Разбор задачи «Урок физкультуры»

Первое замечание, существенно упрощающее понимание решение данной задачи, состоит в том, что нас интересует только соотношение сил остальных учеников с силой Коли, но не соотношение их сил между собой. А именно, заменим все силы учеников, которые слабее Коли, на 0, силу Коли на 1, а силы учеников сильнее Коли на 2. Если какие-то ученики равны Коле по силе, то в случае, когда они стоят левее его, их сила станет равной 0, а если правее – то 2. Нетрудно проверить, что при таком преобразовании ответ в задаче не изменится, и более того, не изменится с точностью до перестановки двоек и нулей результат после каждого прохода Ивана Петровича.

Предположим теперь, что Коля вообще не хочет приседать. Что тогда с ним произойдёт? Пока слева от Коли будет хоть одна двойка, на каждой итерации он будет двигаться на одно место влево, меняясь с одной из таких двоек. А как только слева от Коли останутся одни нули, то он дальше сможет двигаться только вправо. Более того, можно понять, на сколько именно позиций он будет сдвигаться вправо. Разобьём всех учеников правее Коли на группы – группа нулей (возможно, пустая), потом группа двоек, потом опять группа нулей, и так далее. Тогда на каждом проходе учителя ровно одна двойка из каждой группы будет меняться со всеми нулями справа от неё, и тем самым переходить в следующую группу. Одна двойка переходит из первой группы двоек во вторую, одна из второй в третью, одна из третьей в четвёртую, и так далее – в результате просто фактически одна двойка переходит из первой группы в конец. Тем самым, если Коля будет двигаться вправо на x-ом шаге, то он поменяется местами со всеми нулями, стоящими левее x-ой двойки (ведь первые x-1 двойка уйдут в конец, и Коля будет меняться с нулями, пока не упрётся в x-ую двойку).

Отсюда становится ясной оптимальная стратегия Коли – необходимо в последний раз двигаться вправо как можно раньше, т.е. приседать на как можно большем числе последних раундов. Объединяя это соображение с вышеизложенными, решение целиком таково: необходимо не приседать, пока слева есть двойки, а затем присесть либо все оставшиеся разы, либо, если сил на это не хватает, то последние k раз.

Для определения окончательной позиции Коли также можно воспользоваться вышеизложенными соображениями. А именно, левее его не будет ни одной двойки, а нулей будет столько, сколько их в исходной позиции до x-ой двойки, где x – количество шагов, на которых Коля не приседает (эти соображения относятся к нулям и двойкам правее Коли, к ним ещё надо прибавить количество нулей левее).

Такое решение работает за время O(n) и легко укладывается в отведённое время.

Разбор задачи «Электрички на перегонах не меняют»

Хранение входных данных

Построим граф, в котором все вершины будут соответствовать исходным станциям, а ребра – перегонам. Так как по условию в графе не может быть циклов, граф является лесом, и состоит из нескольких компонент связности, каждая из которых – дерево. Для хранения графа можно использовать список. При использовании матрицы смежности из-за большого количества вершин графа возникает переполнение по памяти.

Для хранения маршрутов электричек необходимо использовать списки, либо динамически выделять память под каждый маршрут. Применение двумерного массива недопустимо из-за того, что длина отдельного маршрута и общее количество электричек может быть достаточно большим и это приведет к переполнению по памяти.

Идея решения

В этой задаче существует несколько принципиально разных решений. В каждом решении мы последовательно обрабатываем вершины, ребра или пути электричек при этом мы должны каждой вершине присвоить некоторое уникальное число, которое обозначает тарифный номер станции. Наша цель – ориентировать непротиворечивым образом электрички так, чтобы вдоль каждого маршрута электрички тарифные номера строго возрастали или строго убывали. Такая «ориентация» маршрутов автоматически задает ориентацию ребер, входящих в маршрут.

В процессе решения задачи мы должны учитывать два типа ограничений: ограничения в вершинах и ограничения на ребрах. Ограничение в вершинах означает, что все маршруты, проходящие через одну и ту же вершину должны иметь один и тот же тарифный номер в этой вершины. Ограничения на ребрах означает, что все маршруты, проходящие через одно и то же ребро должны иметь ориентацию, согласованную с ориентацией ребер. Таким образом, задача сводится к построению корректной ориентации ребер графа. Если граф ориентировать не удается (возникает цикл), необходимо вывести ответ «NO». В противном случае граф маршрутов будет ациклическим (исходный граф – дерево), поэтому при помощи топологической сортировки можно восстановить тарифные номера всех станций, через которые проходят маршруты электричек. Для оставшихся вершин тарифные номера можно указать произвольным образом.

Реализация алгоритма

Существует несколько видов «квадратичных решений», с асимптотиками O(kl), O(nl), $O(n^2)$, $O(k^2)$, где n – количество вершин графа, k – количество маршрутов, l – общая длина всем маршрутов электричек (количество ребер). Каждое из этих решений может быть доведено до правильного решения, работающего за линейное время O(n+l) при помощи несложных технических приемов. Все квадратичные решения набирают не более 70 баллов.

Решение с асимптотикой O(kl) обрабатывает маршруты электричек в некотором порядке и для каждого маршрута устанавливает такую тарифную нумерацию (нумерацию), чтобы не возникало противоречий в вершинах и на ребрах. Если мы будем рассматривать пути в хаотичном порядке, тогда получим некоторую эвристику, которая набирает от 20 до 40 баллов. Чтобы это решение стало правильным, мы должны сначала обрабатывать маршруты, для которых можно согласовать ограничения на ребрах, затем маршруты, для которых можно согласовывать ограничения в вершинах. Если маршруты, для которых необходимо согласовывать ограничения отсутствуют, можно перейти к рассмотрению произвольного маршрута. Фактически в этом решении мы нумеруем первый маршрут, затем все маршруты, у которых есть общие ребра с данным маршрутом, затем все маршруты, у которых есть общие вершины и так далее. Для поиска подходящего маршрута потребуется просмотреть ребра всех маршрутов, то есть потребуется O(l) времени, а так как всего k маршрутов общая оценка работы данного алгоритма O(kl). Преимущество этого алгоритма в том, что он не использует топологической сортировки, так как строит нумерацию непосредственно в ходе работы, и совершенно не использует объект «ребро», а вместо этого оперирует с объектом «пара вершин». Таким образом, мы можем в этом решении полностью отказаться от хранения графа.

В предыдущем решении мы рассматривали маршруты, а затем ориентировали ребра. В этом решение будем выполнять то же самое в обратном порядке: сначала ориентируем ребра, а после этого ориентируем все маршруты, содержащие данное ребра. Заведем очередь и будем в нее добавлять все ребра графа, на которых уже задана нумерация. Извлечем очередное ребро из очереди и рассмотрим все маршруты, которые содержат данное ребро. Ориентируем каждое из этих ребер в соответствие с ориентацией маршрута и добавим их в очередь. Перейдем к рассмотрению следующего ребра. На

каждом шаге алгоритма количество ориентированных ребер увеличивается на единицу. Таким образом, нам потребуется не более чем O(n) итераций. На каждой итерации нам придется по заданному ребру определять множество маршрутов, которые проходят через заданное ребро. Для этого нам необходимо пробежаться по всем ребрам маршрутов, то есть потребуется O(l) операций. Если для каждого ребра заранее завести список маршрутов, проходящих через ребро, время выполнения этой операции будет O(1). Кроме того, нам необходимо для каждого ребра маршрута определять, какое это будет ребро в исходном графе. В зависимости от реализации время выполнения этой операции будет O(n), O(Log(n)) или O(1). Таким образом, суммарное время работы данного алгоритма может быть O(nl), $O(n^2)$, O((n+l)Log(n)) или O(n+l).

Третий тип решения заключается в том, что мы строим вспомогательный граф, вершинами которого являются маршруты. Проводим ребро в этом графе в том случае, если указанная пара маршрутов содержит общее ребро в исходном графе. Затем разбиваем граф на несколько компонент связности и восстанавливаем ориентацию исходных ребер графа. Для поиска компонент связности потребуется порядка $O(k^2)$ ребер. Это решение наиболее сложное для реализации. Для того чтобы получить линейное время исполнения, необходимо поиск компонент связности в новом графе объединять с одновременной ориентацией ребер в исходном графе.

Во всех решениях кроме первого, на заключительном этапе необходимо применять топологическую сортировку. В зависимости от реализации алгоритма для сортировки потребуется O(n) O(n Log(n)) или $O(n^2)$ операций.

Разбор задачи «Ударим мостом по бездорожью»

Если город *A* находится правее города *B*, то обменяем их местами. После этого добавим города *A* и *B* к множеству вершин ломаной. При этом, если точка не совпадает с вершинами ломаной, то содержащее ее ребро разбивается на два. В дальнейшем будем считать, что *A* и *B* — индексы соответствующих вершин ломаной.

Отметим, что при малых смещениях моста вверх или вниз между парой склонов его длина изменяется линейно, при этом длина пути из A в B так же изменяется линейно (см. рис. 1). Следовательно, зависимость дины пути от высоты моста является кусочно-линейной функцией, и, соответственно, оптимальный мост будет либо иметь длину *L*, либо иметь общие точки с вершинами ломаной.

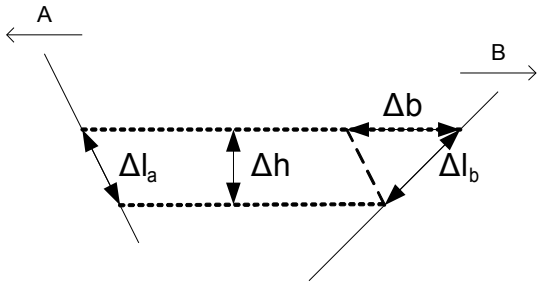

Рис 1. Изменение длины кратчайшего пути

Также заметим, что мост иногда выгодно не только поднимать, но и опускать, как, например, в случае на рис. 2. При этом движение в начале производится в направлении обратном ожидаемому, что требуется учитывать при вычислении расстояний..

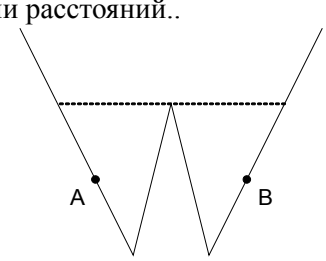

Рис 2. Пример выгодности опускания моста

В процессе решения нам потребуется быстро находить расстояние от точки на ломаной до вершин *A* и *B*. Для этого подсчитаем функцию *d(i)* — расстояние от первой до *i*-й вершины ломаной. Данный предподсчет может быть выполнен за линейное время проходом по ломаной слева направо.

Возьмем пустой стек *S*. Будем двигаться по ломаной слева направо и заносить вершины в стек *S* по следующему правилу: пока в стеке есть вершины с высотой, меньшей высоты текущей вершины, будем удалять вершины из стека *S*, а затем добавим в стек текущую вершину. Очевидно, что при таком построении стека все вершины в нем будут идти по невозрастанию высоты (рис 3). Таким образом, чтобы проверить наличие в стеке вершин с высотой меньшей высоты текущей вершины, достаточно проверить самую последнюю вершину. Отметим, что после обработки *i*-й вершины в стеке находятся верхние вершины ломаной, которые не закрываются первыми *i* отрезками ломаной при взгляде справа.

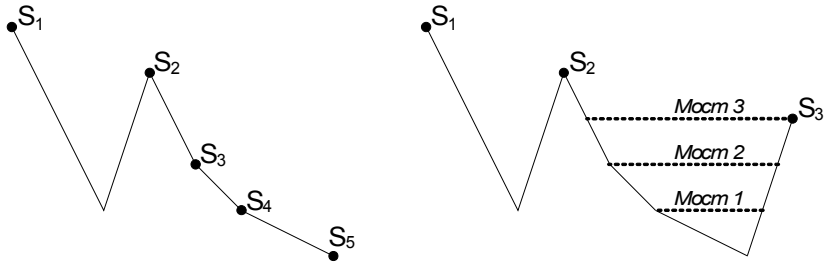

Рис 3. Добавление ребра ломаной и полученные мосты

Пусть в процессе обработки *i*-го отрезка ломаной из стека *S* удаляется вершина с номером *j*, тогда Тогда, если мы проведем мост из вершины *j* вправо, то он упрется в ребро ломаной, ведущей из вершины *i* вниз и влево (мосты 1 и 2 на рис. 3). В тот момент, когда высота вершины, лежащей на вершине стека превысит высоту рассматриваемой вершины, то если мы проведем мост из вершины *i* влево, то он упрется в ребро ломаной, ведущей из вершины, находящейся на вершине стека, вниз и

влево (мост 3 на рис. 2).

Если длина моста, построенного таким образом, превышает L, то сдвинем его вниз так, чтобы его длина стала равна L. Это легко сделать из простых геометрических вычислений, основанных на подобии треугольников. Если полученный мост оказался расположен не ниже, чем вершина, лежавшая в стеке сверху от него, то данный мост является допустимым, так как не пересекает ребер ломаной.

В полученном таким образом решении разбираются все случаи, кроме случая гряды пиков одинаковой высоты. Отметим, что данное решение является линейным по количеству звеньев ломаной.

Добавим обработку гряд пиков. Повторим операцию со стеком. Вершины, образующие гряду будут извлекаться из стека последовательно, так что выделить их не представляет труда.

Пусть мы выделили гряду $a_1, a_2, \ldots, a_k$. На этой гряде можно построить $k+1$ сегмент моста: от вершины a_1 влево, от a_1 до a_2, … , от a_{k-1} до a_k и от a_k вправо (причем сегменты мы строим строго на высоте гряды, даже если они при этом получаются длиннее, чем L). Наш мост будет составлен из одного или более подряд идущих сегментов. Заметим, что если нам удалось построить на гряде наилучший мост, то любой его сегмент будет сокращать расстояние между A и B. Значит можно отбросить первые и последние сегменты, которые либо не уменьшают длины пути между A и B, либо имеют длину больше L. Из оставшихся сегментов мост можно конструировать жадно с помощью двух указателей.

Эта часть решения также работает за линейное время.