Домашнее задание №7

Томинин Ярослав, 778 группа

30 марта 2018 г.

1.

Докажем это от противного. Допустим, что это не так, тогда поймем, что найдется хотя бы один цикл. Рассмотрим этот цикл. Найдем из этого цикла вершину с наибольшим номером, мы знаем, что в эту вершину есть ребро из другой вершины. Следовательно из врешины с меньшим номером идет ребро в вершину с большим номером. Противоречие. Доказали что граф DAG.

2.

Алгоритм: Докажем для начала, что турнир содержит простой путь с n-1 вершиной. Сделаем это по индукции:

База:n=2. Очевидно сущестует путь длины 1

Переход: допустим, что у нас есть простой путь для n вершин, добавим n+1 вершину и составим путь длины n. Вершины в пути длиной n-1 обозначим как 1,2,3,...,n. Тогда посмотрим есть ли ребро из n+1 в 1: если есть, то мы составили путь n+1,1,2,...,n; иначе перейдем на следующую и посмотрим на наличие ребра между n+1 и этой вершиной. Допустим, что мы нашли это ребро на i вершине. Тогда составим путь 1,2,3,...,i-1,n+1,i,...,n. Если же мы не нашли такого ребра, то просто добавим эту вершину в конец пути.

Теперь наш алгоритм будет заключаться в следующем мы будем строить сначала путь длины 2, потом дополнять его до пути длины 3 и так далее до длины n.

Корректность: Мы доказали, что любой путь мы можем дополнить до большей длины, поээтому мы всегда сможем получить путь длины n.

Время работы:Наш алгоритм, составляя из пути длины i, путь длины i+1 i раз обащается к массиву ребер за O(1). Следовательно общая сложность равна

$$\sum_{i=2}^{n} O(i) = O(n^2)$$

3.

а)**Алгоритм:** Рассмотрим ребро e (a->в). И рассмотрим время открытия вершин a,b. Если выполняется условие d[a]<d[b]<f[b]<f[a] и при этом ни для одной вершины, смежной с a (smeg) не выполняется d[b]=d[a]+1 или f[smeg]+1=d[b], то это прямое ребро.

Корректность: Докажем, что если выполняется условие d[a]<d[b]<f[b]<f[a], то ребро может быть либо прямым, либо ребром дерева. Это выражение означает, что b достижимо из a, но a не достижимо из b. Но тогда b является потомком a, а к предка и потомка может соединять либо ребро дерева, либо прямое ребро. Условие d[b]=d[a]+1 или f[smeg]+1=d[b] для

всех вершин smeg, смежных с a говорит нам при его выполнении, что это ребро дерева(так как вершина b должна быть соединена с a), а иначе это ребро будет прямым.
Время работы: Нашему алгоритму требуется рассмотреть время открытия и закрытия $2+deg_+a$ вершин, по условию доступ осуществляется за O(1). Следовательно время работы равно $O(deg_+a) = O(n)$, где n - количество вершин в графе.
б)**Алгоритм:** Аналогично первому пункту узнаем степени наших вершин a,b. Если выполняется условие, что d[a]<f[a]<d[b]<f[b] или d[b]<f[b]<d[a]<f[a] , то наше ребро является перекрестным.
Корректность: Поймем, что ребро перекрестное, если a не является предком b и b не является предком a. Тогда, если один из них откроется первым, то он закроется раньше, чем второй так как второй не является потомком первого. **Время работы:** Узнаем время открытия и закрытия двух вершин. Следовательно O(1).
4.
Алгоритм: Поймем, что в этой задаче от нас требуется найти компоненты сильной связности. Для этого нам нужно проделать следующий алгоритм. Проведем поиск в глубину и отсортируем вершины в порядке убывания времени закрытия. Обернем ребра в графе и Рассмотрим до каких вершин мы сможем добраться начиная с вершины с максимальным временем закрытия, в которой мы еще не побывали. (переходить можно только в те вершины в которых мы еще не были). Тогда наши вершины разобьются на компоненты сильной связности.
Корректность: Докажем, что в нашем графе конденсаций все ребра идут от компоненты с большим временем закрытия к компоненте с меньшим временем.
Допустим, что ребро e соединяет компоненты (C,K). Тогда если сначала открылась компонента C, то из нее будут достижимы и все точки компоненты K, следовательно время закрытия всех точке компоненты K будет меньше, чем время закрытия точки из C с которой мы начали.
Если же мы попали в компоненту K, то мы не сможем попасть из нее в компоненту C, поэтому время закрытия всех точек компоненты K будет меньше6 чем время открытия компоненты C, а, следовательно, время закрытия точек компоненты C будет больше времени закрытия точек компоненты K. Следовательно f[C]>f[K].
Теперь поймем, что если мы транспонируем наш граф, то транспонируется и наш граф конденсаций, это означает, что если мы пойдем из вершины с наибольшим временем закрытия по графу, то мы попадем только во все вершины сильной связности, соответствующие компоненте связности с наибольшим уровнем закрытия. Далее удалим эти вершины

из рассмотрения и найдем из непомеченных вершин вершину с максимальным временем закрытия, аналогично мы знаем, что в транспонированном графе из компоненты связности, ссответствующей этой вершине не исходит ни одного ребра(мы не рассматриваем помеченные вершины и идущие в них ребра). Поэтому мы всегда сможем находить компоненты связности, начиная от компоненты с наибольшим временем закрытия и заканчивая компонентой с наименьшим.

Время работы: Наш алгоритм делает два обхода в глубину, поэтому его сложность O(|V|+|E|).

5.

Алгоритм: Если мы приходим в комнату у которой в центре еще не лежит монет, то мы будем класть тута кол-во монет, соответствующее ее порядку прохождения. Так же после выхода из коридора мы будем класть около выхода кол-во монет, соответствующее той комнате, из которой мы пришли. Мальчик каждый раз будет выбирать коридор, у которого перед выходом не лежит ни одной монетки, ложить монетку и идти в него. Если же все коридоры помечены, то мальчик будет возвращаться обратно.(до тех пор, пока он не дойдет либо до комнаты с непомеченными коридорами, либо до комнаты с номером 1 и со всеми помеченными коридорами) Тогда, если мальчик вернется в 1 комнату и все ее коридоры будут помечены, то это будет означать, что у этого лабиринта нет выхода.

Корректность: Заставим мальчика запоминать последовательность комнат, которые он посетил. Когда у мальчика не будет вариантов выбора он будет возвращаться по коридорам до тех пор, не появится выбор. Поймем, что по всем коридорам он проходит не более, чем 2 раза. Так как он идет по помеченному коридору с меткой той комнаты, из которой он попал в эту комнату. Если же он уже поднимался по этому коридору, то это означает, что он два раза попал из предыдущей комнаты в эту комнату. А такого быть не может. Противоречие.

Теперь поймем, почему мальчик побывает во всех комнатах лабиринта. Докажем что он пройдет по всем ребрам хотя бы 1 раз. Мальчик всегда возвращается по обратному пути из комнат, в которых уже нет непомеченных коридоров. Следовательно он пройдет по всем коридорам, исходящих из комнат, в которых он побывал. Рассмотрим это множество комнат, если часть комнать осталась непосещенной, то это означает, что нет ни одного коридора соединяющие эти две кучки комнат(так как иначе был бы непосещенный коридор). Но у нас граф имеет одну компоненту связности. Следовательно мальчик пройдет через все комнаты или найдет выход.

Сложность по времени: Мальчик проходит по всем ребрам не более

2 раз(это доказано в корректности). Следовательно наш алгоритм обладает заявленной сложностью O(m).

6.

Алгоритм: Отсортируем ребра, которые ориентированны влево, в порядке возрастания той вершины, в которую они идут.(если отрезки указывают в одинаковую вершину, то берем тот, у которого больше номер точки, из которой он исходит). Создадим переменную MAX, которая будет содержать в себе максимальную вершину, из которой исходило одно из рассмотренных ребер.(в начале равна вершине, из которой исходит 1 ребро) Тогда посмотрим на 1 элемент массива. Он увеличивает количество компонент на 1. При добавлении следующего сравниваем левую координату с MAX, если левая координата больше MAX, то увеличиваем кол-во компонент на 1, иначе в MAX кладем max(правая вершина ребра, MAX). Так делаем де тех пор, пока не закончатся ребра.

Корректность: Поймем, что если у нас граф состоял из к компонент связности и мы прибавили к нему ребро, идущее от i к j, где i<j, то количество компонент не изменилось. Докажем от обратного. Допустим, что кол-во компонент изменилось, тогда появился путь между вершинами u и v, которого не было. Но любой такой путь, содержащий новое ребро, можно заменить j-i-1 ребрами, соединяющие соседние верщины. Следовательно такой путь был. Противоречие.

Покажем еще одно наблюдение: если к пути прибавить ребро идущее от i к j, где i>j, то у нас появится компонента сильной связности из вершин i...j.(будет цикл).

Теперь будем добавлять к нашему пути по одному ребру каждый раз. Мы знаем, что добавление ребер, идущих направо, не пеняет количество компонент.

Докажем лемму: если i и j принадлежат одной компоненте связности, то все вершины между i и j принадлежат той же компоненте связности. Это следует из того, что мы можем найти путь от i вершины до к вершины и обратно. Тогда, если мы расположим вершины на координатной прямой в порядке их нумерации, то наш отрезок будет делиться на непересекающимися отрезками компонент связности.(Потому что если они пересекаются, то это одна компонента связности.)

В нашем алгоритме, если левая координата больше, то увеличиваем кол-во компонент на 1.(Это означает, что наше ребро сделало путь i...j циклом и это означает, что количество компонент увеличилось на 1)

Если же левая координата меньше MAX, то это означает по лемме, что количество компонент не увеличилось, но последняя компонента стала больше(если правая вершина больше MAX, в этом случае мы должны записать в MAX правую вершину). Иначе (если правая вершина меньше

MAX) последняя компонента не изменяется и мы продолжаем добавлять ребра.

Сложность по времени: В нашем алгоритме происходит сортировка максимум m элементов(это займет $O(n \log n)$) и максимум m сравнений и изменений значений MAX. Следовательно наш алгоритм работает за $O(n \log n)$

7.

Алгоритм: Докажем лемму, что паросочетание максимально тогда и только тогда, когда мы не можем найти увеличивающую относительно него цепь. Необходимость доказывается просто: если же существует увеличивающая цепь, то вычтем из нее все ребра текущего паросочетания и получим новое, так как из одной вершины будет выходить только одно ребро, поймем, что это паросочетание будет больше предыдущего на один в силу того, что цепь была увеличивающей. Достаточность: для ее доказательства предположим, что у нас есть два просочетания K и P. Допустим, что P максимальное и у K нет увеличивающей цепи, тогда получим граф Q, который будет состоять из ребер K и P за вычетом тех ребер, которые содержаться и в K, и в P. Тогда ребра будут составлять либо пути, либо циклы(не будут пересекаться друг с другом в силу того, что степень каждой вершины не больше 2). Докажем, что циклы и пути имеют четную длину и состоят из чередования ребер из K и P. Докажем для пути. Если же его длина нечетна, то это означает, что либо для K, либо для З существует увеличивающая цепь. Но этого быть не может, так как P максимально, а у K нет увеличивающей цепи. Следовательно все пути четны и ребра в них чередуются исходя из построения. В циклах обстоит все так же, если мы предположим, что цикл нечетной длины, то это будет означать, что из одной вершины исходит два ребра из одного парообразования, а этого не может быть.Следовательно P равно K и мы пришли к противоречию.

В алгоритме будем просматривать вершины v первой доли графа. Если текущая вершина v уже насыщена паросочетанием (т.е. уже выбрано какое-то смежное ей ребро), то эту вершину пропускаем. Иначе — алгоритм пытается насытить эту вершину. Поиск будем выполнять следующим образом: просматриваем все рёбра из этой вершины. Если вершина i_0 ещё не насыщена паросочетанием, то увеличивающая цепь (v,i_0) Иначе продолжаем обход в глубину. Когда находим увеличивающую цепь, то применяем нашу теорему. Если в итоге увеличивающей цепочки не найдено, то ее не существует, то есть мы нашли максимальное паросочетание в силу теоремы, доказанной выше.

Сложность по времени : DFS вызываем не больше, чем n раз, следовательно $O(n^3)$, где n - число вершин.