# Алгоритмы и анализ сложности, 3 семестр ПИ, Ответы на билеты

Собрано 22 декабря 2022 г. в 20:50

---

## Содержание

# Раздел #1: ? TODO: проблемы и доказательства

## 1.1. Машины Тьюринга и тезис Чёрча

**Определение 1** (Машина Тьюринга). Да что за «Машина Тьюринга»?

- Абстрактная вычислительная машина.
- Формализация понятия алгоритма.
- Расширение конечного автомата.
- Лента (бесконечная).
- Головка записи-чтения (управляющее устройство), способная находиться в одном из множества состояний, которое конечно и точно задано.
- Это управляющее может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита.
- Существует $\epsilon$-символ, который заполняет все пустые клетки ленты.
- Управляющее устройство работает согласно правилам перехода, которые представляют алгоритм, реализуемый данной МТ. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей ячейке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо (существует некий «синтаксический сахар» — остаться на месте). Некоторые состояния могут быть помечены как терминальные, и переход в любое из них означает конец работы, остановку алгоритма.

Это все очень интересно, но как насчет формализма?
Формально: $M = \{Q, G, \epsilon, \Sigma, \delta, q_0, F\}$ — запомните этот набор из семи элементов!

- $Q$ — конечное, не являющееся пустым, множество состояний.
- $G$ — конечный, не являющийся пустым, набор символов ленточного алфавита.
- $\epsilon$ — единственный пустой символ.
- $\Sigma = G \backslash \{\epsilon\}$ — набор входных символов.
- $\delta : (Q \backslash F) \times G \nrightarrow Q \times G \times \{L, R\}$ — частичная функция, называемая функцией перехода, где $L$ — сдвиг влево, а $R$ — сдвиг вправо. Если $\delta$ не определена для текущего состояния и символа ленты, то машина останавливается.
- $q_0$ — это начальное состояние.
- $F \subset Q$ — набор конечных состояний.

**Замечание.** А что если мы хотим такое состояние, которое будет являться term при одном символе на ленте и nonterm при другом символе?
Если немного подумать, то здесь все в порядке, поскольку мы можем сделать это состояние nonterm, перейти в другое term состояние, а при определенном символе сдвинуться, например, вправо, записав на ленту такой же символ, что был на ней.

**Замечание.** Хоть любой конечный алфавит и не ограничен одними цифрами 0 и 1, очевидно, что мы всегда можем его представить в виде двоичных чисел, введя на нем порядок.

**Определение 2** (Частичная функция). частичная функция $f$ из множества $X$ в множество Y — это функция из подмножества $S$ из $X$ (возможно, самого $X$) в $Y$ (обозначение: $\nrightarrow$).

**Определение 3** (Детерминированная и недетерминированная машины Тьюринга). Машина Тьюринга называется детерминированнй, если каждой паре состояни и ленточного символа соответствует не более одного правила. В ином случае, машина является недетерминированной.

**Определение 4** (Тезис Черча/Тьюринга/Черча-Тьюринга). Есть ли отличие?
На самом деле, все они говорят об одном, просто Черч в свое время придумал $\lambda$-исчисления, Тьюринг придумал Машину Тьюринга, а позже было показано, что эти формализмы эквивалентны.
Сам тезис сформулируем следующим образом: Класс алгоритмически вычислимых частичных функций совпадает с классом всех функций, вычислимых на машине Тьюринга.

**Замечание.** Стоит понимать, что это именно тезис, а не теорема, ведь понятие «алгоритмически вычислимая частичная функция» неформально.

**Определение 5** (Вычислимая функция). Вычислимые функции — это множество функций вида, $f\colon N \to N$, которые могут быть реализованы на машине Тьюринга.
В качестве множества $N$ обычно рассматривается множество $B^*$ — множество слов в двоичном алфавите $B = \{0, 1\}$, с оговоркой, что результатом вычисления может быть не только слово, но и специальное значение «неопределённость», соответствующее случаю, когда алгоритм «зависает». Таким образом, можно дать следующее определение $N$:
$N = B^* \cup \{\text{undef}\}$, где $B = \{0, 1\}$, а undef — специальный элемент, означающий неопределённость.
Роль множества $N$ может играть множество натуральных чисел, к которому добавлен элемент undef, и тогда вычислимые функции — это некоторое подмножество натуральнозначных функций натурального аргумента. Удобно считать, что в качестве $N$ могут выступать различные счётные множества — множество натуральных чисел, множество рациональных чисел, множество слов в каком-либо конечном алфавите и др.

## 1.2. Классы $RE$, $R$ и $co-RE$, доказательство $R = RE \cap co-RE$

**Определение 6** (Классы $RE$ и $co-RE$). Класс $RE$ (recursively enumerable) — класс decision problems (проблемы принятия решения), для которых ответ «да» может быть проверен машиной Тьюринга за конечное время.

- Если на проблему ответ «да», то существует некоторая процедура, которая требует конечного времени для определения этого.
- Ложь здесь отсутствует.
- Если на проблему ответ «нет», то машина Тьюринга может остановиться, а может и не остановиться.

Класс $co-RE$ является дополнением к классу $RE$: ответ «нет» можно получить за конечное время абсолютно всегда, получение противоположного ответа может занять вечность.

**Определение 7** (Формальный язык). Формальный язык (или просто язык) — это множество конечных слов над конечным алфавитом.

**Определение 8** (Класс $R$). $R$ — класс decision problems, решаемых на МТ (набор всех рекурсивных языков).

**Определение 9** (Рекурсивный язык). Формальный язык является рекурсивным, если существует полная машина Тьюринга (машина Тьюринга, которая останавливается для каждого заданного ввода), которая, когда на вход подается конечная последовательность символов, принимает ее, если она принадлежит языку, и отвергает ее в противном случае.

**Теорема 1.** $R = RE \cap co-RE$.

**Доказательство.** Обозначим за $X$ класс decision problems, содержащихся в классе $RE$, ответы «да» и «нет» на которые можно получить за конечное время. Очевидно, что $X \subset co-RE$ по определению, а также никакая другая задача, содержащаяся в $RE$ не содержится в $co-RE$. То есть, $X = RE \cap co-RE$. Однако $X = R$, так как это в точности класс decision problems, решаемых на машине Тьюринга, то есть: $R = X = RE \cap co-Re$. □

## 1.3. Проблемы Acceptence, Halting, Emptiness

**Определение 10** (Halting Problem). Проблема останова машины Тьюринга...

# Раздел #2: ? TODO: полностью

# Раздел #3: ?

## 3.1. Построение универсальной машины Тьюринга

**Определение 11** (Универсальная машиина Тьюринга). Универсальная машина Тьюринга — такая машина, которая может заменить собой любую машину Тьюринга. Получив на вход программу и входные данные, она вычисляет ответ, который вычислила бы по входным данным машина Тьюринга, чья программа была дана на вход.

**Определение 12** (Построение УМТ).

# Раздел #4: 4?

# Раздел #5: 5?

## 5.1. Определение классов сложности $P$, $NP$, $co-NP$

**Определение 13** (Класс сложности $P$). Классом $P$ называются все проблемы принятия решений, которые могут быть решены детерминированной машиной Тьюринга с использованием полиномиального количества времени вычислений или полиномиального времени.

**Пример** (Задача из класса сложности $P$). Пусть у нас есть массив натуральных чисел, состоящий из $n$ элементов. Вопрос: содержится ли число 5 в этом массиве?
Решение: Простой перебор элементов массива (если массив упорядочен, то можно воспользоваться бинарным поиском).

**Определение 14** (Класс сложности $NP$). Классом $NP$ называют множество задач принятия решения, решение с ответом «да» каждой из которых можно проверить на детерминированной машине Тьюринга за время, не превосходящее какой-либо полином.

**Пример** (Задача из класса сложности $NP$). Дано число $n$. Вопрос: раскладывается ли данное число на три простых?
Решение: заметим, что нельзя точно утверждать, содержится ли данная задача в классе $P$, поскольку единственное решение, которое пока что придумано — это простой перебор, занимающий более чем полиномиальное время работы. Однако если у нас на руках существует решение данной задачи с ответом «да», то мы легко можем проверить данное решение, перемножив три числа, содержащихся в решении. Если перемножение дает верный ответ, то решение верно, и наоборот. Доказательство того, что проверка занимает не более чем экспоненциальное время оставим в качестве упражнения читателям.

**Замечание.** На самом деле, решение прошлой задачи с ответом «нет» тоже можно проверить за экспоненциальное время. Само решение будет состоять в том, чтобы показать, что число раскладывается не на три простых, а на какое-либо другое количество. Проверка решения аналогична: перемножить и убедиться, либо же опровергнуть корректность решения. Это значит, что описанная нами задача также принадлежит и классу $co-NP$, о котором сказано ниже.

**Замечание.** Очевидно: $P \subset NP$ (если мы можем решить задачу за полиномиальное время, то мы можем проверить решение задачи, просто решив ее).

**Определение 15** (Класс сложности $co-NP$). Классом $co$-$NP$ называют множество задач принятия решения, дополнение к которому лежит в классе $NP$. Это означает, что каждая задача, решение которой с ответом «нет» можно проверить на детерминированной машине

Тьюринга за время, не превосходящее какой-либо полином, лежит в классе $co\text{-}NP$.

## 5.2. Определение полиномиальной сводимости и класса $NP$-полных языков

**Определение 16** (Полиномиальная сводимость). Любой язык $L_1$ называется сводимым по Карпу к языку $L_2$, если существует функция $F: \Sigma^* \mapsto \Sigma^*$, вычисляемая за полиномиальное время, где F(x) принадлежит $L_2$ в том случае, если x принадлежит $L_1$.

**Определение 17** (Класс сложности $NP\text{-}complete$). Класс $NP\text{-}complete$ — множество задач принятия решения из класса $NP$, к каждой из которых можно свести **любую** другую задачу из этого класса за полиномиальное время.

**Замечание.** Найдя алгоритм для решения любой задачи из класса $NP\text{-}complete$ за полиномиальное время, возможно решать каждую задачу $NP$ за полиномиальное время, а это решает проблему $P = NP$.

## 5.3. Взаимосвязи этих классов

**Замечание.** Взаимосвязи:
$P \subset NP$;
$P \subset co - NP$;
$NP \cap co - NP \neq \varnothing$.

## 5.4. Доказательство того, что если $NP$-полный язык лежит в $co - NP$, то $NP = co - NP$

**Теорема 2.** Если $NP$-полный язык лежит в $co\text{-}NP$, то $NP = co\text{-}NP$.

**Доказательство.** Пусть $L$ — это язык из класса $co - NP$. Заметим, что дополнение к этому языку лежит в классе $NP$. Сведем это дополнение к $NP$-полному языку (за полиномиальное время), который лежит в $co - NP$. Получается, что дополнение к языку $L$ лежит в классе $co - NP$. Данное рассуждение мы можем проделать для любого $co - NP$ языка. Получается, что дополнения к каждой задаче лежат в $co - NP$, но эти дополнения по определению лежат в $NP$, а значит $NP = co - NP$. □

# Раздел #6: x

# Раздел #7: x

## 7.1. Теорема Кука-Левина об NP-полноте задачи CNF-SAT

**Определение 18** (Булева формула). Булева формула — формула логики высказываний.

**Замечание.** Формула называется тождественно истинной (ложной), если она истинна (ложна) при любых значениях переменных. Две булевы формулы называются эквивалентными тогда и только тогда, когда они истинны на одном и том же подмножестве множества значений аргументов.

**Определение 19** (Задача SAT). Задача SAT — это задача выполнимости булевых формул. Экземпляром задачи является булева формула, состоящая только из имён переменных, скобок и операций $\wedge$ (И), $\vee$ (ИЛИ) и $\neg$ (НЕ). Задача заключается в следующем: можно ли назначить всем переменным, встречающимся в формуле, значения ложь и истина так, чтобы формула стала истинной.

**Определение 20** (Задача CNF-SAT). Определение почти аналогично, однако на булеву формулу накладывается ограничение: она должна быть записана в конъюнктивной нормальной форме (должна иметь вид конъюнкции дизъюнкций литералов).

**Замечание.** Любая булева формула может быть приведена к КНФ.

**Пример.** Формулы в КНФ:
$\neg A \wedge (B \vee C)$;
$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$;
$A \wedge B$.
Формулы не в КНФ:
$\neg(B \vee C)$;
$(A \wedge B) \vee C$;
$A \wedge (B \vee (D \wedge E))$.

**Определение 21** (Задача 3-SAT). Задача 3-SAT — частный случай задачи SAT, где булева формула записана в 3-конъюнктивной нормальной форме.

**Определение 22** (k-КНФ). k-конъюнктивной нормальной формой называют конъюнктивную нормальную форму, в которой каждая дизъюнкция содержит ровно k литералов.

**Пример.** Следующая формула записана в 2-КНФ: $(A \vee B) \wedge (\neg B \vee C) \wedge (B \vee \neg C)$.

**Определение 23** (Задача 1-in-3-SAT). Задача 1-in-3-SAT — частный случай задачи 3-SAT, где каждая дизъюнкция содержит три литерала, только один из которых может быть правдив (TRUE).

**Теорема 3** (Теорема Кука-Левина). Задача CNF-SAT выполнимости является NP-полной. То есть она находится в NP, и любая задача в NP может быть сведена за полиномиальное время детерминированной машиной Тьюринга к булевой задаче выполнимости.

**Доказательство.** TODO □

**Замечание.** Из того, что CNF-SAT является NP-complete, следует то, что SAT является NP-complete (TODO: why?).

## 7.2. co-NP-полнота задачи TAUTOLOGY

**Определение 24** (Тавтология). Тавтологией называется формула или утверждение, которое верно во всех возможных интерпретациях.

**Пример.** $x \neq y \vee x = y$.

**Определение 25** (co-NP-complete). Язык $L$ называется co-NP-complete, если любой co-NP язык можно свести к этому языку за время, не превосходящее какой-либо полином.

**Определение 26** (Задача TAUTOLOGY). Задача TAUTOLOGY — задача определения тавтологии в булевой формуле. Если ответ на экземпляр задачи «да», то необходимо показать, что при любых значениях литералов булева формула верна. Если ответ «нет», то достаточно продемонстрировать один контрпример.

**Теорема 4.** Задача TAUTOLOGY является co-NP задачей.

**Доказательство.** Это, на самом деле, довольно очевидно, поскольку решение с ответом «нет», вместе с которым представлен контрпример, можно легко проверить на корректность, просто подставив в булеву формулу значения литералов и убедившись в правдивости решения (в его ложности). □

**Теорема 5.** Задача TAUTOLOGY является co-NP-полной задачей.

**Доказательство.** Проблема определения того, существует ли какая-либо оценка, которая делает формулу истинной — проблема булевой выполнимости. Проблема проверки тавтологии эквивалентна этой проблеме, потому что проверка того, что предложение S является тавтологией, эквивалентна проверке того, что не существует оценки, удовлетворяющей $\neg S$. Известно, что проблема булевой выполнимости является NP-полной. Следовательно, тавтология co-NP-полна. □

# Раздел #8: x

## 8.1. Доказательство NP-полноты простых задач: 3-SAT, 1-in-3-SAT, SUBSET SUM, SYSTEM OF INCONGRUENCES

**Теорема 6.** 3-SAT является NP-полной задачей.

**Доказательство.** Мы можем преобразовать любую задачу L ∈ NP в задачу CNF-SAT за полиномиальное время (Теорема Кука-Левина). Значит, нам надо доказать, что задачу CNF-SAT можно свести до 3-SAT задачи за полиномиальное время.
Пусть представление CNF исходной задачи SAT равно:

$$\phi = C_1 \ \wedge C_2 \wedge ... \wedge C_n.$$

Представление $\phi$ выполнимо, если все $C_i$ выполнимы.
Без потери общности предположим, что предложение $C_r$ содержит больше, чем 3 литерала:

$$C_r = (a_1 \vee a_2 \vee ... \vee a_m), m > 3,$$

где каждый $a_i$ выбирается из набора литералов:

$$x_1, x_2, ..., x_m, \neg x_1, \neg x_2, ..., \neg x_m.$$

Пусть

$$A = a_3 \vee a_4 \vee ... \vee a_m;$$
$$C_r = a_1 \vee a_2 \vee A.$$

Определим $C'_r$ следующим образом:

$$C'_r = (a_1 \vee a_2 \vee y) \wedge (\neg y \vee A),$$

где $y$ — новая переменная, которую мы вводим, чтобы составить предложение с 3 литералами.
Эта же процедура может быть применена повторно ко второму предложению в $C'_r$, пока не останется предложений с более чем тремя оставшимися литералами.
Теперь нам нужно доказать, что $C'_r$ и $C_r$ равновероятны.
> Покажем, что если $C'_r$ выполнимо, то и $C_r$ выполнимо:

$$(a_1 \vee a_2 \vee y) = 1, \ (\neg y \vee A) = 1$$

Если $y = 0$:

$$C'_r = (a_1 \vee a_2 \vee 0) \wedge (1 \vee A)$$
$$= (a_1 \vee a_2) \wedge (1)$$

$$(a_1 \vee a_2) = 1$$

Очевидно, что, если $C_r$ выполнимо, то и $(a_1 \vee a_2) = 1$ выполнимо.
Если $y = 1$:

$$C'_r = (a_1 \vee a_2 \vee 1) \wedge (0 \vee A)$$

$$(A) = 1$$

Очевидно, что, если $C_r$ выполнимо, то и $(A) = 1$ выполнимо.
$>$ Покажем, что если $C_r$ выполнимо, то $C'_r$ выполнимо.
Если $a_1 = 1$ или $a_2 = 1$, то $C'_r = 1$, потому что:

$$C'_r = (a_1 \vee a_2 \vee y) \wedge (1 \vee A)$$

Если $a_1 = 0$, $a_2 = 0$, тогда $A = 1$. Назначая $y = 1$, мы можем сделать $C'_r$ выполнимым.
Используя описанную выше процедуру, $C_r \forall r$, где количество литералов больше 3 может быть преобразовано в предложения с не более 3 чем равновероятными переменными (если литералов меньше трех, можем добавить нули в предложения, дополнив количество до трех). Из приведенного выше доказательства мы видим, что для этого требуется полиномиальное время по количеству литералов в каждом предложении. □

**Теорема 7.** 1-in-3-SAT является NP-полной задачей.

**Доказательство.** Для доказательства достаточно свести 3-SAT задачу к задаче 1-in-3-SAT за полиномиальное время.
Пусть $(x \vee y \vee z)$ будет предложением в формуле 3-SAT. Добавим 4 новые булевые переменные $a_1, ..., a_4$, которые будут использоваться для моделирования этого предложения и никаких других. Тогда формула

$$R(\neg x, a_1, a_2) \wedge R(a_2, y, a_3) \wedge R(a_3, a_4, \neg z)$$

выполнима некоторой настройкой свежих переменных тогда и только тогда, когда по крайней мере одно из значений x, y или z является истинным (иными словами, существует такой набор из 4 новых булевым переменных, чтобы все три предложения в формуле выше содержали ровно(!) 1 истинное значение). Таким образом, любой экземпляр 3-SAT с m предложениями и n переменными может быть преобразован в равноудовлетворяемый экземпляр 1-in-3-SAT с $3m$ предложениями и $n + 4m$ переменными. □

**Определение 27** (Задача SSP). Задача о сумме подмножеств (SUBSET SUM PROBLEM) заключается в нахождении (хотя бы одного) непустого подмножества некоторого набора чисел, чтобы сумма чисел этого подмножества равнялась нулю.

**Пример.** Пусть задано множество $\{-1, -3, -2, 5, 8\}$, тогда подмножество $\{-3, -2, 5\}$ даёт в сумме ноль.

**Теорема 8.** Задача SSP является NP-полной.

**Доказательство.** TODO: https://neerc.ifmo.ru/wiki/index.php?title=NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D1%82%D0%B0_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8_%D0%BE_%D1%81%D1%83%D0%BC%D0%BC%D0%B5_%D0%BF%D0%BE%D0%B4%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B0 □

**Определение 28** (Задача SYSTEM OF INCONGRUENCES). TODO

# Раздел #9: ?

## 9.1. NP-полнота задачи о линейных диофантовых уравнениях в N, класс задач NP-hard

**Определение 29** (Класс NP-hard). Задача $X$ является NP-сложной, если существует NP-полная задача $Y$, такая, которая сводится к $X$ полиномиальному времени.

**Замечание.** Поскольку любая NP-полная задача может быть сведена к любой другой NP-полной задаче за полиномиальное время, NP $\cap$ NP-hard $=$ NP-complete.

**Определение 30** (Диофантовы уравнения). Диофантовыми уравнениями называются уравнения в целых числах.

**Теорема 9.** Задача о линейных диофантовых уравнениях в N является NP-полной.

**Доказательство.** TODO (доказываем, что задача лежит в NP + доказываем, что задача лежит в NP-hard). □

# Раздел #10: ?

## 10.1. Задачи VERTEX COVER, CLIQUE и INDEPENDENT SET

**Определение 31** (Вершинное покрытие). Вершинное покрытие для неориентированного графа $G = (V, E)$ — это множество его вершин $S$, такое, что, у каждого ребра графа хотя бы один из концов входит в вершину из $S$.

**Определение 32** (Задача VERTEX COVER). В общем случае, задача о вершинном покрытии состоит в поиске вершинного покрытия наименьшего размера для заданного графа (этот размер называется числом вершинного покрытия графа). Примером задачи вершинного покрытия является граф $G = (V, E)$ и натуральное число $k$, и задача состоит в том, чтобы проверить, существует ли вершинное покрытие размера не более $k$ в $G$.

**Теорема 10.** Задача VERTEX COVER является NP-полной задачей.

**Доказательство.** Докажем, что задача VERTEX COVER является NP задачей, а потом, что она является NP-трудной.
> Если какая-либо проблема находится в NP, то, учитывая «сертификат» (решение) проблемы и экземпляр проблемы (в данном случае граф $G$ и положительное целое число $k$), мы сможем проверить (проверить правильность данного решения) сертификат за полиномиальное время. Сертификатом задачи покрытия вершин является подмножество $V'$ множества $V$, которое содержит вершины покрытия вершин. Мы можем проверить, является ли множество $V'$ вершинным покрытием размера $k$, используя следующую стратегию (для графа $G(V, E)$):
* пусть *count* будет целым числом
* установить счетчик на 0
* для каждой вершины $v$ в $V'$
* удалить все ребра, смежные с $v$, из множества $E$
* увеличить счетчик на 1
* если *count* = $k$ и $E$ пусто, тогда данное решение верно
* иначе данное решение неверно.

Легко видеть, что это можно сделать за полиномиальное время. Таким образом, задача покрытия вершин относится к классу NP.
> Чтобы доказать, что вершинное покрытие является NP-трудным, мы возьмем некоторую задачу, которая уже доказана как NP-сложная, и покажем, что эту задачу можно свести к задаче о вершинном покрытии. Для этого мы рассмотрим задачу CLIQUE (о ней можно прочитать ниже), которая является NP-полной (и, следовательно, NP-трудной).
Примером проблемы клики является граф $G(V, E)$ и целое неотрицательное число $k$, и нам нужно проверить существование клики размера $k$ в графе $G$.
Теперь нам нужно показать, что любой экземпляр $(G, k)$ задачи клики можно свести к экземпляру задачи вершинного покрытия. Рассмотрим граф $G'$, который состоит из всех

вершин $G$,однако мы уберем все ребра из $G$, а каждую пару вершин, которая не была соединена в графе $G$ соединим в нашем новом графе. Назовем этот граф дополнением к $G$. Теперь задача о том, существует ли в графе $G$ клика размера $k$ — это то же самое, что и задача о том, существует ли вершинное покрытие размера $|V|-k$ в $G'$. Нам нужно показать, что это действительно так.

В одну сторону: предположим, что в $G$ есть клика размера $k$. Пусть множество вершин клики равно $V'$. Это означает $|V'| = k$. В дополнительном графе $G'$ выберем любое ребро $(u, v)$. Тогда хотя бы один из $u$ или $v$ должен принадлежать множеству $V - V'$. Это связано с тем, что если бы и $u$, и $v$ принадлежали множеству $V'$, то ребро $(u, v)$ принадлежало бы $V'$, что, в свою очередь, означало бы, что ребро $(u, v)$ принадлежит $G$. Это невозможно, так как $(u, v)$ не принадлежит $G$. Таким образом, все ребра в $G'$ покрываются вершинами множества $V - V'$.

В другую сторону: теперь предположим, что существует вершинное покрытие $V''$ размера $|V| - k$. Это означает, что все ребра в $G'$ соединены с некоторой вершиной в $V''$. В результате, если мы выберем любое ребро $(u, v)G'$, оба они не могут быть вне множества $V''$. Это означает, что все ребра $(u, v)$, такие что и $u$, и $v$ лежат вне множества $V''$, принадлежат $G$, т. е. эти ребра составляют клику размера $k$.

Таким образом, можно сказать, что клика размера $k$ в графе $G$ существует тогда и только тогда, когда существует вершинное покрытие размера $|V|^{\smile}k$ в $G'$, и, следовательно, любой пример задачи о кликах может быть сведен к примеру задачи о вершинном покрытии. Таким образом, вершинное покрытие является NP-сложной задачей. Поскольку вершинное покрытие относится к классам NP и NP-hard, оно является NP-complete задачей.

□

**Определение 33** (Задача CLIQUE). Кликой в неориентированном графе называется подмножество вершин, каждые две из которых соединены ребром графа. Иными словами, это полный подграф первоначального графа. Размер клики определяется как число вершин в ней. Задача о клике существует в двух вариантах: в задаче распознавания требуется определить, существует ли в заданном графе $G$ клика размера $k$, в то время как в вычислительном варианте требуется найти в заданном графе $G$ клику максимального размера (нас, по большей части, интересует первый вариант).

**Теорема 11.** Задача CLIQUE является NP-полной.

**Доказательство.** Докажем, что задача является NP задачей, а потом, что она является NP-трудной.

> Сертификат (решение) представляет собой подмножество вершин $V'$, состоящее из вершин, принадлежащих клике. Мы можем проверить это решение, проверив, что каждая пара вершин, принадлежащих решению, является смежной, просто проверив, что они имеют общее ребро друг с другом. Это можно сделать за полиномиальное время, то есть $O(V + E)$, используя следующую стратегию для графа $G(V, E)$:

* $flag = true$
* Для каждой пары $\{u, v\}$ в подмножестве $V'$:

- Проверить, что эти вершины $\{u, v\}$ имеют общее ребро.
- Если ребра нет, то $flag = false$, затем $break$.

* Если $flag = true$: Решение верно
* Иначе: Решение не является верным.
> Чтобы доказать, что задача о кликах NP-сложна, воспользуемся помощью задачи, которая уже является NP-трудной, и покажем, что эту задачу можно свести к задаче о кликах.
Для этого рассмотрим задачу о независимом множестве (INDEPENDENT SET, о которой можно прочитать ниже), которая является NP-полной (и, следовательно, NP-сложной). Каждый экземпляр задачи о независимом множестве, состоящий из графа $G(V, E)$ и целого числа $K$, может быть преобразован в требуемый граф $G'(V', E')$ и $K'$ задачи клики. Построим граф $G'$ следующими модификациями:
$V' = V$, то есть все вершины графа $G$ являются частью графа $G'$, $E' = \overline{E}$ (дополнение то есть ребра, отсутствующие в исходном графе G).
Граф $G'$ является дополнительным графом $G$. Время, необходимое для вычисления дополнительного графа $G'$, требует обхода всех вершин и ребер. Временная сложность: $O(V + E)$.
Теперь мы докажем, что проблема вычисления клики действительно сводится к вычислению независимого множества. Редукция может быть доказана следующими двумя утверждениями:

- Предположим, что граф $G$ содержит клику размера $K$. Наличие клики означает, что в $G$ имеется $K$ вершин, каждая из которых соединена ребром с остальными вершинами. Это также показывает, что, поскольку эти ребра содержатся в $G$, они не могут присутствовать в $G'$. В результате эти $K$ вершин не смежны друг с другом в $G'$ и, следовательно, образуют независимое множество размера $K$.
- Предположим, что дополнительный граф $G'$ имеет независимое множество вершин размера $K'$. Ни одна из этих вершин не имеет общего ребра с другими вершинами. Когда мы дополняем граф, чтобы получить $G$, эти $K$ вершин будут иметь общее ребро и, следовательно, станут смежными друг с другом. Следовательно, в графе $G$ будет клика размера $K$.

Таким образом, мы можем сказать, что в графе $G$ есть клика размера $K$, если в $G'$ существует независимое множество размера $K$ (дополнительный граф). Следовательно, любой случай проблемы клики может быть сведен к примеру проблемы независимого множества. Таким образом, проблема клики является NP-трудной. □

**Замечание.** Размышления в прошлом доказательстве выглядят довольно массивными, однако все это достаточно очевидно, если немного подумать.

**Замечание.** В доказательстве NP-полноты задачи INDEPENDENT SET мы используем задачу CLIQUE, поэтому предыдущее доказательство не является совсем корректным, по-

скольку доказательство циклично. Поэтому докажем, что задача о кликах NP-сложна, сведя булеву проблему выполнимости к задаче решения клики.
Пусть логическое выражение будет следующим: $S = (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3)$, где $x_1, x_2, x_3$ — переменные. Обозначим предложения за $C_1, C_2, C_3$. Рассмотрим вершины как

$$< x_1, 1 >; < x_2, 1 >; < \overline{x_1}, 2 >; < \overline{x_2}, 2 >; < x_1, 3 >; < x_3, 3 >,$$

где второй член в каждой вершине обозначает номер предложения, к которому они принадлежат. Соединим эти вершины так, чтобы:

- никакие две вершины, принадлежащие одному предложению, не были связаны;
- никакая переменная (первый элемент вершины) не связана со своим дополнением (одинаковые переменные должны быть связаны).

Таким образом, граф $G(V, E)$ построен так, что:

$$V = \{< a, i > | a \in C_i\},$$

$$E = \{(< a, i >, < b, j >) | i \neq j; b \neq \overline{a}\}.$$

Рассмотрим подграф $G$ с вершинами $< x_2, 1 >; < \overline{x_1}, 2 >; < x_3, 3 >$. Он образует клику размера 3 (рис. 1). Соответственно этому для присваивания $< x_1, x_2, x_3 >=< 0, 1, 1 >$ S принимает значение true. Следовательно, если у нас есть $k$ предложений в нашем выражении выполнимости, мы получаем максимальную клику размера $k$ и для соответствующего присвоения значений выражение выполнимости оценивается как истинное. Следовательно, для конкретного случая проблема выполнимости сводится к проблеме решения клики. Следовательно, проблема решения клики является NP-трудной. Стоит отметить, что мы рассмотрели частный случай, однако для любой булевой проблемы, сведенной к КНФ, это верно (это необходимо доказывать, но это напрямую вытекает из того, какие ограничения мы накладываем на ребра). Помимо этого, по-хорошему, необходимо доказать, что сведение происходит за полиномиальное время. Убедитесь в последнем самостоятельно.

**Определение 34** (Задача INDEPENDENT SET). Задача INDEPENDENT SET заключается в нахождении множеств вершин в графе размера $k$, никакие две вершины из которых не являются смежными. Decision problem вопрос можно поставить следующим образом: существует ли множество определенного размера.

**Теорема 12.** Задача INDEPENDENT SET является NP-полной.

**Доказательство.** Доказательство принадлежности к классу NP оставляем в качестве упражнения читателям. Доказательство принадлежности к классу NP-hard легко проделать, изучив доказательство теоремы выше. Обратите внимание на замечание после предыдущей теоремы, поскольку оно важно. □

## 10.2. Определение FPT-алгоритмов. Примеры FPT-алгоритмов для

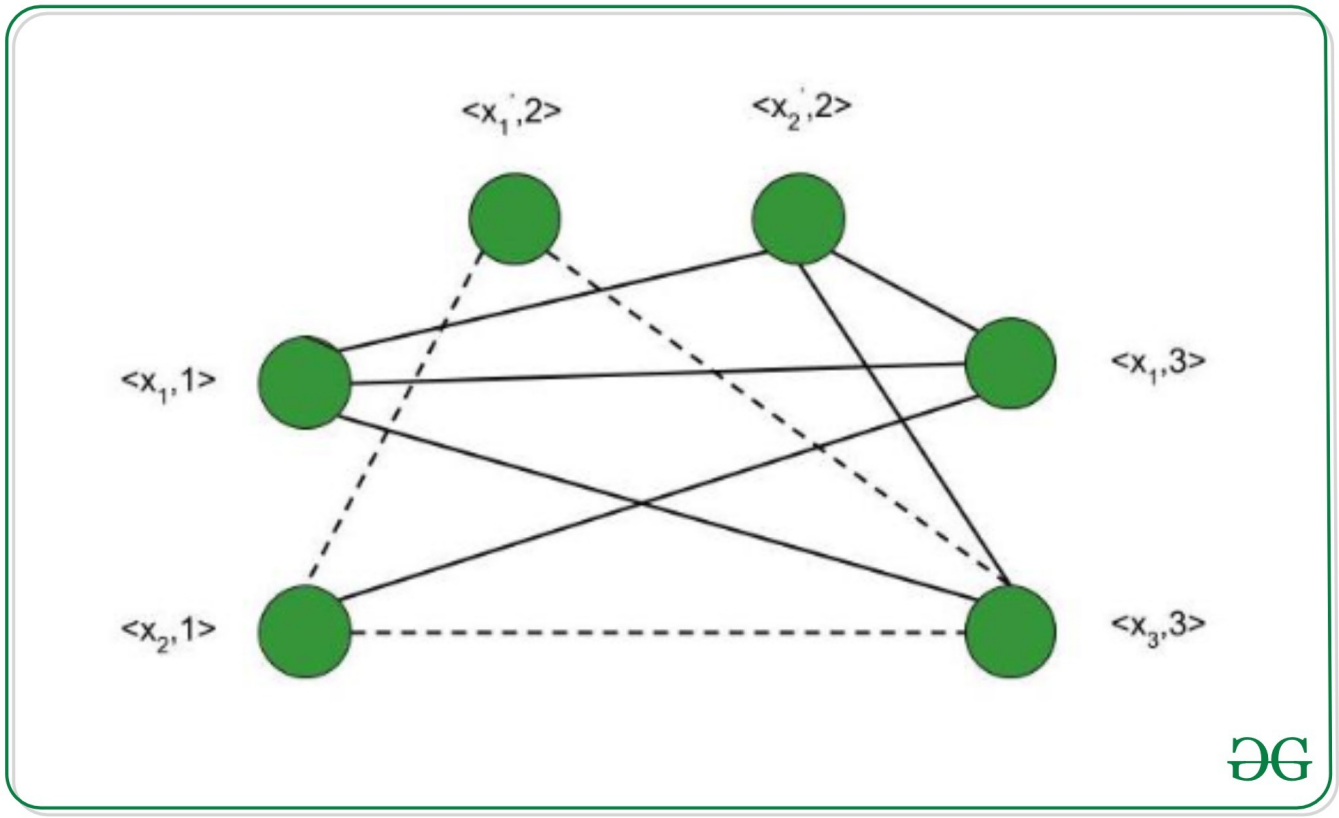


Рис. 1: Граф, построенный по указанным правилам

# VERTEX COVER и CLIQUE

**Определение 35** (FPT-алгоритм). Параметризированная задача P считается разрешимой с фиксированным параметром, или FPT-разрешимой (Fixed-Parameter Tractable), если она может быть решена некоторым параметризированным алгоритмом за время

$$t(n,k) = O(n^{O(1)} * f(k))$$

для функции $f$, зависящей только от параметра $k$.
Порядок роста функции $f(k)$ не ограничивается. Так, возможно $f(k) = 2^{o(k)}$ или $f(k) = 2^{O(k)}$. Важно, что исключаются функции вида $f(n,k)$, например $f(n,k) = n^k$.

**Замечание.** Класс всех разрешимых с фиксированным параметром задач обозначается FPT. Соответствующие параметризированные алгоритмы, решающие такие задачи, называются FPT-алгоритмами.

**Пример.** Примером FPT-разрешимой задачи может служить задача о вершинном покрытии графа (VERTEX COVER), когда параметром выступает размер покрытия. В самом деле, вершинное покрытие размера $k$ в графе $G$ с $n$ вершинами можно определить за время $O(n * 2^k)$. Если в данной задаче в качестве параметра взять древовидную ширину $tw(G)$ графа $G$, то метод динамического программирования способен отыскать наибольшее вершинное покрытие за время $O(n * 2^{tw(G)})$ (обход в ширину ??? TODO). Следовательно, параметризация данной задачи относительно $tw(G)$ приводит к FPT-разрешимости.

**Пример.** Пример с CLIQUE ??? TODO

# Раздел #11: ?

## 11.1. Определение классов функций FP и #P и класс языков PP

**Определение 36** (Функциональная задача). Функциональная задача — это вычислительная задача, в которой результат более сложный, чем у задачи принятия решения. Для функциональных проблем вывод не просто TRUE или FALSE. Разница между FP и P заключается в том, что задачи в P имеют одноразрядные ответы «да»/«нет», в то время как задачи в FP могут иметь любой результат, который может быть вычислен за полиномиальное время.

**Пример.** Сложение пяти чисел является проблемой FP, в то время как определение того, является ли их сумма нечетной, находится в P.