Консультация перед midterm. 21 марта 2019 г.

Подготовил: Горбунов Э.

Ключевые слова: AMB, midterm

Литература: [Кормен 1], [Кормен 2], [ДПВ], [Виноградов], [лекции Мусатова Д.В.]

Задача №1. Оценить асимптотику $T(n)$, если $T(n) = T(n-1) + 2T(n-2) + \ldots + 2^{n-1}T(0)$, где $T(0) = 1$.
Решение. Из условия получаем:

$$\begin{aligned} T(n+1) &= T(n) + 2T(n-1) + \ldots + 2^n T(0) \\ &= T(n) + 2\underbrace{\left(T(n-1) + 2T(n-2) + \ldots + 2^{n-1}T(0)\right)}_{T(n)} \\ &= 3T(n), \end{aligned}$$

откуда и из начального условия следует, что $T(n) = 3^n$.

Задача №2. Пусть $L \in$ со-$\mathcal{NP}$ и $N \subset L$, причём $N \neq \varnothing$ и $N \neq L$. Верно ли, что $N \in$ со-$\mathcal{NP}$?

Ответ: нет, не верно.
Решение. Пусть $L = \Sigma^*$. Тогда $L \in \mathcal{P} \subseteq$ со-$\mathcal{NP}$. Так как любой язык — это подмножество языка Σ^*, то в качестве N можно взять любой язык не из со-$\mathcal{NP}$. Например, подойдёт любой неразрешимый язык в качестве N.

Задача №3. Записать в терминах O, o, Ω, ω утверждение «функция f растёт быстрее любого полинома».
Решение. $f(n) = \omega\left(n^{O(1)}\right)$.

Задача №4. Язык EUCLID состоит из троек (a, b, c) таких, что НОД$(a, b) = c$ (числа задаются в двоичном виде). Язык NOHAMPATH состоит из описаний графов, которые не содержат гамильтонова цикла. Пусть NOHAMPATH полиномиально сводится к EUCLID. Верно ли, что тогда $\mathcal{NP} =$ со-$\mathcal{NP}$.

Ответ: да, верно.
Решение. Это означает, что со-$\mathcal{NP} = \mathcal{P}$, т.к. язык NOHAMPATH полон в классе со-$\mathcal{NP}$ (так как его дополнение (HAMPATH) полно в $\mathcal{NP}$). Но тогда $\mathcal{P} = \mathcal{NP} =$ со-$\mathcal{NP}$ в силу замкнутости класса $\mathcal{P}$ относительно взятия дополнения.

Задача №5. Язык supw$L$ надслов языка L определим по формуле: supw$L = \{u = wsv \mid w, v \in \Sigma^*,\ s \in L\}$. Верно ли, что $\forall L \in \mathcal{P} \hookrightarrow$ supw$L \in \mathcal{P}$.

Ответ: да, верно.
Решение. Нужно найти подстроку из языка L. Для этого будем перебирать все подстроки длины 1, длины 2, длины 3 и т. д. Пусть $p(n)$ — полином, ограничивающий время работы алгоритма, распознающего язык L, на входе длины n. Тогда время работы построенного алгоритма не превосходит $n \cdot p(1) + (n-1) \cdot p(n) + \ldots + 2 \cdot p(n-1) + p(n) \leqslant n^2 \cdot p(n)$. Последнее неравенство верно не для всех полиномов, но мы можем взять в качестве $p(n)$ возрастающий полином (просто взять $p(n)$ так, чтобы все его коэффициенты были положительными; это не уменьшит его значения во всех положительных точках).

Задача №6. Определим граф G_n на множестве $\{1, 2, \ldots, n\}$, где вершины — это числа, а две вершины соединены ребром тогда и только тогда, когда одно число делится на другое. Верно ли, что язык $L = \{(n, k) \mid \text{в } G_n \text{ есть клика размера } k\}$ принадлежит классу $\mathcal{NP}$-complete? Числа задаются двоичной записью.

Ответ: нет, неверно.

Решение. Оценим размер максимальной клики в графе G_n. Для этого пройдём в ней от минимума до максимума. Получим последовательность $1 \to k_1 m \to k_1 k_2 m \to \ldots \to k_1 k_2 \ldots k_l m$, где все $k_j \geqslant 2$ (в максимальную клику всегда входит число 1, потому что оно делит любое число). Поэтому чисел в максимальной клике не более $\lfloor \log_2 n \rfloor + 1$. Поэтому, если $k > \lfloor \log_2 n \rfloor + 1$, то выдаём ответ 0. В противном случае в графе есть клика нужного размера: числа $1, 2, 2^2, \ldots, 2^{k-1}$ образуют клику либо нужного размера. При этом наибольшее число в этой клике меньше n, т. к. $2^{k-1} \leqslant 2^{\lfloor \log_2 n \rfloor} \leqslant n$. Таким образом, разрешающий алгоритм работает за одно сравнение.

Задача №7. В массиве $a[1 \ldots N]$ записано N целых чисел. Все встречаются по 2 раза, кроме одного, которое встречается 3 раза. Требуется найти число, встречающееся 3 раза. Ограничения следующие: время работы — $O\left(N \log\left(\max\limits_i a[i]\right)\right)$, память — $O\left(\log\left(\max\limits_i a[i]\right)\right)$.

Решение. Сделаем побитовую операцию XOR всех записанных чисел (это делается за время $O\left(N \log\left(\max\limits_i a[i]\right)\right)$; при этом память, которую мы используем есть $O\left(\log\left(\max\limits_i a[i]\right)\right)$). Полученное число — это искомое число (проверьте это).

Задача №8. В массиве $a[1 \ldots N]$ записано N целых чисел. Все встречаются по 3 раза, кроме одного, которое встречается 1 раз. Требуется найти число, встречающееся 1 раз. Ограничения следующие: время работы — $O\left(N \log^3\left(\max\limits_i a[i]\right)\right)$, память — $O\left(\log\left(\max\limits_i a[i]\right)\right)$.

Решение. Переведём числа в троичную систему счисления. Для каждого отдельного числа это делается за $O(\log(a[i]))$ делений на число 3 (вычитания мы не учитываем). В итоге за $O\left(N \log^3\left(\max\limits_i a[i]\right)\right)$ можно преобразовать все числа в троичную систему счисления. Далее мы складываем числа поразрядно по модулю 3. То, что получится в итоге, и есть искомое число.

Задача №9. Покажите, что 2-SAT принадлежит $\mathcal{P}$.

Решение. Рассмотрим граф на $2n$ вершинах, соответствующих парам литералов, которые встречаются в формуле. Для любого дизъюнкта вида $a \vee b$, встречающегося в формуле, проведём два ориентированных ребра: $\neg a \to b$ и $\neg b \to a$ (опираемся на тот факт, что $a \vee b$ эквивалентно $(\neg a \to b) \wedge (\neg b \to a)$; см. рисунок 1). Смысл в том, что если b не равняется единице, то a равняется единице (и наоборот); здесь на ребро можно смотреть как на импликацию. Во-первых, в полученном графе существует путь из вершины a в вершину b тогда и только тогда, когда существует путь из $\neg b$ в $\neg a$. Действительно, если есть путь $a \to a_1 \to \ldots \to a_k \to b$, то есть и путь $\neg b \to \neg a_k \to \ldots \to \neg a_1 \to \neg a$, так как построению для любого ребра $a \to b$ есть ребро $\neg b \to \neg a$.

Теперь покажем, что исходная КНФ выполнима тогда и только тогда, когда не существует пути из x в $\neg x$ и не существует пути из $\neg x$ в x одновременно для некоторой переменной x. Пусть в графе есть путь из x в $\neg x$ и путь из $\neg x$ в x. Предположим, что существует выполняющий набор. Тогда, если $x = 1$ в этом наборе, то в пути $x \to x_1 \to \ldots \to x_{l-1} \to x_l \to \neg x$ все литералы должны принемать значение равное единице (в силу того, что ребру $\alpha \to \beta$ соответсвует импликация $\alpha \to \beta$, которая эквивалентна дизъюнкту $\neg\alpha \vee \beta$, который в свою очередь равен нулю только на наборе $\alpha = 1, \beta = 0$). Но тогда $x = \neg x = 1$, противоречие. Если же $x = 0$, то $\neg x = 1$, и, аналогично рассматривая путь $\neg x \to y_1 \to \ldots \to y_{t-1} \to y_t \to x$, получаем противоречащее равенство $\neg x = x = 1$.

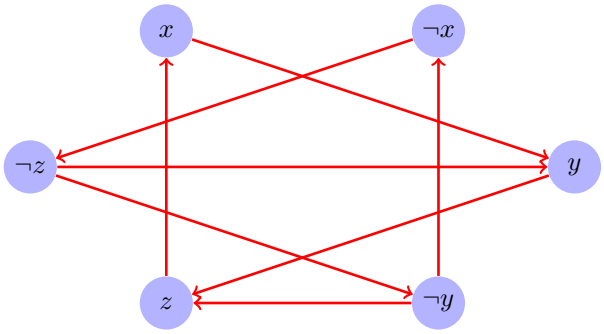

Рис. 1: Граф, построенный по формуле $(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$

В обратную сторону: пусть для любой переменной x из вершины x нельзя достичь $\neg x$ и из вершины $\neg x$ нельзя достичь x одновременно. Покажем, что тогда существует выполняющий набор. Не умаляя общности, пусть из x не достижима вершина $\neg x$. Тогда не существует такой вершины y, что из x достижима вершина y, из которой достижима $\neg y$. Действительно, если бы такая вершина существовала, то существовал бы и путь из $\neg y$ в $\neg x$, как мы отметили ранее. Но тогда можно было бы из x попасть в вершину y, затем в $\neg y$, а потом в $\neg x$, то есть существовал бы путь из x в $\neg x$. Противоречие. Тогда выполняющий набор можно построить следующим способом: рассмотрим литерал x такой, что из x не достижим $\neg x$. Далее рассмотрим все вершины, достижимые из вершины x (включая x) и присвоим соответствующим литералам значение 1. Конфликтов возникнуть не может, т.к. если из x достижим и y и $\neg y$, то есть путь из x в y, а затем из y в $\neg x$, то есть путь из x в $\neg x$. А таких путей для вершины x нет по предположению. Таким образом мы получим множество литералов A_x, значения которых равны единице и которые достижимы из x. Этому множеству литералов также соответствует множество литералов B_x, состоящее из отрицаний литералов множества A_x. Значит, во множестве B_x все литералы равны нулю, конфликтов быть не может. При этом во множество B_x больше не входит никаких рёбер не из множества B_x, а из множества A_x не выходят рёбра в вершины не из множества A_x. Это означает, что мы частично (возможно и полностью) присвоили значения литералам так, что получился граф, у которого из присвоенных вершин выходит ребро либо в присвоенные вершины, а в неприсвоенные вершины выходит ребро только из нулевых вершин; аналогично, рёбро входит в присвоенную вершину, если ей присвоено значение 1, либо если это ребро проведено из другой присвоенной вершины. Иными словами, мы можем отбросить присвоенные вершины (вместе с рёбрами входящими и исходящими из них) и присваивать оставшимся вершинам значения по аналогии с тем, как мы это сделали для вершины x. При этом конфликтов, связанных с рёбрами, которые мы откинули не будет: мы отбросили рёбра, соответствующие импликациям вида $a \to 1$ и $0 \to b$, которые всегда выполнено, вне зависимости от a и b. Такими дийствиями мы получим набор, который выполняет все импликации, которые соответствуют рёбрам, а значит, и все дизъюнкты в исходной формуле.

Поэтому достаточно проверить, что в графе нет путей между x и $\neg x$, $\neg x$ и x для каждой переменной, встречающейся в формуле. Это можно сделать для каждой пары поиском в ширину.

Задача №10. Определим язык DOUBLE-SAT — язык описаний КНФ, имеющих по крайней мере два выполняющих набора. Доказать, что DOUBLE-SAT $\in \mathcal{NP}$-complete.

Решение. Во-первых, язык DOUBLE-SAT лежит в $\mathcal{NP}$ (сертификатом будет не один набор, как для SAT (ВЫПОЛНИМОСТЬ), а два набора; проверяющая процедура будет вызывать проверяющую процедуру для языка SAT для кадого из двух наборов). Теперь построим сведение по Карпу языка SAT к языку DOUBLE-SAT. По КНФ φ построим КНФ $\psi = \varphi \wedge (y_\psi \vee \neg y_\psi)$, где y_ψ — новая переменная (не встречающаяся в φ). Если φ выполнима, то существует набор переменных, на котором она равна единице, а значит, существует хотя бы два набора переменных, на котором ψ равна единице (y_ψ можно взять и нулём, и единицей, а остальные значение переменных — как в выполняющем наборе для φ). Обратно,

если ψ имеет хотя бы 2 выполняющих набора, то тем более существует набор, выполняющий формулу ϕ.

Задача №11. Пусть

$$\text{SUBSET-SUM} = \left\{ (S, \sigma) \mid S = \{x_1, \ldots, x_k\} \subset \mathbb{N}, \sum_{i=1}^{k} \alpha_i x_i = \sigma \text{ для некоторого булевого набора } \alpha_1, \ldots, \alpha_k \right\},$$

а UNARY-SSUM — это точно тот же язык, но числа во множестве S и число t задаются в унарном алфавите. Докажите, что SUBSET-SUM $\in \mathcal{NP}$-complete и UNARY-SSUM $\in \mathcal{P}$.

Решение.

(a) Сведём язык 3-SAT (3-ВЫПОЛНИМОСТЬ) к языку SUBSET-SUM. Рассмотрим произвольную 3-КНФ φ, в которой есть n переменных $x_1, \ldots, x_n$ и m дизъюнктов $c_1, \ldots, c_m$. Построим множество S и число σ следующим образом. Для каждой переменной построим два числа длины $n+m$ в десятичной системе счисления: t_i и f_i. Зададим их по правилам:

- для $i = 1, 2, \ldots, n$ i-е цифры чисел t_i и f_i равны единице;
- число t_i имеет j-ю цифру равную 1 для $j = n+1, n+2, \ldots, n+m$, если литерал x_i есть в дизъюнкте c_{j-n};
- число f_i имеет j-ю цифру равную 1 для $j = n+1, n+2, \ldots, n+m$, если литерал $\neg x_i$ есть в дизъюнкте c_{j-n};
- остальные цифры чисел t_i и f_i положим равными нулю.

Получили $2n$ чисел. Добавим во множество S ещё $2m$ чисел длины $n+m$. Для каждого дизъюнкта c_i построим числа y_i и z_i, которые имеют $(n+i)$-ю цифры равную единице, а остальные цифры — нули. Например, для 3-КНФ $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$ мы получим следующие числа (для удобства запишем это всё в виде таблицы: в каждой строчке, начиная со второй, записаны соответствующие числа; после первых n цифр запишем разделитель опять-таки исключительно для удобства).

	1	2	3	#	1	2	3	4
t_1	1	0	0	#	1	0	0	1
f_1	1	0	0	#	0	1	1	0
t_2	0	1	0	#	1	0	1	0
f_2	0	1	0	#	0	1	0	1
t_3	0	0	1	#	1	1	0	1
f_3	0	0	1	#	0	0	1	0
#	#	#	#	#	#	#	#	#
y_1	0	0	0	#	1	0	0	0
z_1	0	0	0	#	1	0	0	0
y_2	0	0	0	#	0	1	0	0
z_2	0	0	0	#	0	1	0	0
y_3	0	0	0	#	0	0	1	0
z_3	0	0	0	#	0	0	1	0
y_4	0	0	0	#	0	0	0	1
z_4	0	0	0	#	0	0	0	1

Числом σ у нас будет число длины $n+m$ в десятичной записи, у которого i-я цифра равна 1 для $1 \leqslant i \leqslant n$ и j-я цифра равна 3 для $n+1 \leqslant j \leqslant n+m$. Таким образом, мы построили полиномиально вычислимую сводящую функцию. Утверждается, что исходная 3-КНФ выполнима тогда и только тогда, когда полученная пара (S, t) лежит в SUBSET-SUM.

Во-первых, если формула выполнима, то нужно взять в сумму t_i, если $x_i = 1$ на выполняющем наборе, и f_i — в противном случае; кроме того, возьмём y_i в сумму, если в c_i не более двух литералов

равно единице, а если ровно один литерал — возьмём ещё и y_i. Тогда первые n цифр суммы будет равны единице, а последние m цифр будут равны 3, что и требовалось, то есть (S, σ) лежит в SUBSET-SUM.

Обратно: пусть (S, σ) лежит в SUBSET-SUM. Покажем, что формула выполнима. Выполняющий набор строится по слагаемым, входящим в сумму точно так же, как мы выбирали слагаемые в сумму по выполняющему набору (нужно только всё в обратном порядке проделать).

(b) Теперь покажем, что UNARY-SSUM лежит в $\mathcal{P}$. Заметим, что размер входа вырос экспоненциально. Если перевести его в двоичную систему счисления, то получим задачу SUBSET-SUM, которая полна в NP. Поскольку $\mathcal{NP} \subseteq$ EXPTIME и экспоненциальное по входу языка SUBSET-SUM время — это полиномиальное по входу языка UNARY-SSUM время, то UNARY-SSUM лежит в $\mathcal{P}$.

Задача №12. Покажите, что если на вход подаётся двоичное число, гарантированно являющееся полным квадратом, то извлечь квадратный корень из него можно за полиномиальное время.

Решение. Будем использовать бинарный поиск[1]: делим отрезок на две равные (или почти равные части) и берём «среднее» число c (договоримся, что будем брать $c = \left\lfloor \frac{a+b}{2} \right\rfloor$, где a и b — левая и правая границы текущего отрезка соответственно; в самом начале $a = 1$ и $b = n$). Если $c^2 > n$, то переходим к правой половине: $a := c$ и $b := b$; если $c^2 < n$, то переходим к левой половине: $a := a$ и $b := c$; если же $c^2 = n$, то останавливаемся и выдаём ответ c. Заметим, что делений отрезка пополам мы сделаем не больше $\log_2 n$ делений отрезка пополам. После каждого деления отрезка пополам мы вычисляем результат деления одного числа на другое, возводим число в квадрат и сравниваем два числа (длины всех получаемых в процессе работы чисел не больше $2\log_2 n$, так как $\log_2 n^2 = 2\log_2 n$). Сложность алгоритма: $O(\log_2^3 n)$. Отметим, что этот алгоритм может ответить также и на вопрос, является ли целое число полным квадратом, если в конце возвращать «нет», если не будет найдено нужно число, когда левая и правая границы текущего отрезка совпадут.

Задача №13. Сведите задачу о гамильтоновом пути в **ориентированном** графе к задаче о гамильтоновом пути в **неориентированном** графе.

Решение. Рассмотрим ориентированный граф $G = (V, E)$. Построим по нему неориентированный граф $G' = (V', E')$. Каждую вершину $v \in V$ "расщепим"на три вершины v_{in}, v_{mid} и v_{out} следующим образом: если в графе G было ребро (u, v) (то есть входящее в вершину v), то добавим в граф G' неориентированное ребро (u_{out}, v_{in}); аналогично, если в графе G было ребро (v, h) (то есть выходящее из вершины v), то добавим в граф G' неориентированное ребро (v_{out}, h_{in}). Для каждой пары вершин v_{in} и v_{out} добавим рёбра (v_{in}, v_{mid}) и (v_{mid}, v_{out}) в граф G'. Кроме того, добавим ещё 4 вершины a, b, c, d: вершина a будет соединена только с вершиной b, вершина d — только с вершиной c, а вершину b соединим со всеми вершинами, имеющими пометку in, а вершину c — всеми вершинами, имеющими пометку out. Заметим, что процедура построения графа G' по графу G полиномиально вычислима. Если в графе G существовал гамильтонов путь $v_1, v_2, \ldots, v_n$, то и в графе G' существует гамильтонов путь, а именно $a, b, v_{1,in}, v_{1,mid}, v_{1,out}, v_{2,in}, v_{1,mid}, v_{2,out}, \ldots, v_{n,in}, v_{n,mid}, v_{n,out}, c, d$. Далее заметим, что вершины с пометками in в графе G' соединены рёбрами только с вершинами с пометками out и ровно с одной вершиной с пометкой mid; вершины с пометками out в графе G' соединены рёбрами только с вершинами с пометками in и ровно с одной вершиной с пометкой mid; вершины с пометками mid в графе G' соединены рёбрами ровно с одной вершиной с пометкой in и ровно с одной вершиной с пометкой out. Кроме того, если в графе G' есть гамильтонов путь, то он начинается (заканчивается) в вершине a и заканчивается (начинается) в вершине d. Тогда, если в графе G' есть гамильтонов путь, то он имеет вид $a, b, v_{i_1,in}, v_{i_1,mid}, v_{i_1,out}, v_{i_2,in}, v_{i_2,mid}, v_{i_2,out}, \ldots, v_{i_n,in}, v_{i_n,mid}, v_{i_n,out}, c, d$. Это следует из двух простых фактов. Во-первых, в вершины a и d можно прийти только из вершин b и c соответственно. Во-вторых, из вершины b можно попасть только в вершину с пометкой in; если на каком-то шаге мы пойдём из вершины с пометкой in в вершину без пометки mid, то соответствующая вершина с пометкой mid не будет

[1] Пояснение для 678-й группы: здесь применяется тот же метод, что и в задаче №4 с семинара №3.

посещена: в неё можно будет прийти только из соответствующей вершины с пометкой *out*, но выйти из неё нельзя, не побывав второй раз в некоторой вершине. Значит, вид гамильтонового пути может иметь только указанный вид (либо в обратном порядке). Но тогда в исходном графе тоже есть гамильтонов путь: $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$.

Задача №14. Пусть L — это язык пар массивов $A[1..n]$ и $B[1..n]$, имеющих одинаковую длину и удовлетворяющих $A[i] = (B[i])^2$ для всех $i = 1, 2, \ldots, n$. Рассмотрим следующую вероятностную процедуру, которая использует функцию $\textsc{Rand}(n)$, возвращающую случайное целое число от 1 до n (все варианты равновероятны).

```
1: procedure NONSENSE(A, B)
2:     Считать длину массивов: n := |A| и m := |B| (за |A| + |B| итераций)
3:     if |A| ≠ |B| then
4:         return FALSE
5:     end if
6:     for i = 1, 2, 3 do
7:         j := RAND(n)
8:         if A[j] ≠ (B[j])^2 then
9:             return FALSE
10:        end if
11:    end for
12:    return TRUE
13: end procedure
```

Процедура $\textsc{Nonsense}(A, B)$ применяется для разрешения языка L.

(i) Оцените вероятности ошибок первого и второго рода.

(ii) Чему равно время работы процедуры в худшем случае? Оцените время работы в среднем (для данного входа; разобрать случаи, когда $(A, B) \in L$ и $(A, B) \notin L$)?

(iii) Показывает ли данный алгоритм принадлежность L к одному из вероятностных классов языков, введённых на прошлом семинаре?

Считать, что все вычисления с числами производятся на $O(1)$.

Решение.

(i) Найдём вероятность ошибки второго рода, то есть вероятность такого события, что $(A, B) \in L$, но наш алгоритм выдал FALSE. Эта вероятность равна 0: если $(A, B) \in L$, то IF в строчке 8 никогда не выполнится, а значит, алгоритм завершит цикл и выдаст ответ TRUE. Если же $(A, B) \notin L$, то рассмотрим случай, когда $|A| = |B| = n$ (в противном случае алгоритм не ошибается и вероятность ошибки равна нулю). Пусть свойство $A[i] = (B[i])^2$ не выполнено в $1 \leqslant k \leqslant n$ позициях. Тогда вероятность того, что мы этого не обнаружим равна $\left(\frac{n-k}{n}\right)^3$ (на каждой итерации цикла попадаем в ячейки, где это свойство выполнено).

(ii) Время работы в худшем случае равно $f(A, B) + 3$, где $f(A, B)$ — время нахождения длин массивов A и B. Время работы в среднем равняется:

1) $f(A, B) + 3$, если $(A, B) \in L$;
2) $f(A, B)$, если $|A| \neq |B|$;
3) $(f(A, B) + 1)\frac{k}{n} + (f(A, B) + 2)\frac{n-k}{n}\frac{k}{n} + (f(A, B) + 3)\left(\frac{n-k}{n}\right)^2 = f(A, B) + \frac{kn+2kn-2k^2+3n^2-6kn+3k^2}{n^2} = f(A, B) + 3 - \frac{3k}{n} + \frac{k^2}{n^2}$.

(iii) Что мы имеем: алгоритм полиномиален в худшем случае, вероятность ошибки второго рода всегда равна нулю, а вероятность ошибки первого рода равна $\left(\frac{n-k}{n}\right)^3$, где k — число «плохих» позиций. Если $k = 1$, то при больших n эта вероятность стремится к единице. Если $k = n$, то эта вероятность

равняется нулю. В общем случае ничего сказать нельзя, как вероятность ошибки первого рода ограничена, поэтому этот алгоритм не доказывает принадлежность языка L ни к одному из введённых вероятностных классов.