7.1. Дан неориентированный граф. Разбить множество вершин на две доли так, чтобы у каждой вершины был сосед в другой доле.

Построим в графе остовное дерево – это умеет делать DFS (оно имеет вид дерева поиска). Как известно, дерево – двудольный граф (по критерию: для этого необходимо и достаточно чтобы графе не было циклов нечетной длины, а в дереве вообще нет циклов). Построим двудольный граф, то есть разобьем вершины в дереве на две доли – это просто сделать: будем при обходе красить вершины, стоящие на четном расстонии от корня в первый цвет, а на нечетном – во второй. Получили что нужно: у каждой вершины есть хотя бы одно ребро (в силу связности), которое идет в другую долю (в силу двудольности). Дорисуем остальные ребра (ведь граф может быть с циклами, кратными ребрами и пр.) – очевидно, они не изменят этого свойства.

Сложность, очевидно, асимптотически равна сложности DFS, т.е. O(V+E)

7.2. Распараллеливание задач.

Так как граф зависимости ациклический и ориентированный, то его можно топологически отсортировать.

Сделаем это следующим образом: найдем все вершины со входящей степенью ноль, запомним ее, удалим из графа со всеми исходящими ребрами и повторим пока в графе есть хоть одна вершина. Реализация такова:

- Сначала посчитаем входящие степени всех вершин (это делается просто циклом по всем вершинам и увеличении степени всех дочерних вершин текущей вершины на 1) – O(V+E).
- Пройдем по всем вершинам и положим в очередь все вершины степени ноль – O(V)
- По каждой вершине в очереди (запомним ее и удалим из очереди) уменьшим входящую степень всех ее дочерних вершин на единицу и если она стала ноль – положим в конец очереди.

Можно делать отметки у верших в очереди, чтобы отмечать какие вошли в первую итерацию, какие во вторую, чтобы получить разбиение вершин по блокам, каждый из которых можно запускать параллельно.

Дальше нужно посчитать время на весь проект. Будем в каждой вершине-задаче хранить время, за которое делается эта задача.

Сделаем серию обходов в глубину, причем в каждой следующей задаче будем накапливать время, нужное для ее выполнения – оно равно времени в предке + время на саму задачу. Причем, как только наш обход в глубину достиг вершины, которую мы уже посещали (то есть для этой задачи, назовем ее A, есть как минимум две задачи, которые должны выполниться), корректно обновим время завершения задачи A: оно равно максимуму из того, что там уже стояло и новым значением. Однако не будем пересчитывать время окончания задачи A (иначе нам придется пересчитать и все задачи, стоящие после нее – а на это сразу можно привести пример, где этот алгоритм будет работать квадрат), а пометим, что у нас есть поправочный коэффициент.

Таким образом, еще одна серия обходов в глубину даст нам времена окончаний всех задач и, местами, поправочные коэффициенты. Дальше, нужно аккуратно обновить времена всех задач с учетом коэффициетов: сделаем что-то похожее на поиск в ширину: сначала просмотрим все вершины из нулевого блока (истоки) – ну там нет поправочных коэффициентов, это очевидно. Дальше вершины из первого блока – пересчитаем времена там, где есть ненулевые поправочные коэффициеты и пронесем коэффициенты в дочерние вершины (там, где они нужны: а именно если время в родительской больше, чем в

дочерней). И так далее. Этот этап работает также как и все перед ним за O(V+E) – мы просматриваем все вершины и, в худшем случае (почти) все ребра – в момент пронесения коэффициентов.

Заметим, что время окончания задач у нас неуменьшается, каждый дочерний элемент соберет в себе весь необходимый минимум времен, который нужен ему для начала работы – то есть время окончания каждого дочернего элемента минимально.

Дальше нужен еще один проход по вершинам – ищем максимум, он и будет ответом.

7.3. Игра на дереве.

Сначала сделаем обход в глубину из корня (там где стоит фишка – там корень) и определим расстояние от корня до каждой вешины (оно равно минимальному в силу единственности – так как у нас дерево). Назовем вершину нечетной, если она находится на нечетном расстоянии от корня и четной в противном случае (корень – четная вершина, ее расстояние равно нулю).

Дальше опрашиваем дерево обходом в глубину. Правила такие: если вершина нечетная, то ход вторго, если четная - то первого.

Будем помечать вершины единицей и двойкой, если в нем выигрывает соответственно первый или второй.

Когда обход в глубину выходит из вершины, это означает что просмотрены все его дочерние поддеревья (в случае листа поддеревьев нет и это значит, что как только вошли в вершину, сразу из нее выходим), а это значит, что можно определить кто выиграет в этой вершине. Правила такие: если лист четный, то выиграет второй, если нечетный - то первый. Если в поддереве ход первого и есть хоть одна единица, то выиграет первый (иначе второй), елси ход второго и в поддереве есть хоть одна двойка, то выиграет второй (иначе первый). И так до корня дерева.

Время работы: два обхода в глубину + опрос всех ребер = O(V+E). А так как у нас дерево, а в нем E = V-1, то общая сложность O(V).

7.5. Бочки с водой.

- Ответ на первую часть (первое число – количество литров в 4-литровой бочке (большая бочка – чуть больше банки!), второе – в 7-ми и третье – в 10-ти литровой): 4,7,0 –> 0,7,4 –> 0,1,10 –> 4,1,6 –> 0,5,6 –> 4,5,2
- Как решить в общем случае. Построим граф всех состояний: а их понятно сколько – O(a*b), так как в первой бочке не может быть больше a+1 литра, во второй – b+1, применяем правило произведения, получаем ограничение.
  Корнем дерева будет вершина (a, b, 0) будем эмулировать обход в глубину – из каждого состояния можно **максимум** перейти в 6 состояний (каждое действие делаем только если оно возможно, это очевидно проверяется за O(1)): перелить из первой во вторую (или наоборот), из второй в третью (или наоборот) или из третьей в первую (или наоборот). Если мы идем в состояние, в котором уже были, то не «рисуем» ребро (получается, что циклов у нас быть не может). Если из состояния нет новых состояний – получили лист дерева и разворачиваем рекурсию. И так пока не встретим нужный нам объем: останавливаем алгоритм и всплываем по рекурсии наверх. Или не встретим: тогда ответ «нельзя».
  Осталось решить вопрос, как определить, что какое-то состояние у нас уже было: можно пожертвовать памятью и завести трехмерный массив размером a*b*c и ставить туда 1 при входе в вершину.

  Сложность, очевидно, равна сложности обхода в глубину, то есть O(V+E), а так как у нас получилось дерево, то по той же причине, что и в предыдущей задаче, сложность получается O(V) = O(a*b).