d.

Теперь печать гарантированно возможна, поэтому уменьшим значения для тех секций картриджа, которые соответствуют ненулевым битам рассматриваемого фрагмента.

По завершении процесса обработки фрагментов получим списочную структуру, содержащую моменты замен; ее размер как раз будет количеством замен.

Подзадачи 1 и 3 позволяли оперировать с чистыми цветами; иными словами, для печати фрагмента всегда расходовалось содержимое только одной секции, что немного упрощало симуляцию (можно было обойтись без битовых операций).

Кроме того, подзадачи 1 и 2 позволяли выполнять определение битов цветов и проверку возможности печати фрагмента практически любым разумным способом.

Разбор задачи «Планы на будущее»

Опишем полное решение задачи.

Заведем дополнительный массив $z[]$, элемент $z[j]$ этого массива — момент времени, в который здание $\#j$ будет фактически освобождено от сетки. Как понятно, $z[j] = \max(t[j-1], t[j], t[j+1])$.

Теперь пройдем по массиву $z[]$ (слева направо) и сгруппируем подряд расположенные элементы с одинаковыми значениями. Такие элементы будут соответствовать зданиям, которые одновременно будут освобождаться от сетки. Сохраним такие группы в виде (*номер первого элемента группы*, *номер последнего элемента группы*, *значение элемента группы*), а затем упорядочим их по значениям элементов групп, а при равных значениях — по номеру первого элемента. После этого останется только вывести номера первого и последнего элементов каждой группы в полученном порядке.

Минимальность количества групп (нарядов) при таком решении достигается естественным образом. Объединять в один наряд можно только расположенные подряд здания, укрытые сеткой; снимать же сетку следует немедленно, как только возникает такая возможность.

При решении первой подзадачи можно было действовать любыми разумными методами — например, пошагово симулировать процесс и на каждом шаге перебирать все здания для формирования групп.

Вторая подзадача предполагала более аккуратную симуляцию процесса — например «сжатием» временной шкалы (т.е. учетом лишь тех моментов времени, в которые происходили какие-то события).

Наконец, третья подзадача требовала не только работы со сжатыми по времени данными, но и достаточно эффективного выбора подгрупп.

Разбор задачи «Обмен ресурсами»

Для начала формализуем задачу. Нам дан граф из n вершин, изначально ребер между вершинами нет. В процессе нам добавляют ребра по одному.

В задаче требуется определить номер первого ребра, нарушившего двудольность графа (возможность разбить вершины на две доли так, чтобы ребра соединяли между собой только вершины разных долей). Если такое ребро было, нужно вывести его номер. Если же нет — вывести любое допустимое разбиение графа на две доли.

Разберем решения различных подзадач.

Сразу скажем, что если какое-то ребро встречается во вводе несколько раз, то нас интересует только его первое появление — в дальнейшем это ребро никак не повлияет на возможность разбить граф на две доли.

Подзадача 1.

В этой подзадаче было не более 10 вершин и, следовательно, не более 45 уникальных ребер.

Переберем все возможные разбиения вершин на две доли с помощью двоичных масок за $O(2^n)$. Для каждого разбиения будем перебирать ребра в порядке их добавления в граф, если какое-то ребро нарушило разбиение на две доли — запомним его номер для этой маски. Если для какой-то маски нет такого ребра — значит, эта маска задает корректное разбиение графа на две доли, и ответ найден.

В противном случае для каждой маски существует ребро, которое нарушает разбиение графа на две доли. Выберем максимальный из номеров ребер (по маскам) — он и будет ответом. Действительно, если мы выбрали в качестве ответа некоторое ребро x, то для данного разбиения все предыдущие ребра ничего не нарушали, а значит, маски, у которых соответствующий номер ребра меньше x, просто были неоптимально выбраны. Выбрать же маску, у которой номер ребра больше x, мы не можем, так как по нашему алгоритму x — максимальный номер ребра среди всех масок.

В итоге получили решение за $O(n^2 \cdot 2^n)$ (n^2 — верхняя оценка уникальных ребер в графе из n вершин), что легко укладывается в ограничения первой подзадачи.

Подзадача 2.

Количество вершин выросло до 100, соответственно количество уникальных ребер увеличилось до 4950.

Давайте научимся проверять двудольность графа за $O(m+n)$. Воспользуемся следующим свойством: граф является двудольным, если в нем нет нечетных циклов (утверждение довольно очевидное, так что его доказательство мы оставим за рамками данного разбора). Давайте рассмотрим какую-либо компоненту связности и запустим волновой алгоритм (обход в ширину) от какой-либо вершины до всех остальных. Если в какой-то момент между вершинами a и b, принадлежащими одной «волне» алгоритма, есть ребро — это соответствует наличию нечетного цикла в графе. В самом деле, для данных вершин существует вершина c на пути от a и b до стартовой, такая, что кратчайшие пути (a, c) и (b, c) не имеют общих ребер, а длины путей равны (иначе вершины не были бы в одной волне). Это значит, что суммарная длина цикла $c-a-b-c$ равна $2 \cdot k+1$, т.е. является нечетным числом.

Будем добавлять ребра в порядке добавления в граф, после добавления будем проверять, не нарушилась ли двудольность. Если после добавления всех ребер граф двудольный, то разбиение найти несложно — в одну долю пойдут все вершины из долей одинаковой четности.

Такое решение работает за $O(n^2 \cdot (n+n^2)) = O(n^4)$, что укладывается в отведенное время.

Подзадача 3.

Количество вершин увеличилось до 10^5, количество возможных уникальных ребер увеличилось до примерно $10^{10}/2$, что достаточно много, поэтому в данной подзадаче вступает в силу ограничение на m — 10^5.

Зададим функцию $f(i)$: 1, если граф является двудольным, если в него добавлено первых i ребер; 0 —иначе. Вычислить ее значение можно за $O(n+m)$ для одного запроса: просто пустим обход в ширину только по ребрам, индекс которых не превосходит i.

Если $f(m) = 1$ — граф остался двудольным после добавления всех ребер, и ответ уже найден.

Иначе мы имеем функцию, такую что:

- $f(0) = 1$ — ребер нет, а значит нет и нечетных циклов
- $f(m) = 0$ — мы это явно проверили
- если $f(i) = 1$, то и $f(i-1) = 1$ — если граф с i ребрами двудольный, то без одного ребра нечетных циклов не появится
- если $f(i) = 0$, то и $f(i+1) = 0$ — если в графе с i ребрами появился нечетный цикл, то дополнительные ребра никак не повлияют на его наличие
- из последних двух утверждений следует свойство $f(i) \geqslant f(i+1)$.

Итого мы имеем функцию, у которой на границах различные значения и она является монотонно невозрастающей. В таком случае где-то на отрезке $(0; m]$ найдется индекс i такой, что $f(i-1) = 0$, $f(i) = 1$ — именно добавление i-го ребра нарушило двудольность, а значит это ребро и является ответом на задачу. Для поиска такого индекса мы можем воспользоваться бинарным поиском по ответу. Данный подход решает задачу за $O(\log(m) \cdot (n+m))$.

Подзадача 4.

Данная подзадача представляла собой графы с очень большим количеством вершин и ребёр — до $750 \cdot 10^3$ каждое.

Вернемся к истокам. Двудольность — это отсутствие нечетных циклов в графе. Ранее мы использовали тот факт, что если нечетный цикл есть, то его можно найти, используя дерево кратчайших путей от одной вершины (обход в ширину). Но фактически нас не интересовала длина пути, нас интересовала лишь его четность.

Давайте научимся при добавлении ребра быстро определять, существует ли для концов ребра какая-либо вершина, до которых у них есть путь одинаковой четности. В таком случае мы решим задачу за $O(m \cdot check(n, m))$, где $check()$ — данная проверка.

Такую проверку можно осуществлять за $O(alpha(n))$ или за $O(\log^*(n))$, где $alpha()$ — обратная функция Аккермана, а $\log^*$ — итерированный логарифм. Обе эти функции растут очень медленно, а

встречаются при применении структуры данных DSU (СНМ, система непересекающихся множеств) с ранговой эвристикой и эвристикой сжатия пути.

Детальное рассмотрение структуры DSU выходит за рамки данного разбора, поэтому сразу рассмотрим её модификацию для поиска нечетных циклов. Для каждой вершины v будем хранить $par[v]$ — четность пути (0 или 1) до её предка в DSU. В таком случае четностью пути до представителя будет являться xor (исключающее или) четностей всех вершин на данном пути. Реализуется данное вычисление параллельно с поднятием до представителя (базовая операция DSU).

Когда объединяются две независимые компоненты, то фактически у всех вершин «подвешенной» компоненты меняется четность пути, так как меняется представитель. Но вместо этого мы можем изменить четность пути только для «подвешенного» представителя, а четности остальных вершин вычислять по пути через него.

Если мы добавили ребро $(from, to)$, которое соединило независимые компоненты и их представители соответственно a и b (будем считать b подвешенным), то $par[b] = parity(from)\ xor\ 1\ xor\ parity(to)$, где $parity(v)$ — итоговая четность пути до представителя.

В ином случае мы добавили ребро в уже связанную компоненту, что могло создать нечетный цикл. Данное условие записывается как $parity(from) == parity(to)$.