Домашнее задание №5

Гунаев Руслан, 776 группа

19 апреля 2019 г.

1.

Пусть язык $L \in PR \cap co - RP$. Существует для алгоритма A, который никогда не ошибается на неверном входе, и B – не ошибается на правильном входе(вход правильный, если слово принадлежит языку). Построим алгоритм C. Сначала он будет запускать первый, затем второй алгоритм. Если первый алгоритм выдал 1, то слово точно лежит в языке, если второй выдал 0, то точно не лежит. Пусть первый выдал 0, второй 1. Запустим алгоритм еще раз, будем делать так пока не получим либо от алгоритма A ответ 1, либо от B–0. Матожидание количества повторений алгоритма равно 2, а значит алгоритм в среднем полиномиальный. Получили, что язык $L \in ZPP$.

Пусть $L \in ZPP$. Пусть алгоритм , обрабатывающий слово и отвечающий на вопрос принадлежности этого слова языку в среднем работает за полином $t(n)$. Запустим тот же алгоритм на новом слове. Будем считать шаги, если шагов $2t(n)$ и алгоритм не дал ответ на вопрос, то выведем 0. Если закончил, то ответ оставим прежним. Пусть ξ – случайная величина, показывающая количество шагов алгоритма при работе на слове. Воспользуемся неравенством Маркова.

$$P(|\xi| \geqslant 2t(n)) \leqslant \frac{t(n)}{2t(n)} = \frac{1}{2}$$

Если слово не принадлемжит языку, то в любом случае мы получим ответ не более чем за полиномиальное время, если же слово языку принадлежит, то ответ да за полиномиальное время мы получим с вероятностью не меньше, чем $\frac{1}{2}$. А это значит $L \in RP$.

Аналогично с $co - RP$. Что и требовалось доказать.

2.

1)

Поскольку если есть машина Тьюринга, распознающая язык с вероятностью ошибки p, то точность можно сколь угодно хорошо улучшить за счёт относительно небольшого прироста времени. Если мы запустим машину $k = |x|$ раз подряд, а в качестве результата возьмём результат большинства запусков, то вероятность ошибки упадёт до $(2\sqrt{p(1-p)})^k$, а время останется равным $poly(|x|)$. Здесь k запусков машины рассматриваются как схема Бернулли с k испытаниями и вероятностью успеха $1-p$, а формула, выражающая ошибку, — вероятность неудачи не менее чем в половине случаев. Если теперь запустить машину k^2 раз подряд, то время все еще будет $poly(|x|)$, а вероятность ошибки упадёт до $(2\sqrt{p(1-p)})^{k^2}$. Таким образом, с ростом показателя многочлена, оценивающего время, точность растёт экспоненциально, и можно достичь любого нужного значения.

2)

Пусть вероятностный алгоритм V на входе (x, r) работает за время $T_V(x, r) : E[T_V(x, r)] \leqslant p(|x|)$, где p– некоторый полином. Рассмотрим новый вероятнотностный алгоритм V', который на этом же входе будет делать не более $9p(|x|)$ шагов алгоритма V на заданном входе. Если раньше чем за это время будет получен результат, то выводим ответ да, если нет, то ответ нет. Заметим, что по неравенству Маркова

$$P(T_V(x, r) \geqslant 9p(|x|)) \leqslant \frac{E[T_V(x, r)]}{9p(|x|)} \leqslant \frac{1}{9}.$$

Это означает, что если $x \in L$, то

$$P(V'(x, r) = 1) \geqslant P(V(x, r) = 1) - P(T_V(x, r) \geqslant 9p(|x|)) \geqslant 2/3 - 1/9 = 5/9 > 1/2$$

Аналогично если не принадлежит получим

$$P(V'(x, r) = 0) \leqslant 4/9.$$

То есть $L \in BPP_{4/9} = BPP$.

3.

1)

Запустим генератор два раза подряд. Если выпала последовательность 00, то выведем 1. Если выпала 10 или 01, то выведем 0. Если же выпала 11, то заново запустим генератор два раза подряд. Заметим, что такой алгоритм в среднем будет выполняться за константу, так как

$$E_s = 3/4 + 1/4(E_s + 1) \Rightarrow N = 4E_s = 16/3$$

Также для вывода ответа у нас есть только три последовательности, каждая из которых равновероятна, значит итоговый генератор выдает 1 с вероятностью 1/3, 0 – с вероятностью 2/3.

2)

Будем также два раза подряд запускать генератор. Вероятность выпадания 00 равна $4/9$, $10, 01$ – по $2/9$. Если выпала 11 – снова запускаем генератор два раза подряд. Алгоритм снова будет работать в среднем за константу. В качества ответа снова три последовательности. Если выпала 00 – выдаем 1. Если выпала 10 или 01, то выдаем 0. Вероятность событий одинакова и равна 1/2.

3)

Сначала переведем p в 1/2, затем 1/2 в q.

Покажем, как перевести 1/2 в q. Найдем такое k, что $2^{k-1} \leqslant q \leqslant 2^k$. Будем включать генератор k раз подряд, в итоге получим какую-то последовательность из $0, 1$. Вероятность встречи любой из последовательностей равна $\frac{1}{2^k}$. Выберем любые d последовательностей, в результате получения которых мы будем останавливать работу алгоритма, также выберем c из них. Именно на них мы будем говорить да, на оставшихся из набора – нет. Тогда останется $2^k - d$ последовательностей, на которых мы заново запустим генератор на k раз.

Найдем матожидание работы такого алгоритма.

$$E_s = \frac{d}{2^k} + \frac{2^k - d}{2^k}(E_s + 1) \Rightarrow N = k\frac{2^k}{d} \Rightarrow N = O(\log d) = O(\log cd).$$

4.

1)

В случае равенства матриц мы всегда будем получать ответ да. В случае неравенства вероятность ошибки меньше $1/2$ при $N > n$. (см. следующий пункт.) Значит язык лежит в классе $co - RP$.

2)

Заметим, что система $A(Bx) = Cx$ задает n уравнений с n неизвестными, каждое из которых должно быть равно 0. Поэтому если мы сложим квадраты всех уравнений и приравняем к нулю, то получим решения, которые бы мы получили, решив исходную систему, таким образом, данное уравнение зависит от n переменных и имеет порядок 2. По лемме Шварца-Зиппеля вероятность получения ответа ноль в случае неравенства матриц оценивается следующим образом

$$P(f(x_1, x_2, \ldots, x_n) = 0) \leqslant \frac{2n}{N}$$

3)

$$P(f(x_1, x_2, \ldots, x_n) = 0) \leqslant \frac{2n}{N} = p \Rightarrow N \geqslant \frac{2n}{p}$$

4)

В случае возможности повторного запуска алгоритма вероятность будет возводиться в степень, а значит если сделаем k запусков, то

$$N \geqslant \frac{2n}{p^{1/k}}$$

5.

1)

Запишем формулу полного разложения.

$$Det(A) = \sum_{\pi \in S_n} sign(\pi) a_1^{\pi_1} a_2^{\pi_2} \ldots a_n^{\pi_n}$$

Пусть в графе есть паросочетание. Тогда этому паросочетанию будет соответствовать некоторый набор вершин, который будет давать ненулевой вклад в сумму. В силу того, что все переменные независимы, то никаких сокращений в сумме не будет, а значит, детерминант будет отличен от нуля, ровно как и перманент.

Допустим, что детерминант отличен от нуля, тогда однозначно найдутся ненулевые слагаемые в сумме. Заметим, что такое слагаемое в точности определяет полное паросочетание. А значит перманент отличен от нуля.

2)

Заметим, что если в двудольном графе меньше чем n ребер, то полного паросочетания быть не может, поэтому сначала пройдемся по матрице смежности, если ребер меньше n, то выведем ответ нет, если больше или равно будем считать детерминант. Количество вершин в двудольном графе на $2n$ вершинам может быть быть равно любому число от n до n^2.

Оценим вероятность ошибки по лемме Шварца-Зиппеля. Детерминант это многочлен n степени от t переменных, где t–количество ребер в графе. Поэтому вехняя оценка ошибки равна $\frac{n^3}{N}$. Битовая сложность также оценивается как $O(n^2)$.

3)

Давайте выбирать ребро так, чтобы в оставшемся графе детерминант был также отличен от нуля, это будет означать, что выбрав какое-то ребро таким образом, мы сможем восстановить паросочетание, зная оставшуюся часть в меньшем графе. Детерминант мы можем считать за $O(n^3)$. Выбрать ребро за $O(n)$. И проделать так n раз. Грубая оценка для этого алгоритма $O(n^5)$. Если выбрать N достаточно большим, то вероятность ошибки будет крайне мала по второму пункту, данный алгоритм в среднем полиномиальный.

6.

Среди чисел от 1 до $m-1$ есть как взаимно-простые с m, так и не взаимно-простые. Если число не взаимно-простое, то оно ни в какой степени не может давать остаток 1 по модулю m, иначе бы это означало, что для этого числа существует обратное, а значит оно лежит в мультипликативной группе вычетов по модулю m. Составим эту группу. Понятно, что туда попадут только взаимно-простые с m числа и только они.

Пусть у нас имеется t чисел, которые в степени $m-1$ сравнимы с 1. Так же есть число b, которое в данной степени такой остаток не дает. Давайте подействуем на эту группу числом b.

Заметим, что

$$(ba_i)^{m-1} = b^{m-1}a_i^{m-1} = b^{m-1} \not\equiv 1(modm).$$

Таким образом, имея t чисел, дающих в нужной степени остаток 1, мы можем найти столько же чисел, не дающих этот же остаток. А значит, среди чисел от 1 до m найдется хотя бы половина чисел, обладающих нужным свойством.

7.

Заметим, что в силу простоты числа m в группе $\varphi(m) = m-1$ элемент. Пусть образующий элемент c. Рассмотрим произвольный элемент группы $a \equiv c^n$. Посмотрим разложение числа t на элементарные делители. Если n делится на $2^t, t \geqslant s$, то нам достаточно возвести число a в степень d, чтобы получить единицу.

Пусть теперь $t \leqslant s-1$. Заметим, что элемент -1 нашей группы имеет порядок $2^{s-1}d$, ведь нам нужно возвести его в квадрат, чтобы получить 1. Пусть $n = 2^t r, r = 2k+1$. Заметим, что если мы возведем число a в степень $2^w d = 2^{s-1-t}d$, то получим -1.

$$(a)^{2^w d} = c^{n2^w d} = c^{2^{t+s-t-1}dr} = c^{2^{s-1}dr} = (-1)^r = -1.$$

8.

Заметим, что если в последовательности x_k произошел повтор, то в y_k он произошел либо сейчас, либо раньше. Случай, когда повтор произошел раньше будет вести алгоритм к правильному ответу. Поэтому найдем вероятность второго случая. По парадоксу дней рождений первый повтор в псевдослучайной последовательности при выборке из n чисел случится на $O(\sqrt{n})$ элементе. Для оценки ошибки алгоритма будем считать, что он случится именно на этом номере последовательности. Таким образом, найдем вероятность того, что на $\sqrt{p}$ числах при выборке из m все числа будут различны. Воспользуемся для этого парадоксом дней рождений.

$$p < e^{\frac{-(\sqrt{p}-1)^2}{2m}}$$

Нам нужно противоположное событие, то есть, что среди этих чисел нашлась хотя бы пара одинаковых.

$$p_f > 1 - p > 1 - e^{-\frac{p}{2m}}$$