Домашняя работа

Котов Артем, МОиАД2020

4 февраля 2021 г.

Содержание

Task 1

Условие: Найти в неорграфе простой цикл через данную вершину за $\mathcal{O}(E)$.

Решение.

Запустим из данной вершины `dfs`, тогда мы либо найдем цикл, либо пройдемся по всем ребрам, при мы не запускаемся от вершин, которые не достигли из данной стартовой вершины. Почему мы можем позволить себе именно такой запуск? Потому что поиском из вершины получаем все достижимые вершины из данной вершины, то есть если вершина содержится в простом цикле, т.е. есть путь из нее в нее же по уникальным единожды посещенным (кроме нее самой) вершинам, то запуск поиска из другой вершины, принадлежащей этому же пути, приводит к тому же ответу о принадлежности заданной вершины циклу.

Теперь, если мы запустили поиск из вершины, не принадлежащей циклу, содержащему исходную вершину, но достижимому (в смысле цикл достижим) из стартовой вершины, то по достижении этого цикла мы найдем исходную вершину в нем, если она там есть. Так можно проделать для всех циклов, соответственно, если вершина не принадлежит циклу, то мы бы и так ее не нашли. Случаи, когда мы бы не достигали исходной вершины при запуске из какой-то другой вершины, не очень интересные, так как если мы не достигли её просто поиском, то уже и в циклах, которые мы попутно бы нашли, исходная вершина точно не содержится. Следовательно, почему бы сразу не запустить поиск из исходной вершины. ■

Update: Попробую разбить утверждения для чистоты:

1) Если вершина содержится в цикле, то она содержится в компоненте связности, содержащей этот цикл.

2) Чтобы определить, содержится вершина в цикле, нам, естественно, нет смысла запускать поиск в глубину в других компонентах связности.

3) Ну и раз уже мы из любой вершины (в неорграфе) с поиском в глубину достигаем все вершины той же самой компоненты связности, то можем позволить себе просто запускаться сразу от заданной в условии вершины.

4) В плане пройдемся по всем ребрам имеется в виду, что мы посмотрим в худшем случае все ребра компоненты связности, к которой принадлежит стартовая вершина.

Task 2

Условие: Дано дерево $T = \langle V, E \rangle$. За $\mathcal{O}(V+E)$ вычислить для каждого ребра, сколько простых путей проходит через него.

Решение.

Так как исходный граф — дерево, то любое ребро в нем является мостом, то есть если мы разобьем наш граф на две доли по этому ребру, то оно будет единственным ребром, соединяющим эти доли. Будем это эксплуатировать: рассмотрим какие-то две доли дерева (назовем их “кексы”, потому что очень похожи и так чуть-чуть повеселее).

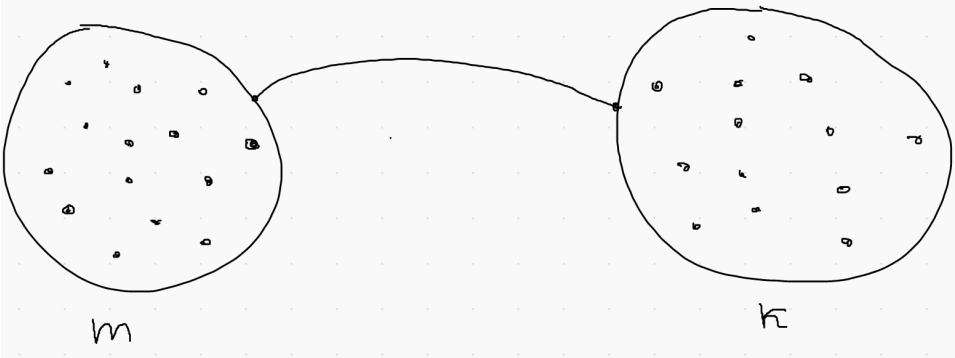

Рис. 1: Два кекса. Количество вершин слева равно m, справа — k, суммарно n

Тогда, количество путей, содержащих рассматриваемый мост равно mn, так как мы из любой вершины слева может достигнуть любую вершину справа. Теперь останется только напечь таких кексов, но это просто: можем для каждой вершины v чем-то похожим на `dfs` посчитать количество вершин в поддереве с корнем v.

Формально это заняло бы $\mathcal{O}(V + E)$, а потом для каждого ребра, как для моста, выпекать два кекса и считать за $\mathcal{O}(1)$ количество путей, так как количество элементов для левого кекса можно взять из корня соответствующего поддерева, а количество элементов в правом кексе есть $n - \#$элементов в левом кексе.

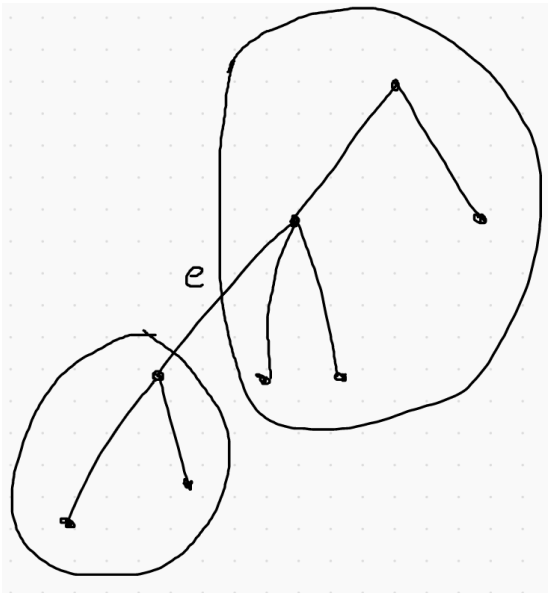

Рис. 2: Поясняющая картинка к дереву и кексам.

Замечание. *Можно заметить, что в дереве, количество вершин $\sim$ количество ребер, то-*

гда реально сложность будет $\mathcal{O}(E)$.

■

Update: Как именно мы считаем количество вершин в поддереве с корнем в v? Это можно сделать следующим образом: мы обновим значение количества вершин в поддереве как сумму вершин в детях плюс одну за сам корень, предположим, что мы поиском в глубину спустились до листа, когда начинаем из него выходить (подниматься наверх) мы обновляем количество вершин в поддереве этого листа (очевидно, что там всего 1 вершина), если у родителя этого листа есть еще дети, то мы спускаемся в них. Для простоты рассмотрим пример, где у родителя всего два листа:

1) мы спустились в первый лист, посчитали количество вершин в поддереве этого листа, как на корне (то есть записали для него 1).
2) вернулись к родителю, увидели второго ребенка
3) спустились во второго ребенка, посчитали количество вершин бла-бла-бла, записали 1.
4) вернулись опять к этому родителю, увидели, что больше детей тут нет и надо подниматься выше
5) обновили значение у этого родителя (сумма значений в детях $+1$ за себя, то есть в этом примере будет записана 3 в эту вершину)
6) продолжаем такую процедуру выше (надеюсь, я понятно переложил мысль в слова).

Task 3

Условие: Найдите лексикографически минимальный из всех топологических порядков. $V, E \leq 10^6$.

Решение.

Так-с, итерация номер...

Рассмотрим множество всех вершин. Посчитаем степень входящих ребер для каждой вершины, в целом, для этого нам надо будет взять вершину и посчитать количество всех входящих ребер, в худшем случае такое нам будет стоить $\mathcal{O}(V + E)$.

Теперь создадим мин-кучу в лексикографическом смысле. Поместим в кучу все вершины с нулевой входящей степенью (назовем такие вершины истоки).

Теперь, чтобы гарантировать, что мы берем минимальный в лексикографическом смысле вершины, будем брать минимальный элемент из кучи (`extract_min`), обозвем его текущей вершиной. Для текущей вершины, во-первых, добавим ее в качестве очередной вершины топсорта, во-вторых, вычтем из каждой соседней с ней вершины 1 из входящей степени (то есть из тех вершин, в которые есть выходящее ребро из текущей вершины), если в процессе возникают вершины, у которых входящая степень стала равна 0, то добавляем их в мин-кучу из истоков, так как эта вершина — новый исток.

Замечание. *В простом варианте топологической сортировки мы помещали истоки в очередь.*

Затем, когда мы перебрали все соседние вершины текущей вершины, мы берем опять минимум из кучи и повторяем процедуру, так будет происходит пока куча не окажется пустой (в целом, это произойдет, когда все вершины не пройдут через кучу).

В худшем варианте, мы будем иметь кучу на V элементах, следовательно, каждая операция добавления новой вершины и удаления минимальной вершины будет стоить нам $\mathcal{O}(\log V)$. Всего вершин $V \implies$ в итоге (с учетом просмотра выходящих ребер) будем иметь сложность $\mathcal{O}(\underbrace{V \log V}_{\text{heap}} + E)$ ■

Task 4

Условие: Про Васю, Петю и вечное желание занять чужое место.

Ох ну и странная, как мне кажется, задачка. Какой-то Xzibit с мониторами получается (простите, не удержался не вставить этот боян).

Решение.

Рассмотрим пары вершин (i,j) исходного графа. Зададим для каждой такой пары вопрос $f(i,j) \geq d$? Если да, то сопоставим такой **паре вершин** вершину в новом графе, которую обозначим так же, как и саму пару. Примечательно, что в самом худшем случае новый граф содержит V^2 вершин (т.е. вообще все пары вершин доступны, и Петя с Васей могут гулять, где хотят).

Теперь надо понять, что есть ребра в новом графе. Рассмотрим ребро $e(a,b)$ исходного графа. Тогда, это мы могли бы использовать это ребро (в смысле, кто-то из друзей мог бы по нему гипотетически пройти), то есть в новом графе между вершинами (a,i) и (b,i) (аналогично для (i,a) и (i,b)) для всех допустимых (с точки зрения функции f) вершин i есть ребро, если в исходном графе есть ребро между (a,b). В самом богатом случае мы бы имели VE ребер (как каждое ребро с каждой вершиной).

Теперь, когда у нас есть такой граф на графе, нам надо найти самый короткий путь из вершины (v,p) в вершину (p,v). Вроде как, можем просто воспользоваться поиском в ширину от (v,p).

Почему поиск в ширину? Потому что в процессе обхода графа поиск в ширину также вычисляет минимальное количество ребер между входной вершиной и каждой достижимой из нее вершиной, что как раз нам и надо. Это можно интуитивно (строгое доказательство что-то большое, как мне кажется) увидеть через сравнение с поиском в глубину: в `dfs` мы спускались глубже сразу же, если могли спуститься глубже, а в поиске в ширину мы видим, что можем пойти, но сначала смотрим куда мы еще можем пойти, то есть поиск в глубину мог бы гипотетически поместить в своем дереве конечную вершину в какой-нибудь очень глубокий лист, даже если реально исходная вершина сразу же соединена с "конечной", а тут мы увидим, что она рядом и сможем сразу к ней перейти (помахал руками в воздухе).

По сложности на составление нового графа мы потратим $\mathcal{O}(V^2 + VE)$, а поиск в ширину стоит нам, вообще говоря, столько же (количество вершин плюс количество ребер). ■

Task 1.1 (Случайный доп.)

Условие: У каждой вершины не более 3 врагов. Вражда – симметричное отношение. Разбить вершины на 2 доли так, чтобы с вершиной в долю попало не более 1 врага. $\mathcal{O}(V + E)$.

Решение.

Честно, я случайно начал ее решать, но раз уже есть какое-то решение, то приведу его.

Представим эти отношения враждебности как ребра на графе, т.е. из условия следует, что степень каждой вершины не больше 3. Заведем два цвета для двух долей: красный и синий. И будем отслеживать активный цвет, например, на старте красный. Запустим поиск в глубину и будем красить вершины в активный цвет. Если наткнемся на ситуацию, в которой число смежных вершин, покрашенных в активный цвет больше 1, то меняем активный цвет на другой.

Почему это, вероятно, работает? Рассматриваем только связные графы, иначе можно было бы разбить задачу на несколько подзадач на каждой из компонент связности. Поиском в глубину можно построить своего рода подвешенное дерево на вершине, из которой запускаемся. Если такое дерево содержало бы вообще все ребра исходного графа, то было бы все просто, так как мы бы просто красили ветки в разные (чередующиеся) цвета (слова передаются с трудом). Но такое дерево не видит ребер, которые могут идти в одной ветви из одной вершины в другую (на семинаре уже "показали", что ребер между ветками быть не может). Тогда если бы у нас была ветка одного цвета, то мы бы покрасили ее неправильно, ну как раз для этого мы в какой-то момент меняли цвет раскраски, чтобы такие вершины были разного цвета. Так как у нас максимум степень вершины 3, то у нас как раз хватит двух цветов, чтобы покрасить ее и ее соседей так, чтобы не было больше одной одноцветной пары для одной и той же вершины (опять сложное перекладывание мысли в слова). ■