07-hw-PA

vladrus13rus

October 2021

1 Некий медленный топологический граф

Подумаем, из чего вообще состоит наш алгоритм. Наш алгоритм - это просто куча dfs, чередующиеся пересечением или отрицанием множеств. Однако линейная работа со множествами - это просто $O(n)$, а вот dfs работает за $O(m) >= O(n)$, то есть давайте выбросим из рассмотрения работу со множествами и делает только dfs

Давайте разберем, что же нам нужно, что бы у нас был алгоритм, который постоянно делает dfs на как можно большем числе вершин. Во первых давайте разберемся с C в какой либо из итераций (рекурсивной) алгоритма. Какого размера (число вершин) оно должно быть? Допустим, его размер x. В него идут вершины из P, из него идут вершины из S. И все, нигде более он не влияет. Но зачем тогда нам делать C большим? Пусть будет минимально малого размера, зачем нам попусту тратить вершины? То есть заметим такой факт - если у нас есть компонента сильной связанности размера x, то мы БЕЗ потерь со стороны ответ или чего либо (кроме количества вершин) можем сжать ее в одну.

Решено, пусть каждая вершина будет компоненотой сильной связанности. Тогда мы будем n раз вычленять компоненту сильной связанности. Уже неплохо! Осталось достигнуть того, что бы каждый раз dfs шел долго.

Но что значит, что каждая вершина - компонента сильной связанности? Это значит, что в этом графе не будет циклов. Иначе говоря, нам бы хорошо было сделать некую топологическую сортировку на нем. То есть у нас есть последовательность вершин слева направо (от 1 до n), и ребра идут тоже только слева направо. То есть нам выгодно рассмотреть дерево.

Теперь подумаем над этапом dfs. Теперь он выполняется n раз, надо сделать так, что бы он ел как можно больше вершин. Допустим, мы отцапали какое то поддерево алгоритмом. Мы помним доказательство с лекции, что мы брали топсорт по всем вершинкам ($V_{p(1)}, V_{p(2)}$ и так далее). Мы смотрели на работу алгоритма как на $d(n) = \frac{1}{n}d(S_{V_{p(1)}}) + \frac{1}{n}d(S_{V_{p(2)}}) + ... + \frac{1}{n}d(S_{V_{p(n)}})$. Заметим, что мы как раз эту сумму и хотим увеличить, это и есть те самые этапы dfs. Что же такое у нас было $S_{V_{p(i)}}$. Это размер множества *succ*, когда мы запустились из вершинки $V_{p(i)}$. На лекции так же говорили про топсорт, то есть мы сортировали эти вершинки. Раз мы хотим добиться большего результата на $d(n)$, нам нужно максимизировать $\frac{1}{n}d(S_{V_{p(i)}})$, а значит, максимизировать $d(S_{V_{p(1)}})$, а значит, максимизировать $S_{V_{p(1)}}$.

Посмотрим на время работы алгоритма. Будем поддерживать такой инвариант: у нас есть дерево. Пусть мы выделили какую то вершинку. Попытались по ней порезать. В succ лежат все вершинки-дети, в pred - все родители. Однако у нас появилось в A все братья родителей нашей вершины. Непорядок, мы же хотели поддерживать всегда дерево. Тогда уберем их. Получается бамбук. То есть теперь мы берем только бамбуки. Тогда в pred все вершинки с меньшими номерами в topsort, а в succ - большими.

Рисунок 1. Я, когда решал задачу всеми возможными графами, а потом оказалось, что ответ бамбук

Теперь у нас есть последовательность вершин, нам нужно как то связать их m ребрами, что мы ничего не сломаем, что у нас будет по прежнему крутиться дфс постоянно и так далее. Можно было просто доказать через быструю сортировку, ведь здесь происходит та же самая вещь - мы делим массив (в нашем случае топсорт) на две части, за O(количество ребер в подграфе, нужно что бы оно было всегда больше размера подграфа) мы делаем dfs и уходим снова в рекурсию. Что же может пойти не так? Мы всегда делим как то вершины, получаем $n = n_1 + n_2$ вершин, $m = m_1 + m_2$ ребер, идем в рекурсию... Тааак, стоп. Почему же мы получаем $m = m_1 + m_2$? В этом и состоит загвоздка - мы не можешь бесконечно увеличивать ребра алгоритма до n^2. Надо понять, до каких пор мы можем это делать. Равенство $mlogn$ обуславливается с точки зрения алгоритма тем, что мы максимизировали до равенства $S_{V_{p(1)}}$. Там дальше по доказательству пойдут знаки равенства и в конце будет $T(n) = c_1 nlog(n)$. Осталась загвоздка в виде того сколько именно ребер мы можем добавить. Минимальное число делений для до каждой вершины log(n), оно прервет не более $\frac{n}{log(n)}$ ребер потому что частей будет именно столько минимальное, больше ставить - все портить. В общем, больше $\frac{n}{log^2(n)}$ ребер ставить нельзя, будет больно. Но это все равно меньше чем n^{2-eps}, так что мы выиграли