№1

Используем dfs:

```
dfs(curNode, curCost) {
        if (curNode == endNode) {
                res = min(res, curCost);
                return;
        }
        for (edge in curNode.edges)) {
                dfs(edge.nextNode, max(curCost, edge.cost) );
        }
}
```

Асимптотика dfs – O(n + m)

№2

а) Ответ – наибольшая по количеству вершин компонента двусвязности.

Докажем сначала, что если граф двусвязный, то существует такая ориентация, что для любой вершины существует путь во все вершины (любая вершина вершина достижима из любой).

Докажем индукцией по n – количеству вершин. Если n = 3, очевидно. Пусть при n = k выполнено. Рассмотрим n = k + 1 и обозначим новую вершину vNew. Т.к. граф двусвязный, то существует как минимум 2 ребра, для которых vNew – концевая (проще говоря, из vNew исходят как минимум 2 “неориентированных” ребра). Пусть вторые концевые вершины для этих ребер – p и s, а путь между ними при ориентации при n = k следующий: $p = v_1\text{-}v_2\text{-}v_3\text{-}\ldots\text{-}v_i = s$, а следующая за вершиной s – v_{i+1}. Тогда ориентируем новый граф следующим образом: $p = v_1\text{-}vNew\text{-}(v_i = s)\text{-}v_{i-1}\text{-}v_{i-2}\text{-}\ldots\text{-}v_3\text{-}v_2\text{-}v_1 = p$. Ориентацию остальных ребер при этом оставляем без изменений. Таким образом, мы получили новую ориентацию, при которой все вершины, достижимые при старой ориентации, остались достижимыми при новой, и новая вершина также стала достижимой из всех остальных.

Теперь докажем, что ответ достигается при такой ориентации, что все мосты, направлены в сторону максимальной компоненты двусвязности KMax, а любая вершина в каждой компоненте достижима из любой:

Тогда min(c(v)) = nMax – размер этой компоненты. Действительно, в этом случае мы из любой вершины v !€ KMax можем прийти в KMax --> c(v) > nMax, а из любой v € KMax c(v) = nMax. Пусть какой-то мост ведет не в сторону KMax. Тогда идя по нему, мы рано или поздно придем в компоненту

К двусвязности, из которой не выходят другие мосты. Для любой вершины v из этой компоненты c(v) = size(K) < nMax.

Поиск компонент связности и их размера осуществляется за O(n + m).

№4

Сначала делаем topsort всех вершин (O(k + n + m)). Результат получаем в массиве sorted. Затем проверяем, чтобы в нем сначала шли города, а потом – деревни (O(k + n)). Если это не так, то не из каждого города можно попасть в любую деревню (например, если sorted[2] – деревня, а sorted[3] – город, то город следует за деревней, поэтому в силу ацикличности из этого города нельзя попасть в эту деревню). Затем для каждой деревни посчитаем min[i], 1 <= i <= n – минимальная сумма ребер такая, что из любого города можно добраться до деревень с 1-й по i-ю (порядок соответствует sorted). Делаем это следующим образом:

Для i = 1 это, очевидно, сумма всех ребер, идущих из всех городов в эту деревню. Если из какого-то города пути нет, то ответ не существует. Далее для каждой k-й деревни считаем сумму длин ребер из каждого города в эту деревню – curSum. Затем ищем минимальное ребро из деревень 1…k – 1 в k-ю – curMin. Если из какого-то города дороги нет и не существует ни одной дороги из предыдущих деревень, то ответ не существует. Если из какого-то города дороги нет или если curSum > curMin, то min[k] = curMin. Иначе же min[k] = curSum. Так как мы проходим по каждому ребру ровно 1 раз, то асимптотика – O(m).

Ответом будет min[n].

№7

Используем dfs. При проходе будем хранить вершину, с которой начинали, а также длину текущего пути и минимальную найденную длину. Если при очередной итерации мы попали в начальную вершину, то обновляем, если нужно, длину минимального пути. Если в начальную вершину мы не попали, и при этом из текущей вершины больше нет путей в непосещенные вершины, то ничего не делаем, выходим из текущей итерации. Если в результате полного обхода значение минимального пути не изменилось (равно какому-то специальному первоначальному значению, например, -1), то цикла для данной вершины не существует. Делаем такой обход для каждой начальной вершины.

Так как вершин n, а асимптотика dfs – O(n + m), то общая асимптотика – O(n(n + m)).