Гавриленко Арсений *777*

Домашняя работа 3

Дисклеймер Доделал 1.2 и 1.3, Оставил 1.1 для наглядности. Немного изменил ход мысли в 4ой , сделал 5ую, доделал 8ую.

Задача 1.

1. Проблема - мы не знаем, где в слове из нашего нового языка образованного конкатенацией заканчивается слово из первого языка и начинается из втого, поэтому мы будем эмулировать всевозможные исходы. Для этого потребуется 2 ленты и две МТ, которые разрешают язык A и B соотвественно. Для лент:

- Входная - с нее будем считывать наше слово
- Лента счетчик количества букв, которые мы уже "отщепили и проверили". По факту - это счетчик позиции разделителя.

Далее будет небольшое словоблудие, так как я сам не очень то представляю как все это записать по-человечески, но я попробую. Распишем алгоритм, но не будем опускаться в мельчайшие подробности. То есть будем счиать что внутренее устрйоство МТ для языков A и B известно и корректно. Пусть мы находимся на iтом шаге, все головки стоят на начале всех лент тогда:

- на 2ой ленте сейчас лежит предыдущая позиция разделителя(допустим там просто i символов каких то написано). Начнем синхронно двигаться по этой цой ленте а также по входной , при этом на каждом шаге копируя вход на ленту МТ A. В какой то момент мы доходим до конца символов(пусть G) на нашей ленте счетчике, в этот момент у нас скопировано на МТ A ленту i+1 символов (как было на предыдущем шаге), тогда увеличим счетчик на 1(допишем на ленту счетчик еще один G) и допишем на нее в конец пустой символ.
- Теперь мы также продолжаем двигаться по ленте входа, но теперь уже копируем на ленту МТ B, когда дойдем до конца входного слова w допишем в конец 4ой ленты пустой символ.
- Итак, сейчаси имеем два подслова, которые в конкатенции дают входное w, также счетчик позиции разделителя, причем первое подслово подано на вход машине A , второе подслово - машине B.

- Далее запустим обе эти машины. Так как они по отдельности разрешают свои языки полиномиально.

- Наша МЕГА МТ состоящая из двух МТ + двух лент - входной и счетчик ленты будет принимать слово, если обе машины приняли свои подслова и счетчик не превысил длину входа, если же хотя бы одна из подМТ попала в Reject и мы еще не дошли счетчиком до конца слово - запускаем на новых двух подсловах, если ще оба попали в Reject и разделитель дошел до конца слова - то МЕГА МТ не принимает слово , т.е. попадает в Reject. Полиномиальность скорости работы от длины входа очевидна, так как это сумма из пар полиномов работы машин A и B.

Задача 1.

2. Внимательный читатель заметит, что есть решение вот тут : https://neerc.ifmo.ru/wiki
здесь совсем не указано как хрнаить концы, а это вроде важно при построении МТ)
В целом воспользуемся концепцией описанной выше. Сократим словоблудие до минимума.

- утверждение - Если наше входное слово w, то оно является конкатенаций неких слова из A.

- Немного понизим градус абстракции. По факту нам нужно проверить всевозможные нарезки слова w на подслова и попробовать найти те, в которых, все подслова являются словами из A.

- Делаем МЕГА МТ, у нее есть входная лента, где лежит слово w, есть МТ которая принимает слова из A и должен быть некоторый счетчик, опишем его далее.

- Построим индуктивный алгоритм. Пусть на каком-то шаге мы знаем, что первые k симоволом нашего слова w составляют конкатенацию слов из A(возможно там одно слово или один символ или даже пустое слово(оно точно будет прималделать A*) - неважно). Тогда на следующем шаге начнем проверять все подслова вида $w[k+1;j]$, где $j \in [k+2, |w|]$, все слова такого вида будем "отправлять"на МТ A и чекать, принимаются ли они, если нет, то идем дальше, если да, то занесем позицию j(то есть конце этого подслова) на нашу ленту счетчик.

- Устройство ленты счетчика (самая проблемная часть)- туда будем заносить позиции концов подслов(не будем опускаться в детали их кодирования) из A, как только доходим до конца слово w ставим разделитель. Получается для каждого уже существующего набора подслов, которые покрывают первые k символов у нас будет набор(возможно пустой) концов. Построим некоторое дерево (которое можно хранить в памяти) и не будем вдаваться в подробности (всегда можно всунуть адрес ребенка и адрес родителя в какой-то кусок ленты, чтобы имелось дерево). Тогда проход по всем нарезкам - это просто поиск в глубину.
- Получается, что у нас максимум n^2 концов, проход в глубину - полином, значит мы получили полиномиальный алгоритм.

Задача 1.

3. Декомпозируем задачу на построения языка B = AA (первая задача) и языка G = B* - вторая задача. Та-дам.

Задача 2.

Задача 3.).

Задача 4 Чтобы проверить, что w не принадлежит L, можно проверить, что w принадлжеит не L, что эквивалентно (см. курс ТРЯП) тому, что слово w принимается автоматом, который является дополнением автомата для языка L. Обозначим длину входа как длину РВ L + длину w.

1. Если язык задан РВ, то можно построить автомат, принимающий слова из этого языка. Давайте рассмотрим алгоритм построенения НКА по РВ (Хопкрофт стр 122).

- Всего есть три операции в РВ - "или "конкатенация "итерация".
- Заметим, что операция или, которая обьедняет два слогаемых(условно говоря) прибавляет два состония и 4 перехода к исходному количеству.
- Итерация 4 перехода и два состояния.
- конкатенация добавляет 1 переход

- Ну и в самый первый момент мы выделим по два состояния и одному переходу из 1го во 2ое состояние соотвественно для каждоыго символа из РВ.

Как мы видим из обьяснения, любой символ РВ или любая операция РВ линейное влияет на колиество переходов и количество состояний автомата, значит можем сказать, что размер НКА линейно зависит от размера РВ.

Но теперь, когда у нас есть автомат мы пойдем в другую сторону - построим по нему грамматику, это займет у нас линейное время $\Theta(n)$ - где n колво состояний. Теперь вспомним (при помощи коллег, котоыре сейчас пересдают ТРЯП), что существует теорема 7.32, страница 309 Хопкрофта, которая говорит, что для грамматике длиной n можно построить грамматику в нормальной форме хомского за $O(n^2)$, причем ее длина будет $O(n^2)$.

Дальше - лучше. В курсе ТРЯП был Алгоритм Кока-Янгера-Касами разбора грамматики в НФХ, который по грамматике и слвоу проверяет лежит ли слово в языке. Асимптотика $O(|w|^2 * |N|)$, |N|- колво правил . То есть алгоритм мы построили, теперь давайте приведем асимптотику:

- РВ -> НКА $\Theta(|PB|)$
- НКА -> Грамматика $O(n)$
- Грамматика -> НФХ $O(N_{rules})$
- Проверка принадлежности слова $O(|w|^2 * |N|)$
- Итого полином.

<u>**Задача 5**</u> В процессе переделывания 4ой у меня появились мысли по поводу 5.

- По каждому РВ строим НКА(в 4ом номере я доказал полиномиальность)
- Строим пересечение всех автоматов = нужно построить их декартово произведение и расставить нужные переходы(существет алгоритм, доказывать не буду), в любом случае это $O(n_1 * n_2 * .. * n_2 019)$ - то есть также полином.

- нужно проверить, что построенный автомат принимает хоть что-то. Можно сделать с помощью многих алгоритмов - например Флойда, он тоже полином от размера графа. Итого все полинм, полином от полинома - тже полином.

<u>**Задача 7**</u> Идея правильная - но правильный ли номер? Я знал, что задача гуглится, но не хотел читать так как какой тогда смысл в дз, поэтому возможно не очень корректно записана мысль с счетчиком.

Насколько я помню, вроде я когда то читал разбор этой задачи, попробую правильно записать. Перед тем как решать, давайте подумаем. Операция сравнения - функция двух элементов, то есть если мы даже собираемся пройти хоть раз и хранить где то результаты сравнения, уже будет O(n) памяти. Значит нужно построит какой-то счетчик элементов. Построим следующий алгоитм:

- Будем хранить количество активных элементов(далее уточню что это) и один(последний) активный
- Считываем новый элемент. Если у нас сейчас в активных нет ни одного, то делаем этот активным (заносим в переменную), счетчик ++
- Считываем новый элемент, но у нас уже есть активные(и значение одного даже знаем), в таком случае сравниваем активный и вошедший, если они не равны, то вошедний делается активным, счетчик++, если не равны, то счетчик – и активный выкидывается(кладется NULL к примеру).

когда мы дойдем до конца, мы будем знать значение самого встречающегося элемента. Причем его значение останется в активном элементе, тогда пройдем еще раз и посчитаем сколько раз он в массиве, тогда сможем ответить на вопрос задачи.

Теперь докажем, что этот алгоритм действительно находит самый часто встречающийся элемент. Для начала рассмотрим случай, что наши элементы(которые встр N/2 раз стоят как раз один через один, тогда не останется активных элементов, но тогда можно просто прогнать и посчитать вхождения последнего и предпоследнего элемента, никаие кроме этих двух не могут быть(при такой постановке). Иначе(если не один через один), то где то будут точно стоять два или более подряд одинаковых элемента, в таком случае они занесутся в счетчик. То есть по построению алгоритма видно, что в активном лежит наиболее частый или последний(если у всех одинаковая частота).

Итого мы прогнали 2 раза для общего случая и еще два дря проверки случая раз через раз, то есть $\Theta(n)$.

<u>**Задача 8**</u> Ниже представлен алгоитм, который позволяет найти за $n + logn + c$. Тогда давайте просто докажем, что меньше нельзя - см последний пункт. Для начала покажем, что для нахождения минимума из x обьектов нужно x-1 сравнений, что по факту является x + c сравнений. И так , если у нас есть X обьектов, то найти минимальный обьект можно МИНИМУМ за X-1 сравнение, потому что иначе граф сравнений будет несвязный, то есть об отношении порядка каких то двух обьектов мы знать не будем. Теперь перейдем к нашей задачи.

1. Давайте устроим турнир в виде дерева сравнений(я надеюсь не нужно подробно расписывать как он устроен). То есть просто попарно сравниваем обьект, наименьший проходит дальше. Оговорим, что рассматриваем $n = 2^k$, иначе у нас на некоторых "этажах"добавятся некоторые элементы, но они влияют не больше чем на константу.

2. Такой турнир нам позволит найти минимум за n сравнений (Сумма n/2 + n/4 + n/8 ...) Но как нам найти второй минимум? Давайте докажем, что второй минимум находится в группе тех элементов, которые сравнивались с нашим минимальным элементом и проиграли ему. Заметим пока, что их $log_2(n)$, так как их столько же, сколько этажей нашего дерева турнира.

3. Два варианта, либо наш второй минимум сравнивался с нашим минимум , либо нет. Пусть нет, тогда был элемент A который меньше нашего второго минимума, но в то же время он меньше нашего самого минимального, тогда получается, что A - второй минимум, значит предположение было неверным. Тогда получаем, что второй минимум был одним из тех, что когда то сравнивался с минимумои и проиграл.

4. Найти второй минимум легко взяв все элементы, который сравнивались с минимумом и проиграли ему (их, как уже было сказано $log_2(n)$ и найти среди них минимум, что можно сделать за $log_2(n) +$ как уже было доказано. Т.е. утверждение доказано.

5 Докажем, что это оптимальный алгоритм, и меньше нельзя. Знаем, что чтобы найти минимум среди элементов нужно $n - c_1$(в случае

дерева), теперь подумаем, как бы нам устроить сравнения так, чтобы быстрее чем за n + logn + c также найти и второй минимум. Второй минимум - как бы мы не строили сравнения(турнир неважен) мог проиграть только один раз - когд сравнивался с первым минимум. И так у нас есть некое множество элементов проигравших только один раз. Также у нас известно сколько раз выигрывали эти элементы и каких именно элементов (можем построить турнир и дать счетчик каждом элементу), теперь вопрос - насколько быстро мы сможем найти среди этого множества минимум(который будет для нас вторым). Будем брать элементы с наибольшим количеством побед и сравнивать, если допустим элемент A меньше элемента B, то понятно, что все те с которыми сравнивался A и выиграл(то есть большие чем A) - точно не минимумы, таким образом рекурсивно сравнивая, мы каждый раз будем логарифмечески уменьшать количество потенциальных минимумов(делиться на два, если турнир идеальный и симметричный).