

FPT, Абстрактный Университет, 2018

Параметризованные алгоритмы

Собрано 27 декабря 2018 г. в 17:04

Содержание

1. Билеты по FPT части	1
1.1. Определение FPT алгоритма. Мотивация для построения FPT алгоритмов.	1
1.2. Определение ядра. Эквивалентность существования ядра и FPT алгоритма	1
1.3. Простейшее ядро $\mathcal{O}(k^2)$ и FPT алгоритм для Vertex Cover быстрее $2^k \cdot \text{poly}(n)$. . .	2
1.4. FPT алгоритм для Vertex Cover и Closest String	2
1.5. Ядро для Edge Clique Cover	2
1.6. Ядро для Feedback Arc Set In Tournaments	3
1.7. Ядро для Maximum Satisfiability	4
1.8. $2k$ kernel for Vertex Cover via Linear Programming	4
1.9. Sunflower lemma	5
1.10. Ядро для Hitting Set	6
1.11. Iterative Compression	6
1.12. Feedback Vertex Set in Tournaments by iterative compression	7
1.13. Feedback Vertex Set using randomization	8
1.14. Color Coding and k-path	9

Билет #1: Билеты по FPT части

26 декабря

1.1. Определение FPT алгоритма. Мотивация для построения FPT алгоритмов.

У любого человека бывает такое состояние, когда ему очень хочется решить **Vertex Cover** (далее **VC**). Пусть вершин $n = 1000$. 2^{1000} — многовато. Но вдруг мы откуда-то узнали, что $|VC| \leq 10 = k$. Будем искать любое решение (возможно, не минимальное), подходящее под это условие. $\binom{1000}{10}$ тоже много, но уже более-менее нормально.

Но давайте посмотрим на задачу по-новому. Рассмотрим вершины, из которых нет ребер. Они точно не входят в **VC**. Далее рассмотрим какую-то вершину, из которой исходит $> k$ рёбер. Если мы не возьмём её в ответ, то придётся взять всех её соседей и мы проиграем \Rightarrow берём её в ответ. Пока есть такие вершины, удаляем их и сводимся к той же задаче, но с меньшим k .

Теперь таких вершин нет \Rightarrow степень каждой вершины $\leq k$. То есть если у нас есть более k^2 рёбер, то мы обречены на поражение, так как какое-то ребро мы точно не сможем покрыть. Рёбер меньше $k^2 \Rightarrow$ вершин меньше $2k^2$ (все вершины нулевой степени повыкидывали). Переберём и получим $\binom{2k^2}{k}$. Это уже меньше, чем было, но всё равно много.

Далее посмотрим на вершины степени 1. Нам всегда выгодно брать их соседей. Отсюда количество вершин $\leq k^2$. $\binom{k^2}{k}$ при $k = 10$ примерно 10^{13} . Это мы сделали кернелизацию (как это правильно написать по-русски?) и получили ядро размера k^2 .

Зайдём с другой стороны. Каждое ребро должно быть покрыто \Rightarrow должен быть взят хотя бы один из его концов. Запустим следующий перебор на глубину k :

1. Если $k < 0$, то выходим с поражением
2. Если рёбер не осталось, то победа
3. Выбираем любое оставшееся ребро
4. Поочерёдно запускаемся рекурсивно от каждого из концов (уменьшили k на единицу, удалили вершину-конец)

Это работает за $\mathcal{O}(2^k(n + m))$. Уже умеем добиваться $m \leq \frac{kn}{2}$, то есть $\mathcal{O}(2^k kn)$. При наших k, n это будет 10^7 .

Def 1.1.1. Алгоритм, работающий за время $f(k)n^c$ (константа c не зависит ни от n , ни от k) называется *fixed-parameter algorithm* или **FPT алгоритмом**.

А ещё есть **XP** алгоритмы (*slice-wise polynomial*), которые работают за $f(k) \cdot n^{g(k)}$

Мораль: если каким-то образом получается так, что k достаточно маленькое, то мы умеем решать задачу почти за полином и детерминированно.

1.2. Определение ядра. Эквивалентность существования ядра и FPT алгоритма

Def 1.2.1. \mathcal{A} — алгоритм кернелизации (или просто ядро) для параметризованной задачи Q , если на паре из задачи и k (I, k) , он отработает за полиномиальное время и вернёт эквивалентную пару (I', k') такую, что $|I'| + k' \leq f(k)$.

То есть мы за полином сжимаем задачу к эквивалентной, размер которой меньше какой-то произвольной функции от k .

Теорема 1.2.2. Параметризованная задача Q кернализируема \Leftrightarrow она FPT.

Доказательство.

" \Rightarrow " За полином сделаем кернелизацию. Осталась задача размера, зависящего только от k . Ну её как-нибудь решим за функцию, зависящую только от k .

" \Leftarrow " $Q \in \text{FPT} \Rightarrow$ есть алгоритм \mathcal{A} , работающий за $f(k) \cdot |I|^c$. Построим алгоритм кернелизации следующим образом: запустим \mathcal{A} на $|I|^{c+1}$ шаг. Это полином. Если он успел завершиться, то мы смогли решить задачу. Иначе выполняется $f(k) \cdot |I|^c > |I|^{c+1} \Rightarrow I < f(k) \Rightarrow I + k < f(k) + k$. То есть размер задачи изначально меньше какой-то функции от k , то есть исходная задача уже подходит под определение ядра.

1.3. Простейшее ядро $\mathcal{O}(k^2)$ и FPT алгоритм для Vertex Cover быстрее $2^k \cdot \text{poly}(n)$.

Ядро нужного размера уже построили в пункте 1.

Научимся решать VC за время $T(k) = T(k-1) + T(k-2) + \text{poly}(n) = 1.62^k \cdot \text{poly}(k)$.

Уже знаем, что если есть вершина степени 1, то выгодно брать её соседа \Rightarrow можем считать, что степени всех вершин хотя бы 2. Возьмём любую вершину. Если мы берём её в ответ, то делаем рекурсивный вызов, который отработает за $T(k-1)$. Иначе мы должны взять всех её соседей (которых хотя бы 2), то есть сделать рекурсивный вызов за $T(k-2)$.

На практике научились за $T(k) = T(k-1) + T(k-3) = 1.46^k$, разобрав графы без вершин степени > 2 .

1.4. FPT алгоритм для Vertex Cover и Closest String

Для VC построили в прошлых пунктах.

Closest String: нам даны k строк над алфавитом Σ , все длины L и дано число d . Нужно узнать, существует ли такая строка y , что $d_H(y, x_i) \leq d \forall i$. $d_H(x, y)$ — расстояние Хэмминга (количество различающихся позиций).

Запишем строки в матрицу. x_{ij} — j -й символ строки x_i .

Назовём столбец матрицы *хорошим*, если все символы в нём совпадают.

Редукция 1. Удаляем все хорошие столбцы.

Заметим, что если ответ на задачу «Да», то останется не более kd колонок (так как для каждый символ в строке y не совпадает с соответствующим символом хотя бы в одной строке).

Редукция 2. Если столбцов больше kd , то ответ «Нет»

Решим задачу за $\mathcal{O}(kL + kd(d+1)^d)$.

Начнём со строки $y = x_1$. Если она подходит под ответ, то всё хорошо. Иначе существует строка x_j , такая, что $d_H(y, x_j) \geq d+1$. Возьмём любые $d+1$ отличия. Хотя бы одно из них должно быть устранено \Rightarrow делаем $d+1$ рекурсивный запуск от строки y , с соответствующим поменным символом. Глубину рекурсии ограничим числом d , так как мы не должны далеко уйти от стартовой строки x_1 . Итого $\mathcal{O}^*((d+1)^d)$.

1.5. Ядро для Edge Clique Cover

Edge Clique Cover: можно ли покрыть все рёбра графа k кликами?

Построим ядро размера 2^k .

Пусть есть две смежные вершины u и v с одинаковым набором соседей.

Если этот набор соседей пустой, то мы обязаны покрыть это ребро новой кликой размера 2 (то есть свелись к $k - 1$, $m - 1$).

Понятно, что если удалим v из графа, то оптимальное покрытие содержит не большее количество клик. Оно содержит не меньшее количество клик, так как к любой клике, в которой содержится u , можем добавить вершину v .

Таким образом, вершину v можем удалить из графа и результат не поменяется.

Теперь у всех смежных вершин разные наборы соседей. Тогда любые две смежные вершины лежат в разном наборе клик. Но тогда если вершин больше 2^k , то невозможно все рёбра покрыть k кликами \Rightarrow ответ «Нет».

1.6. Ядро для Feedback Arc Set In Tournaments

Feedback Arc Set In Tournaments: дан ориентированный турнир (ребро между любыми u, v ориентировано в одну сторону). Нужно найти наименьшее подмножество рёбер A , удалив которое, граф станет ациклическим.

Параметризованная версия: узнать, есть ли **Feedback Arc Set** размера $\leq k$.

Lm 1.6.1. Ориентированный граф G ациклический \Leftrightarrow возможно перенумеровать вершины таким образом, чтобы для каждого ориентированного ребра $u \rightarrow v$ выполнялось $u < v$

Доказательство.

" \Rightarrow " топсорт

" \Leftarrow " очевидно

Пусть $F \subseteq E(G)$. Обозначим за $G \odot F$ граф, в котором все рёбра из F перевернули.

Замечание 1.6.2. $G \odot F$ ациклический $\Rightarrow F$ — **Feedback Arc Set**.

Lm 1.6.3. F минимальный по включению **Feedback Arc Set** $\Leftrightarrow F$ — минимальный по включению набор рёбер такой, то $G \odot F$ ациклический.

Доказательство.

" \Rightarrow " Пусть $G \odot F$ не ациклический. Тогда в нём есть ориентированный цикл C . Тогда в C есть хотя бы одно ребро из F (иначе это не **Feedback Arc Set**). Обозначим рёбра из $C \cap rev(F)$ за f_1, \dots, f_l в порядке следования по циклу. Соответствующие (не развёрнутые) рёбра из G обозначим e_1, \dots, e_l .

Так как F минимальный по включению, для каждого e_i в G существует ориентированный цикл C_i в G такой, что $F \cap C_i = \{e_i\}$ (иначе могли бы удалить все остальные рёбра и остался бы **Feedback Arc Set**). Теперь пойдём по циклу C в графе G , но, когда наткнемся на ребро e_i , будем идти по циклу C_i (TODO: картинка). Получили цикл в графе G . Противоречие.

Минимальность следует из замечания выше.

" \Leftarrow " По замечанию выше F — **Feedback Arc Set**. Пусть есть $F' \subset F$ такой, что F' — **Feedback Arc Set**. Тогда по уже доказанной стрелке $G \odot F'$ ациклический. Противоречие с минимальностью по включению F .

Теперь построим ядро размера $k^2 + 2k$.

По лемме можем не удалять рёбра, а разворачивать. Тогда граф остаётся турниром.

Редукция 1. Если ребро e содержится в хотя бы $k + 1$ треугольнике, то его надо удалить (то есть развернуть и уменьшить k на 1).

Это так потому что треугольники, содержащие это ребро, пересекаются только по нему.

Редукция 2. Если вершина v не содержится ни в одном треугольнике, то можем удалить её из графа.

Тут надо нарисовать картинку графа, левая доля которого состоит из вершин, рёбра из которых входят в v , а правая из вершин, рёбра в которые выходят из v . Так как v не принадлежит ни одному треугольнику, то нет рёбер из правой доли в левую, то есть ни одно из связанных с v рёбер не лежит на цикле \Rightarrow не повлияет на **Feedback Arc Set**. Значит можем удалять вершину v , и это не повлияет на решение.

Теперь покажем, что если ответ «Да», то после редукций размер графа не больше $k(k+2)$. Пусть у нас есть граф T , над которым произвели все редукции, и в нём есть **Feedback Arc Set** A размера $\leq k$. Для каждого ребра $e_i \in A$ есть максимум k вершин, которые лежат в треугольнике, содержащем e_i . Так как A покрывает все треугольники, и каждая вершина содержится в каком-то треугольнике, получаем, что всего вершин не более $2k + k^2$ (не более $2k$ вершин — концы рёбер из A , по k вершин на каждое ребро из A).

1.7. Ядро для Maximum Satisfiability

В **Maximum Satisfiability** мы хотим удовлетворить хотя бы k клозов у выражения, данного в КНФ форме.

Построим ядро с k переменными и $2k$ клозами.

Пусть у нас есть m клозов. Тогда можем удовлетворить $\frac{m}{2}$ из них (поставить все переменные в **true** или все в **false**). Поэтому ищем решения только для случая $m < 2k$. Таким образом, задача свелась просто к нахождению ядра с $n < k$ переменными.

Если переменных и так меньше k , то мы уже победили.

Построим двудольный граф. Левая доля — переменные, правая доля — клозы. Есть ребро между переменной и клозом, если клоз содержит эту переменную или её отрицание. Найдём максимальное паросочетание. Если размер паросочетания больше, чем k , то можем обозначить переменные так, чтобы выполнилось больше, чем k клозов (смотрим, в какой клоз ведёт ребро паросочетания из переменной. Если там переменная с отрицанием, то присваиваем переменной **false**, иначе **true**).

Остался случай, когда размер паросочетания меньше k . Все изолированные вершины выкинем. Паросочетание не совершенное (так как переменных больше k), значит по лемме Холла можем найти минимальное множество C переменных так, что $|N(C)| < |C|$. Выкинем из C одну вершину x . Тогда (по минимальности C) у нас будет совершенное паросочетание на $C \setminus x$, $N(C)$. То есть переменные из $C \setminus x$ могут выполнить все $N(C)$ клозов, при этом не повлияв на другие клозы (потому что они с ними не связаны) \Rightarrow выгодно выполнить эти $N(C)$ клозов. Таким образом уменьшили k и n на $N(C)$. Повторив несколько раз, получим ядро нужного размера.

1.8. $2k$ kernel for Vertex Cover via Linear Programming

Применим ILP к VC.

Получится задача вида:

$$\begin{aligned} \sum_{v \in V(G)} x_v &\rightarrow \min \\ x_u + x_v &\geq 1 \quad \forall uv \in E(G) \quad (\text{все рёбра покрыты}) \\ 0 \leq x_v &\leq 1, \quad x_v \in \mathbb{Z} \quad \forall v \in V(G) \end{aligned}$$

Очевидна равносильность этой задачи и VC.

Посмотрим на **Linear Programming** версию этой задачи:

$$\sum_{v \in V(G)} x_v \rightarrow \min$$

$$x_u + x_v \geq 1 \quad \forall uv \in E(G)$$

$$0 \leq x_v \leq 1, \quad \forall v \in V(G)$$

Отличие в том, что теперь x_v могут быть нецелыми. Равносильность **VC** потерялась. Но зато **LP** умеют решать за полином.

Посмотрим на оптимальное решение **LP**.

$$V_0 = \{v \in V(G) : x_v < \frac{1}{2}\},$$

$$V_{\frac{1}{2}} = \{v \in V(G) : x_v = \frac{1}{2}\}$$

$$V_1 = \{v \in V(G) : x_v > \frac{1}{2}\}$$

Теорема 1.8.1. Существует минимальное вершинное покрытие S графа G такое, что

$$V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$$

Доказательство.

Рассмотрим S^* какое-то вершинное покрытие. Рассмотрим $S = (S^* \setminus V_0) \cup V_1$. Убедимся, что это вершинное покрытие. Если какое-то ребро перестало покрываться, то оба его конца лежали в V_0 . Такого быть не может.

Поймём, что $|S| \leq |S^*|$. Пусть не так. Тогда $|S^* \cap V_0| < |V_1 \setminus S^*|$ (выкинули вершин меньше, чем добавили).

Обозначим $A = S^* \cap V_0$, $B = V_1 \setminus S^*$

$\varepsilon := \min\{|x_v - \frac{1}{2}| : v \in V_0 \cup V_1\}$ (заметим, что $\varepsilon > 0$ из-за неравенства на A и B)

Пусть x_i — решение **LP**. Прибавим ε ко всем $x_i : i \in A$, а из всех $x_i : i \in B$ вычтем ε .

Покажем, что все неравенства из **LP** сохраняются.

$e = uv \Rightarrow x_u + x_v \geq 1$. Проблема могла произойти только с рёбрами, которые шли из B в $V_0 \setminus A$ (сумма у них уменьшилась на ε).

Но заметим, что таких рёбер не существует, так как S^* не покрывает их.

Получили решение линейного программирования меньше, чем оптимальное \Rightarrow противоречие.

Таким образом получили редукцию:

Редукция. Оставляем $V_{\frac{1}{2}}$, уменьшаем k на $|V_1|$

Построим ядро размера $2k$.

Замечание 1.8.2. Ответ к **LP** не больше, чем ответ к **ILP**.

На каждой итерации будем решать **LP** и проводить редукцию (если решение **LP** не больше k).

Надо убедиться, что после проведения редукций граф остался маленьким.

То есть если $V(G') = V_{\frac{1}{2}}$, то $|V(G')| \leq 2k$

$$|V(G')| = |V_{\frac{1}{2}}| = \sum_{v \in V_{\frac{1}{2}}} 2x_v \leq 2 \sum_{v \in V(G)} x_v \leq 2k. \text{ Что и требовалось.}$$

1.9. Sunflower lemma

Def 1.9.1. Подсолнухом с k лепестками и ядром Y называется набор множеств S_1, \dots, S_k такой, что $S_i \cap S_j = Y \quad \forall i \neq j$. Лепестками называются множества $S_i \setminus Y$. Требуется, чтобы они были непустыми. Ядро может быть пустым.

Теорема 1.9.2. Пусть \mathcal{A} — семейство множеств (без повторяющихся) над объемлющим множеством U . Размер каждого множества из \mathcal{A} ровно d . Тогда если $|\mathcal{A}| > d!(k-1)^d$, тогда в \mathcal{A} существует подсолнух с k лепестками и его можно найти за полином от $|\mathcal{A}|$, $|U|$ и k .

Доказательство.

Делаем индукцию по d . Для $d = 1$ очевидно.

Возьмём максимальное по включению семейство непересекающихся множеств из \mathcal{A} . Его можем найти жадно за полином. $G = \{S_1, \dots, S_l\}$. Если $l \geq k$, то задача решена.

Пусть $S = \bigcup S_i$. $|S| \leq d(k-1)$. Так как G максимально по включению, каждое множество A из \mathcal{A} пересекается с S . Значит (по принципу Дирихле) существует элемент $u \in U$, который содержится в хотя бы

$$\frac{|\mathcal{A}|}{|S|} > \frac{d!(k-1)^d}{d(k-1)} = (d-1)!(k-1)^{d-1}$$

множествах из \mathcal{A} .

Возьмём эти множества, удалим из них элемент u , по индукции построим подсолнух, добавим к каждому множеству u .

1.10. Ядро для Hitting Set

d-Hitting Set: даны множества A_1, \dots, A_n размера $\leq d$, есть ли множество H размера $\leq k$, пересекающееся со всеми A_i .

Дополним все множества A_i уникальными фиктивными элементами так, чтобы их размер стал ровно d . Ответ от этого не изменился.

Если есть подсолнух с $k+1$ лепестком (а он есть, если множеств хотя бы $d!k^d$), то **Hitting Set** должен пересекаться с ядром. Удаляем все множества из подсолнуха, вместо них добавляем ядро (оно размера $\leq d$). Свелись к меньшей задаче с $n' = n - (k+1) + 1 = n - k$.

Таким образом, получили ядро размера $d!k^d$

1.11. Iterative Compression

Снова решим **Vertex Cover**.

Найдём 2-приближение. Это можем сделать, выбирая очередное непокрытое ребро, и беря в вершинное покрытие оба его конца. Получилось покрытие C размера T . Если $T > 2k$, то ответ сразу «Нет». Иначе применим технику, называемую **Iterative Compression** (итеративное сжатие).

Посмотрим на наш граф. Он разбился на две части: C и $B = G \setminus C$. При чём B — независимое множество.

Посмотрим на оптимальное решение. Оно содержит какие-то (возможно, никакие) вершины из B и какое-то (возможно, пустое) множество W вершин из C . Переберём это W ($\leq 2^{2k}$ вариантов).

Рассматриваем конкретное W . Если есть ребро в $C \setminus W$, то такое W точно не подходит (так как мы точно не берём в вершинное покрытие никакие вершины из C , кроме W).

Мы обязаны взять в новое вершинное покрытие $N(S \setminus W)$ (по тем же соображениям).

Заметим, что таким образом мы построили вершинное покрытие (так как B независимое множество, а все остальные рёбра мы покрыли).

Более того, так мы нашли минимальное вершинное покрытие. За время $\mathcal{O}^*(2^{2k})$

Рассмотрим эту задачу с другой стороны. Посмотрим на граф, образованный вершинами v_1, \dots, v_k . В нём, очевидно, есть вершинное покрытие размера k (все вершины). Добавим новую вершину (и в вершинное покрытие тоже). Применив прошлый алгоритм, сможем найти вершинное покрытие размера $\leq k$ (или сказать, что его не существует) за время $\mathcal{O}^*(2^k)$. По то-

му же принципу будем добавлять остальные вершины, сохраняя инвариант «знаем ввершинное покрытие размера $\leq k$ на первых i вершинах». Тогда всю задачу сумели решить за $\mathcal{O} * (2^k)$.
Общий алгоритм для большинства случаев:

- Упорядочим вершины
- Как-то легко найдём решение на первых k вершинах
- Переходим от графа G_n (в котором есть решение размера $\leq k$) к графу G_{n+1} , в котором по решению в графе G_n можем легко построить решение размера $\leq k + 1$
- По задаче $(G, k + 1)$ либо получаем задачу (G, k) , либо заключаем, что решения нет.

Требуем, чтобы ответ для H был не больше ответа для G , если $H \subseteq G$. Такое требование, например, не выполняется для задачи нахождения доминирующего множества.

1.12. Feedback Vertex Set in Tournaments by iterative compression

Дисклеймер: этот билет написан максимально плохо. Его гораздо проще понять, чем формализовать.

Feedback Vertex Set in Tournaments: есть ориентированный турнир. Нужно удалить не более k вершин, чтобы граф стал ациклическим.

В графе v_1, \dots, v_k удалим все вершины и циклов не будет. Если есть граф, для которого знаем решение размера k , то добавив к нему одну вершину, можем получить ответ $\leq k + 1$ (к предыдущему ответу нужно прибавить эту вершину).

Теперь хотим сделать переход от $(G, k + 1)$ к (G, k) .

$G = A \sqcup B$, где A — решение размера $k + 1$. Тогда в B нет циклов. Снова будем перебирать $W \subset A$ такое, что W входит в ответ. Надо решить disjoint версию задачи. Если в $A \setminus W$ есть цикл, то такое W сразу не подходит.

Считаем, что $C = A \setminus W$ ациклический. Нужно удалить в B $k' = k - |W|$ вершин так, чтобы $\sqcup B$ стал ациклическим.

В порядке топсорта упорядочим вершины из C и из B . (топсорт на ациклическом турнире единственный)

В C получили последовательность $\{c_i\} c_1, c_2, \dots, c_l$. Аналогично для B построим $\{b_i\}$

Имеем свойство: если есть ребро $c_i c_j$, то $i < j$.

Посмотрим на вершину b_i из B . Если она образует цикл с вершинами из C , то её точно нужно удалять. Иначе, посмотрим на её позицию в топсорте графа C , если бы она была в нём. Это место определено однозначно, так как граф — турнир:

$$c_1, \dots, c_{j_i}, b_i, c_{j_i+1}, \dots, c_l$$

Для каждого b_i запомнили соответствующую позицию j_i (если b_i стоит перед c_1 , то $j_i = 0$).

Отсортируем $\{b_i\}$ по j_i , а при равенстве по i . Получим последовательность $\{d_i\}$.

Рассмотрим любой ациклический граф, в котором есть все вершины $\{c_i\}$ и какие-то вершины $\{b_j\}$. Для него выполняется:

- $\{c_i\}$ стоят в порядке топологической сортировки C
- $\{b_j\}$ стоят в порядке топологической сортировки B (то есть является подпоследовательностью $\{b_i\}$)

- Ближайшее c_k слева к любому b_i — это c_{i_j} (то есть $\{b_j\}$ являются подпоследовательностью $\{d_i\}$)

Обратно, нетрудно убедиться, что любая последовательность $\{c_i\}$ и $\{b_j\}$, подходящая под эти правила образует корректный топсорт \Rightarrow ациклический граф.

Таким образом, чтобы найти ациклический граф наибольшего размера нужно найти наибольшую общую подпоследовательность $\{d_i\}$ и $\{b_i\}$.

1.13. Feedback Vertex Set using randomization

Feedback Vertex Set удобно решать на мультиграфах.

Обычные редукции:

Редукция 1. Если есть петля, то удаляем (т.е. добавляем в ответ) соответствующую вершину и уменьшаем k на 1.

Редукция 2. Если есть ребро кратности больше 2, то сделаем его кратности ровно 2.

Редукция 3. Если есть вершина степени ≤ 1 , то удалим её.

Редукция 4. Если есть вершина степени 2, то удали её, а соседей соединим ребром.

Таким образом, мы научились избавляться от вершин степени ≤ 2 . Докажем следующую лемму:

Lm 1.13.1. G — граф на n вершинах с минимальной степенью ≥ 3 . Тогда для любого Feedback Vertex Set X более чем половина рёбер G имеют хотя бы один конец в X .

Доказательство.

Пусть $H = G - X$. Каждое ребро из $E(G) \setminus E(H)$ инцидентно хотя бы одной вершине из X , значит утверждение леммы эквивалентно $|E(G) \setminus E(H)| > |E(H)|$. Но H — лес, значит $|V(H)| > |E(H)|$ и достаточно показать $|E(G) \setminus E(H)| > |V(H)|$.

За J обозначим рёбра, у которых один конец в X , а другой в $V(H)$ (все они входят в $E(G) \setminus E(H)$, отсюда $|E(G) \setminus E(H)| \geq |J|$). За $V_{\leq 1}$, V_2 и $V_{\geq 3}$ обозначим множества вершин, у которых степень в H не более 1, ровно 2 и хотя бы 3 соответственно.

В G степень ≥ 3 , значит у каждой вершины из $V_{\leq 1}$ есть хотя два ребра из J . Аналогично у вершины из V_2 есть хотя бы одно ребро в J . Так как H лес, то $|V_{\geq 3}| < |V_{\leq 1}|$ (так как каждая вершина степени 3 «порождает» два новых листа).

Получаем:

$$|E(G) \setminus E(H)| \geq |J| \geq 2|V_{\leq 1}| + |V_2| > |V_{\leq 1}| + |V_2| + |V_{\geq 3}| = |V(H)|$$

Это и хотели.

Предъявим полиномиальный рандомизированный алгоритм, который при существовании решения найдёт его с вероятностью 4^{-k} :

1. Применим все редукции
2. Выберем случайное ребро, а у него один из концов (v). С вероятностью $\frac{1}{4}$ это вершина из ответа
3. Удалим вершину v из графа

Всего делаем k итераций.

Повторив этот алгоритм 4^k раз, получим алгоритм, выдающий ответ с константной вероятностью. Он будет работать за $\mathcal{O}^*(4^k)$

1.14. Color Coding and k-path

k-path: найти в графе несамопересекающийся путь длины ровно k .

Покажем вершины графа случайно в k цветов. Если у нас был путь длины k , то посмотрим с какой вероятностью он будет содержать по одной вершине каждого цвета: