10.1. Распределение заявок на занятия.

Будем действовать жадно: для каждой новой заявки будем брать минимально возможный номер аудитории.

Чтобы сделать это эффективно, заведем 2 массива – массив начала занятий и массив конца и отсортируем их по возрастанию.

Будем пробегать по этим двум массивам в порядке увеличения времени (что-то типа online merge). Если текущий момент времени t есть время начала занятия – то дадим ей свободную аудиторию, если же это время конца занятия – освободим аудиторию.

Будем поддерживать очередь свободных аудиторий, и, если на момент начала занятия очередь пуста, то создадим новую комнату.

Алгоритм занимает минимально возможное число аудиторий: пусть их число на момент окончания алгоритма было равно m. Пусть занятие a – первое занятие, которое попросило себе аудиторию номер m. Значит, на момент начала занятия все m-1 аудитории были заняты, значит в этот момент одновременно проходило m занятий – то есть любой алгоритм должен использовать не меньше m аудиторий, значти этот алгоритм оптимальный.

Сложность: две сортировки + линейный просмотр = O(N*lgN)

P.S. Эту задачу на теормине только что сдал.

10.2. Построить граф по наборам степеней.

Алгоритм придумался еще на практике: будем действовать жадно. Отсортируем набор степеней по убыванию: (d1, d2, d3… dN)

Самая большая степень будет соответствовать веришине с наибольшей степенью и так далее. Проведем из этой вершины (степени, скажем, d1 = d) d ребер в ближайшие d вершин (ближайшие – в списке степеней) и у каждой вершины, куда мы провели ребро уменьшим степень на единицу. Выкинем из набора число d1. Получим набор: (d2-1, d3-1,…dN), где минус единица написана у первых d чисел. Сведем задачу к предыдущей: найдем наибольшую степень, проведем от нее ребра к ближайшим и так далее.

Критерий, когда такого графа нет:

- Мы пытаемся провести ребро в вершину степени ноль, например: (4, 1, 1, 0, 0)
- В конце успешно получили последовательность из 0 и 1, но в ней нечетное число единиц

Почему это работает. С этим сложнее.

Нужно доказать, что для последовательности (d1, d2, d3… dN) **(1)** граф существует тогда и только тогда, когда он существует для последовательности (d2-1, d3-1,…dN) **(2)**.

- Достаточность очевидна: Пусть граф для **(2)** существует. Тогда добавим в него новую вершину, соединим ее с вершинами, соответствующими степеням d2-1, d3-1… -- получим граф, последовательность степеней которого **(1)**
- Необходимость. Пусть дан граф G, последовательность степеней которого **(1)**
  - Пусть из наибольшей вершины (назовем так вершину степени d1) идут ребра в точности в d1 следующих в последовательности вершин. Тогда все просто – удалим эту вершину (и ребра инцидентные ей), получим граф, у которого последовательность вершин **(2)**
  - Пусть это не так: тогда вершина «d1» не смежна хотя бы с одной из следующих за ней d1 вершин. Назовем эту вершину x.
    Значит, эта вершина (d1) смежна с какой-то другой вершиной z.
    Так как наша последовательность невозрастающая, то d(x) не меньше d(z).
    Тогда найдется такая вершина y, которая смежна с x, но не смежна с z. (Это понятно: в противном случае все вершины, смежные с x смежны и с

z, но раз z смежна с первой, то степень вершины z больше, чем степень x, что невозможно).

Раз так, то все просто: перебросим ребра так, как нам нужно: соединим вершину x с «первой» (удалив при этом ребра x-y и d1-z) и, чтобы сохранить последовательность степеней (а именно степени вершин y и z) добавим ребро y-z.

Так мы свели задачу к первому случаю. (если таких вершин несколько, повторим эту операцию пока не сведем к хорошему случаю).

10.3. Найти самый легкий путь между двумя выделенными вершинами в графе по метрике «вес пути равен сумме двух самых тяжелых ребер в нем».

Хочется применить агоритм Дейкстры сразу к метрике «сумма двух самых тяжелых ребер», но он не работает. Есть простой контрпример.

Можно предложить следующий способ: решим два раза задачу 9.4(b) из прошлой домашней работы: сначала построим дерево кратчайший путей из вершины s, затем из вершины t (причем уже в транспонированном графе).

Таким образом, мы для каждой вершины будем знать два самых тяжелых ребра в пути **от s** и в пути **до t**.

Далее, пробежимся по всем ребрам и для каждого ребра (u-v) сделаем предположение, что это ребро входит в кратчайших путь. Посчитаем вес этого пути: это сделать просто: у нас есть два тяжелых ребра в пути (от s до u) и в пути (от v до t) плюс само это ребро. За O(1) можно корректно найти два самых тяжелых ребра и посчитать их сумму.

Проделав это для каждого ребра можно найти минимум.

Чтобы вывести путь, достаточно вместе с минимумом поддерживать ребро, на котором этот минимум нашелся, а затем восстановить этот путь по деревьям, найденным во время выполнения алгоритма Дейкстры дважды.

10.4. Найти кратчайший цикл.

Мы уже умеем решать задачу “найти кратчайший цикл проходящий через данное ребро u-v” за линию (делали на практике: напомню, он равен расстоянию между вершинами u-v в графе без ребра u-v плюс один).

Запустим этот алгоритм для каждого ребра и среди всех циклов выберем минимальный.

10.5. Найти отрицательный цикл.

На лекции мы изучили, что если алгоритм Беллмана-Форда после n итераций сделал хотя бы одну релаксацию, то это значит, что в графе есть отрицательный цикл. Иначе, такого цикла нет.

Осталось научиться выводить его.

Для этого нужно уметь выводить не только расстояние до какой-либо вершины, но и сам кратчайший путь. Это делается стандартным методом: заведем массив «предков» и когда ребро нужно прорелаксировать, для соответствующей вершины обновим «предка».

Чтобы вывести отрицательный цикл, если он есть, нужно взять вершину, расстояние до которой изменилось на последней итерации алгоритма (то есть то ребро, которое прорелаксировалось) и пойти от нее по предкам пока не войдем в цикл – он и будет искомым.