Задание на вторую неделю.

№1

1) Рефлексивность: $A \leq_p A$, т. к. если взять $f(x) = x$, то условие сводимости очевидно выполняется.
Транзитивность: Пусть $f_1(x)$ функция приведения от A к B, а $f_2(x)$ функция приведения от B к C. Тогда из определения сводимости:

$$x \in A \Leftrightarrow f_1(x) \in B \Leftrightarrow f_2(f_1(x)) \in C.$$

Значит композиция этих функций $f_2 \circ f_1$ сведет полиномиально(т. к. f_2 и f_1 вычисляются за полиномиальное время) A к C. $\Rightarrow A \leq_p C$, ч. т. д. .
2) Приведем алгоритм проверки принадлежности x к языку L:

- Вычислить $f(x)$. Полиномиальная сложность.
- Проверить принадлежность $f(x)$ к языку B. Полиномиальная сложность, т. к. $B \in P$.
- Из определения сводимости, если оказалось, что $f(x) \in B$, то $x \in A$, иначе x не принадлежит A.

Таким образом, алгоритм имеет полиномиальную сложность, $\Rightarrow A \in P$.
3)Пусть $f_1(x)$ функция приведения от A к B, вычисляемая за полиномиальное время. Т. к. $B \in NP$, то существует алгоритм V с полиномиальным временем работы, такой что $x \in B \Leftrightarrow \exists s : V(x, s) = 1$. Введем новый алгоритм верификации V_1: $V_1(x, s) = V(f(x), s)$. Тогда из определений сводимости и NP языка:

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow \exists s : V(f(x), s) = 1 \Leftrightarrow \exists s : V_1(x, s) = 1 \Leftrightarrow A \in NP,$$

ч. т. д. .

№2

1) Заметим, что язык является пустым, т. к. если в графе есть хотя бы один треугольник, то он не является двудольным(три смежные

вершины невозможно раскрасить правильно в два цвета). Значит, язык полиномиален, т. к. проверка принадлежности к нему занимает $O(1)$ (для любого x ответ: нет).
2) Воспользуемся алгоритмом из курсов алгоритмов для проверки графа на связность с помощью обхода в глубину(во время обхода считаем количество посещенных вершин, если после завершения алгоритма оно совпало с количеством вершин, то граф связен, иначе не связен), который работает за полиномиальное от длины входа время. После еще раз запустим поиск в глубину для выявления циклов(во время обхода будем при входе в вершину окрашивать ее в красный, если в какой-то момент придем в уже красную вершину, то граф содержит цикл, если же алгоритм завершится, и таких вершин не найдется, то в графе нет циклов). По результатам двух обходов можно определить принадлежность языку. $\Rightarrow$ язык принадлежит P.
3) Длина входа n^2. В матрице порядка n можно выбрать 2018^2 различных квадратных подматриц размером $n - 2018$ (2018 способов выбрать, сколько оступить слева и справа по горизонтали, чтобы суммарный отступ был на 2018, и аналогично по вертикале). Для каждой такой подматрцы сравним все ее элементы с единицами: это займет $O((n - 2018)^2) = O(n^2)$. Таким образом, за $2018^2 \cdot O(n^2)$, полиномиальное от длины входа время, можно проверить, есть ли в матрице единичная подматрица размером $n - 2018$, а это определяет принадлежность матрицы языку. $\Rightarrow$ Данный язык принадлежит классу P.

№4

2)Предположим, что $L \in NP$. Тогда существует алгоритм V с полиномиальным временем работы, такой что $x \in L \Leftrightarrow \exists s : V(x, s) = 1$.
Пусть $x \in L^*$, тогда x можно разбить на некоторый упорядоченный набор c слов из L. Обратно, если для x существует такое разбиение c, то $x \in L^*$.
Рассмотрим алгоритм $V_1(x, S)$, принимающий на вход слово x и аргумент S, состоящий из упорядоченного набора слов c из языка L, разбивающего слово x, и сертификатов $s(c)$, соответствующих данным словам(т.к. эти слова принадлежат L, то такие сертификаты обязательно найдутся). Работа алгоритма заключается в том, чтобы

проверить, что упорядоченный набор слов c, состоит из слов языка L(это он делает с помощью соответстующих сертификатов $s(c)$ и верификации V за полиномиальное время) и что он является разбиением слова x(сложность этого $O(|x|)$).
Из сказанного вначале следует, что если x не принадлежит L^*, то для него нельзя подобрать такое c, а значит и S, такое чтобы $V_1(x, S) = 1$. А если $x \in L^*$, то такое c, а значит и S подобрать можно. При этом длина c будет полиномиальна(даже равна ей) от длины входа, соответствующие им сертификаты будут также полиномиальной длины(из определения $L \in NP$), а число сертификатов равно числу слов в разбиении, т. е. не более длины входа. Таким образом, длина подобранного S, равная $c + s(c)$, полиномиальна от длины входа. Значит $x \in L^* \Leftrightarrow \exists S : V_1(x, S) = 1$. $\Rightarrow L^* \in NP$. А сертификатом принадлежности служит описанный выше набор величин S.
1) Пусть $L \in P$. Предположим, что для слова длиной n мы умеем проверять принодлежность языку L^* полиномиальным алгоритмом. Построим алгоритм для слова x длиной $n + 1$:
Слово лежит в $L^* \iff$ у него есть такой префикс, что он лежит в языке L, а оставшаяся часть слова(без этого префикса, дальше будем называть ее окончанием) лежит в L^* *(усл. (1))*.
Будем перебирать пары слов: (префикс, соответствующее окончание), по длине(начиная с префикса длины 1 до $|x|$), проверяя у каждой такой пары принадлежит ли префикс языку L(это полиномиальное действие) и принадлежит ли окончание языку L^* (тоже полином по предположению). Перебрав всевозможные пары(т. е. $|x|$ штук), проверим выполнимость для данного слова усл. (1), а оно определяет принадлежность слова языку. Таким образом, мы построили полиномиальный алгоритм проверки принадлежности языку L^*для слова длины $n + 1$.
Заметим, что для слова длины 1 мы уже умеем строить искомый полиномиальный алгоритм, т. к. однобуквенное слово принадлежит $L^* \iff$ это слово принадлежит L.
$\Rightarrow$ Для слова любой длины существует полиномиальный алгоритм проверки принадлежности к L^*. $\Rightarrow L^* \in P$, ч. т. д. .

№5

Пусть дана система линейных уравнений. Введем алгоритм верификации $R(x, s)$, который принимает на вход систему линейных уравнений x и число s, и проверяет, что x несовместна. Это можно сделать по модифицированному алгоритму Гаусса из задачи три за полиномиальное время. Т.е. $R(x, s)$ не зависит от s. Тогда $x \in L \Leftrightarrow \exists s$ (например, можно всегда брать сертификат $s = 0$): $R(x, s) = 1$. $\Rightarrow L \in NP$.

№6

1) Возьмем в качестве сертификата делитель $1 < d \leq M$, его длина $\leq \log N$, а алгоритм верификации $V((N, M), s)$ будет проверять, что $d \leq M$(сложность этого, побитовое сравнение, $O(\log M)$) и что N делится на d(сложность $O(n^2)$). Очевидно, что $x \in L_f \Leftrightarrow \exists s : V((N, M), s) = 1$. $\Rightarrow L_f \in NP$.
2) $L_{nef} = \overline{L_f} = \{(N, M) \in \mathbb{Z}^2 \mid 1 < M < N$ и $(N, d) = 1\ \forall d : 1 < d \leq M\} \cup \overline{L_0}$, где $L_0 = \{(N, M) \in \mathbb{Z}^2 \mid 1 < M < N\}$.
Пусть алгоритм $V(x, s)$ принимает на вход x: пару чисел $(N, M) = x$, и набор чисел $s = n_1, n_2, \ldots, n_k$. При этом $V(x, s) = 1$, тогда и только тогда, когда x лежит в $\overline{L_0}$ ИЛИ выполняются одновременно следующие условия $(1) - (3)$:

1. $N = n_1 \cdot n_2 \ldots \cdot n_k$
2. $\forall i : 1 \leq i \leq k \rightarrow n_i \in Pr$, где Pr язык простых чисел.
3. $\forall i : 1 \leq i \leq k \rightarrow n_i > M$.

Оценим сложность работы алгоритма. Проверка принадлежности x языку $\overline{L_0}$ занимает $O(\log N + \log M)$. Для проверки условия (1) необходимо перемножить числа $n_1, \ldots n_k$. Чисел больших единицы не более $\log_2 N$, а значит операцию умножения, сложность которой $O(\log^2 N)$, надо повторить не более $\log N$. Т.е. сложность проверки (1): $O(\log^c N)$. Для проверки условия (2) достаточно полиномиального алгоритма: доказано, что язык простых чисел принадлежит классу P, а значит необходимо провести не более $\log_2 N$ полиномиальных проверок на простоту. Условие (3) проверяется за $O(\log N)$ сравнений, т. е. за

$O(\log^2 N)$. Таким образом, сложность алгоритма $V(x, s)$ полиномиальная.
Если $x \in L_{nef}/\overline{L_0}$, то за s можно взять разложение N на простые множители и $V(x, s) = V((M, N), s)$ будет равно 1, т. к. условия $(1) - (2)$ очевидно выполняются, а если предполить, что условие 3 не пыполнено для некоторого n_i, то N делится на $1 \leq n_i \leq M$, а значит $x \in L_f = \overline{L_{nef}}$, а это невозможно. При этом длина $s = O(\log N \log N)$, полиномиальна. Если $x \in \overline{L_0}$, то можно взять любое s и получим, $V(x, s) = 1$.
Если $x \in L_f$, то у N есть делитель $d : 1 < d < M$. Предположим, что для такого x нашлось $s : V(x, s) = 1, s = n_1, n_2, \ldots, n_k$. Тогда, т. к. $d \leq M < n_i \forall i$ и n_i простые, то $(n_i, d) = 1 \forall i$. Но тогда $(N, d) = (n_1 \ldots n_k, d) = 1$. Противоречие. $\Rightarrow$ такого s не существует. $\Rightarrow x \in L_{nef} \Leftrightarrow \exists s : V((N, M), s) = 1$. $\Rightarrow L_{nef} \in NP$.
Из пунктов 1) и 2) получаем, что $L_f \in Np \cap Co - NP$.

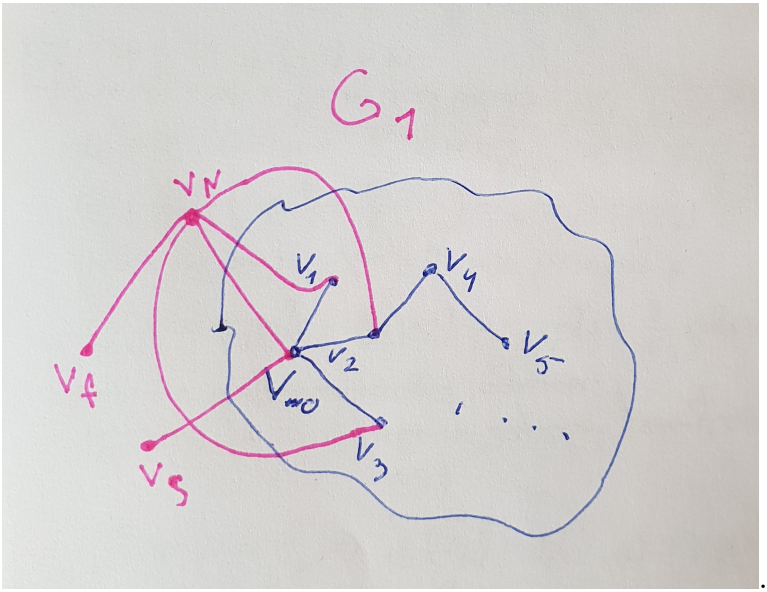
№8

Пусть n длина регулярного выражения, будем считать ее длину входа $n + |w|$. Из курса ТРЯП, мы знаем операции с автоматами, которые эквивалентны регулярным операциям(как построить автомат для сложения, звезды Клина, и переумножения языков автоматы которых уже построены). Т. е. можно построить автомат по регулярному выражению и это займет полиномиальное время. Так же можно рекурсивно показать, что количество вершин в построенном так автомате не более $2n$.
Тогда пусть алгоритм преобразует данное рег. выражение в автомат с $K = O(2n)$ вершинами. Теперь задача сводится к проверки, допускается ли автоматом данное слово w. Будем поочередно считывать символы слова, и записывать для каждого i-го символа: сначала в "состояния этого символа": $S(i)$, все возможные состояния, в которые можно перейти по i-му символу из состояний $S(i-1)$, а затем добавим еще состояния из e-замыкания уже добавленных вершин. При $i = 0$ $S(i)$ содержит состояния из e-замыкания множества начальных состояний. Сложность такой операции для каждого символа равна произведению числа состояний в $S(i-1)$ (их не больше K) и нахождением для фиксированного состояния, куда из него можно перейти

по данному символу(или e), это делается проверкой существования такого перехода (за $O(1)$) для каждого из оставшихся $K-1$ состояния. То есть для каждого символа слова сложность $O(n^2)$. Общая сложность: $O(n^2 \cdot |w|)$. $w \in L$, если в состояние $S(|w|)$ будет хотя бы одно финальное состояние, иначе w не лежит в L. Проверка этого займет $O(n)$.
Таким образом, данный алгоритм решает задачу за $O(n^2 \cdot |w|)$, где n длина регулярного выражения.

№7

1) Покажем сводимость ГП(GP) к ГЦ(GC). Пусть $f(x)$, принимающая на вход граф G, выдает на выходе граф G_1, полученный из графа G добавлением новой вершины V_N, соединенной со всеми остальными вершинами графа.
Если $G \in GP$, то очевидно, что $f(G) \in GC$: рассмотрим в графе G_1 путь, который является гамильтоновым в G, соединим начальную и конечную вершину пути с V_N. Получим, гамильтонов цикл.
Если $G_1 = f(G) \in GC$, то в ней есть вершина V_N, которая соединена ребрами со всеми остальными. Запустим гамильтонов цикл из этой вершин(если таких вершин несколько, то можно проверить для каждой), обозначим за V_1 вершину, в которую цикл приходит из V_s и за V_f, из которой он возвращается в V_N. Заметим, что часть цикла между посещением вершин V_s и V_f является гамильтоновым путем в графе G. $\Rightarrow G \in GPf(G) \in GC$. Т. е. ГП сводится к ГП. Функция полиномиальна, сложность равна сложности добавления в матрицу смежности новой строки.
2) Сводимость ГЦ к ГП. Пуст $f(G) = G_1$, где G_1 полученный из G следующим образом:
Добавим в G новую вершину V_N. Возьмем произвольную вершину G V_0 и соединим вершину V_N новыми ребрами со всеми вершинами, которые соединены ребром с V_0. Также соединим V_N и V_0 между собой ребром. Добавим вершины V_s и V_f и соединим их ребрами с V_0 и V_N соответственно.



Если $G \in GC$, то покажем, что $f(G) = G_1$ имеет гамильтонов путь: Из V_s в V_0, далее так как G_1 включает в себя все элементы G из V_0, а $G \in GC$, то можно обойти все вершины графа, кроме V_N, V_f, V_s, по одному разу и вернуться в V_0. Сделаем это. Далее из V_0 в V_N, а из V_N в V_f. Таким образом, гамильтонов путь явно найден. $\Rightarrow G \in GC \Rightarrow f(G) \in GP$.

Если $f(G) \in GP$, то покажем, что $G \in GC$:

Т. к. в графе $G_1 = f(G)$ вершины V_f и V_s имеют нечетную степень, то эти вершины должны быть концами гамильтонова пути. Для определенности V_s начало, V_f конец. Однозначно определяется первое и последние ребро этого пути V_fV_N и V_sV_0. Если отбросить эти ребра и вершины V_f, V_N, то в оставшемся графе будет существовать гамильтонов путь с началов в V_0 и конце V_f. Т. к. каждую(начальная и конечная уже не совпадают) вершину можно пройти только один раз, то ребра V_nV_0 в данном пути не будет. Уберем это ребро, рассматриваемый путь от этого не изменится. Т. к. V_N и V_0 абсолютно симметричны, то их можно совместить. Тогда рассматриваемый гамильтонов путь превратится в гамильтонов цикл. Заметим, что преобразованный граф это $f^{-1}(G_1) = G$, т.к. все приведенные преобразования были обратными преобразованиями функции f(Значит, $G \in GC$.

$\Rightarrow G \in GCf(G) \in GP$. Т. е. ГЦ сводится полиномиально(добавление трех новых вершин и соответствующих ребер имеет полиномиальную сложность: добавление трех строк в матрицу смежности) к ГП.