Федер Евгений, М3139, HW10

Задание 1)

Пусть у нас есть дерево. Пройдемся по нему обходом в глубину и пронумеруем вершины. – О(n)  
Теперь отсортируем запросы в порядке нумерации их в дереве – О(m).  
Пойдем опять обходом в глубину и будем дополнительно хранить вектор пути от вершины до корня. Если мы пришли в вершину и на нее есть запрос(за О(1) мы смотрим это), то смотрим в вектор и отвечаем на него. – О(n)  
Время: О(m) + 2O(n) = O(m+n)

Задание 2)

Пусть мы умеем выполнять LA(k-ый предок)  
Вспомним решение LCA через двоичные подьемы. В том методе мы предподсчитывали массив прыжков на степень двойки для каждой вершины. Давайте теперь вместо этого массива использовать LA.   
Таким образом когда нас просят сделать LCA для двух вершин будем использовать O(logn) LA вместо прыжков с помощью массива.  
  
Задание 3)  
Давайте в каждой цепочке хранить минимальное ребро в этой цепочке. Считать его будем при создании цепочек – давайте выбирем ребро, как и раньше, но дополнительно будем считать в вершине – минимум из выбранного ребра и значения, которое пришло из того поддерева.  
  
Мы знаем, что можем делать LCA за <n, 1> (на лекции мы свели это к RMQ+-1) . Также посчитаем для двоичных подьемов минимум.

То есть теперь все происходит так: ищем LCA(u, v) и смотрим ее место в цепочке. Так как для каждой вершины в цепочке мы сохранили минимум от самой нижней вершины, до текущей, то мы можем достать min внутри цепочки за O(1)  
Итог:  
предпроцес: О(n) на множества для отрезанных деревьев, LCA, цепочки и двоичные подьемы с минимумами = O(n)

Запрос: тоже самое, только считать параллельно минимум.

Задание 4)

Сейчас у нас наш предподсчет устроен f[c][l][r] -> памяти О(s^2) для каждой маски. Давайте для каждой маски хранить sparse table как и в RMQ(по +-1 мы можем узнать какие числа, зафиксировав начальное –при подсчете мы все равно обходим все вершины). Теперь у нас f[c][l][k], что O(slogs) памяти.