Разбор задач

XVI командного чемпионата по информатике, программированию и математике среди школьников Самарской области

(8 февраля 2015 г.)

Задача А. Календарь праздников

При чтении данных сформируем, во-первых, список фиксированных праздников, содержащий пары «номер — дата», а также построим граф (в виде списка списков). В графе для каждого праздника будут храниться его непосредственные «последователи».

Отсортируем фиксированные праздники по дате (это, вообще говоря, можно сделать и позже).

Для каждого из фиксированных праздников подсчитаем, сколько назначаемых праздников должно состояться после него. Кроме того, выполним такой же подсчёт и для праздника ватрушкопечения. Это несложно сделать с помощью серии обходов в ширину, которые следует запустить от соответствующих вершин графа.

Заметим, что, поскольку граф фактически является деревом, то при обходе в ширину нет необходимости помечать пройденные вершины: повторное попадание в них исключено. Однако при обходе в ширину можно выяснить, к какому из фиксированных праздников «привязан» праздник ватрушкопечения.

Теперь воспользуемся следующей «жадной» стратегией. Рассмотрим сначала фиксированный праздник с наиболее поздней датой. Пусть после этого праздника должно состояться f_1 назначаемых праздников, причём среди них нет праздника ватрушкопечения. Тогда, чтобы «сохранить» наиболее удалённые даты для возможного размещения праздника ватрушкопечения, поставим все назначаемые праздники в первые f_1 дней после этого фиксированного праздника (мы всегда можем это сделать, поскольку нам гарантировали корректность исходного календаря). После этого у нас останется некоторое (возможно, нулевое) количество дней до конца года.

Запомним этот отрезок — пусть это даты с w_1 до n .

Теперь рассмотрим предпоследний по дате фиксированный праздник и проделаем с ним то же самое (мы по-прежнему полагаем, что праздник ватрушкопечения не входит в список праздников, которые должны состояться после рассматриваемого).

Точно также все f_2 назначаемых праздников свяжем с первыми днями, следующими за этим праздником. Если необходимо, придётся «захватить» какую-то часть отрезка $[w_1..n]$, или же, напротив, у нас появится еще один «свободный отрезок» $[w_1..d_1-1]$ (где d_1 — дата самого позднего фиксированного праздника).

Мы будем продолжать так действовать, пока очередной фиксированный праздник не будет содержать в своём списке назначаемых праздник ватрушкопечения. В этом случае мы поступим иначе — отсчитаем по свободным отрезкам количество дней, которые необходимы, чтобы назначить праздники после праздника ватрушкопечения. Первый же свободный день и будет ответом.

Задача B. Сложный процент

Несложно заметить, что изменение величины разрешённого увеличения производства в процентах для некоторого предприятия, производящего m_i сметаны, далеко не всегда влечёт за собой реальное увеличение производства в процентах. Действительно, для этого нужно, чтобы выполнялось равенство $m_i \cdot p/100 = z_i \cdot q$, где z_i — некоторое целое положительное число. В этом случае предприятие сможет приобрести z_i производственных линий. Далее, пока будет справедливо неравенство $z_i \cdot q < m_i \cdot p/100 < (z_i + 1) \cdot q$, предприятие реально не сможет нарастить объём производства.

Конечно, для каждого значения m_i «диапазон изменения процента без изменения производства» будет свой. Однако это позволяет нам каждый раз рассматривать «эффективный» процент, т.е. перебирать величину процента с шагом, возможно, существенно превышающим 1.

Для организации такого перебора можно воспользоваться следующими вспомогательными структурами данных: множеством (*set* в C++, *TreeSet* в Java) пар «величина процента — номер предприятия» и массивом для хранения последнего рассмотренного значения z_i для каждого предприятия. Множество должно быть упорядочено по величине процента.

Каждый раз, когда мы будем извлекать очередную пару из множества, следует сделать ещё два действия: во-первых, извлечь из множества все пары с таким же значением процента, как у извлечённой (для этих предприятий производство также возрастёт), во-вторых, для всех номеров предприятий из извлечённых пар сформировать новые пары (с увеличенным на единицу значением z_i) и поместить их во множество.

Когда для некоторого p разность между величиной s и дополнительным объёмом производства достигнет наименьшего по абсолютной величине значения, останется только определить, на сколько ещё можно увеличить p. Это легко сделать, взяв очередную пару из множества и уменьшив процент в ней на единицу.

Вообще говоря, даже инициализация элементов массива z нулями позволяет уложиться в ограничения по времени с запасом (на C++ и Java). Однако можно (особенно в специфическом случае вида «s близко к 10^5, q близко к 1») ускорить перебор, выполнив оценку процента снизу. Для этого достаточно вычислить, на какой процент (с округлением вниз до целого) следовало бы увеличить производство, если бы не было требования приобретать производственные линии фиксированного объёма производства. Понятно, что даже в «идеальном» случае ($GCD(m_i, i = 1, 2, \ldots, n) \cdot p/100 = Z \cdot q$) мы не получим значения, большего, чем нужно.

Задача C. Отыщи всему начало

Сформируем ориентированный граф g, в котором каждая вершина (продукт) связана ребром с теми продуктами, в производстве которых она непосредственно используется. Определим для каждой вершины v_i максимально возможную длину цепочки, которая приводит к вершине, не имеющей потомков. Очевидно, что

$$maxChain[v_i] = max\{maxChain[u], u \in g[v_i]\} \quad ,$$

где $g[v]$ — список всех потомков вершины v .

Теперь останется найти максимальное значение среди всех $maxChain[v_i]$, соблюдая условие, ограничивающее сверху цену организации производства продукта величиной s .

Задача D. Проблема выбора

В этой задаче требовалось найти центр неориентированного графа. Для этого потребуется сначала получить матрицу расстояний между всеми вершинами графа. Если мы располагаем такой матрицей $distance[v][u]$, то всё, что нам требуется — отыскать в каждой строке этой матрицы максимум (определить расстояние от вершины v до наиболее удалённой от неё вершины u), а затем найти среди этих максимумов минимум:

$$ans = min\{max\{distance[v][u], u \in g\}, v \in g\},$$

где перебор по v и по u — перебор по всем вершинам графа.

Та вершина ans , которой будет соответствовать этот минимум, и будет ответом.

Получить матрицу расстояний (в данных ограничениях) можно разными способами: можно, например, воспользоваться алгоритмом Флойда (отыскивающим кратчайшие расстояния между всеми вершинами графа за $O(n^3)$, где n — количество вершин в графе) или же алгоритмом Дейкстры (в «классической реализации» отыскивающим кратчайшие расстояния от заданной вершины до всех остальных вершин графа за $O(n^2+m)$, где n — количество вершин графа, а m — количество рёбер), применив его n раз.

Задача E. Пастбище

Чтобы обеспечить корове ежедневное пропитание, трава должна расти как минимум со скоростью $w = m/x$. Поэтому, если скорость роста травы без внесения удобрений $k_i \geqslant w$, то стоимость решения — это просто стоимость семечка g_i . Если же $k_i < w$, то придётся вносить удобрения.

Поскольку нам недостаёт $w - k_i$ единиц скорости, это значит, что требуется как раз столько единиц удобрения на единицу площади, т.е. стоимость решения составит $g_i + (w - k_i) \cdot u_i$. Единственное, что надо проконтролировать — это то, что $(w - k_i) \leqslant v_i$.

Чтобы избежать операций деления, можно сравнивать m с $k_i \cdot x$, $m - k_i \cdot x$ с $v_i \cdot x$, а вместо стоимости решения для единицы площади вычислять величину $g_i \cdot x + (m - k_i \cdot x) \cdot u_i$. Ограничения позволяют рассчитывать, что ни одна из величин в процессе вычисления не превысит 10^8 .

Задача F. Хорошая замена

Будем решать задачу методом динамического программирования. Обозначим как $dp[i]$ максимальное число подстрок, не являющихся палиндромами, на которое можно разбить префикс строки длины i. Тогда $dp[i]=1+max\{dp[j],\ j<i\}$, где максимум берется только по тем j, для которых $S[(j+1)..i]$ не является палиндромом.

Проведём некоторые оптимизации. Для начала пропустим все случаи, когда выделяемая подстрока, проверяемая на «непалиндромность», состоит из одинаковых букв. Кроме того, будем рассматривать j в порядке убывания и при этом останавливаться, если уже найденное значение не хуже, чем любое получаемое для меньших j значение. Наконец, для префикса вида $ababa...ba$ для позиций a будем сразу помечать, что искомого разбиения не существует.

Можно показать, что суммарное число итераций уменьшения j после указанных оптимизаций есть $O(|S|)$, где $|S|$ — длина строки. Доказательство этого факта предлагаем читателю выполнить самостоятельно (оно длинное и скучное), основываясь на следующих идеях:

1. Из $dp[j]$ и $dp[j+1]$ хотя бы одно является максимумом среди всех $dp[k]$ для $k\leqslant j+1$.

2. Среди двух проверяемых подстрок $S[j..i]$ и $S[(j+1)..i]$ хотя бы одна не является палиндромом.

3. Утверждения (1) и (2) не доказывают оценку для числа итераций только в случае, когда рассматриваемый префикс имеет вид $...abababa$.

4. Если префикс не имеет вид $abababababa$, то вдоль участка $...abababa$ значения $dp[j]$ для чётных и нечётных позиций возрастают и мало отличаются друг от друга (а значит, сработает проверка на то, что ответ уже не может быть лучше).

Задача G. Всё идёт по плану

Предположим, что мы построили некоторый план распределения грантов $t_1, t_2, \ldots, t_n$, который позволяет достичь (как минимум) необходимого уровня для каждого из технологических направлений.

Покажем, что любой такой план может быть преобразован к плану, для которого выполняются два условия:

1. Любое технологическое направление, которое в итоге должно быть развито больше, в любой момент времени развито не меньше. Иными словами, для любых q $(1\leqslant q\leqslant n)$, i и j $(i\neq j,\ 1\leqslant i,\ j\leqslant k)$ верно, что если $b_i>b_j$, то $a_i^{(q)}\geqslant a_j^{(q)}$, где $a_i^{(q)}$ и $a_j^{(q)}$ — уровни развития технологических направлений i и j после выделения q грантов.

2. Любое технологическое направление продвигается первым из грантов, который может продвинуть это направление и не может продвинуть направление, которое в итоге должно быть развито больше. В терминах предыдущего пункта: грант $(q+1)$ выделяется

некоторому направлению j только в том случае, если $a_j^{(q)} \leqslant g_{q+1}$ и при этом не существует направления i, такого, что $b_i > b_j$ и $a_i^{(q)} \leqslant g_{q+1}$.

Допустим, что построенный нами план нарушает пункт 1. Найдём первую такую ситуацию (наименьшее значение q) для некоторой пары направлений i и j, что $b_i > b_j$ и при этом $a_i^{(q)} < a_j^{(q)}$.

Найдём после этой ситуации первую ситуацию (такое значение p), в которой направления развиты одинаково $\left(a_i^{(p)} = a_j^{(p)}\right)$.

Поменяем местами распределение грантов: грант q выделим направлению i, а грант p выделим направлению j. Несложно показать, что за конечное число шагов мы преобразуем план так, чтобы пункт 1 выполнялся.

Теперь допустим, что построенный нами план нарушает пункт 2. Найдём ситуацию, в которой некоторое направление j не было развито первым доступным после проверок «более важных» направлений грантом.

Если этот грант (пусть это будет грант q) не был потрачен на улучшение какого-либо другого технологического направления (был выделен на направление с $a_i^{(q-1)} > g_q$), мы просто можем его использовать вместо следующего использованного на улучшение направления j гранта (пусть это будет грант p).

Если же грант q был потрачен на улучшение «менее важного» направления l (с $b_l < b_j$), то передадим грант p этому «менее важному» направлению, а грант q направим на улучшение направления j.

Предполагая, что пункт 1 уже выполнен, заметим, что план по-прежнему корректен (по текущему направлению ничего не изменилось; по «менее важному» направлению гранта p достаточно согласно пункту 1, а все промежуточные гранты продвигают направления с меньшими итоговыми уровнями развития).

Если после преобразования нарушен пункт 1, проведём описанную выше процедуру обмена грантов (она не будет влиять на изменяемое направление и вообще на любое направление важнее «менее важного»).

Из вышеописанного получается, что задача может быть решена следующим алгоритмом. Будем рассматривать все гранты последовательно. Из множества всех направлений, которые могут быть продвинуты текущим грантом (и ещё не достигли требуемого уровня развития), будем выбирать то направление i, которое в итоге должно иметь наибольший уровень развития (b_i).

Для эффективной реализации этого алгоритма необходима структура данных, позволяющая быстро отыскивать подходящее технологическое направление, например, *set* (в C++) или *TreeSet* (в Java). Кроме того, удобно предварительно упорядочить направления по неубыванию итогового уровня развития. Оценка сложности решения — $O(n \cdot \log n)$ времени и $O(n)$ памяти.

Задача H. Важные показатели

Будем действовать следующим образом. Пойдём по числу справа налево (от младших разрядов к старшим). Поскольку мы хотим, чтобы разница между изменённым числом и исходным была минимальной, все цифры, которые можно исправить на меньшие, следует «уменьшать». А как только мы встретим цифру, которую можно исправить на б**о**льшую, мы исправляем её и завершаем процесс.

Если мы дошли до конца числа и так и не встретили цифру, которую можно исправить на большую, значит, увеличить число не получится, и следует вывести -1 .

Разумеется, число удобнее хранить в строке или массиве (символов или целых чисел).

Задача I. Спор

Рассмотрим подстроку из m элементов и определим (для неё), во-первых, может ли Силантий выиграть спор и, во-вторых, какое минимальное количество раз ему придётся солгать, чтобы выиграть (если, разумеется, это возможно).

Чтобы ответить на «принципиальный вопрос2 о возможности выигрыша, будем действовать «жадно» и полагать, что Силантий будет говорить, что знает об исследовании, в каждом случае, когда у него не попросят подтверждающую его слова ссылку. Конечно, в тех случаях, когда Силантий действительно знает об исследовании, он обязательно об этом скажет (и сможет подтвердить свои слова).

Если при таком подходе окажется, что Силантий может переспорить Фалалея, попробуем минимизировать количество безосновательных утверждений Силантия. Понятно, что Силантий для победы в споре должен набрать $\lfloor m/2 \rfloor + 1$ очков. Если при «жадном» подходе он набирает $p > (\lfloor m/2 \rfloor + 1)$ очков, и при этом q из них $(q \leqslant p)$ получены им честно, то ему будет достаточно сказать неправду $\lfloor m/2 \rfloor + 1 - q$ раз (если $\lfloor m/2 \rfloor + 1 > q$) или же вообще не нужно будет говорить неправду (0 раз, если $\lfloor m/2 \rfloor + 1 \leqslant q$).

Выполним такой подсчёт для каждой подстроки длины m и определим наилучший (с наименьшим количеством безосновательных утверждений) результат.

Задача J. Выигрыш

Рассмотрим некоторый кратчайший путь от Волка до Зайца (любой, не обязательно соблюдающий приоритеты при выборе направления движения Волка). Ходы Зайца будут состоять в том, чтобы бежать навстречу Волку по этому кратчайшему пути до тех пор, пока Волк не окажется перед ходом Зайца на расстоянии двух или одной клетки. Если расстояние составит две клетки, ход Зайца будет состоять в том, чтобы остаться на месте и позволить Волку подойти на соседнюю клетку (т.е. сократить расстояние до одной клетки).

Рассмотрим теперь клетку с Волком как если бы она была запрещённой клеткой лабиринта. Заметим, что если Заяц может (в предположении, что вместо Волка запрещённая клетка) из текущей клетки достичь либо некоторую другую клетку, соседнюю с Волком, либо клетку, входящую в некоторый цикл в лабиринте, то он может бегать сколь угодно долго —

Волк просто будет бегать за ним следом. В противном случае Заяц должен будет убегать в самую дальнюю от своего текущего положения клетку.

Под циклом в лабиринте мы понимаем последовательность клеток, в которой из каждой клетки (этой последовательности) можно непосредственно перейти в следующую клетку, а из последней клетки — в первую.

Для доказательства того, что описанный выше алгоритм является оптимальным для Зайца, заметим следующее: если Заяц не может «оббежать» Волка в его положении в момент встречи, и в области с Зайцем нет цикла, то область, которая находится «за» положением Волка, для Зайца в принципе недостижима.

Обозначим клетку, в которой окажется Волк в момент встречи, символом M. Любой путь от начального положения Волка до области с Зайцем проходит через клетку M, а значит, проходит и любой кратчайший. Отсюда, в частности, следует, что, пока Заяц бежал навстречу Волку, Волк бежал к клетке M по кратчайшему пути. Заяц не успел добраться до клетки M (а он тоже бежал по кратчайшему пути к ней!), и потому не мог добраться до M ни при каких условиях (точнее, мог, но следующим ходом был бы пойман — это не имеет значения).

Если в области, которую изолирует клетка M, нет ни одного цикла, это значит, что любой путь между клетками этой области единственный и, соответственно, дальнейший маршрут Волка от клетки M до момента поимки Зайца будет кратчайшим, т.е. кратчайшим будет и весь маршрут волка от начальной точки до момента поимки Зайца.

Следовательно, число ходов, которые может сделать Заяц, определяется числом ходов по кратчайшему пути Волка до места встречи; после того, как Заяц бежал навстречу Волку, он убегал в самую дальнюю клетку от M, что делает его поведение оптимальным.

Задача K. Архитектурное решение

Рассмотрим фрагмент улицы, на каждом конце которого расположен памятник архитектуры, и никаких других (кроме этих двух) памятников архитектуры в этом фрагменте нет. Понятно, что гостиницу максимальной вместимости в этом фрагменте можно построить следующим образом: снести все «обычные» здания между памятниками архитектуры и заменить их зданием с максимально возможной высотой (равной наибольшей из высот памятников архитектуры).

Если фрагмент улицы ограничен с одной из сторон концом улицы, то формально можно считать, что на этом конце улицы расположен памятник архитектуры нулевой длины и нулевой (или единичной) высоты, и определить максимально возможную вместимость гостиницы так, как описано выше.

Теперь остается рассмотреть попарно ближайшие друг к другу памятники архитектуры и, выполнив для каждой пары подсчёт вместимости гостиницы между ними, найти максимальную величину.

Задача L. Глобальное мышление

Заведём целочисленный массив из трёх элементов $points$, элемент $points[k]$ будет соответствовать количеству очков, которое набирает предложение k $(k=1,2,3)$. Вероятно, наиболее коротким объяснением будет псевдокод:

для строки i $(i=1,2,\ldots,n)$:

для элемента $m_j^{(i)}$ $(m_j^{(i)}\in\{1,2,3\},\ j=2,1,0)$:

$points[m_j^{(i)}]=j$

Здесь $m_j^{(i)}$ — это собственно номер предложения, которое министр i поставил на позицию j.

Во внутреннем цикле применён следующий «трюк»: позиции элементов в строке (предложений очередного министра) занумерованы от 2 до 0 (в обратном порядке). Таким образом, позиция предложения в строке равна количеству очков, начисляемому за эту позицию. Разумеется, можно обойтись и без этого «трюка», код станет чуть длиннее.