Алгоритмы. Домашка 5

Стрелец Владимир

28 мая

Тематические задачи

Задача 1

Отметим все те рёбра исходного графа, входящие в какой-либо кратчайший путь от A до B. Для этого найдём минимальные расстояние от A до каждой из вершин алгоритмом Дейкстры с поиском минимума с помощью бинарной кучи. Затем пройдёмся двумя поисками в глубину по графу из вершины A. Первым поиском отметим дуги, путь по которым вообще ведёт в B (дошли до B — значит это последня дуга вела в B, а затем любая из дуг, ведущая в вершину с исходящей дугой, ведущей в B, становится дугой, ведущей в B). Вторым поиском отметим дуги, лежащее на кратчайшем пути из A в B. Если какая-либо дуга (v, w) является ведущей в B, а также идёт из v в w, причём $min\_destination(v) + weight((v, w)) = min\_destination(w)$, то это значит, что это ребро принадлежит одному из кратчайших путей из A в B.

Затем остаётся лишь найти мосты в графе, состоящем из отмеченных дуг, лежащих на кратчайших путях из A в B.

Сложность описанного алгоритма: $O(m \log n)$ на Дейкстру, $O(m + n)$ на каждый из поисков в глубину, $O(m + n)$ на поиск мостов. Сложность будет $O(n + m \log n)$. Чтобы n стало равным $O(m)$, перед описанным алгоритмом оставим в графе лишь достижимые из A вершины и дуги.

Задача 2

Рассмотрим минимальное остовное дерево T_{min} для исходного графа. Понятно, что для любого ребра, лежащего в этом дереве, минимальное остовное дерево, содержащее это ребро, будет уже построенным деревом. А, следовательно, ответом для таких рёбер будет вес этого дерева.

Теперь рассмотрим любое ребро e_k, не принадлежащее построенному дереву. Если добавить его в это дерево, то в построенном графе $T_{min} + e_k$ появится один цикл. Покажем, что минимальным остовным деревом, содержащим e_k, будет этот граф без максимального по весу ребра e'_k в цикле (но нельзя убирать e_k, так как оно должно быть в дереве). Обозначим этот граф T_{min,e_k}.

Будем строить минимальное остовное дерево алгоритмом Крускала, только начальным состоянием будет не n изолированных вершин, а ребро e_k и $n - 2$ вершин, не инцидентных этому ребру. Рёбра рассматриваются по неубыванию весов. Оптимальность построенного дерева очевидна (исходя из корректности алгоритма Крускала). Ребро e'_k было самым последним в рассмотрении ребром из найденного цикла при таком алгоритме построения. Поэтому оно просто будет образовывать цикл в уже построенной конструкции, а поэтому и не будет добавлено в дерево. А в дерево будут добавлены все остальные рёбра минимального остовного дерева

исходного графа, причём не будет ни одного нового ребра (за исключением e_k). Поэтому искомым деревом как раз будет T_{min,e_k}. Его вес будет равен $weight(T_{min,e_k}) = weight(T_{min}) + weight(e_k) - weight(e'_k)$.

Теперь укажем способ быстрого поиска веса $weight(e'_k)$ для каждого из $O(m)$ рёбер e_k не из T_{min} за $O(\log m)$. Если добавить ребро $e_k = (v, w)$ в дерево T_{min}, то в полученном графе $T_{min}+e_k$ образуется цикл $v \to LCA(v, w) \to w \to v$. Искомое ребро e'_k будет самым тяжёлым на пути $v \to LCA(v, w) \to w$. Поэтому при поиске $LCA(v, w)$ в каждой вершине в дополнение к вершине, в которую мы попадём при "прыжке"вверх по дереву на 2^m рёбер вверх, храним вес максимального ребра на этом "прыжке". А также помним, что $\max\limits_{v \in M \bigsqcup N} weight(v) = \max\{\max\limits_{v \in N} weight(v), \max\limits_{v \in M} weight(v)\}$.

Итак, алгоритм. Изначально построим минимальное остовное дерево T_{min} алгоритмом Крускала за $O(m \log m)$. А далее пройдёмся по всем рёбрам графа с помощью dfs. И если текущее ребро $e_k = (v, w)$ находится в T_{min}, то ответом будет $weight(T_{min})$. Иначе — ответом будет $weight(T_{min})+weight(e_k)-max\_edge\_weight(v-LCA(v,w)-w)$.

Задача 3

[Предположительно, не решено]

Рассмотрим любую вершину, до которой есть несколько простых рёберно непересекающихся путей из вершины A. Пусть мы ищем минимальную длину пути по введённому весу "наивно то есть проходом от A до этой вершины по всевозможным путям из A с обновлением $min\_length(w) = maxmin\_length(v); weight((v, w))$ при проходе по ребру (v, w) от v к w. Понятно, что минимальный путь будет найден.

Теперь покажем, что если запустить поиск в глубину с подобной релаксацией по ребру при проходе по нему в любую из сторон, то [будут пройдены и "пролелаксированы"все пути из A в каждую из рассматриваемых вершин, до которых путь минимален; а второй поиск в глубину найдёт веса минимальных путей и до всех остальных вершин] ничего, к сожалению, не случится. Сразу думал за 2 поиска в глубину всё это решить, но нашёл пример, ломающий всё :C

На самом деле сделаю ставку (попытка решения без доказательства xD), что за три dfs с подобными "направленными релаксациями"в B будет храниться длина минимального пути. После 2 dfs для вершин с несколькими до них простыми рёберно непересекающимися путями всё станет на свои места, а после третьего — и для всех остальных (с одним путём).

А вообще можно сделать всё с помощью классического алгоритма Дейкстры (например, на бинарной/Фибоначчиевой куче), запущенного из вершины A. Только ассимптотика будет не та. Но ведь так часто делается в жизни — неоптимально C:

Задача 4

Заметим, что если и заменять вес какого-либо ребра на неотрицательное число, то уж заменять на ноль! Ведь длина любого из путей, а, соответственно, и минимального, от этого не увеличится, но длина некоторых уменьшится.

Построим новый граф, в котором для каждой из вершин v исходного графа сделаем $k+1$ новых вершин вида $(v, i), i = \overline{0, k}$, где вторая координата будет обозначать

количество заменённых (обнулённых) ребёр на пути к вершине v в исходном графе (если проводить аналогии между поисками кратчайших путей в исходном и построенном графах).

Какие рёбра будем проводить в этом графе? Если было ребро между вершинами v и w, веса $weight((v,w))$, то в новом графе будут введены дуги (не рёбра) вида $((v,i),(w,i))$ и $((w,i),(v,i)), i = \overline{0,k}$ весов $weight((v,w))$ каждая (отвечающие переходу без обнулений ребёр), а также дуги вида $((v,i),(w,i+1))$ и $((w,i),(v,i+1)), i = \overline{0,k-1}$ весов 0 каждая (переход с обнулением ребра и увеличением количества обнулённых рёбер). Также добавим дуги вида $((B,i),(B,i+1)), i = \overline{0,k}$ (на тот случай, если по пути к вершине B в исходном графе было занулено слишком мало рёбер а этот путь оказался минимальным. Такое возможно, если минимальный путь до вершины B содержит меньшее, чем k, число положительных по весу рёбер). Итак, количество дуг в построенном графе будет равно $m' = (4k+2)m + k = O(m)$, а количество вершин $n' = n(k+1) = O(n)$.

Вводим дуги вместо рёбер, так как в минимальном пути к вершине B исходного графа мог содержаться как переход вида $v \to w$ с обнулением ребра, так и переход вида $w \to v$ с обнулением ребра.

Теперь если мы запустим алгоритм Дейкстры на куче Фибоначчи из вершины $(A,0)$, то ответом на задачу будет минимальное расстояние до вершины (B,k).

Ассимптотическая сложность алгоритма будет составит

$$O(m' + n' \log n') = O((4k+2)m + k + (n(k+1)) \log (n(k+1))) = O(m + n \log n).$$

Задача 5

Заметим, что студенты не могут встретиться на ребре, так как если они идут в одну сторону, то из-за одинаковой скорости никто никого не догонит, а идти в разные стороны они не могут, так как в одной расстояние от общаги до одной из вершин любого ребра не больше, чем от общаги до второй вершины. А, следовательно, студент не пошёл бы от ближайшей вершины по ребру к более дальней вершине.

Итак, студенты могут встретиться лишь в вершинах и лишь в них они решают, кому продолжить нелёгкий вечерне-ночной путь до общаги, а кому не выдержать испытаний и, выйдя из графа, сесть на асфальт и расплакаться.

Любые 2 студента с минимальными путями разных длин не встретятся в вершинах. Поэтому соревноваться за [главный приз] путь к общаге могут лишь те, кто находятся на одинаковом удалении от общаги (т.е. студенты из одного [братства] класса эквивалентности).

Теперь рассмотрим один из таких классов эквивалентности. Найдём суммарный максимальный поток (пропускные способности каждого ребра равны 1) от общаги до всех вершин со студентами из этого класса эквивалентности (учитывая, что если в вершине стоит x студентов, и был найден поток величины x до этой вершины, то до неё искать ничего больше не стоит.). Это будет ответом на вопрос "сколько студентов из этого класса эквивалентности доберуются сегодня до общаги?".

Просуммировав эти значения для каждого из классов эквивалентности, получим ответ на задачу.

Теперь что касается реализации. Для нахождения минимальных расстояний от общаги до всех вершин графа воспользуемся алгоритмом Дейкстры на куче Фибоначчи ($O(m + n \log n)$). Чтобы разделить вершины по классам эквивалентности,

отсортируем все вершины по расстоянию ($O(n \log n)$), и будем просматривать из по неубыванию и начинать новый класс эквивалентности после изменения расстояния до вершины. Максимальный поток будем искать поиском в ширину. Для этого введём дополнительную фейковую вершину для каждого из классов эквивалентности и заменим в графе все рёбра на парные дуги, а также соединим все вершины со студентами из одного класса эквивалентности только исходящими дугами с фейковой вершиной. Причём пропускные способности этих новых рёбер будут равны количеству студентов в вершинах, из которых исходят дуги. Так как всего студентов k, то суммарный максимальный поток по всем студентам не превосходит k. А это — не более k поисков в ширину.

Полученная сложность алгоритма — $O(n \log n + (k+1)m) = O(n \log n + km)$.

Задача 6

Рассмотрим граф, в котором будет N вершин, соответствующих сыновьям Чингиса, N вершин, соответствующих девушкам, а также рёбра, соединяющие парней и тех девушек, которые понравились парням. Этот граф — двудольный.

Таким образом, в этом графе найдёно совершенное паросочетание. Попробуем изменить для i-го парня девушку с $a_{i,1}$ на $a_{i,l}$. Тогда нужно и парню, который выбрал девушку $a_{i,l}$, сменить девушку на какую-либо другую. И следующему парню. Если существует такая комбинация, которая заканчивает череду смен на исходной девушке $a_{i,1}$, то это значит, что парню i можно сменить девушку на $a_{i,l}, 2 \leq l \leq k_i$, и при этом остальные парни смогут каким-то образом договориться. Это вообще означает, что в построенном графе существует простой цикл, в котором лежит ребро между i-м парнем и девушкой $a_{i,l}$.

А теперь алгоритм. Строим ориентированный граф, в котором введены следующие дуги: если мудрец изначально выбрал для i-го парня девушку $a_{i,1}$, то добавляем дугу $(a_{i,1} \to i)$ (это значит переход к парню i для перевыбора девушки, так как девушка $a_{i,l}$ уже перевыбрана другим парнем). А если девушка "ничего так но не выбрана мудрецом, то добавляем дугу $(i \to a_{i,l})$ (возможность перевыбрать девушку $a_{i,l}$ вместо $a_{i,1}$).

Алгоритм. Запускаем dfs для каждой вершины графа (из множества непомеченных) и если так оказалось, что найден цикл, это будет значить, что все текущие перевыбранные девушки в цикле являются вполне себе неплохими заменами для перевыбравших их парней. Для определения этих возможных замен будем просто при возвращении из обхода по ребру тянуть с собой флаг, говорящий, что такая замена возможна и добавлять её в список ответов. Если же флаг установлен в $false$, то флаг не переустанавливаем в $true$, ведь такая замена может поссорить братьев. Но если хоть из одного ребра ниже по дереву пришёл флаг $true$, то существует корректная замена. Поэтому после установки флага в $true$ он неможет измениться на $false$. То есть флаг при переходе по древесному ребру равен $false$, а после нахождёния обратного ребра превращается в $true$.

Что касается сложности. Ответ (тот, который с размером T) имеет размер $O(m+n)$, так как количество парней n, а количество девушек, которые могут быть перевыбраны не больше, чем число рёбер в графе. А вот $N + A$ как раз и равно $O(m+n)$. Таким образом, итоговая сложность для алгоритма — $O(m+n) = O(N+A) = O(T+N+A) = O(\max(T, N+A))$.

Задача 7

[Не решено]

Задачи на повторение

Задача 8

Рассмотрим любое ребро $e = (v, w)$ в исходном дереве. Найдём количество простых путей, в которые входит это ребро. Одним концом любого из простых путей будет будет вершина, лежащая по одну сторону этого ребра, а вторым концом — вершина, лежащая по вторую сторону (следует из того, что любая вершина в дереве — мост).

Поэтому если количество вершин по одну из сторон будет равно $count\_vertices(v)$, то количество вершин по вторую сторону будет равно $n - count\_vertices(v)$, а число всех простых путей, которым принадлежит исходное ребро —

$$count\_ways((w, v)) = count\_vertices(v) \cdot (n - count\_vertices(v)).$$

Понятно, что сумма длин всех простых путей в дереве будет равна сумме показанных величин. Для нахождения этой суммы воспользуется поиском в глубину из любой вершины дерева, в котором будем поддерживать количество вершин в поддереве с корнем в текущей вершине (эта величина равна 1 для листьев и равна $count\_vertices(v) = 1 + \sum\limits_{u \in sons(v)} count\_vertices(u)$ для всех остальных вершин). В этом поиске после возвращения из какого-либо ребра (v, w) величина $count\_vertices(w)$ уже определена, а значит уже можно добавить в итоговую сумму число

$$count\_ways((v, w)) = count\_vertices(w) \cdot (n - count\_vertices(w)).$$

Понятно, что изначально сумма равна 0.

Итак, ответ:

$$\sum_{(v,w) \in E(Tree)} count\_ways((v, w)) = \sum_{(v,w) \in E(Tree)} count\_vertices(w) \cdot (n - count\_vertices(w)).$$

Задача 9

Рассмотрим любое значение $x \in A$.

Последовательность значений $x, f(x), f(f(x)), \ldots, f^k(x), \ldots$ (если изобразить её в виде графа с дугами вида $f^i(x) \to f^{i+1}(x)$) будет выглядеть как ветвь, ведущая к циклу.

Теперь рассмотрим весь граф. Он выглядит как несколько циклов, в которые, возможно, ведут ветви. Ветви могут сливаться до того, как зашли в цикл. Таким образом, граф — несколько компонент связности, каждая из которых является циклом с входящими в него деревьями деревьями.

Чтобы $f^k(x)$ была идемпотентна, она должна была уже привести наш исходный x в цикл. Также k должен делиться на длины каждого из циклов. То есть, $k = c \cdot \text{НОК}\{ loop_1; loop_2; \ldots; loop_l \}$, где c – минимальная натуральная константа, такая что k больше длин всех ветвей.

Таким образом, воспользовавшись, например, алгоритмом Флойда, мы сможем найти длины всех циклов, а также длины всех ветвей (и, соответственно, длину самой длинной ветви) суммарно за $O(|A|)$. Для того, чтобы не рассматривать разные циклы по несколько раз, после нахождения длины какого-либо цикла поставим метки $used(v) = true$ для вершин этого пути. Также на вершинах ветви этой пути расставим расстояние в дугах до цикла. Теперь же если мы попали в вершину w с $used(w) == true$, то это значит, что рассматриваемый текущий путь содержит цикл, который уже был рассмотрен, а значит далее его рассматривать не стоит, но стоит обновить максимальную длину ветви (максимум из текущей максимальной длины и суммы длина текущего рассматриваемого пути до первой вершины, $used$ которой равен $true$, + длина от этой вершины до цикла).

Итак, мы имеем несколько чисел $loop_i, i = \overline{1, l}$, сумма которых равна $O(|A|)$, НОК которых требуется найти, а также длину самой длинной ветви. Изначально применим алгоритм решета Эратосфена к массиву чисел от 1 до $|A|$ за $O(|A|)$. Далее разложим на простые множители числа $loop_i$ (каждое за $O(\log loop_i)$, суммарно за $O(l + \sum_{i=1}^{l} \log loop_i) = O(|A| + |A|) = O(|A|)$, так как сумма логарифмов чисел, сумма которых равна $O(|A|)$, не больше, чем $O(|A|)$).

Затем заведём массив $LCM - array$ для подсчёта простых множителей, входящих в искомый НОК. Теперь пробежимся по найденным разложениям и для каждого из простых чисел p в $LCM - array$ будем обновлять максимальное количество простых делителей p из всех разложений $loop_i$ (простой делитель — индекс в массиве $LCM - array$). А затем просто найдём НОК, найдя произведение $\prod_{index=0}^{|A|-1} index^{LCM-array[index]}$.
Также нужно найти минимальную такую натуральную константу c, что $k \geq length\_max\_brunch$. Так как НОК получится не меньше 1, то такая константа найдётся за $O(|A|)$ простым инкрементированием, начиная с 1.

Так как число всех простых делителей было $O(|A|)$, то эта часть действа, пройдёт за $O(|A|)$, а всё действо — за $O(|A|)$, а не за $O(|A|log|A|)$ (сортировать длины циклов ведь не надо, нам не важен порядок их просмотра для поддержания максимальной степени каждого из простых делителей из-за коммутативности операции максимум).

Найденное k и будет ответом. Заметим, что, таким образом, такое минимальное k всегда существует.

Практические задачи

Ма-аленькое вступление

Вспомнил твои, Алексей, слова: "расставляйте приоритеты и осознал: "не закожу".

Задача 10

[Даже не пытался]

Задача 11

[Даже не пытался]

Задача 12

[Даже не пытался]