Домашняя работа №6

Бредихин Александр

19 апреля 2020 г.

Задача 1

Задача: на вход задачи подаётся граф G и его верины s и t. Постройте алгоритм, который за время $O(|V| + |E|)$ проверяет, что вершина t достижима из вершины s. Решите задачу как в случае, когда G неориентированный граф, так и в случае, когда G ориентированный граф.

Запускаем DFS из вершины s и храним, как обычно вершины, которые достигаются алгоритмом, их время открытия и закрытия. Известно, что алгоритм обходит все вершины, которые достигаются из s, как для ориентированного графа, так и нет.

После запуска этого алгоритма проходим по полученному массиву встретившихся вершин и проверяем есть ли в нём вершина t или её нет. Если она есть, то она достижима, если нет, то нет (так как алгоритм просматривает все достижимые вершины).

Сложность: DFS работает за $O(|V| + |E|)$. В худшем случает нам придётся рассмотреть все вершины (все вершины будут достигаться из s), поэтому в массиве после выполнения будет $|V|$ вершин. Значит вершину t в нём в худшем случае мы найдём за $O|V|$.
Получается сложность всего алгоритма будет: $O(|V| + |E|) + O(|V|) = O(|V| + |E|)$ (как и требуется в задаче).

Задача 2

Задача: докажите, что каждый турнир на n вершинах содержит (простой) путь длины $n - 1$. Постройте алгоритм, который получив на вход турнир, находит в нём такой путь, и оцените асимптотику его времени работы.

1) По индукции по количествам вершин в турнире докажем, что что каждый турнир на n вершинах содержит (простой) путь длины $n-1$.

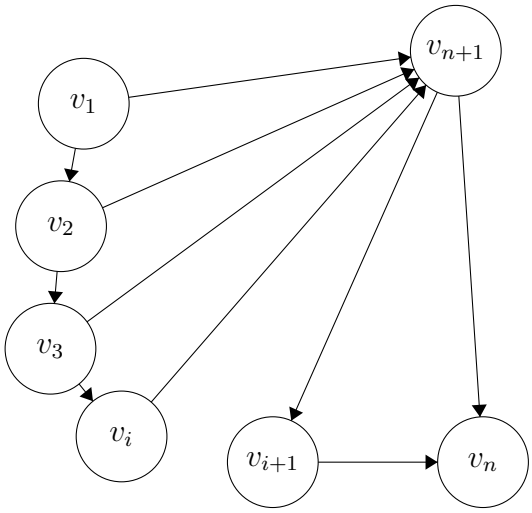
Б.И. для $n=2$: всегда будет путь либо из v_1 в v_2 либо наоборот (по определению турнира). Понятно, что это простой путь и его длина 1 – выполнено.

Ш.И. пусть для турнира с n вершинами мы нашли такой простой путь: $v_1, v_2, \dots v_n$ (длина $n-1$), докажем, что при добавлении новой вершины v_{n+1} мы найдём простой путь. Возможны 3 случая:

1) При добавленнаии вершины v_{n+1} в турнире будет ребро (v_n, v_{n+1}). Тогда простой путь будет иметь вид: $v_1, v_2, \dots, v_n, v_{n+1}$ и его длина будет n.

2) При добавленнаии вершины v_{n+1} в турнире будет ребро (v_{n+1}, v_1). Тогда простой путь будет иметь вид: $v_{n+1}, v_1, v_2, \dots, v_n$ и его длина будет n.

3) Не первые 2 пункта, тогда найдётся вершина v_i такая что будут рёбра: (v_i, v_{n+1}) и (v_{n+1}, v_{i+1}). Так как если это не первые 2 пункта, то в турнире будут рёбра: (v_1, v_{n+1}) и (v_{n+1}, v_n). Поэтому если есть ребро (v_{n+1}, v_2), то $i = 1$, если нет, то должно быть наоборот и рассматриваем дальше, но у нас точно есть ребро (v_{n+1}, v_n), поэтому такая вершина точно найдётся (самый максимальный i будет $n-1$)



Тогда наш простой путь будет выглядеть следующим образом: $v_1, v_2, \dots, v_i, v_{n+1}, v_{i+1}, \dots, v_n$ длиной n

Доказали, что всегда найдётся простой цикл в турнире.

2) Построим алгоритм, который его находит.
Понятно, что мы можем найти путь $v_1 \to v_2$. Начнём добавлять туда вершины, чтобы получался простой путь.
Рекурсивный алгоритм: пусть мы уже построили простой путь из k вершин, то есть у нас есть $v_1, v_2, \dots v_k$ покажем, как в него добавляется ещё не использованная вершина: берём произвольную вершину из оставшихся. Проверяем на условия первого и второго пунктов, если они выполняются добавляем эту вершину соответствующим образом, если это не так, то ищем минимальный i для 3го пункта и добавляем после v_i как в пункте 3). Делаем так, пока не будут использованы все вершины.

Корректность алгоритма следует из доказательства наличия простого пути (делается, как шаг индукции).

Сложность: используем двусвязный список, добавление в него работает за $O(1)$. В худшем случае нам придётся для каждой вершины выполнять пункт 3) и всегда минимальный $i = k - 1$. То есть каждый раз нужно будет проходить весь построенный путь. Получается $O(n^2)$.

Задача 3

Задача: в графе G был проведён поиск в глубину. Время открытия и закрытия вершин сохранено в массивах d и f. Постройте алгоритм, который используя только данные из массивов d и f (и описание графа) проверяет, является ли ребро e графа G a) прямым ребром; b) перекрёстным ребром.

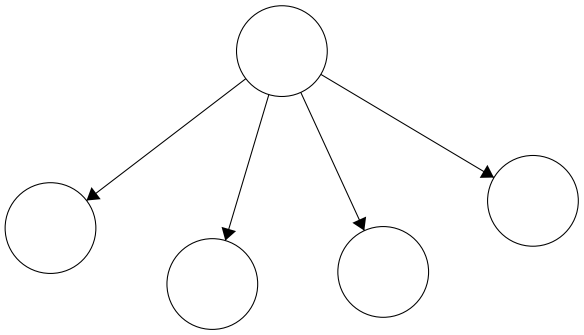
a) Прямое ребро (u, v):
докажем, что уловия: $d[u] < d[v] < f[v] < f[u]$ и $d[v] - d[u] > 1$ необходимым и достаточны для того, чтобы ребро (u, v) было прямым (не совсем, смотреть доказательство).

Ребро (u, v) – прямое, следовательно, это ребро от предка к потомку и не ребро дерева.

То, что это ребро от предка к потомку равносильно тому, что $d[u] < d[v]$ и $f[v] < f[u]$ (так как иначе не выполняется лемма правильной скобочной последовательности, то есть должна быть такая вложенность: [()]). Неравенство $d[v] < f[v]$ выполняется для любой вершины по опре-

делению массивов d и f. Получаем условие: $d[u] < d[v] < f[v] < f[u]$

То, что это не ребро дерева равносильно, что после открытия вершины u открывается не вершина v а другая, то есть между временами открытия есть промежуток $\Leftrightarrow d[v] - d[u] > 1$. Также нужно проверять после нахождения всех «прямых рёбер» ещё такое условие: время закрытия и открытия должны различаться больше, чем на единицу, иначе это будут рёбра дерева (вот пример зачем это нужно, для такого графа дерево поиска выглядит также)



То есть более формально мы должны пройтись по вершинам в которые есть прямой путь из u и проверить для ребра (u, v) чтобы оно не было ребром дерева условия: a – ребёнок. $f[a] \neq d[v] + 1$ и $f[v] \neq d[a] + 1$
При хранении массивов d и f проверка занимает $O(|V|)$ так как в худшем случае (нарисованный пример) все вершины – дети u.

b) Перекрёстное ребро (u, v):
по определению это ребро не от потомка к предку (это бы было обратным ребром), не от предка к потомку (это бы было прямым ребром или ребром дерева). Это равносильно тому, что $d[v] > f[u]$, то есть что мы закрываем вершину u раньше, чем открываем вершину v (скобочное выражение будет выглядить так: $[\ldots]\ldots(\ldots)$)
Проверка этого условия при условии хранения массивов d и f занимает также $O(1)$.

Задача 4

Задача: в государстве между n городами есть m односторонних дорог. Было решено разделить города государства на наименьшее количество областей так, чтобы внутри каждой области все города были достижимы друг из друга.

Предложите эффективный алгоритм, который осуществляет такое разделение, докажите его корректность и оцените асимптотику.

Алгорим: понятно, что в задче предполагается ориентированный граф с n вершинами и m рёбрами и в нём нужно найти все компоненты сильной связности (множество вершин из любой можно добраться в любую). Докажем корректность такого алгоритма:

1) Запускаем поиск в глубину в графе из произвольной вершины

2) Инвертируем грфа (то есть меняем направление всех рёбер на противоположные)

3) Запускаем поиск в глубину из непосещённых вершин с максимальным временем закрытия (этот массив мы получаем из 1) пункта). Вершины,попадающию в одну итерацию образуют компаненту сильной связности.

Корректность: для начала заметим такие 2 факта:

Пусть c_1 и c_2 компоненты сильной связности и из какой-то вершины $u \in c_1$ есть ребро (u, v) где $v \in c_2$. Тогда:

1) Если мы запустим поиск в глубину из любой вершины c_2 тогда все вершины из c_1 не будут найдены (так как есть ребро только в одну сторону) (если будет ребро в другую сторону, то получим одну компоненту связности).

2) Если же мы запустим поиск в глубину из любой верщины из c_1 то по определению DFS все вершины из c_2 закроются раньше любой вершины из c_1. Так как мы всегда дойдем до $u \in c_1$ (так как это компонента сильной связности) затем перейдём по ребру (u, v) обойдём все вершины c_2 и закроем их, так как нет обратного ребра в c_1.

Эти два факта говорят, что все вершины c_2 закрываются всегда позже, чем вершины c_1. То есть если из одной компоненты сильной связности можно попасть в другую компоненту сильной связности, то все вершены закроются раньше в той, в которую можно попасть.

Получается, если у u максимальное время закрытия по всем вершинам графа, то это вершина лежит в компоненте сильной связности, которая является «истоком». То есть из другой компоненты сильной связности в неё нельзя попасть.

После инвертации графа и запуске поиска в глубину из этой вершины u (с максимальным временем закрытия) мы обойдём все вершины этой компоненты сильной связности (так как при инвертации компонента сильной связности не изменится: просто меняется порядок обхода), но больше мы никуда не попадём, так как раньше в эту вершину нельзя было попасть из других компонент сильной связности, а теперь понятно, что из неё нельзя выйти (также основываясь на первых двух фактах из доказательства). Получается всё то что мы обойдём и будет компонентой сильной связности, аналогично делаем для других.

Сложность: нам потребуется константное количество обходов в глубину, которые работают за $O(|E| + |V|)$ и инвертация графа – $O(|E|)$. Получается весь алгоритм работает за $O(|E| + |V|)$

Задача 5

Задача: вам нужно выбраться из лабиринта. Вы не знаете, сколько в нем комнат, и какая у него карта. По всем коридорам можно свободно перемещаться в обе стороны, все комнаты и коридоры выглядят одинаково (комнаты могут отличаться только количеством коридоров). Пусть m - количество коридоров между комнатами. Предложите алгоритм, который находит выход из лабиринта (или доказывает, что его нет) за $O(m)$ переходов между комнатами. В вашем расположении имеется неограниченное количество монет, которые вы можете оставлять в комнатах, причем вы знаете, что кроме ваших монет, никаких других в лабиринте нет, и вы находитесь в нем одни.

Формализуем задачу: у нас есть комнаты – вершины графа. Они между собой связаны m коридорами, по которым можно ходить в 2 стороны (то есть граф неориентированный). Выход из лабиринта – какая-то из комнат, войдя в которую мы поймём, что это выход. Начинаем мы из произвольной комнаты.

Получается наша задача обойти все вершины неориентированного графа начиная с произвольной. Хотим использовать DFS. Для его реализации нам нужно хранить время открытия вершины (время входа в

комнату) и время её закрытия.
Мы можемм ввести также глобальную переменную $time$. Если время открытия какой-то комнаты равно d, то мы кладём $2d$ монет. Если время закрытия равно f то оставляем (отдельно от первой кучки открытия) $2f + 1$ монет (делаем так, чтобы понимать какое число соответствует времени открытия, а какое времени закрытия).
В итоге мы однозначно определяем время закрытия и открытия вершин, следовательно, обходим все вершины графа, используя DFS и находим, есть ли в лабиринте выход или его нет.

Корректность и сложность следует из алгоритма DFS. Сложность: $O(|V| + |E|)$. Из того, как мы формализовали задачу следует, что $|E| = m$, и $|V| \leq \sum\limits_{v \in V} deq(v) = 2|E|$, то есть время работы всего алгоритма $O(2m + n) = O(m)$ (как и требовалось в условии задачи).

Задача 6

Задача: дан орграф на n вершинах ($V = \{1, \ldots, n\}$), который получен из графа-пути (рёбра, которого ведут из вершины i в $i + 1$) добавлением ещё каких-то m данных ребер. Найдите количество сильно связных компонент за $O(m \log m)$.

Изначально у нас был граф-путь. Понятно, что все его вершины образуют собственные компоненты связности. Если мы добавим ребро (a_j, a_i) то вершины $(a_j, a_{j+1}, \ldots, a_i)$ становятся сильно связными. Заметим, что расширяет эту КСС только ребро с меньшим номером, которое идёт от одной из $(a_j, a_{j+1}, \ldots, a_i)$ вершин к ней. Построим алгоритм, опираясь, на эти факты.

Храним m добавленных рёбер в массиве пар вершин. Заметим, что можем сразу отбросить те рёбра, у которых первый номер меньше, чем второй, так как они не внесут новых КСС. Сортируем это массив по первой вершине в порядке убывания.

Идём по этому массиву по первым вершинам и делаем то, что описано выше: для это создаём переменную cur_node. Перваночально $cur\_node = v_1$ (где v_1 – правая вершина первой пары (u_1, v_1)). Рассмотрим случаи:

1) $cur\_node < u_2$ и $cur\_node > v_2$ тогда меняем $cur\_node = v_2$ иначе просто переходим дальше cur_node оставляем прежним

2) $cur\_node \geqslant v_2$ то новые вершины уже не добавятся, следовательно, мы нашли КСС и можем её отделить и начинаем анолагичные действия с u_2 (вершины между u_2 и v_1 будут образовывать каждый свою КСС как и до добавления новых рёбер).

Проходя так как по отсортированным вершинам $u_1, \ldots, u_m$ ищем КСС. Корректность показана в начале решения.

Сложность: сортировка занимает $O(m \log(m))$. Затем проходим циклом – $O(m)$. Получается суммарно $O(m \log(m))$.

Задача 7

Задача: на вход задачи поступает описание двудольного графа $G(L, R, E)$, степень каждой вершины которого равна двум. Необходимо найти максимальное паросочетание в G (которое содержит максимальное количество рёбер). Предложите алгоритм, решающий задачу за $O(|V| + |E|)$.

Заметим, что степень каждой вершины – чётная $\longrightarrow$ это Эйлеров граф (будет содержать себе один или несколько Эйлеров циклов). Это следует из таких рассуждений: от противного, допустим в графе существует вершина с нечетной степенью. Рассмотрим эйлеров обход графа. Заметим, что при попадании в вершину и при выходе из нее мы уменьшаем ее степень на два (помечаем уже пройденые ребра), если эта вершина не является стартовой(она же конечная для цикла). Для стартовой(конечной) вершины мы уменьшаем ее степень на один в начале обхода эйлерова цикла, и на один при завершении. Следовательно вершин с нечетной степенью быть не может. Наше предположение неверно.

Но в нашей задаче степень вершины не просто чётная, а равна 2, следовательно, в каждой вершине мы побываем по 1 разу, то есть будет несколько Эйлеровых циклов без пересечений – простые циклы (так как при входе и выходе мы уже уменьшим степень на 2 и она станет 0, то есть в неё уже нельзя будет войти).

Используем тот факт, что наш граф ещё и двудольный: так как он состоит из объединения простых циклов, то они будут чётной длины (чтобы попасть в себя нужно перейти в другое множество и затем обратно, то есть 2 ребра). Значит $|E|$ - чётно. Также из того, что у нас граф это объединение простых циклов следует то, что $|E| = |V| = 2n$ (понятно, что в простом цикле количество вершин и рёбер совпадает).

Размер максимального паросочетания не сможет превышать n, так как нужно бы было больше концов чем $2n$, а вершин $|V| = 2n$ (принцип Дирихле). Покажем, что n всегда достижимо: для этого мы можем занумеровать рёбра в каждом простом цикле. Их там будет чётное число, так как всего рёбер чётное количество. И будем брать только чётные, понятно, что их концы будут разными, следовательно, это будет паросочетанием. Объединяя все чётные рёбра в простых циклах получаем паросочетание размером n (так как всего $2n$ и берём каждое второе).

Получается в алгоритме: с помощью DFS ищем все простые циклы, затем в каждом из этих циклов нумеруем рёбра и затем выбираем только чётные, получаем максимальное паросочетание (корректность следует из рассуждений выше).

Сложность: нам нужно константное количество запусков DFS, которые работают за $O(|V| + |E|)$ и затем нумерация всех рёбер, то есть $O(|E|)$. Суммарно получаем $O(|V| + |E|)$.

Задача 8

Задача: все степени вершин в неориентированном графе равны $2k$. Все его ребра покрашены в несколько цветов. Предложите $O(V + E)$ алгоритм, который находит в этом графе эйлеров цикл, в котором цвета всех соседних ребер разные (либо выводит, что такого цикла нет).

Используем алгоритм на основе циклов:

1) Берём произвольную вершину v и пока есть цикл проходящий через неё находим его сохраняя порядок обхода а попавшие в цикл рёбра убираем из графа. Это делаем с помощью DFS учитывая, что друг за другом не могут идти рёбра с одинаковым цветом.

2) идём по вершинам, в том же порядке в котором они попадались в первом пункте (храним их, например, в массиве) и делаем то же самое, что и в первом пункте из этой вершины (а её добавляем в ответ)

Корректность алгоритма следует из того, что удаление ребра гарантирует то, что по нему мы не можем пройти два раза. Различность цветов у соседних рёбер учитывается в DFS, также если нужный эйлеров

цикл существует, то мы его построим. От противного: пусть после алгоритма останется непройденное ребро поскольку граф связен, должно существовать хотя бы одно не пройденное ребро, инцидентное посещенной вершине. Но тогда мы должны были бы запустить из неё DFS и так как степень у вершины чётная, то это ребро тоже было бы включено в цикл по построению алгоритма. Противоречие.

Сложность: нам нужно просмотреть кажждую вершину и ребро, поэтому $O(|V| + |E|)$