

**Минимизация ДКА.** Два автомата изоморфны, если они изоморфны как графы с буквами на ребрах и с учетом терминальных вершин. Мы хотим найти автомат минимального размера, который будет принимать тот же язык, что и заданный автомат.

Выделим классы эквивалентности по вершинам автомата. А именно, каждой вершине в соответствие ставим множество строк, которые из нее читаются. Мы покажем, что автомат, где вершинами будут классы эквивалентности, минимален.

*Замечание.* Классы будут либо терминальными, либо нет, в зависимости от того, принадлежит ли классу пустая строка.

Почему такой построить можно? Потому что если склеить вершины с одинаковым множеством читаемых строк, то множество строк, принимаемых автоматом, не изменится. Поэтому алгоритм построения такого автомата будет тривиальным — нам надо просто объединять вершины с одинаковым множеством строк.

Пусть есть автомат меньшего размера, принимающий такой же язык. Для представителя каждого класса эквивалентности выберем строку, которая в него ведет. Тогда какая-то пара строк будет вести в одну и ту же вершину минимального автомата. Язык, соответствующий этой вершине, не совпадает хотя бы с одним из языков классов эквивалентности.

Если автомат был ациклическим, то нам надо взять вершины в порядке топологической сортировки, после чего надо вершине ставить в соответствии множество  $terminal, class(go_{c_1}), class(go_{c_2}), \dots, class(go_{c_n})$ . Тогда достаточно просто смотреть, был ли класс с таким множеством раньше. Для этого можно воспользоваться хэш-таблицей, и получить алгоритм сложностью  $O(m)$ .

**Алгоритм Хопкрофта.** Теперь наш автомат может иметь циклы. Мы будем наращивать классы постепенно. А именно, если вершины относятся к разным промежуточным классам, то они уже точно разные, а вершины внутри одного класса еще могут быть одинаковыми. Изначально классы определяются терминальностью вершины. Потом можно брать и определять новый класс с помощью хэша от классов вершин-потомков. Тогда надо делать операцию, пока классы меняются. Почему этого хватает? Потому что если две вершины были разными, и отличались строкой  $L$ , и  $|L|$  было минимальным из возможных, то тогда на каждом шаге  $1, 2, \dots, |L|$  одна вершина будет менять свой класс. Работать алгоритм будет за  $O(nm\sigma)$ , но это наш медленный алгоритм.

Теперь, чтобы улучшить алгоритм, воспользуемся идеей о классах-сплиттерах. Если нашим промежуточным классом мы разбили другой класс по символу  $c$  (то есть, если для какого-то  $A$  верно, что из него есть переход по  $c$  и в класс  $B$ , и вне класса  $B$ ), то тогда для его непересекающихся подклассов  $B_1, B_2$  верно, что в дальнейшем можно будет разбивать только по одному из них. Доказательство из теории множеств — если мы уже разбили по двум классам, то по информации  $x \stackrel{?}{\in} B, x \stackrel{?}{\in} B_1$  однозначно достраивается  $x \stackrel{?}{\in} B_2$ .

Кроме того, число разбиений линейно — если взять сумму  $|C| - 1$  по всем классам, то каждое разбиение уменьшает эту сумму на 1. Поскольку эта величина неотрицательна и изначально  $O(m)$ , то и количество ненулевых разбиений будет линейно. А количество нулевых разбиений не более чем в два раза превышает количество ненулевых.

Ну тогда достаточно сохранить обратные переходы в автомате, рассматривать классы в порядке очереди, а при разбиении класса на два класса поменьше можно добавить в очередь только меньший из  $A_1, A_2$ . Тогда рассматривать конкретный обратный переход в автомате мы будем не более, чем  $O(\log m)$  раз, а значит суммарное время работы будет  $O(m\Sigma \log m)$ . Тут, правда, важно, что каждое разбиение должно быть проведено за  $O(|B| + \sum |A_i|)$ , где  $B$  — это класс, по которому мы разбиваем, а  $A_1$  — это меньший из двух непустых классов разбиения (тут важно, что сумма  $A_1$  оценивается как  $O(m\Sigma \log m)$  только если  $|A_1| < |A_2|$ ), но это не очень сложно сделать, если поддерживать всякие счетчики, и при

перенаправлении указателей для разбиения делать новый класс меньшим из двух.