Домашнее задание 7

Христолюбов Максим, 771

Задание 1

Т.к. ребра идут только от вершин с большим номеров к вершинам с меньшим номером, то из вершины с номером n можно попасть только в вершины с номером $k < n$, значит, в вершину с номером n нельзя попасть из вершины n. Так как это верно для всех n, то граф - DAG.

Задание 2

Очевидно для турниров с 2 вершинами утверждение верно. Пусть оно верно для турниров с k вершинами, рассмотрим турнир с $k+1$ вершиной. Выберем вершину v_0. Тогда по нашему предположению в оставшихся k вершинах есть путь длины $k-1$ v_1,v_2,...v_k. Если в последовательности, описывающей путь есть v_i,v_{i+1} такие, что существует ребро из v_i в v_0 и из v_0 в v_{i+1}, то путь длины k v_1,..v_i,v_0,v_{i+1},..v_k - искомый. Если таких v_i,v_{i+1} не существует, то возможны 2 варианта. Если есть ребро (v_1,v_0), то значит все ребра ориентированы в v_0, следовательно, последовательность v_1,..v_k,v_0 - искомая, если же есть ребро (v_0,v_1), то последовательность v_0,v_1,..v_k - искомая. По индукции получаем требуемое утверждение.

Алгоритм поиска пути A_n выглядит так

$A_n = waysearch(G_n)$:

if $n = 2$

return: путь из одной вершины графа во вторую

Удаляем произвольную вершину v_0

$A_{n-1} = waysearch(G_{n-1})$

Из A_{n-1} получаем A_n прибавляя к A_{n-1} вершину v_0 как показано после предположения индукции.

return A_n

В худшем случае на каждом k-ом шаге придется перебирать все $k-1$ вершин на существования ребра из них в v_0 и делать фиксированное кол-во операций. Всего рекурсивных вызовов $n-2$, $\sum\limits_{k=3}^{n} k-2 = \Theta(n^2)$ операций

Задание 3

а) Прямым ребром является ребро графа, соединяющее предка и потомка в дереве, но не входящего в него. Значит вторая вершина B ребра e должны быть в поддереве первой вершины A, т.е. время $d_A < d_B$ и $f_B < f_A$, но A - не родитель B. Проверить выполнение данных условий можно за $\Theta(1)$

б) Перекрестное ребро - ребро графа, которое соединяет вершины, которые не являются для друг друга предком и потомком. Значит, $f_A < d_B$ или $f_B < d_A$

Задание 4

Воспользуемся алгоритмом поиска сильных компонент связности, они и будут этими областями. Для этого проведем топологическую сортировку, проведя поиск в глубину и расположив вершины в порядке убывания закрытия вершин. В сопряженном графе проведем поиск в глубину по вершинам в порядке топологической сортировки. Каждое дерево будет компонентой сильной связности, так как из всех вершин, в которые можно попасть из начальной, так же можно попасть в начальную, поскольку начальная находится левее в топологической сортировке сопряженного (изначального) графа. Ни в какие же лишние попасть не получится, так как из вершины можно попасть только в лишнюю вершину, которая находится левее в топологической сортировке сопряженного графа, но во всех них мы уже побывали. Таким образом, полученные компоненты сильной связности - искомые области.

Задание 5

Будем обходить лабиринт поиском в глубину и искать выход, если после обхода всех комнат лабиринта (вершин графа) выхода не будет, значит его нет. Для того чтобы не запутаться по каким коридорам (ребрам) уже проходили, при переходе в другую комнату у входа в коридор будем класть k монеток, где k - номер комнаты (нумерация с 1 - начальной комнаты, в каждой новой посещенной комнате увеличивается на 1), если мы по нему сейчас пойдем. Так же при входе в $k+1$ комнату у того прохода из которого пришли кладем k монет. Это необходимо чтобы различать ребра по которым мы пришли в вершину и ушли. Очевидно, что по коридорам, помеченным монеткой не надо ходить. Если мы пришли в комнату, а в ней у одного из проходов уже есть монетки, значит мы здесь уже были и надо вернуться назад, не оставляя никаких монет в этой комнате. Если все коридоры, выходящие из комнаты были пройдены, то возвращаемся по коридору с наименьшим кол-во монет. Таким образом реализуется поиск выхода в глубину, так как по каждому коридору мы пройдем не более 2 раз, то за $O(m)$ переходов получится найти выход.

Задание 6

Если добавленное ребро соединяет предка со своим потомком, то оно не влияет на связность графа. Если же ребро соединяет потомка r с предком l, то появляется цикл из вершин $(r-l)$. Если добавленны 2 ребра v и u ($l_v \leq l_u$ для определенности), причем $l_u \leq r_v$, то группа вершин $(l_v - max(r_v, r_u))$ - компонента сильной связности. Связность такого графа можно изобразить графически на прямой, на которой отмечены вершины $a_1,...a_n$, а добавленные ребра - это отрезки $[r;l]$. Тогда если отрезки пересекаются, то их объединение образует компоненту сильной связности. Задача сводится к нахождению отдельных объединений отрезков, после нанесения на прямую m отрезков.

Отсортируем добавленные ребра (отрезки) по возрастанию l_k, тогда если $l_{k+1} \leq r_k$, то объединим отрезки в один $[l_k; max(r_v, r_u)]$. Будем сравнивать полученный отрезок с $k+2$, пока не получим, что $l_{p+1} \geq r_p$, где r_p - правый конец отрезка полученного на предыдущем шаге. Т.к. отрезки отсортированны по возрастанию это будет значит конец цельного объединение отрезков, которое является компонентой сильной связности. Перейдем к следующему отрезку, найдем следующую компоненту сильной связности. Будем повторять эти действия начиная с $k=1$, пока массив вершин не закончится. Полученные объединения $[b_p; c_p]$ - искомые компоненты связности из вершин от b_p-ой до c_p-ой.

На быструю сортировку добавленных ребер уйдет $\Theta(m \log m)$ операций, на объединение отрезков $\Theta(m)$ операций, всего - $\Theta(m \log m)$.

Задание 7

Исходный граф состоит из компонент связности. В каждой компоненте связности существует эйлеровый цикл, поскольку степень каждой вершины четна. Построим эйлеровый цикл по алгоритму поиска эйлерова цикла (лекция).

Запустим на графе поиск в глубину, начиная с какой-то вершины, определяющую первую компоненту связности, причем будем заносить в 2 массива L_k и R_k посещенные вершины из L и R и считать их количество. Когда вершины, достижимые из той, откуда начался поиск в глубину, закончатся все вершины из этой компоненты связности. В ней ровно $min(|L_k|,|R_k|)$ ребер из паросочетания. Действительно, так как по компоненте связности построен эйлеров цикл, то ребра, соединяющие вершины из $V = min(L_k, R_k)$ с следующими за ними вершинам в эйлеровом цикле, будут входить в паросочетание. Докажем, что оно максимально. Так как все вершины из $V = min(L_k, R_k)$ уже задействованы, то больше ребер с незадействованными вершинами нет, значит, это паросочетание - максимально.

Проведем поиска максимального паросочетания по остальным компонентам связности, просто беря еще не пройденную алгоритмом поиска в глубину вершину. Тогда максимальным паросочетанием для графа будет обьединение максимальных паросочетаний его компонент связности.