1 Доказать принадлежность P

1.1 Язык двудольных графов

Очевидно (по принципу Дирихле), что если мы берём три любые вершины двудольного графа, то хотя бы две из них будут в одной доле. По условию задачи нам нужно 2018 треугольников (т.е. три вершины, связанные между собой попарно), но по определению двудольных графов, вершины внутри каждой доли не связаны, поэтому этот язык пустой и распознаётся за $T = O(1)$, т.е. $\in \mathbf{P}$

1.2 Язык несвязных графов без циклов

Известный факт: поиск циклов в графе удобно организовывать с помощью поиска в глубину (DFS), который работает за $O(V + E)$. Если во время DFS мы натыкаемся на уже отмеченную вершину - цикл найден.
Получается, что если наш граф задан матрицей смежности, то длина входа $n^2 \equiv V^2$. И т.к. число рёбер ограничено $0 \leq E \leq (V-1)^2$ (для полного графа). Хотя можно доказать, что в итоге этот алгоритм будет $O(n)$(при отсутствии циклов граф будет более разреженным), нам достаточно и оценки сверху: $O(n+n^2) = O(n^2)$. Таким образом ассимптотика поиска цикла $T(n) < O(n^2)$ и язык полиномиален.

1.3 Язык квадратных матриц

Самый грубый способ проверки без каких либо оптимизаций: будем просто из каждой ячейки матрицы с координатами $i \in \overline{1, 2018}$ и $j \in \overline{1, 2018}$ проходиться по всем элементам квадратной подматрицы от i, j до $i + n - 2018, j + n - 2018$ и прерываться при нахождении хотя бы одного 0. Такой алгоритм будет работать в худшем случае ассимптотически за $O(n^4)$, что доказывает его принадлежность **P**

1.4 Язык остатков от деления степеней числа

Язык задают числа $a, m \in \mathbb{N}$. Заметим, что раз мы берём остаток от деления по m, то в самом языке не более m чисел, значит он конечный. А это значит. Что $\forall i \in \mathbb{N} \exists i_k : x_{i_k} = x_0$ и все остатки замкнутся по кругу.
Значит программе проверки достаточно перебрать (может и очень большое, но конечное и не зависящее от входа вообще никак) число остатков и записать их во множество, с которым мы будем сравнивать наш вход. В итоге для входного числа n (длина входа $O(logn)$), принадлежность которого надо проверить, мы совершим m сравнений за $O(logn)$ каждое.
Получим, что этот алгоритм выполняется за (огромную) константу + линию. Таким образом он линеен от входа, а значит $\in \mathbf{P}$

2 Показать замкнутость класса P

2.1 Относительно объединения и пересечения

У нас есть два языка $A, B \in \mathbf{P}$. Значит существуют МТ, разрешающие их за полиномиальное время. Тогда построим МТ объединения по следующей конструкции. Её ленты это объединение лент МТ A и МТ B (входное слово дублируется на вторую МТ), и ещё одна лента - бинарного

результата, из двух ячеек. На первую мы поместим 0, если входное слово $\omega \in A$, и 1 если $\omega \in A$. Аналогично во вторую ячейку положим $0B$, иначе положим 1. Таким образом на ленте бинарного результата лежит одна из комбинаций: $11, 10, 01, 00$. Первая означает условие $\omega \in A \cap B$, а первые три комбинации значат $\omega \in A \cup B$. И поскольку полиномиальность языков A и B значит, что поотдельности операции выполняются за $O(n^a)$ и $O(n^b)$ соответственно, то мы просто запустим проверки последовательно, и затем посмотрим на итог работы на ленте бинарного результата. Тогда итоговая ассимптотика: $O(n^a + n^b + c) = O(n^{max(a,b)})$

2.2 Относительно операции Клини

```
N=|v|;
ends={0};
for (i=1; i<=N; i++) {
        for (j in ends)
                if (MT(v[j+1...i])){
                        if(i==n) return true;
                        ends.include(i)
                }
        }
}
return false;
```

Суть алгоритма - фактически за $O(n^2)$ мы перебираем возможные разбиения v на подслова (индексы, по которым разбиваем хранятся в ends). Останавливаемся, когда слово удалось непрерывно (без непринадлежащих языку подслов) разбить. MT(v[j+1...i]) значит проверку на принадлежность слова языку. Оно полиномиально т.к. сам язык по предположению полиномиален. Тогда мы просто повышаем степень полинома на два, и получаем полином. Итоговый язык $L^* \in \mathbf{P}$

3 Пример языка, не принадлежащего P

Поскольку этот язык задаёт нам R, можно взять как пример задачу о клике (Clique Problem) - поиск максимального полного подграфа в графе.
Доказательство NP полноты этой задачи, и, как следствие, её непринадлежности классу **P**(при условии, что $\mathbf{P} \neq \mathbf{NP}$, разумеется) приводится в книге Томаса Кормена на 1137 странице (в 3ем издании).

4 Вызов полиномиальной программы

4.1 Конечное число раз

Дан вход длины n. И есть подпрограммы, выполняющиеся каждая поотдельности за $O(n^{i_k})$, где k - номер подпрограммы. Наша программа каждую из подпрограмм вызывает соответственно c_k раз. В итоге ассимптотика будет $O(\sum_{k=1}^{m}(c_k \cdot n^{i_k})) = O(max(c_k \cdot n^{i_k})) = O(n^{\max(i_k)})$ что является полиномом от длины, значит и сама программа $\in \mathbf{P}$

4.2 линейное по входу число раз

А вот тут уже не факт. Ведь из 1 семестра матана нам известно разложение по формуле Тейлора. В данном случае просто применяем её для очень больших n к ряду:

$$T(n) = \sum_{i=1}^{n} \overbrace{c_i}^{\text{вызовов}} \cdot \underbrace{n^i}_{\text{подпрограмма}} > \sum_{i=0}^{n} \frac{n^i}{i!} = e^n$$

Таким образом мы из линейного повторения полиномиальных по сложности функций получили экспоненциальный рост (ну в каком-то его приближении). И в данном примере время не полиномиально, а экспоненциально.

5 Полиномиален ли алгоритм?

5.1 Деление в столбик нацело

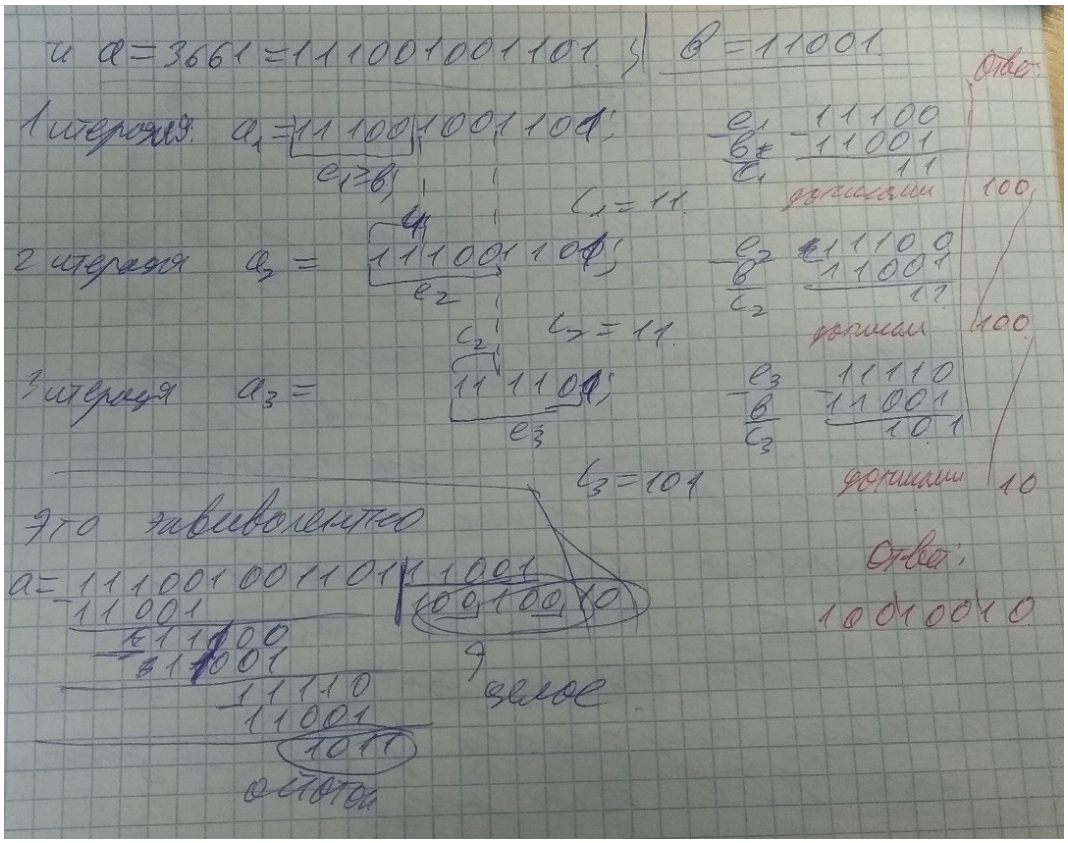
Ответ: да.
Сам алгоритм выглядит так: У нас даны числа a -делимое и b - делитель, заданные в бинарном алфавите (т.е. длина входа $n = \log_2 a + \log_2 b$).

1. ищем число e - минимальную часть a сверху $\geq b$ (берём из a первые $log_2 b$ или $log_2 b + 1$ знаков за константное время). Если знаков $log_2 b + 1$, то дописываем в ответ 0.

2. $c := b - e$. Дописываем в ответ 1 и $log_2 b - log_2 c - 1$ нуль

3. Начинаем алгоритм с первого пункта, написав вместо в начало a вместо e, значение c

Всё это происходит, пока мы можем выделить такое число e. Для помощи вот картинка с примером:



Видно, что мы делаем в самом худшем случае (при большом a и малом b) порядка $O(log_2 a)$ итераций, в которых осуществляются только константные (ну или ценой $O(log_2 b)$) операции. Таким образом данный алгоритм полиномиален.

5.2 Деление в столбик нацело

Ответ: да

На семинаре был разобран алгоритм быстрого возведения в степень (со степенями двойки). Внешний цикл прокручивается $log_2 b$ раз, поскольку мы делим b каждый раз на два, пока он не станет равным 0. В одной итерации совершаются только операции деления на 2(его можно делать за $O(1)$ побитовым сдвигом) и умножения, которые могут занять $O(log_2^2 a)$. Таким образом итоговая ассимптотика $O(log_2^2 a \cdot log_2 b) = O((log_2 max(a,b))^3$ т.е. полиномиально.

5.3 Проверка на простоту за $\sqrt{n}$

Ответ: нет

Нам дан вход - число n, которое надо проверить. Пусть оно дано нам в k-ичной (k>1) системе исчисления (2 или 10, например). Тогда длина входа $log_k n$. Но сам алгоритм предполагает проверку $\sqrt{n}$ раз. Даже если мы будем считать, что проверка делимости (которая на самом деле осуществляется за $log_k n$) операций), каким-то чудом работает за $O(1)$, мы получим ассимптотику

$$O(\sqrt{n}) \sim \Big/ \sqrt{n} = (\log_k n)^x \Rightarrow x = \frac{1}{2}\log_{log_k n} n \Big/ \sim O\left((logn)^{\frac{1}{2}log_c n}\right)$$

Что является полиномом от длины.
Более того, вроде бы, если брать основание равным единице(унарная система исчисления), то вход имеет длину n и, казалось бы, всё будет хорошо. Но нет, тут уже мы обязаны учитывать то, что все эти операции (взятие корня, деление) будут не полиномиальны, а экспоненциальны.