Разбор задачи «Акции»

Будем решать эту задачу методом динамического программирования.

Заведём массив $adp[i]$ и связанный с ним $add[i]$, а также массив $bdp[i]$.

Элемент $adp[i]$ будет хранить максимально возможное количество акций, которое может быть у Стива в момент времени $\#i$, элемент $add[i]$ — количество денег, имеющихся у Стива в этот же момент наряду с акциями (деньги могли остаться при покупке акций; при прочих равных это также максимальное количество денег).

Элемент $bdp[i]$ будет хранить максимально возможное количество денег, которое может быть у Стива в момент времени $\#i$ при условии, что на руках у него только деньги, а акций нет.

Инициализация: $adp[0] = n$, $add[0] = 0$, $bdp[0] = 0$.

Переходы:

для акций — можно либо оставить уже имеющийся пакет, либо купить акции на все деньги:

$adp[i+1] = \max(adp[i],\ bdp[i] \,/\, b[i+1])$

для связанного массива $add[i]$ — если вариант выбирается однозначно (оставлены акции из предыдущего момента времени или же куплены), то элемент массива вычисляется в соответствии с этим вариантом; если же оба варианта равноценны, то выбирается максимальное из двух возможных значений:

$$add[i+1] = \begin{cases} add[i], & \text{если } adp[i] > bdp[i] \,/\, b[i+1] \\ bdp[i] \mod a[i], & \text{если } adp[i] < bdp[i] \,/\, b[i+1] \\ \max\left(add[i],\ bdp[i] \mod a[i]\right), & \text{если } adp[i] = bdp[i] \,/\, b[i+1] \end{cases}$$

для денег — можно либо оставить деньги, имеющиеся в предыдущий момент времени, либо продать все акции (из предыдущего же момента времени) и учесть деньги, которые наличествовали вместе с акциями:

$bdp[i+1] = \max(bdp[i],\ adp[i] \cdot a[i+1] + add[i])$.

Почему этот подход корректен?

Во-первых, очевидно, что в какой-то момент Стив должен продать акции. В какой бы момент он ни собрался их продавать, за большее количество акций он выручит больше денег. Поэтому мы стремимся максимизировать в каждый момент времени количество акций, а затем выясняем, какую сумму Стив сможет выручить при продаже.

Количество денег, которые Стив получает, продав акции, мы также стремимся максимизировать. Действительно, если в какой-то момент времени выгодно приобрести акции (чтобы потом их продать), то лучше приобрести их как можно больше. А чтобы приобрести как можно больше, нужно иметь в распоряжении больше денег.

Возможно, стоит пояснить, почему у нас не возникает ситуации, когда выгодно «докупить акции» к уже имеющемуся пакету на «остаток денег». Как понятно, этот «остаток» образовался, когда Стив приобретал акции. Следовательно, этот остаток был меньше, чем стоимость одной акции на момент покупки (согласно условию задачи). Если в какой-то момент этого «остатка» может хватить на приобретение ненулевого количества акций, то цена этих акций стала ниже, чем в тот момент, когда Стив их покупал. Поскольку мы стремимся максимизировать количество акций, то выгоднее не покупать их тогда, когда покупал Стив, а дождаться именно этого момента, когда акции подешевеют, и приобрести большее количество на всю сумму.

В качестве ответа необходимо предъявить не только максимально возможную сумму (очевидно, она будет храниться в $bdp[m]$), но и последовательность действий Стива.

Чтобы корректно восстановить ответ, можно действовать, например, так. Заведём ещё два вспомогательных массива $afrom[i]$ и $bfrom[i]$, в которых будем хранить «историю» для массивов $adp[i]$ и $bdp[i]$ соответственно. Как один из вариантов, будем хранить в соответствующих элементах этих вспомогательных массивов символ `A`, если элемент $\#i$ интересующего нас массива был получен переходом из элемента массива $adp[i-1]$, и символ `B`, если элемент $\#i$ был получен переходом из элемента массива $bdp[i-1]$.

Когда массивы $adp[i]$ и $bdp[i]$ будут заполнены, для получения последовательности действий Стива достаточно будет пройти в обратном порядке по массивам $afrom[i]$ и $bfrom[i]$ и сформировать требуемую строку (очевидно, что в этой строке не может быть двух одинаковых символов `A` или `B` подряд).

Асимптотика решения $O(n)$ (все действия выполняются за линейное по количеству элементов время).

Разбор задачи «Блокирующий пакет»

Поскольку требуется минимизировать количество сделок, нужно каждый раз приобретать максимально возможное (на текущий момент) количество акций. Чтобы организовать такой процесс, предварительно отсортируем владельцев акций по параметру b_i (по неубыванию).

Далее будем действовать следующим образом. Пусть z — количество акций, которым обладают скупщики в текущий момент (понятно, что в начальный момент времени $z = 0$). Рассмотрим всех владельцев акций, у которых $b_i < z$ (и которые ещё не продали свои акции), и выберем из них того, у которого значение a_i наибольшее. После этого увеличим z ($z = z + a_i$) и, возможно, добавим к уже имеющемуся множеству владельцев акций кого-то ещё.

Понятно, что выполнять поиск максимума каждый раз достаточно долго, поэтому следует использовать сортирующую структуру данных — например, очередь с приоритетами (PriorityQueue) или множество (set, TreeSet). В этой структуре элементы должны упорядочиваться по значению a_i.

Каждый раз, когда изменилось значение z (а также в начальный момент времени), будем брать очередные подходящие элементы из отсортированной по b_i последовательности и помещать их в сортирующую структуру. Когда все подходящие элементы помещены в структуру, удалим из неё элемент с наибольшим значением a_i, обновим z и повторим всё описанное выше.

Процесс заканчивается либо когда после очередного обновления z оказалось больше или равно s, либо когда для некоторого z ($z < s$) нельзя добавить новые элементы в сортирующую структуру (очередной владелец акций имеет слишком высокие требования или же все владельцы акций продали свои акции, но нужной суммы так и не набралось).

Асимптотика решения: $O(n \cdot \log n)$ на предварительную сортировку, $n \cdot O(\log n)$ на добавление элемента в сортирующую структуру. Извлечение максимального элемента не требует перестройки сортирующей структуры, проход по отсортированным по b_i элементам осуществляется за линейное время (каждый раз будем запоминать, где остановились).

Разбор задачи «Разделяй, если не властвуешь»

Проведём небольшой анализ. Сначала подсчитаем, по сколько контрактов было заключено с каждой из компаний за уже прошедшие m дней и найдем максимальное значение среди этих величин.

Понятно, что ответ не может быть меньше этого максимального значения (обозначим его за z). Поэтому базовое предположение будет состоять в том, что каждая компания заключила по z контрактов.

Теперь нам нужно придумать процедуру, которая оптимально «соберёт» эти контракты в пары. Здесь (как и в задаче B) нам окажется полезной сортирующая структура (PriorityQueue или set / TreeSet). Будем хранить в такой структуре пары (*количество ещё не заключённых контрактов, номер компании*), отсортированные по первому параметру. Каждый раз будем извлекать из этой структуры по два элемента, имеющие наибольшее количество ещё не заключенных контрактов, образовывать из этой пары сделку очередного дня, уменьшая количество ещё не заключённых контрактов на единицу в каждом из этих элементов, после чего возвращать изменённые элементы в структуру.

Конечно, может случиться так, что по окончании этой процедуры в структуре может остаться один элемент с ненулевым количеством ещё не заключённых контрактов. В этом случае добавим по одному дополнительному контракту всем компаниям и повторим процедуру. Будем делать так до тех пор, пока не получим в итоге «пустую» структуру.

Почему эта процедура приведёт нас к правильному (оптимальному) ответу? Если у нас получилось распределить контракты, не увеличивая их количество ни разу, лучший результат мы получить просто не могли (поскольку существует ограничение снизу).

Допустим, что нам пришлось один раз увеличивать количество небольших контрактов на единицу. Это могло произойти только в том случае, когда в сортирующей структуре оставался единственный ненулевой элемент. Очевидно, исходно (когда структура впервые формировалась) этот элемент

был максимальным (возможно, одним из нескольких максимальных). Заметим, что если этот ненулевой элемент содержал отличное от единицы количество ещё не заключённых контрактов, то он оставался максимальным всё время, пока мы выполняли процедуру. Т.е. каждый день заключался контракт именно с этой компанией и какой-либо другой, и при этом каких-либо других суммарно попросту не хватило. В этом случае понятно, что лучшего способа заключать контракты у нас не было, и на этом шаге решение ещё остаётся оптимальным.

Вторая возможность состоит в том, что ненулевой элемент содержит один ещё не заключённый контракт. Это могло произойти как в случае, когда этот элемент был единственным максимальным, так и в случае, когда максимальных элементов было несколько. Но в обоих случаях важно другое: общее количество ещё не заключённых контрактов было нечётным, что не давало возможности разделить все контракты на пары. Поскольку это количество небольших контрактов возникло либо из нижней оценки (z), либо из корректного предыдущего построения с единственным максимумом, уменьшено оно быть не может, остаётся только увеличивать его.

Оценим асимптотику решения.

Обозначим ответ ans, тогда нам придётся обработать с помощью структуры данных $\sum_{i=1}^{n} (ans - d_i)$ элементов, где d_i — это количество уже заключённых в течение m дней контрактов с компанией $\#i$.

Ответ ans можно оценить сверху как $2 \cdot m$ (например, при $n = 3$ можно предположить, что в каждый из m дней заключались контракты с компаниями #1 и #2, а с компанией #3 не было заключено ни одного контракта; в других случаях можно обойтись и меньшим количеством).

Поэтому оценка сверху для суммы составит $2 \cdot m \cdot n$ — не более такого количества операций нам потребуется совершить. В действительности, конечно, операций будет меньше: какое-то количество контрактов у нас уже заключено.

Каждая такая операция будет осуществляться за $O(\log n)$ шагов, поскольку в структуре никогда не будет более n элементов (по количеству компаний).

Итоговая асимптотика составит $O(m \cdot n \cdot \log n)$.

Разбор задачи «Обновления»

Внимательно прочитав описание, можно понять, что новые функции образуют одну или несколько древовидных структур. Действительно, если считать новые функции вершинами, то у каждой вершины есть не более одного предка.

Легко видеть, что самую дорогую функцию нужно искать среди тех, у которых предков нет, а стоимость такой функции определяется количеством вершин в дереве, корнем которого она является. Таких функций может быть несколько (но совершенно точно имеется хотя бы одна).

Так что, вероятно, один из наиболее очевидных путей решения этой задачи состоит в том, чтобы сформировать граф в виде списка (или массива) списков, сопоставляя каждой вершине список тех вершин, которые зависят от неё непосредственно, а затем запустить от каждой вершины, не имеющей предков, обход графа (например, поиск в глубину, depth first search).

Формирование графа можно выполнить за линейное по количеству рёбер время; за линейное по количеству вершин время отыскиваются вершины, не имеющие предков. Наконец, асимптотика алгоритма dfs составляет $O(n + m)$, где m — количество рёбер (в нашем случае оно не превышает количества вершин).

Ещё одной возможностью является построение системы непересекающихся множеств (disjoint set union). При таком подходе можно сразу формировать деревья из поддеревьев и определять их корень. (Вероятно, это более сложный для понимания подход, нежели описанный выше).

Подробно про алгоритмы обхода графа и систему непересекающихся множеств можно прочитать в литературе (например, в книге «Алгоритмы. Построение и анализ», Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн) или на различных интернет-ресурсах (например, на e-maxx.ru).

Разбор задачи «Переподготовка»

В этой задаче можно использовать «жадный» подход.

Отсортируем сотрудников по неубыванию количества дней, которое требуется им для подготовки к экзамену. Пусть в результате сортировки получится последовательность $d_{i_1}, d_{i_2}, \ldots, d_{i_n}$, $(d_{i_1} \leqslant d_{i_2} \leqslant \ldots \leqslant d_{i_n})$.

Сотрудник $\#i_1$ может сдавать экзамен самое позднее в день $d_{i_1} + m$ (иначе его квалификация не будет подтверждена). Все остальные сотрудники могут быть разделены на две большие группы — с $d_{i_j} < d_{i_1} + m$ $(j = 2, 3, \ldots b)$ и $d_{i_j} \geqslant d_{i_1} + m$ $(j = b+1, b+2, \ldots, n)$ (здесь b — некий «барьерный» элемент).

Из первой группы (с $j \leqslant b$) возьмём первых $k-1$ сотрудников (одного сотрудника — $\#i_1$ — мы уже взяли) и включим их в первую группу экзаменующихся. Если таковых окажется меньше, чем $k-1$, возьмём всех; группа будет неполной, но выбора нет: сотрудник $\#i_1$ не сможет сдать экзамен позже.

После формирования первой экзаменующейся группы точно таким же образом сформируем вторую: выберем сотрудника, который будет готов сдавать экзамен раньше всех (оставшихся), определим для него предельный срок сдачи экзамена и постараемся включить в экзаменующуюся в этот день группу максимально возможное количество сотрудников (не большее k, разумеется).

Будем продолжать такой процесс формирования экзаменующихся групп, пока все сотрудники не окажутся включёнными в какую-либо экзаменующуюся группу.

Представляется достаточно очевидным, что такой подход гарантирует минимальное количество сформированных экзаменующихся групп.

Действительно, обязательно нужно назначить экзамен в такой день, в который его сможет сдать сотрудник, подготовившийся раньше всех. В нашем решении такой экзамен назначается в последний из возможных дней; допустимо его назначить в любой день, начиная с $d_{i_1} + 1$ по $d_{i_1} + m$. Если назначить экзамен раньше, возможны два варианта: либо все, кто попадал на экзамен в день $d_{i_1} + m$, попадут на него и в более ранний день (и тогда ничего не изменится), либо кто-то не успеет подготовиться, и экзаменующаяся группа будет меньше. В этом случае те, кто не успевает сдать экзамен в первой группе, будут сдавать его во второй (или в какой-либо ещё из следующих групп). Это не «улучшит» ситуацию: придётся потесниться кому-то из тех, кто в нашем решении попал во вторую и следующие группы. Поэтому назначить первый экзамен в последний из возможных дней для сотрудника $\#i_1$ является выбором, который как минимум не ухудшает решение.

Рассуждая аналогичным образом, придём к тому, что любое оптимальное решение может быть преобразовано к построенному, и количество групп при этом не изменится.

Асимптотика: сортировка выполняется за $O(n \cdot \log n)$, формирование групп за $O(n)$, итоговая асимптотика $O(n \cdot \log n)$.

Разбор задачи «Учебный план»

Будем последовательно выполнять подсчёт для каждого менеджера, который любит отправлять документы «на доработку», сколько дней потребуется, чтобы получить его подпись. Непосредственный подсчёт может оказаться слишком долгим, поэтому будем учитывать результаты подсчётов для менеджеров с меньшими номерами.

Рассмотрим первого такого менеджера, имеющего номер $\#k_1$. Чтобы получить его подпись, придётся проделать следующее:

- собрать $k_1 - 1$ подписей за $k_1 - 1$ дней;
- потратить один (1) день, чтобы получить отказ от этого менеджера;
- снова собрать $k_1 - 1$ подисей за следующие $k_1 - 1$ дней;
- потратить один (1) день, чтобы получить подпись этого менеджера.

Таким образом, мы потратим $2 \cdot k_1$ дней.

Теперь рассмотрим второго такого менеджера — с номером $\#k_2$.

К нему мы отправимся с подписью менеджера $\#k_1$ (которую получили за $2 \cdot k_1$ дней). Добраться после получения этой подписи до $\#k_2$ мы сможем за $k_2 - k_1$ дней. Однако менеджер $\#k_2$ отправит план на доработку, и нам придётся сначала пройти заново весь путь с получением подписи менеджера $\#k_1$, а затем потратить еще $k_2 - k_1$ дней, чтобы получить подпись $\#k_2$.

Итого:

$2 \cdot k_1 + (k_2 - k_1) + 2 \cdot k_1 + (k_2 - k_1) = 2 \cdot (k_1 + k_2)$.

Дальнейший наш путь лежит к менеджеру $\#k_3$.

Рассуждая аналогично, получим:

$2 \cdot (k_1 + k_2) + (k_3 - k_2) + 2 \cdot (k_1 + k_2) + (k_3 - k_2) = 2 \cdot (2 \cdot k_1 + k_2 + k_3) = 4 \cdot k_1 + 2 \cdot k_2 + 2 \cdot k_3$.

Для менеджера $\#k_4$:

$4 \cdot k_1 + 2 \cdot k_2 + 2 \cdot k_3 + (k_4 - k_3) + 4 \cdot k_1 + 2 \cdot k_2 + 2 \cdot k_3 + (k_4 - k_3) = 8 \cdot k_1 + 4 \cdot k_2 + 2 \cdot k_3 + 2 \cdot k_4$

Продолжая, можно прийти к формуле со степенями двойки. А можно просто воспользоваться формулой в «изначальном виде» и подсчитать значение для очередного менеджера, зная значение для предыдущего. Разумеется, если $k_m < n$, к ответу нужно будет не забыть прибавить $n - k_m$.

Разбор задачи «Объединяя усилия»

Применим в этой задаче «классическую» технику. Будем описывать границу отрезка времени парой значений: собственно временем (соответствующим этой границе) и типом границы (началом или концом отрезка эта граница является).

Поместим эти границы в список и отсортируем их по времени, при равном времени — по типу границы. После этого сформируем структуру отрезков, для каждого элемента которой можно сказать, сколькими исследованиями «покрыт» тот или иной отрезок. Для этого установим счётчик количества исследований равным нулю и пройдём от самой первой границы до последней, прибавляя единицу к счётчику каждый раз, когда проходим границу типа «начало отрезка» и отнимая единицу из счётчика каждый раз, когда проходим границу типа «конец отрезка». Когда значение счётчика меняется, формируем очередной отрезок.

Если бы было задано конкретное значение k, для которого требуется определить суммарное количество времени, в которое нужно провести дополнительные исследования, можно было бы просто пройти по полученной структуре и выполнить непосредственный подсчёт. Однако может быть предложено достаточно много запросов, поэтому такой подход окажется слишком медленным.

Впрочем, понятно, что если для некоторого k какой-то из отрезков времени исследован достаточно, то и для всех меньших k этот отрезок также исследован достаточно. Поэтому можно подсчитать (за один проход), сколько времени «закрыто» в точности одним, двумя и т.д. исследованиями. А затем, начав с самого большого количества, просуммировать это время (и сохранить результаты в массиве или списке). Теперь, когда будут поступать соответствующие запросы, достаточно вычесть из полного времени, которое интересует заказчиков, то время, которое уже «закрыто» подходящим количеством исследований.

Разбор задачи «Бизнес-история»

Вероятно, это одна из самых простых задач комплекта.

Предположим, что исходно компания располагала нулевым количеством собственных средств, и будем считать ноль текущим значением. Будем считать нарастающим итогом сумму a_i (в том порядке, в котором эти элементы содержатся во входных данных). Если сумма становится меньше текущего значения, сделаем текущим значением эту сумму.

Действительно, если компания «уходит в минус», на её счёте изначально должны были быть деньги (иначе нечего было бы тратить).

Таким образом, пройдя по всему массиву и выполнив подсчёт суммы, найдём минимальное значение, которое она может принимать. Заметим, что если минимальное значение будет положительным, текущее значение (исходно равное нулю) не изменится (что также логично: исходно компания не может иметь отрицательное количество денег на счёте).

Разбор задачи «В движении»

Задачу можно решить симуляцией процесса.

Заведём переменную, хранящую «текущее время», счётчик трамваев, а также переменную, указывающую, на какой остановке находится Стив. Будем двигаться по последовательностям трамваев следующим образом:

- если Стив находится на остановке B, пропустим все трамваи, которые придут на остановку B раньше текущего времени. После этого вычислим время $probTime$, в которое он окажется на остановке A, если уедет на первом трамвае, приходящем в текущее (или более позднее) время.

Если окажется, что Стив приедет в этом случае на остановку A строго раньше, чем на эту остановку приедет трамвай с незнакомцем, меняем текущее время на $probTime + 1$ (единицу добавляем, чтобы избежать проблем с возможной пересадкой обратно), увеличиваем счётчик трамваев на единицу и меняем значение переменной, указывающей на остановку (теперь это остановка A).

Если же выяснится, что Стив не сможет попасть на остановку A при указанных условиях, можно прерывать цикл: Стив больше не сядет ни на какой трамвай и будет ожидать незнакомца на остановке B.

- если Стив находится на остановке A, пропустим все трамваи, которые придут на остановку A раньше текущего времени (важно не забыть вычесть из времени прихода трамваев величину d). После этого вычислим время $probtime$, в которое Стив может оказаться на остановке B, если уедет на первом трамвае, приходящем в текущее или более позднее время.

  Если окажется, что Стив приедет на остановку B строго раньше, чем на остановку A прибудет трамвай с незнакомцем, меняем текущее время на $probTime + 1$ (так же, как и в предыдущем случае), увеличиваем счётчик трамваев на единицу и меняем значение переменной, указывающей на остановку (теперь уже B).

  Если же выяснится, что Стив не сможет при этих условиях попасть на остановку B, то прерываем цикл.

По окончании цикла останется лишь вывести содержимое переменной, указывающей на остановку, и счётчика трамваев.

Этот подход можно несколько оптимизировать, если не перебирать все трамваи подряд в поисках первого, который может увезти Стива на другую остановку, а воспользоваться бинарным поиском (который можно применить, поскольку входные данные упорядочены).

Разбор задачи «И ещё несколько остановок»

В этой задаче нужно было аккуратно выполнить симуляцию процесса.

Будем хранить в переменной $time$ время, прошедшее с момента старта трамвая. Когда трамвай отъезжает от очередного перекрёстка, будем прибавлять к этой переменной соответствующее значение a_i — время, за которое трамвай достигнет следующего перекрёстка. После этого потребуется определить, сможет ли трамвай проехать перекрёсток сразу или же ему придётся провести какое-то время в ожидании зелёного сигнала.

Поскольку нам известно, за сколько секунд до старта трамвая на светофоре загорался зелёный свет, мы можем посчитать, сколько прошло полных циклов «зелёный свет + красный свет» до этого момента и сколько секунд «неполного цикла» осталось: $rem = (time + d[i]) \mod (g[i] + r[i])$.

Если $rem < g[i]$, трамвай проедет перекрёсток сразу; в момент $g[i]$ включается красный свет, и если $rem \geqslant g[i]$, то время ожидания на светофоре вычисляется как $r[i] + g[i] - rem$; это время и нужно добавить к $time$.

По завершении обработки последнего перекрёстка в $time$ будет храниться ответ.

Разбор задачи «Дельта»

В первую очередь необходимо минимизировать затраченное время, а при фиксированном (минимальном) затраченном времени необходимо минимизировать количество затраченных денег.

Вычислим, сколько раз потребуется увеличить скорость для сервоприводов:

$s = \lceil (u - v)/dv \rceil$

и для рычагов:

$r = \lceil (u - w)/dw \rceil$

В обоих случаях округление вверх; если получаем отрицательную величину, то соответствующее значение считаем равным нулю.

Таким образом, получаем, что одной научной группе потребуется выполнить s шагов, а другой — r шагов.

Для определённости предположим, что $s < r$. Это значит, что нужный результат можно достичь самое меньшее за r лет, если поддерживать для этой научной группы «учетверённое» финансирование, позволяющее ей выполнять один шаг каждый год. Поскольку первой научной группе нужно сделать меньшее количество шагов, нужно выбрать схему финансирования такой, чтобы результат был достигнут также за r лет.

Если финансировать группу «обычным» образом (по f каждый год), то результат будет достигнут через $2 \cdot s$ лет. Конечно, может оказаться так, что $2 \cdot s \leqslant r$; в этом случае просто придётся поддерживать группу r лет, ежегодно выплачивая по f (по условию финансирование не прекращается до запуска в производство).

Если же $2 \cdot s > r$, то, чтобы обе группы закончили работу в одни и те же сроки, нужно будет часть времени финансировать первую группу в учетверённом объёме, а часть времени — в обычном. Разложим s на два слагаемых: $s = a + b$, при этом a шагов будут сделаны с финансированием в учетверённом объёме, а b шагов с финансированием в обычном объёме. Тогда все шаги будут выполнены за $a + 2 \cdot b$ лет. Поскольку нужно добиться одновременного окончания работ, $r = a + 2 \cdot b$.

Таким образом, нам нужно решить систему уравнений

$$\begin{cases} s = a + b \\ r = a + 2 \cdot b \end{cases}$$

Легко видеть, что $b = r - s$, $a = s - b$; откуда получаем необходимый объём финансирования $4 \cdot a \cdot f + 2 \cdot b \cdot f + 4 \cdot r \cdot g$.

Случай $s > r$ рассматривается аналогично; если же $s = r$, то обе группы следует финансировать в учетверённом объёме всё время.

Разбор задачи «Лишние вопросы»

Будем использовать для хранения данных о вопросах-предпосылках список списков (или иную двумерную структуру). Разумеется, в этой структуре вопросы следует упорядочить по возрастанию их номеров — т.е. расположить их в том порядке, в котором Стив будет их задавать.

Также заведём массив, содержащий «ответы по мнению Стива» (в принципе для этих целей можно использовать и массив, исходно хранящий фактические ответы на вопросы).

Изначально положим значение счётчика заданных вопросов равным нулю и будем обрабатывать все вопросы последовательно.

Если у вопроса нет вопросов-предпосылок, скопируем в массив «ответов по мнению Стива» фактический ответ менеджеров, а также зафиксируем, что этот вопрос Стив задал (это можно сделать, либо сохраняя номер вопроса в списке / динамическом массиве, либо выставляя соответствующее значение в булевом массиве, содержащем данные обо всех вопросах).

Если у вопроса есть вопросы-предпосылки, посчитаем количество ответов `Y` и `N` на эти вопросы. Если количество тех или иных ответов окажется больше половины, запишем именно этот ответ в массив «ответов по мнению Стива»; в противном случае скопируем ответ из массива фактических ответов, а также зафиксируем, что и этот вопрос Стив задал.

В случае, если Стив задал вопрос, увеличим на единицу значение счётчика.

По завершении процесса обработки нам останется вывести значение счётчика, а также номера вопросов, которые в процессе были помечены (тем или иным образом) как заданные.

Разбор задачи «Продолжение следует»

Задача решалась достаточно простой формулой.

Действительно, если $z = a$, то количество обновлений определяется разностью $y - b$.

Если же $z > a$, то сначала будет нужно выполнить все обновления версии a (их будет $p - a$), затем выполнить все обновления всех версий вплоть до версии z (их будет $(z - a - 1) \cdot (p + 1)$), и, наконец, выполнить все обновления с версии $z.0$ до версии $z.y$ (их будет $y + 1$). Просуммировав все перечисленные слагаемые, получим ответ.