Домашнее задание №8

Томинин Ярослав, 778 группа

8 апреля 2018 г.

1.

Алгоритм: Проведем поиск в ширину до тех пор, пока на i-ом уровне мы не найдем вершину t.(Проводить алгоритм поиска в ширину будем проводить с особенностью: каждая вершина будет хранить всех своих предков и уровень.) После этого заберем из очереди все вершины с i-1 уровня и поймем, что если мы рассмотрим количество различных предков вершины t, потом прибавим к этому всех предков у предков вершины t и так далее, то мы получим количество вершин.

Корректность:

Докажем лемму: У двух путей вершины могут совпадать тогда и только тогда, когда они находятся на одном и том же уровне. Это понятно, так как иначе мы бы могли составить меньший путь.(если вершина a встречается на k,j уровнях и k<j, то составим путь, содержаший часть первого пути до уровня к и часть второго пути.)

После последней итерации на i уровне, t будет хранить всех предков, из которых можно добраться до нее за i ребер. Следовательно любой наименьший путь мы сможем востановить из предков. Теперь осталось посчитать количество вершин, для этого достаточно посчитать количество различных вершин на каждом уровне.(в силу нашей леммы)

Сложность по времени: Поймем, что по каждому ребру мы проходим не более чем два раза и на каждый раз на требуется константа времени, чтобы записать предка вершины и добавить ее в очередь. Следовательно наш алгоритм работает за O(n)

2.

1) Докажем от обратного . Допустим, что наш алгоритм нашел, путь большей длины до вершины V. Тогда возьмем наименьший путь, для начала поймем, что в нем нет циклов(иначе мы бы могли убрать этот цикл и путь стал бы короче, так как мы заведомо знаем, что у нас нет циклов отрицательной длины). Так же поймем, что каждая часть этого пути является min расстоянием между вершинами, так как в противном случае мы бы просто заменили часть на меньшую длину и путь бы стал короче. Поймем, что каждая предыдущая вершина будет вызывать релаксацию и добавление следующей вершины в очередь(если она еще не добавлена). Поэтому наш алгоритм найдет путь точно не длиннее чем наименьший. Противоречие

Разберем вариант, когда наш алгоритм выдал расстояние, которое меньше самого короткого пути, но мы знаем, что наш алгоритм строит путь путем релаксации соседних вершин, следовательно если мы из вершины V посмотрим на ее предка, а из предка посмотри еще на предка и т.д. , то мы найдем наш путь и придем к противоречию, потому что он будет короче минимального.

2) Доказательство первого пункта не опиралось на неотрицательность ребер.

Сложность по времени: Рассмотрим сколько раз может быть добавлена вершина в очередь.

3)

3.

Не корректен. Покажем это на примере: если наш граф имел 3 вершины а,б,с и нам нужно было найти расстояние между а и с. Длина ребра (а,с) равна 1, длина (а,б) равна 10, длина (б,с) равна -100. Тогда если мы добавим 100, то путь, содержащий 2 ребра, станет длинее пути, содержащее одно ребро.

4.

Алгоритм: "Схлопнем"вершины, имеющие между собой 0 ребро в одну вершину и отразим это в названии новой вершины(одна вершина будет играть роль нескольких вершин). После этого проведем алгоритм поиска в ширину и найдем расстояние до каждой из вершин.

Корректность: Допустим, что наш алгоритм не находит минимального расстояния для вершины С. Допустим, что наш алгоритм выдает большее число, тогда посмотрим на минимальный путь. В нем содержатся как ребра длины 0, так и ребра длины 1. Поймем, что наш алгоритм отожествляет вершины, которые соединяются ребром длины 0. Следовательно такой путь есть в нашем полученном графе и наш алгоритм выдал длину не большую, чем длина этого пути. Допустим, что наш алгоритм выдает длину меньше, чем она есть на самом деле. Рассмотрим путь в модифицированном графе, добавлением 0 ребер между вершин, которые соответствуют одной вершине мы можем сделать путь, который есть в изначальном графе, так как мы делаем обратную операцию(добавление 0 ребер). Заметим, что эта операция не влияет на длину и это означает, что в изначальном графе содержиться путь с минимальной длиной. Противоречие. Следовательно наш алгоритм работает корректно.

Сложность по времени: Для того, чтобы провести операцию "схлопывания"потребуется O(E), так как для того, чтобы убрать 0 ребро нам надо вычеркнуть его из матрицы и создать массив(новую вершинуБ если его еще нет) и добавить туда вершину. Все ребра, исходящие из такой вершины- все ребра, исходящие из всех вершин из массива. Далее мы проводим поиск в ширину за O(n). Тогда общая сложность O(n).

Но если же наш граф ориентированный, то это решение на него не получится распространить. Можем просто модифицировать алгоритм поиска в ширину. На каждой итерации будем добавлять в начало очереди те вершины, до которых мы добрались по нулевому ребру. Тогда наши инварианты будут поддерживаться. На i-ой итерации будут открыты все

пути, меньшие i+1, до неоткрытых вершин расстояние очень большое и в очереди находятся все вершины, которые достижимы из вершин i уровня, но еще не были открыты. Из этого следует, что наш алгоритм корректен и время его работы аналогично времени работы алгоритма поиска в ширину и равно O(n).

5.

Алгоритм: Обернем все ребра в обратном направлении и заменим веса на противоположные получим следующую задачу: найти минимальный путь от нашей вершины до какой-либо другой, но эту задачу мы умеетм решать за линейное время с помощью Беллмана-Форда, если же расстояние будет стремиться к бесконечности, то это означает, что есть цикл отрицательной длины в измененном графе и наш алгоритм выдаст ошибку.

Корректность: При изменении ориентации все пути, идущие к нашей вершине, станут путями из нашей вершины до остальных вершин.При этом у нас не появится нового пути из нашей вершины(в противном случае сделаем перевертывание ребер и получим противоречие, потому что оккажется, что все таки наш путь является путем от какой-то вершины к нашей вершине, а мы предположили обратное). Далее, когда мы ставим знак - на всех весах, мы просто меняем нашу задачу, так как максимальный путь в 1 задаче будет минимальным путем в нашей новой задаче, а алгоритм по нахождению такого пути мы уже знаем.

Время работы: Поймем, что операция перевертывания ребер- замена места, на котором стоит k в матрице смежности(с места i,j на место j,i) и сразу меняем знак на (-к) это займет $O(\mid E \mid)$(на каждое ребро тратим const). Далее применяем алгоритм Беллмана-Форда за $O((\mid E \mid\mid V \mid))$. Следовательно время работы алгоритма $O(\mid E \mid\mid V \mid)$.

6.

Алгоритм: В нашем алгоритме будем записывать максимальное число вершин независимых и идущих ниже нее. Рассмотрим наши листья, в каждый лист мы запишем 1 по понятным причинам. В каждую вершину на i уровне выше мы будем записывать max(сумма значений всех ее потомков, сумма значений всех потомков от ее потомков+1). Тогда после всех итераций в корне будет записано max значение независимых вершин.

Корректность:

Лемма: Если есть две вершины, которые являются потомками третьей вершины, то все вершины, которые были независимыми в поддереве, образованном первой вершиной и которые были независимыми в поддереве, образованном второй вершиной будут независимы.(Если бы было ребро, то это бы означало, что в дереве есть цикл, а такого быть не может)

Докажем по индукции предположение, что max кол-во вершины A на i-ом уровне-max(сумма значений всех ее потомков(i+1 уровень), сумма значений всех потомков от ее потомков+1(i+2 уровень)).

База: Рассмотрим лист. Для него все соблюдается
Переход: Рассмотрим вершину A на i-ом уровне и для всех ее потомком выполняется наше предположение(число соответствует максимольному кол-ву независимых вершин в поддереве). Тогда рассмотрим максимальный список независимых вершин. Он либо включает нашу вершину A, либо не включает. В первом случае мы понимаем, что все потомки вершины A не включаются в список, следовательно оставшееся количество независимых вершин- это сумма чисел, записанных во всех потомках от потомков A (+1).(Этот случай достигается в силу нашей леммы.)(Если было больше, то это означало, что хотя бы в одном поддереве кол-во независимых вершин превосходило бы число, записанное в вершине, но по предположению индукции такого быть не может.) Во втором случае, когда A не содержится в списке, кол-во независимых вершин равно сумме чисел, записанных в потомках.(Случай достигается в силу нашей леммы.) Докажем, что больше быть не может. От противного: если это не так, то тогда есть такие независимые вершины, которые являются потомками(не прямыми) потомка вершины A и при этом их количество больше, чем число, записанное в этом потомке вершины A.(Иначе бы количество независимых вершин было бы меньше или равно сумме чисел, стоящих в потомках вершины A). Но это противоречит нашему индукционному предположению. Следовательно мы доказали наше утверждение и наш алгоритм действительно находит максимальное количество независимых вершин.
Сложность по времени: В нашем алгоритме мы используем ребро не более, чем три раза(допустим наше ребро соединяет к и к+1 уровни, тогда мы его используем 2 раза, когда находимся на к уровне и один раз, когда находимся на к-1 уровне). Следовательно все арифметические операции будут стоить не дороже, чем $3 \mid E \mid c$, где c-время на одну операцию. Тогда сложность нашего алгоритма равна $O(\mid E \mid + \mid V \mid)$
7.
Алгоритм: Научимся находить цикл отрицательной длины с помощью Беллмана-Форда. Мы уже знаем, что мы умеем индентифицировать отрицательный цикл за O(EV). Исходя из этого мы можем сказать, что путь от начальной вершины до той, которая прорелаксировала на n-ой итерации будет содержать цикл. Отлично, мы можем востановить этот путь, если будем хранить предков в каждой вершине, а из востановленного пути мы сможем найти цикл. Это можно сделать следующим обра-

зом: отсортировать названия вершин и сверить попарно рядом стоящие вершины(предварительно запомнив их номера в изначальном массиве). Такая сортировка затратит $O(V \log V)$
Научившись находить отрицательный цикл, устроим бин-поиск. Просто вычтем из всех вершин определенное число и попытаемся найти отрицательный цикл: если он есть, то это означает, что минимальное среднее значение меньше того числа, которое мы вычли, и поэтому мы вдолжны вычесть меньшее число, иначе большее. Начинаем с 0, следующим идет -0,5M или 0,5M в зависимости от результата. Теперь поймем, что за $\log W$ мы сможем найти то самое среднее значение, затрачивая на каждой итерации O(EV). Следовательно общее время $O(VE \log W)$
8.
Рассуждения можно проводить следующим образом: реализуем с помощью алгоритма поиска в глубину нахождение компонент сильной связности и сделаем сортировку этих компонент так, чтобы ребра шли слева направо. На это мы затратим $O(V + E)$Теперь наша задача свелась к тому чтобы дополнить наш полученный граф так, чтобы у него осталась одна компонента связности. Чтобы ответить на этот вопрос введем обозначения: сток-та вершина, из которой не исходит ребер, исток- та вершина, в которую не входят ребра. Поле этого составим двудольный граф и попытаемся найти полное паросочетание,если оно нашлось, то мы можем соединить стоковою вершину из i паросочетания с истоковой вершиной из i+1 паросочетания и у нас получится цикл. В противном случае добавим аналогично ребра и сожмем получившуюся компоненту связности. Теперь рассмотрим ситуацию и оставшиеся ребра добавим и получим компоненту связности(не сильно понятно как). В результает количество ребер- min(количество стоков, количество истоков)