Домашнее задание 9

Христолюбов Максим, 771

Задание 1

Если вершина v_k находится на кратчайшем пути S_n из v_0 в v_n длинной n, то длинна кратчайшего пути из v_0 в v_k и длинна кратчайшего пути из v_n в v_k в сумме дают n. Действительно, любая часть кратчайшего пути должна быть кратчайшим путем, так как если бы существовал более короткий путь S_k, например из v_0 в v_k, тогда бы путь $S_k + (v_k$ в $v_n)$ был бы короче S_n, что невозможно, так как S_n уже кратчайший. Тогда найдем кол-во v_k, удовлетворяющих этим условиям, они то и будут вершинами, принадлежащим одному из кратчайших путей.

Запустим поиск в ширину и получим массив расстояний A от v_0 до всех вершин графа, в том числе расстояние до v_n равное n. После перенаправим все ребра графа в противоположную сторону (просто возьмем симметричную матрицу смежности графа) и запустим на полученном графе поиск в ширину из v_n. Получим массив расстояний B от v_n до всех вершин полученного графа, что то же самое что расстояние от всех вершин до v_n в исходном графе. Теперь пройдемся одновременно по этим 2-ум массивам и подсчитаем кол-во вершин v_k, таких что $A[k] + B[k] = n$, где k проходит по всем вершинам графа. Как показано выше, такие вершины будут искомыми, а для всех остальных вершин $A[k] + B[k] \geq n$.

Перенаправление ребер займет $O(E)$, так как потребуется пройтись по всем ребрам, 2 поиска в ширину займут $O(V + E)$, проход по всем вершинам для подсчета $A[k] + B[k] = n$ $O(V)$, в целом $O(V + E)$ операций.

Задание 2

1) Модифицированный алгоритм на графах с неотрицательными ребрами будет работать абсолютно так же, по скольку, если вершина извлекается из очереди, то кратчайшее расстояние до нее уже найдено и расстояние до нее никогда не будет релаксировано, и она больше не попадет в очередь. Если же вершина попадает в очередь, это значит, что

она была релаксирована в первый раз, абсолютно так же происходит в классической версии алгоритма.

2) Проблема алгоритма Дейкстры в том, что он жадный и считает что кратчайшее расстояние до извлекаемой из очереди вершины v окончательно найденным. Для алгоритма с неотрицательными ребрами это правильный подход, но в остальных случаях такой путь, что его начало длинее уже найденного пути до v, но сам он короче чем найденный путь до v. Если алгоритм будет заканчивать работу не тогда, когда все вершины пройдены (побывали в очереди), а тогда когда очередь пуста (то есть если до вершины обновилось расстояние, то она помещается в очередь, даже если она уже была когда-то пройдена), то есть расстояния до вершин перестали обновляться, то алгоритм будет работать для графа с отрицательными ребрами. Действительно, если расстояние до уже пройденной вершины обновилось (смена меток) и мы снова поместили ее в очередь, то все вершины в которые можно из нее попасть тоже будут обновленны и найдены правильно, либо если какое-то расстояний до пройденной вершины обновится пересчет повторится вновь. На каждом пересчете длинна пути до вершин уменьшается, а так как в графе нет циклов отрицательного веса, то длинна пути не может быть меньше какого-то определенного значения - длинны кратчайшего пути, значит, когда-то расстояния перестанут обновляться, вершины релаксироваться, и на этом алгоритм закончит действие.

Очевидно, что все подпути кратчайшего пути - кратчайшие, значит если путь $s,v_1 \cdots v$ кратчайший, то точно будет правильно найдено расстояние $l(s,v_1)$ без смены меток, так как это путь из одного ребра. За 1 смену меток будет корректно найден $l(v_1,v_2)$, а значит, и $l(s,v_2)$. Таким образом через $n-1$ смен меток будет найден кратчайший путь до вершины v, так как реберная длина любого пути в графе не больше $n-1$. Значит, сложность $O(n^2)$.

3) Если алгоритм после n^2 релаксаций ребер будет находить еще более короткие пути, значит в графе есть циклы отрицательной длинны. Обнаружить смену можно, например, записав прорелаксированные значения в отдельный массив и сравнив его с предыдущим массивом кратчайших путей.

Задание 3

Нет, не корректен. Например, пусть кратчайший путь проходит через ребра $(a,b) = 2$, $(b,c) = -2$ и существует ребро $(a,c) = 1$. Кратчайший путь из a в c очевидно имеет длину 0, но если сделать все ребра по-

ложительными, добавив к ним 3, тогда $(a,b) = 5$, $(b,c) = 0$, $(a,c) = 3$ и теперь кратчайший путь будет проходить уже через (a,c), и даже не важно каким алгоритмам дальше пользоваться - любой правильный алгоритм поиска кратчайшего пути выдаст неверный ответ - кратчайший путь будет проходить через (a,c).

Задание 4

Будем использовать модифицированный поиск в ширину. Перед тем как вносить информацию в массив о длине пути из v в его потомков, которые еще не были посещены, переберем исходящие из s ребра и, если ребро (v,u), u - потомок v, имеет вес 0, произведем схлапывание u и v: заменим все ребра вида (u,x) и (x,u) ребрами (v,x) и (x,v) соответственно и удалим u. Будем производить схлапывания по вершине v, пока у нее не закончатся смежных ребер длины 0. Заметив, что при этом расстояние от v до каждой вершины не изменится - при извлечении вершины v в очередь кладутся вершины, расстояние от v до которых 1, значит кратчайшее расстояние от s до них равное $S(s,v) + 1$ будет высчитываться корректно. При извлечении вершины придется делать дополнительные операции в кол-ве пропорциональном кол-ву потомков v, но так на этом шаге и в классическом поиске в ширину выполнялось обновление элементов массива расстояний в кол-ве равном кол-ву потомков v, то на асимптотику это не влияет, сложность такая же, как у поиска в ширину, - $O(V + E)$.

Задание 5

Перенаправим ребра орграфа в противоположную сторону. Если $S(u,s)$ - кратчайший путь из произвольной вершины u в s в исходном графе, то $\overline{S}(s,u)$ - кратчайший путь в новом графе из s в u, поскольку длину, так как если бы существовал более короткий путь, тогда бы поменяв направление ребер и получив исходный граф имел более короткий путь из u в s. Запустим в перенапрваленном графе алгоритм Дейкстры и найдем расстояния от s до всех вершин, которые равны расстояниям от всех вершин до s в исходном графе. Взятие максимума по массиву даст вершину, от которой s удалена на максимальное расстояние. Перебор ребер для перенаправления займет $O(E)$, алгоритм Дейкстры $O((V+E)\log V)$, взятие максимума $O(V)$, в целом сложность алгоритма $O((V+E)\log V)$.

Задание 6

Максимальных по размеру независимых множеств может быть несколько, покажем, что одно из них включает в себя все стоки графа. Пусть s - сток. В данном дереве существует максимальное по размеру независимое множество M. Тогда предок v стока s может или входить в M или нет.

Если $v \in M$, то $s \notin M$, тогда рассмотрим M' отличающееся от M только тем, что $s \in M'$, $v \notin M'$. Тогда M' - максимальное независимое множество, так как s соединена только с $v \notin M'$, а все остальные вершины в M и M' совпадают. Таким образом, в этом случае s принадлежит максимальному независимому множеству.

Если же $v \notin M$, то, если $s \in M$, утверждение уже верно, если $s \notin M$, то можно построить независимое множество $M' = M \cup \{s\}$, которое больше M, что противоречит тому что M - максимальное. Следовательно, любое максимальных независимых множество M либо уже включает сток s, либо из него можно получить максимальное независимое множество M', которое включает s.

Значит, всегда можно получить максимальное независимое множество, содержащее s. В силу произвольности выбора s можно провести эти действия для всех стоков и получить максимальное независимое множество, которое будет включать все стоки дерева.

1) Будем закрашивать вершины, заведомо принадлежащие максимальному независимому множеству, черными, а не принадлежащие - белым, и вносить их в соответствующие массивы. Тогда, когда все вершины будут закрашены, множество черных вершин будет искомым.

Покрасим все стоки черным, так как они принадлежат максимальному независимому множеству, а их предков белым. Удалим все черные вершины из дерева. Тогда все стоки нового дерева - белые. Действительно, все старые стоки были черными и были удалены, значит новые стоки или белые, или незакрашенные, но все новые стоки раньше не были стоками, значит были получены удалением потомков, которые черные, значит, они белые.

Удалим из дерева все стоки (они все белые), их можно найти перебором белых вершин и проверкой. Некоторые стоки полученного дерева могут быть белыми, удалим их. Будем удалять белые стоки, пока не получим дерево со всеми бесцветными стоками.

На этом итерационный шаг заканчивается - все эти действия можно применить к полученному дереву снова, так как все стоки бесцветны, их некоторые предки - белые, а бывшие потомки текущих стоков были белыми, значит можно закрасить стоки черным, а всех их предков белым. Заметим, что все действия не привели к изменению искомого максималь-

ного независимого множества, так как удалялись только закрашенные вершины, заведомо входящие или не входящие в искомое множество. Будем повторять итерации, пока все дерево не будет закрашено, а значит, найдено искомое множество.

Сложность поиска стоков в первый раз $O(V + E) = O(V)$ (для дерева), перебором всех ребер и записей в массив вершин, у которых есть потомки, а после перебором вершин и отбором тех, что не вошли в массив вершин, имеющих потомков.

После нахождения запишем их в список и будем перебирать элементы списка стоков с конца, и при удалении заносить в конец списка предка удаляемой вершины. Удалять все черные стоки будем простым перебором и удалением элементов с конца массива, пока не дойдем до начала массива.

При удалении белых стоков, пока таковых не останется в дереве, будем кроме того хранить указатель на изначальный конец списка на каждой итерации, чтобы заканчивать перебор элементов с конца списка на этом элементе, для того чтобы не проходить по уже заведомо бесцветным стокам, которые не требуется удалять. Таким образом не придется каждый раз при удалении белых стоков перебирать все стоки для поиска белых. Тогда операция удаления стоков на каждой итерации i займет $O(S_i)$, где S_i - кол-во стоков, которые были удалены. Значит, за все итерации будет произведено $O(S_1) + .. + O(S_k) = O(S_1 + .. + S_k) = O(V)$ действий, сложность алгоритма $O(V)$.

2) Это же решение можно реализовать иначе, например, так:

Проводим поиск в ширину, полученный массив сортируем по возрастанию длины пути. Заносим в искомое множество вершины с максимальной длинной пути и удаляем их из дерева вместе с их родителями - это можно сделать так как к их родителям подсоединены только вершины с максимальной длинной пути, которые тоже буут удалены. Это один итерационный шаг, на k-ом каждом шаге из дерева будут удалены вершины с длинной пути $\max(l) - k + 1$, после делении всех вершин получим искомое множество, док-во корректности совпадает с приведенным выше. Но в этом случае потребуется быстрая сортировка, а значит сложность будет больше - $O(V \log V)$.