

Выбор версий компонентов

Автор задачи: Даниил Орешиников, разработчик: Павел Скобелкин

В **первой подгруппе** имеется ровно одна зависимость. Можно заметить, что в таком случае всем компонентам, которые не зависят от других, можно выбрать одинаковую версию — x , а единственной компоненте, зависящей от какой-то другой — версию $y \geq a \cdot x + b$. Так как мы хотим минимизировать сумму версий компонентов, выберем минимально возможную версию $y = a \cdot x + b$. Теперь решим неравенство $\sum_{i=1}^n d_i \leq X$, значит $x \cdot (n - 1 + a) + b \leq X$, и узнаем X и Y .

Во **второй подгруппе** граф зависимостей имел всего 3 компонента и $X \leq 100$, значит можно было перебрать распределение X по трем компонентам, проверив выполнение всех зависимостей.

Далее в **третьей, четвертой и восьмой подгруппе** было ограничение $X \leq 10 \cdot n$. Это довольно сильно ограничивает ответ, ведь несложно показать, что ответ на задачу (то есть, минимальная из версий компонент) будет не более $\frac{X}{n}$, что в этой серии подзадач не превосходит 10. Значит, в этих подзадачах можно было **перебрать** минимальную из версий компонент и проверить, возможно ли расставить версии так, чтобы минимальная версия была такой.

Для подхода к полному решению нужно было воспользоваться бинарным поиском по ответу. Действительно, если существует распределение версий, которое удовлетворяет заданным ограничениям и имеет минимальную версию m , то есть и распределение версий, в котором минимальная версия равна $m - 1$. Аналогично, если не существует ответа с минимумом M , то не существует и ответа с минимумом $M + 1$.

Исходя из соображений выше, мы свели задачу от нахождения минимума к проверке: теперь нужно проверить, есть ли такое распределение версий, в котором минимальная версия равна m . Будем делать это с помощью динамического программирования.

В **третьей и пятой подгруппе** граф зависимостей имел форму бамбука. Мы знаем, что минимальная версия равна m . Тогда поставим ее в лист нашего бамбука и начнем подниматься вверх (к корню), выбирая текущую версию как минимально возможную (то есть, для зависимости u_i, v_i, a_i, b_i , которая образует ребро $u_i \rightarrow v_i$, выберем $d_{u_i} = a_i \cdot d_{v_i} + b_i$). В этой и следующих подзадачах необходимо было обрабатывать переполнение — например, если в какой-то момент мы назначили вершине версию $> X$, то нужно было закончить проверку (в таком случае, ответа не существует).

В **четвертой и шестой подгруппе** граф зависимостей имел форму дерева. В таком случае можно было воспользоваться обходом в глубину. Таким образом, для вершины v нужно было рекурсивно запустить обход для всех ее соседей и взять как версию текущей компоненты значение $\max_{u \rightarrow v_i} (a_i \cdot v_i + b_i)$, где v_i — это ребра, исходящие из u .

Для решения **седьмой, восьмой и девятой подгруппы** нужно было воспользоваться ДП на ациклических неориентированных графах (ДП на DAG'e). Так как граф ациклический, у него существует топологическая сортировка — давайте рассмотрим ее. Теперь пойдем по ней справа налево и будем считать ДП: сначала поставим в вершину p_i значение m — текущий минимум, который мы проверяем. Тогда значение ДП в текущей вершине это максимум из m и значения линейных функций по исходящим ребрам. Все ребра из этой вершины идут направо, значит все значения ДП, от которых зависит текущее значение, уже посчитаны.

В зависимости от реализации алгоритма это решение проходило 7, 8, 9 подгруппы. Для решения **десятой подгруппы** нужно было аккуратно обработать переполнения, ведь в бинарном поиске значение m могло быть порядка X , то есть 10^{18} . Самый простой вариант — тип `int128` в C++. Также для проверки, что произведение чисел a и b больше 10^{18} ($a \cdot b > 10^{18}$) можно использовать другую технику: $a \cdot b > 10^{18} \Rightarrow a > \frac{10^{18}}{b}$ и проверять такое условие.