1.

1. Ориентированный граф:

Вершина t достижима из s если она является потомком в дереве при обходе в глубину, а также, если они соединены обратным или перекрестным ребром.

Произведем обход графа в глубину (DFS) с запоминанием времен открытия и закрытия вершин в массивы d и f соответственно.

а) Тогда условием того, что вершина t является потомком s будет: $d[s] < d[t]$ и $f[t] < f[s]$ (по лемме о скобках, так как, если расставить для s скобку круглую, а для t – квадратную, то получим [()]). То есть, отрезок $(d[s], f[s])$ полностью содержится в отрезке $(d[t], f[t])$.

б) Условие, что s и t соединены обратным ребром: $d[t] < d[s]$ и $f[s] < f[t]$. Аналогично предыдущему, только теперь отрезок $(d[t], f[t])$ полностью содержится в отрезке $(d[s], f[s])$.

в) Условие, что s и t соединены перекрестным ребром: $d[t] < f[t] < d[s] < f[s]$. То есть, отрезки $(d[s], f[s])$ и $(d[t], f[t])$ не пересекаются.

2. Неориентированный граф:

Неориентированный граф можно рассматривать как ориентированный на V вершинах с $2|E|$ ребрами: как в ориентированном графе и их транспонированных (перевернутых).

Тогда в этом случае условие достижимости t из s будет: $d[s] < d[t]$ и $f[t] < f[s]$ или $d[t] < d[s]$ и $f[s] < f[t]$, так как в зависимости от того, с какой вершины стартует обход в глубину, алгоритм может пройти от t к s или от s к t, а так как граф неориентированный, и то, и другое условие дает достижимость.

При поиске в глубину в неориентированном графе любое ребро оказывается либо ребром дерева, либо обратным, так что оба случая учтены таким образом.

Доказательство этого факта:

Пусть (u, v) – произвольное ребро графа, и пусть, например, $d[u] < d[v]$. Тогда вершина v должна быть обнаружена и обработана прежде, чем закончится обработка вершины u, так как v содержится в списке смежных с u вершин. Если ребро (u, v) первый раз обрабатывается в направлении от u к v, то (u, v) становится ребром дерева. Если же оно первый раз обрабатывается в направлении от v к u, то оно становится обратным ребром.

Корректность:

Алгоритм базируется на лемме о скобках.

Сложность:

Обход в глубину, согласно лекции стоит $O(|V| + |E|)$, а проверка условий на времена открытия и закрытия – O(1). Тогда итовая сложность: $O(|V| + |E|)$.

2.

Простой путь длины $n - 1$ содержит все вершины по одному разу, то есть, это гамильтонов путь.

Доказательство существования:

Докажем по методу математической индукции.

1. Для $n = 1$ утверждение очевидно. Получится простой путь длины 0.

2. Предположим, что это верно для турнира с $n = k$ вершинами.

3. Докажем, что это верно для турнира T с $n = k + 1$ вершинами.

Выделим в этом турнире вершину v_0, тогда $v_1 v_2 ... v_k$ – гамильтонов путь в $T \setminus v_0$. Если $i \in 1, ..., k$ – максимальное число такое, что для любого $j \leq i$ существует ребро из v_j в v_0, тогда $v_1 v_2 ... v_i v_0 v_{i+1} ... v_k$ – искомый путь.

Алгоритм:

Пусть s – искомый путь.

Вначале s пустой. Затем вставляем в него все вершины по очереди по индукции: если уже есть путь из k вершин и нам надо добавить v_{k+1}, то находим такой максимальный номер i, что для любого $j \leq i$ существует ребро из v_j в v_{k+1}.

Вставляем v_{k+1} после v_i и получаем путь: $v_1 v_2 ... v_i v_{k+1} v_{i+1} ... v_k$.

Такую процедуру проделываем для всех вершин.

Корректность:

Построение алгоритма следует из доказательства его существования по методу математической индукции.

Сложность:

На вставке каждой вершины необходимо проверить наличие ребра между v_{k+1} и $v_1, v_2, ... v_k$ в худшем случае вершинами. То есть, операций: $1 + 2 + ... + n = O(n^2)$. Таким образом, итоговая сложность: $O(n^2)$.

3.

Пусть u – начало ребра e и v – конец ребра e.

а)

1. Условие достижимости: $d[u] < d[v]$ и $f[v] < f[u]$

Но этого недостаточно, так как ребро может быть ребром дерева, тогда потребуем, чтобы

2. $\exists y : u \to y \to v$, тогда e – не ребро дерева.

б) Отрезки времен $(d[u], f[u])$ и $(d[v], f[v])$ не должны пересекаться, то есть получим, как на лекции:

$d[u] < f[u] < d[v] < f[v]$ или $d[v] < f[v] < d[u] < f[u]$, что можно упростить до:

$f[u] < d[v]$, так как перекрестные ребра всегда идут справа налево, если изобразить это в оси времени и скобочной структуре (было показано на лекции).

Корректность:

Алгоритм базируется на лемме о скобочной последовательности. Все выкладки были на лекции.

Сложность:

Обход в глубину, согласно лекции стоит $O(|V| + |E|)$, а проверка условий на времена открытия и закрытия – O(1). Тогда итовая сложность: $O(|V| + |E|)$.

4.

1. По сути, это поиск компонет сильной связности в графе, где вершинами являются города, а ребрами – дороги.

Алгоритм был продемонстрирован на лекции: производим обход в глубину в графе, строим транспонированный граф (переворачиваем все ребра), в этом транспонированном графе производи обход в глубину по вершинам в порядке убывания времени закрытия при обходе исходного графа.

Корректность:

Внутри области все города достижимы друг из друга – это в точности определение сильно связной компоненты графа. Корректность алгоритма поиска таких компонент была доказана на лекции.

Сложность:

Два обхода в глубину, так что итоговая сложность $O(n+m)$.

2. Будем действовать аналогично задачи №2. Для начала представим, что между всеми городами есть ровно одна дорога, то есть, построим турнир на этих вершинах. А затем выделим гамильтонов путь, который будет состоять из тех дорог, которые и необходимо построить.

Пусть s – искомый путь.

Вначале s пустой. Затем вставляем в него все вершины по очереди по индукции:

Если уже есть путь из k вершин и нам надо добавить v_{k+1}, то находим такой максимальный номер i, что для любого $j \leq i$ существует ребро из v_j в v_{k+1}.

Вставляем v_{k+1} после v_i и получаем путь: $v_1 v_2 ... v_i v_{k+1} v_{i+1} ... v_k$.

Такую процедуру проделываем для всех вершин.

Корректность:

Гамильтонов путь – простой путь, проходящий через каждую вершину ровно один раз. Таким образом, мы обеспечим проходимость от любой вершины до каждой. А также минимальность, так как этот путь проходит через каждую вершину ровно один раз.

Сложность:

На вставке каждой вершины необходимо проверить наличие ребра между v_{k+1} и $v_1, v_2, ...v_k$ в худшем случае вершинами. То есть, операций: $1+2+...+n = O(n^2)$. Таким образом, итоговая сложность: $O(n^2)$.

6.

Дополнительные ребра добавляются к графу-пути, так что между двумя любыми вершинами в одном направлении (для определенности слева направо) всегда есть путь.

Необходимо найти компоненты сильной связности, то есть, группы вершин, между любыми из которых есть путь. Заметим, что, если между вершинами i и $i-k$ (вершина, расположенная на k вершин левее) есть ребро, то между всеми вершнами, лежащими в промежутке между ними есть путь. Таким образом, необходимо найти, куда можно максимально в обратном направлении переместится из данной вершины (рассматривать вершины удобно с конца, ведь в прямом направлении, условились, что прямое – это слева направо – путь есть всегда в силу графа-пути). Необходимо смотреть, куда можно максимально переместится, так как ребер из данной вершины может быть несколько. Все вершины между данной и той, куда переместимся, будут входить в одну компоненту связности. Далее необходимо проверить, можно ли из вершины, куда переместились, передвинутся еще назад по дополнительным ребрам. Если да, то эти вершины, между той, с которой стартуем и куда приходим, также принадлежат этой компоненте связности. Если переместится уже нельзя, то увеличиваем счетчик компонент связности и переходим к следующему ребру. Также необходимо иметь переменную, в которой хранится общее число вершин, вошедших в какую-либо компоненту связности.

Таким образом, необходимо, отсортировать по убыванию пары вершин, задающих дополнительные ребра по началу этого ребра. Также, если начало ребра меньше, чем конец, то это ребро в прямом направлении, и оно не интересно, так как путь в прямом направлении между двумя вершинами заведомо есть, такие ребра можно отбросить.

Затем по очереди брать эти ребра и реализовывать написанное выше, рекурсивно к каждому ребру. При нахождении очередной компоненты сильной связности увеличивать счетчик $count$ на единицу и увеличивать переменную $count\_all$, которая хранит общее число вершин в компонентах сильной связности.

Чтобы найти итовое число компонент сильной связности, необходимо не забыть про вершины, которые не соединены дополнительными ребрами и которые не будут просмотрены. То есть, итоговое число компонент сильной связности можно вычислить как $count + (n - count\_all)$.

Корректность:

Алгоритм проходит все дополнительные ребра, а только благодаря им могут появиться компоненты сильной связности, отличные от единичной вершины. Все вершины, которые не вошли в эти компоненты, будут вырожденными компонентами с одной вершиной.

Сложность:

Сортировка дополнительных ребер быстрой сортировкой: $O(m \log m)$, проход по всем ребрам и нахождение максимума в каждой группе с одинаковым началом ребра в сумме дадут $O(m)$. Таким образом, итоговая сложность: $O(m \log m)$.

7.

Так как степень вершины 2, то заметим, что получится объединение циклов, причем число вершин равно числу ребер. Таким образом, необходимо выделить эти циклы и взять в них ребра через одно.

Для выделения циклов будем производить обход в глубину. Как только закрылась вершина, с которой начинался обход – получился цикл.

Корректность:

В теории графов доказано, что двудольный граф не может иметь циклов нечетной длины. То есть, все циклы четной длины и, чередуя ребра в циклах, будем последовательно брать ребра у которых начала и концы в одних и тех же долях, чтобы соблюсти определение паросочетания. Таким образом, мы выберем максимальное количество пар ребер в паросочетании, ведь проходя по циклам и выбирая ребра через одно, мы не пропускаем ни одной вершины одной из долей, а больше быть ребер в паросочетании, исходя из определения, быть не может.

Сложность:

Для поиска циклов производится алгоритм поиск в глубину: $O(|V| + |E|)$, затем проходятся все циклы для выбора ребер для паросочетания, что в худшем займет $O(|V|)$. Таким образом, итоговая сложность: $O(|V| + |E|)$.