Основные алгоритмы 9

Ковалев Алексей

1. Достаточно запустить `DFS(s)`, но на каждом шаге дополнительно проверять, не является ли обрабатываемая вершина искомой вершиной t. Тогда в силу леммы о белых путях вершина t будет посещена, если и только если она была достижима из s. Значит алгоритм корректен и работает за то же время, что и `DFS`, так как на каждом шаге затрачивает только $O(1)$ дополнительного времени. То есть время работы, как и требуется, составляет $O(|V| + |E|)$.

2. Будем проводить доказательство по индукции по числу вершин:
База $n = 2$ очевидна.
Переход: мы знаем, что в турнире на $n-1$ вершине есть путь длины $n-2$. Добавим в него новую вершину x (и все ребра, соединяющие ее с другими вершинами). Если эта вершина может быть добавлена в путь первой или последней, то добавим ее туда, получив искомый путь длины $n-1$. Если это невозможно, то найдутся такие ребро (u, v) в этом пути, что также в графе есть ребра (u, x) и (x, v). Тогда удалим из пути (u, v) и добавим (u, x) и (x, v), получив тем самым искомый путь длины $n-1$.
Значит в турнире всегда существует путь длины $n-1$. □
Алгоритм, который находит такой путь можно построить аналогично доказательству: занумеруем вершины графа каким-либо способом и будем действовать "по индукции то есть сначала найдем путь в графе из вершин 1 и 2, затем будем добавлять к нему вершину, удлиняя уже существующий путь на каждой итерации. На каждой i-той итерации рассматривается не более $2(i-1)$ ребер, значит всего рассматривается $1+2+\ldots+n-1 = O\big(n^2\big) = O(|E|)$ ребер, причем рассмотрение каждого ребра требует $O(1)$ времени. Значит асимптотика алгоритма – $O(|E|)$, то есть алгоритм линеен от размеров графа.

3. Будем называть масисив d времен открытия t_{in}, а массив f времен закрытия t_{out}. Рассмотрим некоторое ребро (u, v). Оно будет

- прямым, если $t_{\text{in}}[u] < t_{\text{in}}[v]$, $t_{\text{out}}[u] > t_{\text{out}}[v]$ и $\exists z\colon t_{\text{in}}[u] < t_{\text{in}}[z] < t_{\text{in}}[v],\ t_{\text{out}}[u] > t_{\text{out}}[z] > t_{\text{out}}[v]$.
- перекрестным, если $t_{\text{in}}[u] < t_{\text{in}}[v]$ и $t_{\text{out}}[u] < t_{\text{out}}[v]$ или $t_{\text{in}}[u] > t_{\text{in}}[v]$ и $t_{\text{out}}[u] > t_{\text{out}}[v]$.

4.

1. Эту задачу решает алгоритм Косарайю, который разбивает граф на компоненты сильной связности за линейное от размеров графа время (то есть $O(|V| + |E|)$). (мы, кажется, даже разбирали его на семинаре)
2. Для начала сконденсируем данный граф G, получив граф G'. В нем есть истоки (s штук), стоки (t штук) и изолированные вершины (q штук). Очевидно, что для того чтобы G стал сильно связным, необходимо и достаточно, чтобы G' стал сильно свзяным. Также понятно, что для того чтобы G' стал сильно связаным необходимо как минимум $\max(s, t) + q$ ребер, так как надо добавить входящее ребро в каждый исток и изолированную вершину и исходящее ребро из каждого стока и изолированной вершины. Теперь передем к описанию самого алгоритма, который находит ровно $\max(s, t) + q$ ребер, которые делают G сильно связным. Будем запускать поиск в глубину от всех истоков, запоминая, какие вершины графа уже были посещены, и останавливая его тогда, когда был найден некоторый сток. Найденный сток будем сопоставлять истоку, из которого был запущен поиск в глубину, таким образом формируя пары сток-исток. Это все (конденсация, поиск стоков, истоков, изолированных вершинй в ней, поиск пар) займет линейное от размеров графа время. Далее будем добавлять необходимые для сильной связности ребра: циклически соединим все изолированные вершины и стоки с истоками из пар, то есть добавляем ребро из стока i-той пары в исток $(i+1)$-ой пары и ребра из i-той изолированной вершины в $(i+1)$-ую (из стока последней пары – в первую изолированную вершину, из последней изолированной в истой первой пары). Это даст нам некоторый "цикл из которого могут "торчать"некоторое число стоков и истоков. Далее разобьем оставшиеся стоки и истоки на пары (количество этих пар – минимум из количеств

стоков и истоков) и проведем по ребру в каждой паре от стока к истоку. После этого либо все стоки, либо все истоки закончатся, и мы сможем присоединить оставшиеся стоки / истоки к произвольной вершине графа (если стоки – то ребра из них, если истоки – то ребра в них). Несложно понять, что такой набор ребер имеет размер ровно $\max(s, t) + q$, то есть алгоритм восстанавливает минимальное необходимое число ребер. Время работы алгоритма линейно, так как мы сделали несколько линейных от размеров графа действий с графом (поиск необходимых для добавление ребер также линеен, так как надо добавить не более $|V|$ ребер). (Эту задачу я уже решал и даже напрогал этот алгоритм. Ссылка на мое решеение: `https://codeforces.com/group/PVbQ8eK2T4/contest/374347/submission/151440527`. Оно кривое с точки зрения качества кода, но оно работает).

5. Представим, что лабиринт – это граф, вершины которого – комнаты, а ребра – коридоры между ними. Монетки будем использовать для того, чтобы помечать уже посещенные комнаты. Тогда найти способ выбраться из лабиринта можно с помощью поиска в глубину: запустим его из комнаты со входом, и если выход из нее достижим, то он обязательно будет найден (по задаче 1). Сложность этого поиска можно оценить как $O(m + n)$, где n – количество комнат в лабиринте. Но лабиринт связен, а значит в нем не более $m + 1$ комнат, откуда сложность $O(m)$.

6. Будем для удобства объяснения считать, что изначальный граф-путь выглядит так, что все его ребра направлены вправо. Тогда для решения задачи можно предложить следующий алгоритм: сначала удалим все добавленные ребра, ведущие вправо, то есть соединияющие вершины i и j, такие что $i + 1 < j$ (это необязательно делать, но это действие упрощает понимание и не влияет на асимптотику). Затем создадим массив пар, в котором первый элемент пары будет номером вершины, инцидентной некоторому ребру, а второй элемент пары – нулем, если ребро начинается в этой вершине, и единицей, если заканчивается в ней. Отсортируем полученный мвссив пар по первым элементам в этих парах. Затем при линейном обходе этого массива пар мы можем найти компоненты сильной связности и их размер, а значит и их количество. Иными словами у нас есть массив и какие-то отрезки (их начала и концы – нули и единицы в парах) в нем. Если какие-то отрезки пересекаются, их нужно считать за один отрезок, и тогда надо найти количество этих отрезков и их размер, что легко делается одним линейным обходом этого массива. Если количество таких отрезков – c, а их суммарный размер – s, то количество компонент сильной связности в искомом графе равно $n - s + c$.
Корректность алгоритма: дополнительные ребра, ведущие направо можно выкинуть, так как любое из них равносильно некоторому пути ребер изначального графа-пути. Рассмотрим два дополнительных ребра (u_1, u_2) и (v_1, v_2), ведущих налево:

- Если $v_2 \leqslant u_2 \leqslant u_1 \leqslant v_1$, то ребро (u_1, u_2) никак не влияет на достижимость вершин друг из друга.
- Если $v_2 \leqslant u_2 \leqslant v_1 \leqslant u_1$, то эти ребра с точки зрения достижимости равносильно одному ребру (v_2, u_1).
- Если $v_2 \leqslant v_1 \leqslant u_2 \leqslant u_1$, то при условии отсутствия других ребер из предыдущих пунктов, вершины $\{v_2, \dots, v_1\}$ и $\{u_2, \dots, u_1\}$ не лежат в одной компоненте сильной связности.

Фактически, именно эти варианты и рассматриваются выше в описании алгоритма.
Сложность алгоритма составляет $O(m \log m)$, так как сортировка массива пар займет именно такое время, а несколько линейных обходов не повлияют на нее.

7. Описанный в условии двудольный граф является регулярным, а значит в нем есть совершенное паросочетание. Если данный граф не связен, то найдем все его компоненты связности и запустимся рекурсивно от них, а затем объединим все паросочетания в подграфах. Паросочетание в связном графе строится так: выберем произвольную вершину левой доли и возьмем в паросочетание какое-то из ицидентных ей ребер. Это однозначно определит, какие еще ребра могут лежать в этом паросочетании. Возбмем в него их все. Если это паросочетание является совершенным, то ответ для данного графа найден, иначе совершенное паросочетание содержит все остальные ребра этого графа и не содержит уже взятые. Сложность алгоритма линейна от размеров графа, так как поиск компонент связности требует линейного времени, попытка составить совершенное паросочетние в каждом подграфе – суммарно линейное время, проверка выбранного паросочетания – также линейное время, то есть все действия линейны.