Домашнее задание №9

Гунаев Руслан, 776 группа

3 мая 2019 г.

1.

Пусть граф является деревом. Если между какими-то двумя вершинами есть два простых пути, то тогда точно найдется цикл. Если пути не пересекаются, то цикл будет включать обе вершины, если пересекаются, то можно выделить цикл меньшего размера на вершине пересечения.

Пусть для любой пары вершин существует только 1 простой путь. Во-первых, граф связный. Во-вторых, ациклический, ведь иначе нашлись бы как минимум две вершины имеющие два простых пути. А значит этот граф – дерево.

Пусть граф связен и у него $V-1$ ребро. Докажем по индукции, что этот граф дерево.

1. Если $V=2$, то есть только одно ребро, очевидно такой граф является деревом.
2. Пусть для любого связного графа на n вершинах с $n-1$ ребром выполнено, что он является деревом.
3. Рассмотрим связный граф на $n+1$ вершине с n ребрами. Очевидно найдется вершина степень которой равна 1, ведь иначе ребер $n+1$. Уберем эту вершину и ребро, выходящее из нее, получим связный граф на n вершинах с $n-1$ ребром, тогда этот граф – дерево. Обратно добавим эту вершину и это ребро. Граф останется связным, также не появится цикла, ведь если вершина участвует в цикле, то ее степень должна быть минимум 2. Значит исходный граф также дерево.

Пусть граф – дерево, покажем, что тогда количество ребер равно $V-1$.

1. Для деревьев на одной и двух вершинах очевидно выполняется это свойство.
2. Пусть верно для всех деревьев на $k \leqslant n$ вершинах.
3. Рассмотрим дерево на $n+1$ вершине. Удалим любое ребро. Очевидно граф станет несвязным в силу единственности простого пути между любыми двумя вершинами. Получим две части графа, каждая из которых является связным деревом на вершинах, количество которых меньше либо равно n, но тогда в этих подграфах количество ребер на 1 меньше, чем количество вершин, значит если мы добавим убранное ранее ребро, то получим дерево ровно на n ребрах.

Пусть граф – дерево. Возьмем произвольную вершину, назовем ее корнем, известно, что от нее до любой другой существует только 1 простой путь. Рассмотрим произвольный путь. Ориентируем ребра, которые принадлежат этому пути в порядке перехода от одной вершины к другой. Сделаем так со всеми простыми путями. Тогда получим, что в корень не входит ни одно ребро, а только выходят. Пусть для какой-то вершины выполнено, что в нее входит хотя бы два ребра. По построению это означает, что эта вершина приняла участие в двух простых путях от корня, а значит в исходном неориентированном графе был цикл.

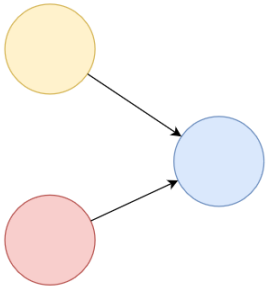
Пусть теперь граф можно ориентировать так, чтобы выполнялось 4 условие. Предположим в графе был цикл. Зафиксируем ребра, которые входили в этот цикл, они каким-то образом ориентированы. Есть два варианта.

1. После ориентирования этот цикл сломался. Это означает, что в этом цикле нашлась вершина,в которую вошло 2 ребра, значит такого быть не может.

2. После ориентирования цикл остался. Вершина, в которую не входит ни одно ребро, попасть в такой цикл не может, так как иначе в нее что-то должно входить. Тогда рассмотрим ориентированный подграф, образующий цикл, назовем его G', и назовем G'' оставшийся подграф. Эти подграфы связны. Тогда либо из графа G'' есть ребро, идущее в G', но тогда в G' найдется вершина с двумя входящими в нее ребрами, либо из G' в G'', из вершины u' в u''. В графе G'' есть вершина, в которую не входит ни одно ребро, назовем ее a. Тогда с одной стороны мы можем попасть в вершину u'', минуя подграф G', ведь тогда попадем в предыдущий случай, с другой стороны в u'' входит ребро из G', значит в графе нашлась вершина с двумя входящими в нее ребрами.

   Таким образом, данный граф является связным ациклическим, а значит дерево.

2.



3.

1)

В обеих процедурах мы сначала добавляем в очередь стартовую вершину. Далее мы смотрим все достижимые из нее вершины. И так далее. Порядок рассмотрения вершин будет одинаковый, если эквивалентны условия, после проверки которого вершина добавляется в очередь.

Очевидно, что если вершина достижима из другой вершины, то максимальное наименьшее расстояние между ними не превосходит $V - 1$, потому что иначе какая-то вершина в этом пути встретится дважды, а значит есть цикл, который можно миновать.

Условие $dist[v] + 1 < dist[u]$ означает, что вершина может быть достижима. Изначально вершина u не лежит в очереди, а $dist[u] = V$, когда мы меняем его на меньшее значение, то вершина автоматически становится достижимой из s.

Если это условие не выполняется, значит вершина уже была осещена ранее. Именно это проверяет первая процедура. Значит процедуры эквивалентны.

2)

Если вершина еще не была посещена, и она смежна с вершиной v, которая достижима из s, то существует путь из s в u длиной $dist[v] + 1$. Данная процедура по сути разбивает все вершины на уровни достижимости. На нулевом находится сама вершина, на первом – все смежные с ней и так далее. То есть, если вершина находится на k-ом уровне, то ,во-первых, найдется вершина на $k - 1$ уровне, смежная с ней, во-вторых, путь длины k будет минимальным, ведь иначе

вершина должна была бы оказаться на более высоком уровне, при переходе на новый уровень, процедура увеличивает расстояние на 1.

3)

Алгоритмы эквивалентны, значит время работы обоих одинаковое. В очередь добавляются только непосещенные вершины, поэтому каждая вершина рассматривается только 1 раз. Далее для каждой вершины рассматриваются все вершины, смежные с ней, то есть мы проходимся по всем ребрам каждой вершины, а значит

$$T(G) = O(max\{V, E\}) = O(V + E).$$

4.

Изначально значение массива *visited* для каждой вершины равно 0. Оно становится равным единице, если из этой вершины был запущен поиск в глубину. Запуститься он может только в том случае, если эта вершина смежна с какой-то уже посещенной вершиной. Занумеруем вершины в порядке обхода из начальной вершины. Начальная вершина сразу становится посещенной. Пусть значения для первых k посещенных вершин в массиве *visited* равны единице, и они достижимы из начальной. Далее мы попали в $k+1$ вершину, значит она смежна с какой-то из первых k. Значит существует путь от s до $k+1$-ой, а $visited[v_{i_{k+1}}] = 1$.

Пусть теперь вершина a достижима, но значение в массиве равно нулю. Рассмотрим путь от начальной вершины до нее. Так как эта вершина отмечена как непосещенная, но она смежна со своим предком в пути, то у предка также значение в массиве равно нулю, ведь иначе от a бы также запустился поиск в глубину, а значение стало единицей. Если у предка значение равно нулю, то и у его предка также, поднимаясь образом, придем к тому, что значение в массиве *visited* для стартовой вершины равно 0, но оно становится равным 1 в самом начале работы алгоритма, противоречие.

Получили, что $visited[a] = 1$ тогда и только тогда, когда вершина достижима из начальной вершины.

5.

a)

Производем поиск в ширину из произвольной вершины; как только в процессе обхода мы пытаемся пойти из текущей вершины по какому-то ребру в уже посещённую вершину, то это означает, что мы нашли цикл, и останавливаем обход в ширину. Запомним вершину a, из которой мы попали в уже посещенную b. Удалим ребро (a, b). Запустим модифицированный алгоритм поиска в ширину из вершины b. Теперь для каждой вершины будем хранить предшественника. Как только вершина a будет посещена, остановим алгоритм, а далее пройдемся по массиву предшественников до b. Все эти вершины будут образовывать цикл. Такой алгоритм будет работать с ориентированным графом. Если в графе есть цикл, то мы можем найти две смежные вершины в этом цикле и, убрав соединяющее эти вершины ребро, все равно сможем попасть из одной вершины в другую по оставшейся дуге цикла, ориентация графа никак не повлияет на это.

b)

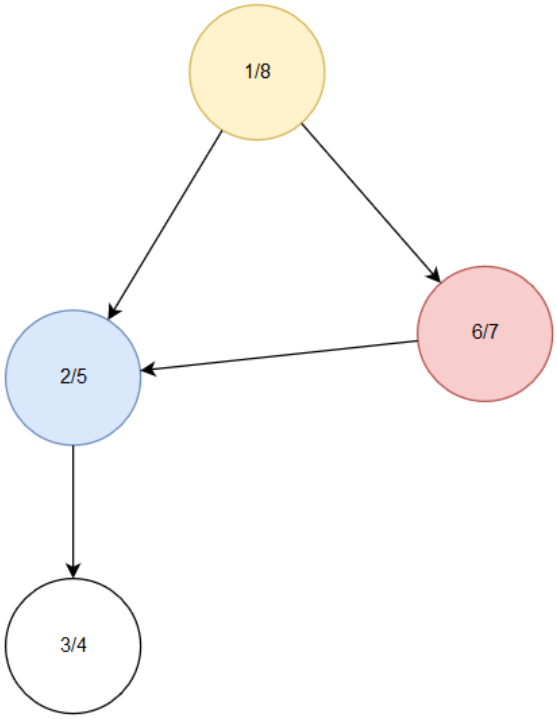
Из каждой вершины, в которую мы ещё ни разу не приходили, запустим поиск в глубину, который при входе в вершину будет красить её в серый цвет, а при выходе из нее — в чёрный. И, если алгоритм пытается пойти в серую вершину, то это означает, что цикл найден.

Для восстановления самого цикла достаточно при запуске поиска в глубину из очередной вершины добавлять эту вершину в стек. Когда поиск в глубину нашел вершину, которая лежит на цикле, будем последовательно вынимать вершины из стека, пока не встретим найденную еще раз. Все вынутые вершины будут лежать на искомом цикле.

6.

В силу теоремы о скобочной структуре дерева обхода в глубину следует, что вершина a является предком b тогда и только тогда, когда вершина a была открыта раньше, чем b, а закрыта позже. (Доказательство в Кормене.) Пусть ребро (b, a) не попало в дерево обхода, это означает, что вершина a уже открыта. Закрыта вершина a быть не может в силу отсутствия ориентации графа, ведь если такое ребро существует, то закрыть мы ее можем только после рассмотрения этого ребра, но раз мы сейчас попали в b да так, что ребро (b, a) не попало в дерево, то пришли в b другим путем, значит вершина a закроется позже, чем b, значит a–предок b.

Пример для ориентированного графа.



Ребро, соединяющее красную вершину с синей не попадет в дерево обхода в глубину, но красная не является предком синей.

7.

а)

Покажем, что остовное дерево, состоящее из минимальных ребер на циклах, минимальное.

Пусть нет. Это значит, что дерево можно улучшить, а значит на каком-то цикле выбрать ребро меньшего веса, не принадлежащее дереву, а значит дерево мы строили не на минимальных ребрах в циклах.

b)

Рассмотрим алгоритм Прима, на каждом шаге которого мы добавляем безопасные ребра, значит итоговое дерево состоит только из безопасных ребер. Рассмотрим разрез полученного дерева (A_1, A_2). Рассмотрим ребро (a, b), пересекающее этот разрез, $a \in A_1, b \in A_2$. Рассмотрим путь в изначальном графе, соединяющий a, b, также рассмотрим некоторое ребро e_i, которое пересекает разрез и лежит в этом пути, такое точно найдется ведь вершины a, b лежат в разных компонентах связности. Но так как (a, b) принадлежит остовному дереву, значит оно безопасное, а значит по теореме о разрезе, $\omega((a, b)) \leqslant \omega(e_i)$. Получили, что все ребра, не попавшие в дерево, не легче ребер из цикла.

8.

a)

Из прошлой задачи следует, что любое ребро не попавшее в дерево не легче ребра из цикла, попавшего в остовное дерево. Это значит , что максимальное ребро, принадлежащее пути по остовному дереву, будет меньше либо равно всех ребер, не попавших в это дерево. Таким образом, какой бы мы путь не взяли, он будет больше пути по остовному дереву.

b)

Возьмем вершины u, v Рассмотрим путь uv,который не лежит в остовном дереве, и путь по дереву.

По условию задачи ω(путь uv) $\geqslant \omega$(пути по дереву)

$\omega(max$ ребро пути $uv) = \omega$(пути $uv) \geqslant w$(пути по дереву) = max $\omega(max$ ребро из пути по дереву)

Значит выполняется условие прошлой задачи, а значит дерево минимальное.

9.

Так как работа алгоритма Форда-Фалкерсона(а лучше Эдмондса-Карпа) продолжается до тех пор, пока в остаточной сети существует путь из s в t, то для нахождения минимального разреза нам нужно найти такой разрез, ребра которого (ребра, из S в T) насыщены. Для этого достаточно проделать алгоритм Эдмондса-Карпа до конца, а потом запустить поиск в ширину из s. Вершины, достижимые из истока, и будут образовывать левую долю в минимальном разрезе.

3

Рассмотрим произвольные две вершины u и v графа. Для каждой пары вершин (x, y), между которыми было ребро (x, y) мы построим два ребра (x, y) и (y, x) с пропускными способностями, равными единице.

Заметим, что наш граф будет реберно k-связным, если максимальный поток для любых двух вершин u и v графа не меньше k, а также хотя бы для одной пары вершин этот должен быть равен k. Действительно, если найдется пара вершин, между которыми максимальный поток будет меньше k, то если мы в исходном графе удалим ребра (их соответственно будет меньше k) соответствующие ребрам в минимальном разрезе, то граф распадется на компоненты связности. Также у нас не может быть в минимальном разрезе для любой пары вершин быть строго больше k ребер, т. к. тогда у нас при удалении любых k ребер будет сохраняться связность.

Таким образом, алгоритм определения на k связность следующий. Для любой пары вершин u и v мы определяем максимальный поток, например, при помощи алгоритма Эдмонда-Карпа,

который полиномиален относительно входа. Проверяем количество ребер в минимальном разрезе и сравниваем это число ребер с k. Так мы делаем для всех пар, количество которых $O(|V|^2)$. Поэтому весь алгоритм останется полиномиальным относительно входа.