# 1 Обязательная часть

1. ***Ремонт дорог.***

   Забудем про дороги, которые можно построить. Сейчас есть только уже построенные. Компоненты связяности, на которые эти дороги разбили города, сжимаем в вершины. Т.е. теперь одна вершина - множество городов, куда можно доехать по уже построенным. Теперь вспоминаем про дороги, которые можно построить, делаем их ребрами в новом сжатом графе. Все, что теперь нужно сделать - найти MST этого графа. Стоимость построенных дорог будет минимальна, будут соединены все компоненты связности, а значит, можно будет доехать из любого города в любой.

2. ***Проверка на минимальность.***

   $\mathcal{O}(m \cdot M(n))$. Умеем искать минимум, следовательно, умеем искать и максимум (делается домножением весов на -1). Рассматриваем все ребра графа. На каждой итерации делаем проверку: правда ли, что $w[pair(a, b)] >= max(a, b)$, где w - вес ребра, max - максимум на пути от a до b. Если найдется такое ребро, что неправда, значит мы могли включить это ребро в остовное дерево вместо найденного и тем самым уменьшить вес дерева.

3. ***Минимум на пути.***

   Для массива строим дерево отрезков с операцией "минимум на интервале". Построение за $\mathcal{O}(n \cdot \log n)$, ответ на запрос за $\mathcal{O}(\log n)$. Получили асимптотику $\mathcal{O}((n + m) \log n)$. А с СНМ не умею.

4. ***2-connective.***

   Поддерживаем лес деревьев компонент реберной двусвязности. Добавление ребра от дерева к другому = добавление + увеличение счетчика количества мостов. Если внутри одной компоненты - let it go. Внутри одного дерева, но не одной компоненты - ищем ближайшего общего предка, идем от них к предку, по пути сливая в одну компоненту. Итак, сделали union СНМа (слили в одну вершину), уменьшили кол-во ребер на суммарное число, на сколько поднялись от них к предку. Можем рассказывать про мосты.

5. ***Connectivity in directed.***

   Запросы удаления и добавления: пусть изменилось ребро (u, v). Для пересчета количества путей из $a$ в $b$ вычитаем при удалении, прибавляем при добавлении кол-во путей $c[a, u] \cdot c[v, b]$. Чтобы как-то бороться с большими числами храним все c[][] по модулю большого простого числа.

6. ***Время работы.***

   $\mathcal{O}(logn)$ в худшем случае. В среднем - почти константа. Если рассмотреть дерево, на котором сжатие путей работает за логарифм, то там есть момент, когда подвешиваем рассматриваемое дерево за новый корень. В случае с рандомом среднее количество подвешеваний будет равно двум, нам это вроде как говорили.

7. ***Код.***

```
vector < vector<int> > g;
vector < vector<int> > res;

//заполнение g ребрами

vector <int> use(n, -1);

//Записываем, от какой вершины (use[i]) мы
//последний раз проводили ребро в i.
//Если хотим провести ребро из u, то:
//use[i] == u => уже проводили => кратное;
//use[i] < u => последний раз проводили из
//какой-то другой вершины, все хорошо.

for (int i = 0; i < n; i++){
    for (int j = 0; j < g[i].size(); j++){
        int v = g[i][j];
        if (use[v] < i && v != i){
            res[i].push_back(v);
            use[v] = i;
        }
    }
}
```