МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

НАУМОВ
Роман Дмитриевич

СЛОЖНОСТЬ РАСПОЗНАВАНИЯ ЖЁСТКОСТИ ГРАФА

Дипломная работа

Научный руководитель:
кандидат
физико-математических
наук, доцент
В. И. Бенедиктович

Допущена к защите

«19» мая 2025 г.

Заведующий кафедрой дискретной математики и алгоритмики
доктор физико-математических наук, профессор В.М. Котов

Минск, 2025

Оглавление

РЕФЕРАТ

Дипломная работа, 39 страниц, 8 иллюстраций, 1 таблица, 1 листинг, 9 источников.

Ключевые слова: ЖЁСТКОСТЬ, ДРЕВЕСНАЯ ДЕКОМПОЗИЦИЯ, ДРЕВЕСНАЯ ШИРИНА, ГАМИЛЬТОНОВОСТЬ, ХОРДАЛЬНЫЙ ГРАФ, РЕГУЛЯРНЫЙ ГРАФ, ПЛАНАРНЫЙ ГРАФ, ФАКТОР.

Объект исследования: задача определения сложности распознавания жёсткости для различных классов графов.

Цель работы: обзор текущего состояния исследований по теме жёсткости графов. Реализация алгоритма для распознавания жёсткости графа.

Методы исследования: системный подход, изучение соответствующей литературы и электронных источников, постановка задачи и её решение.

Результаты: изучена литература, связанная с жёсткостью и другими свойствами графов. Рассмотрены классы графов, для которых есть полиномиальный алгоритм нахождения жёсткости, а также открытые проблемы в этой области. Собрана информация о связи жёсткости с гамильтоновостью для разных классов графов. Изучены особенности распознавания жёсткости в регулярных графах и приведены некоторые примеры доказательств NP-трудности распознавания жёсткости. Разобран и реализован алгоритм поиска жёсткости в графах с ограниченной древесной шириной на языке C++, а также разобран алгоритм поиска древесной ширины и древесной декомпозиции с минимальной шириной.

Область применения: исследования применимых на практике свойств графов.

РЭФЕРАТ

Дыпломная праца, 39 старонак, 8 малюнкаў, 1 табліца, 1 лістынг, 9 крыніц.
Ключавыя словы: КАЛЯНАСЦЬ, ДРАЎНЯНАЯ ДЭКАМПАЗІЦЫЯ, ДРАЎНЯНАЯ ШЫРЫНА, ГАМІЛЬТАНАВАСЦЬ, ХАРДАЛЬНЫ ГРАФ, РЕГУЛЯРНЫ ГРАФ, ПЛАНАРНЫ ГРАФ, ФАКТАР.
Аб'ект даследавання: задача вызначэння складанасці распазнання каляснасці для розных класаў графаў.
Мэта працы: агляд сучаснага стану даследаванняў па тэме калянасці графаў. Рэалізацыя алгарытму для распазнання калянасці графа.
Метады даследавання: сістэмны падыход, вывучэнне адпаведнай літаратуры і электронных крыніц, пастаноўка задачы і яе вырашэнне.
Вынік: вывучана літаратура, звязаная з калянасцю і іншымі свойствамі графаў. Разгледжаны класы графаў, для якіх ёсць паліномны алгарытм знаходжання калянасці, а таксама адкрытыя праблемы ў гэтай галіне. Сабрана інфармацыя аб сувязі калянасці з гамільтанавасцю для розных класаў графаў. Вывучаны асаблівасці распазнання калянасці ў рэгулярных графах і прыведзены некаторыя прыклады доказаў NP-цяжкасці распазнання калянасці. Разабраны і рэалізаваны алгарытм апісання калянасці ў графах з абмежаванай драўнянай шырынёй на мове C++, а таксама разабраны алгарытм пошуку драўнянай шырыні і драўнянай дэкампазіцыі з мінімальнай шырынёй.
Вобласць прымянення: даследванні ўласцівасцей графаў, якія магчыма прымяніць на практыцы.

ABSTRACT

Diploma thesis, 39 pages, 8 figures, 1 table, 1 listing, 9 sources.
Keywords: TOUGHNESS, TREE DECOMPOSITION, TREEWIDTH, HAMILTONIANITY, CHORDAL GRAPH, REGULAR GRAPH, PLANAR GRAPH, FACTOR.
Object of reseach: the problem of determining the complexity of recognizing toughness for different classes of graphs.
Objective: an overview of the current state of research on graph toughness. Implementation of an algorithm for recognizing graph toughness.
Research methods: a systematic approach, studying relevant literature and electronic sources, setting the problem and solving it.
The result: The literature related to toughness and other properties of graphs was studied. The classes of graphs for which there is a polynomial algorithm for finding toughness are considered, as well as open problems in this area. The features in regular graphs are studied and some examples of proofs of NP-hardness of recognition of toughness are given.Information on the relationship between toughness and hamiltonianity for different classes of graphs is collected. The algorithm for detecting toughness in graphs with limited treewidth in C++ is analyzed and implemented.
The scope: research into practical properties of graphs.

ВВЕДЕНИЕ

Графы являются одним из ключевых объектов изучения в теории алгоритмов и дискретной математике. Они находят широкое применение в различных областях науки и техники: компьютерные сети, биоинформатика, логистика, обработка изображений и многие другие. Одной из интересных практических задач графовой теории является задача распознавания жёсткости графа.

Жёсткость графа — это важное свойство, которое описывает, насколько тесно между собой соединены вершины. В практическом смысле, жёсткость графа имеет значение при проектировании устойчивых инженерных конструкций, моделировании механических систем, а также в задачах робототехники и компьютерной графики. Жёсткость сильно связана с гамильтоновостью и многими другими свойствами графа.

Задача распознавания жёсткости графов является не только теоретически интересной, но и вычислительно сложной. Определение жёсткости графа требует анализа его структуры, что включает в себя проверку условий, связанных с подграфами, рёбрами и вершинами. Данная задача является coNP-полной, что делает её решение неэффективным для больших графов при использовании наивных подходов.

Актуальность исследования данной темы заключается в необходимости разработки эффективных методов распознавания жёсткости графов, которые могут быть применены в реальных задачах, а также в поиске классов графов, для которых жёсткость распознаётся за полиномиальное время. Оптимизация алгоритмов и поиск новых подходов к решению этой задачи имеют большое значение для многих прикладных областей.

Целью данной дипломной работы является изучение задачи распознавания жёсткости графов, анализ её вычислительной сложности для разных классов графов. В ходе работы будут рассмотрены теоретические аспекты задачи и трудности, связанные с её решением, а также её связь с другими свойствами графов. Также будет реализован алгоритм, который по графу и древесной декомпозиции определяет жёсткость.

ГЛАВА 1
ПОСТАНОВКА ЗАДАЧИ РАСПОЗНАВАНИЯ ЖЁСТКОСТИ ГРАФА И ЕЁ ПРИМЕНЕНИЕ

1.1 Постановка задачи распознавания жёсткости графа

Обозначим $C(G)$ множество компонент связности графа G. $c(G) = |C(G)|$.

Определение. Граф называется t-жёстким, если $|S| \geq t * c(G - S)$ для любого $S \subseteq V_G$ такого, что $c(G - S) > 1$. Жёсткостью графа G является максимальное t такое, что G является t-жёстким. Жёсткость графа G будем обозначать $\tau(G)$. $\tau(K_n) = \infty$, для любого $n \geq 1$.

Задача: по заданному графу G определить $\tau(G)$.

1.2 Применение

Жёсткость графа имеет широкое применение:

1. проектирование инженерных конструкций,
2. проектирование компьютерных сетей,
3. биоинформатика,
4. логистика,
5. комбинаторика,
6. криптография,
7. в целом, с любая сфера, где применяется теория графов.

Отдельно стоит отметить важность жёсткости в теории графов. Жёсткость является показателем, за счёт которого доказывается масса других свойств с обширным практическим применением.

Следует отметить, что для многих задач размерность графа небольшая, что позволяет использовать экспоненциальные алгоритмы. Поэтому важной задачей также является усовершенствование экспоненциальных алгоритмов.

1.3 Общие сведения о сложности распознавания жёсткости графа

Рассмотрим некоторые свойства жёсткости графа, а также её связь с другими параметрами графа.

Утверждение. [5] Жёсткость дерева T равна $\frac{1}{\Delta(T)}$, где $\Delta(T)$ является максимальной степенью вершины дерева T.

Доказательство. Удаление вершины со степенью $k = \Delta(T)$ приведёт к появлению леса из k деревьев. Удаление любой другой вершины даст не более $k-1$ новых компонент связности. Поэтому можно утверждать, что нет такого множества вершин, которое при удалении из дерева образует больше, чем kS компонент связности. □

Определение. [5] Жёстким подмножеством графа G является подмножество $S \subset G$ такое, что $\tau(G) = \frac{|S|}{c(G-S)}$.

Можно показать, что для неполных графов $\tau(G) \leq \frac{k(G)}{2}$, где $k(G)$ обозначает связность графа. Действительно, есть такое $S \subset V(G)$, что $|S| = k(G)$ и $c(G-S) \geq 2$.

Теорема. [5] $\tau(G) = \frac{k(G)}{2}$ для $K_{1,3}$-свободных графов.

Доказательство. Пусть S является жёстким множеством графа G и $|S| = k$. Каждая компонента графа $G - S$ сожержит $k(G)$ различных соседей в S. G является $K_{1,3}$-свободным, поэтому каждая вершина из S имеет соседа не более, чем в двух компонентах графа $G - S$. Следовательно, $k(G)c(G-S) \leq 2k$ или $\frac{|S|}{c(G-S)} \geq \frac{k(G)}{2}$. Если учесть неравенство, показанное выше, то получим $\tau(G) = \frac{k(G)}{2}$. □

Поскольку каждая компонента графа $G-S$ добавляет как минимум одну вершину в независимое множество графа G (далее $\alpha(G)$), следует установить связь между параметрами $\alpha(G)$ и $\tau(G)$.

Теорема. [5] $\tau(G) \leq \frac{n-\alpha(G)}{\alpha(G)}$ для неполных графов на n вершинах.

Доказательство. Пусть $S = V_G \backslash I$, где I является максимальным независимым подмножеством графа G. Тогда $c(G-S) = \alpha(G)$. Из этого следует, что $\tau(G) \leq \frac{|S|}{c(G-S} = \frac{n-\alpha(G)}{\alpha(G)}$. □

Теорема. [5] $\tau(G) \leq \dfrac{k(G)}{\alpha(G)}$ для неполных графов.

Доказательство. Пусть S будет жёстким подмножеством графа G. Очевидно, что $|S| \geq k(G)$. Также очевидно, что $\alpha(G) \geq c(G - S)$. Отсюда следует, что $\tau(G) = \dfrac{|S|}{c(G-S)} \geq \dfrac{k(G)}{\alpha(G)}$. □

Теорема. [6] Для любого рационального t проблема t-жёсткости является coNP-полной для произвольного графа.

Пусть $\omega(r)$ обозначает класс всех графов G, удовлетворяющих следующему неравенству:
$\delta(G) \geq r|V(G)|$.

Теорема. [2] Пусть $t \geq 1$ любое рациональное число.

1. Любой граф из $\omega\left(\dfrac{t}{t+1}\right)$ t-жёсткий.
2. Для любого бесконечно маленького $\epsilon > 0$, распознавание t-жёсткости отсаётся NP-трудной задачей для графов из $\omega\left(\dfrac{t}{t+1} - \epsilon\right)$.

Жёсткость двудольного графа не может быть больше 1. Однако проблема t-жёсткости не становится легче для двудольных графов.

Теорема. [6] Для любого положительного рационального числа $t \leq 1$ проблема t-жёсткости является coNP-полной для двудольных графов.

Вопрос сложности распознавания жёсткости для многих классов графов до сих пор остаётся открытым:

- двусвязные, планарные, кубические, двудольные графы;
- двусвязные, планарные, кубические графы;
- двусвязные, планарные, двудольные графы;
- двусвязные, кубические, двудольные графы;
- двусвязные, планарные графы;
- 3-связные, двудольные графы;
- 3-связные, планарные графы;

При этом есть некоторые классы графов, для которых есть полиномиальные алгоритмы распознавания жёсткости.

Теорема. [6] $\forall t \geq 0$ t-жёсткий расщепляемый (Рисунок 1.1[9]) граф может быть распознан за полиномиальное время.

Позже данная теорема была расширена для $2K_2$-свободных графов (нетрудно видеть, что расщепляемый граф относится к $2K_2$-свободным).

Есть несколько дополнительных результатов для определённых классов графов. Например, для интервальных графов жёсткость можно определить за полиномиальное время.

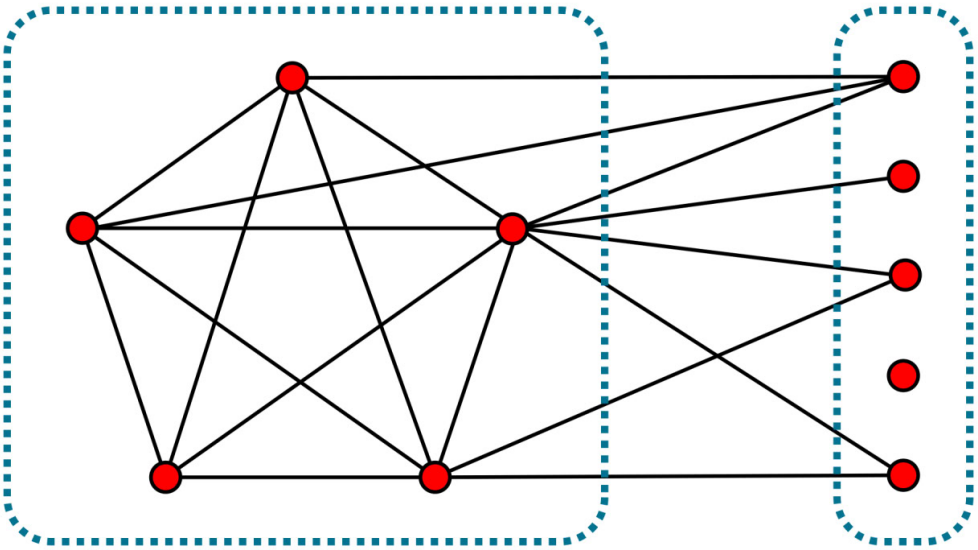

Рисунок 1.1 — Расщепляемый граф

ГЛАВА 2
СВЯЗЬ ЖЁСТКОСТИ С ГАМИЛЬТОНОВОСТЬЮ

Определение. Гамильтоновым циклом является такой цикл (замкнутая цепь), который проходит через каждую вершину данного графа ровно по одному разу; то есть простой цикл, в который входят все вершины графа. Граф, который содержит гамильтонов цикл, называется гамильтоновым.

Теорема. [8] Дан граф G порядка $n \geq 3$. Если $\delta(G) \geq \frac{n}{2}$, тогда G гамильтоновый.

Теорема. [8] Дан граф G порядка $n \geq 3$. Если $d(u) + d(v) \geq n$, для любых несмежных вершин u, v из G, то граф G гамильтоновый.

Теорема. [8] Дан граф G порядка $n \geq 3$. Если для любого $1 \leq j < \frac{n}{2}$ число вершин степени, не превосходящей j меньше, чем j, то G гамильтоновый.

2.1 Связь жёсткости с гамильтоновостью

Наблюдение, что циклический граф C_n является 1-жёстким, ведёт к тому, что гамильтоновый граф должен быть хотя бы 1-жёстким.

Предположение. [8] Любой 1-жёсткий граф является гамильтоновым.

Данное предположение неверно. Контрпример изображён на рисунке 2.1[8].

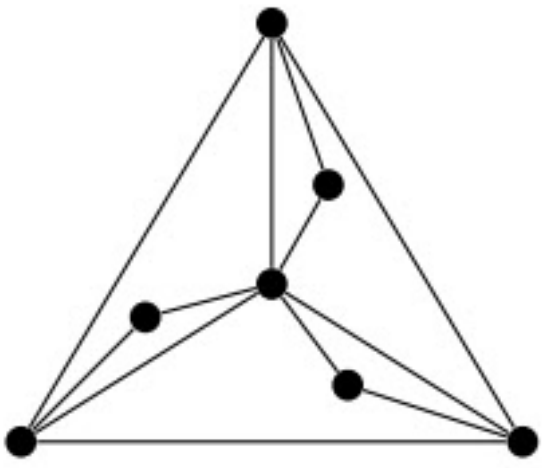

Рисунок 2.1 — 1-жёсткий граф без гамильтонового цикла

Предположение. [8] Существует такое вещественное число t_0, что любой t_0-жёсткий граф является гамильтоновым.

Вопрос справедливости данного утверждения для произвольного графа до сих пор остаётся открытым. Существуют классы графов, для которых данное предположение верно.

Предположение о том, что любой 2-жёсткий граф является гамильтоновым тоже было опровергнуто. Такой результат был показан тем, что для любого $\epsilon > 0$ существует $\left(\frac{9}{4} - \epsilon\right)$-жёсткий граф без гамильтонового пути. Это было достигнуто путём копирования более маленьких графов.

2.2 Определённые классы графов

Для многих классов графов предположение 2 было доказано[5].

- Любой t_0-жёсткий планарный граф является гамильтоновым при $t_0 > \frac{3}{2}$. Причём существует $\frac{3}{2}$-жёсткий негамильтоновый граф.
- Для $K_{1,3}$-свободных графов известно, что $t_0 \leq \frac{7}{2}$.
- Для хордальных графов, то есть графов без индуцированных циклов длины больше 3, известно, что $t_0 \leq 18$. При этом известна нижняя оценка на t_0: существует бесконечно много хордальных графов с жёсткостью близкой к $\frac{7}{4}$, которые не сожержат гамильтонов цикл.
- Каждый расщеплённый граф с жёсткостью не менее $\frac{3}{2}$ являлется гамильтоновым. При этом существует бесконечно много расщеплённых графов с жёсткостью близкой к $\frac{3}{2}$, которые не содержат гамильтонов цикл. То есть найдено точное значение для t_0.
- Для интервальных графов $t_0 = 1$, что очевидным образом нельзя улучшить.
- k-деревом является граф полученный из K_k последовательным добавлением новых вершин и соединением их с k связанными между собой вершинами. k-дерево является хордальным графом. Если G является k-деревом и его жёсткость не менее $\frac{k+1}{3}$, то G гамильтоновый.
- Любой 25-жёсткий $2K_2$-свободный граф гамильтоновый.
- Любой 1-жёсткий $2K_2$-свободный, K_3-свободный граф гамильтоновый.

- Пусть G является t-жёстким графом на не менее 3 вершинах. Если $\delta(G) \geq \frac{n}{t+1} - 1$, то G гамильтоновый.
- Пусть любой R-свободный 1-жёсткий граф гамильтоновый. Тогда R является индуцированным подграфом от $K_1 \cup P_4$.

2.3 Построение жёсткого негамильтонового графа

Сначала введём два графа L и M, изображённые на рисунках 2.2[8] и 2.3[8] соответственно. Также на графах обозначены вершины u, v, которые пригодятся далее.

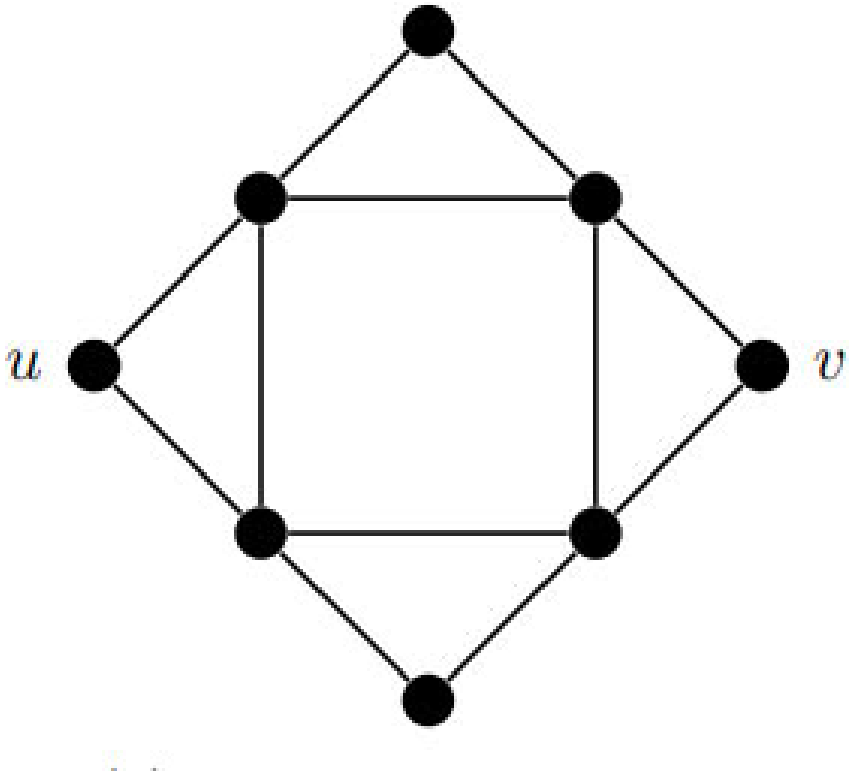

Рисунок 2.2 — Граф L

Дан граф H и две его вершины $x, y \in V(H)$. Граф $G(H, x, y, l, m)$ (l, m целые) определим далее. Граф H копируется m раз. Эти копии называются H_i, а x_i, y_i являются соответствующими копиями x, y из H. Граф F_m является объединением графов H_i с добавлением рёбер между каждой парой вершин из множества $\{x_1, ..., x_m, y_1, ..., y_m\}$. Граф G является объединением графа F_m с графом K_l.

Теорема. [8] Пусть H является графом, где вершины $x, y \in V(H)$ не соединены гамильтоновым путём. Если $m \geq 2l+3$, тогда $G(H, x, y, l, m)$ не содержит гамильтонового пути.

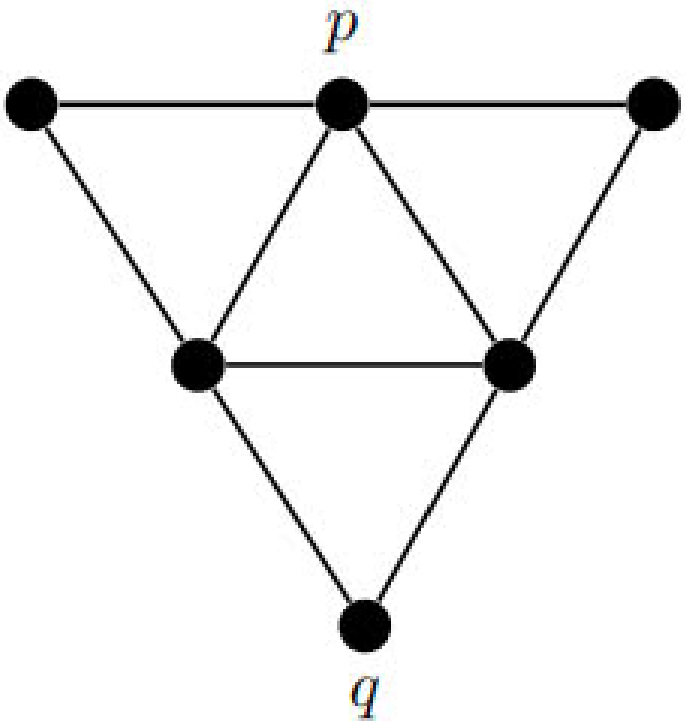

Рисунок 2.3 — Граф M

Теорема. [8] Для $l \geq 2, m \geq 1$ выполняется: $\tau(G(L,u,v,l,m)) = \dfrac{l+3*m}{2*m+1}$

Доказательство. От противного. Предположим, что $G(H,x,y,l,m)$ содержит гамильтонову цепь P. Пересечение P и F_m состоит из списка A состоящего не более, чем из $l+1$ не соединённого друг с другом пути, которые вместе содержат все вершины из F_m. $m \geq 2l+3 = 2(l+1)+1$, поэтому существует подграф H_{i_0} в F_m, такой, что ни одной конечной вершины пути из A, не лежит в H_{i_0}. H_{i_0} – копия графа H, то есть в нём нет гамильтонового пути из x_{i_0} в y_{i_0}. Пересечение P и H_{i_0} покрывает все вершины H_{i_0} и должно начинаться в x_{i_0} и заканчиваться в y_{i_0} или наоборот, потому что только эти вершины соединены с другими копиями H. Это противоречит тому, что в графе H нет гамильтонового пути между x и y. □

Теорема. [8] Пусть H является графом, где вершины $x, y \in V(H)$ не соединены гамильтоновым путём. Если $m \geq 2l+1$, тогда $G(H,x,y,l,m)$ не содержит гамильтонового пути.

Можно показать, что $\tau(G(L,u,v,l,m)) = \dfrac{9}{4} - \epsilon$ для бесконечно молого ϵ, если l и $m = 2l+1$ достаточно большие.

Если в графе G нет гамильтонового пути, то он не гамильтоновый.

Теорема. [8] Для $l \geq 2, m \geq 1$ выполняется: $\tau(G(L,u,v,l,m)) = \dfrac{l+4*m}{2*m+1}$

Можно показать, что $\tau(G(M,u,v,l,m)) = \dfrac{7}{4} - \epsilon$ для бесконечно молого ϵ, если l и $m = 2l+1$ достаточно большие.

2.4 Связь жёсткости с факторами

Определение. k-фактором графа G называется k-регулярный остновный подграф G.

Гамильтонов цикл является 2-фактором.

Теорема. [5] Пусть G k-жёсткий граф на $n \geq k+1$ и kn чётное. Тогда G имеет k-фактор.

Теорема. [5] Пусть $k \geq 1$. Для любого $\epsilon > 0$, существует $(k-\epsilon)$-жёсткий граф на $n \geq k+1$ вершинах с чётным kn, который не является гамильтоновым.

Теорема. [5] Пусть $k \geq 1$ и G – граф на $n \geq k+1$ вершинах (kn чётно). Предположим, что $|S| \geq k * c(G-S) - \frac{7k}{8}$ для всех $S \subseteq V(G)$ таких, что $c(G-S) > 1$. Тогда G является гамильтоновым.

ГЛАВА 3
СЛОЖНОСТЬ РАСПОЗНАВАНИЯ ЖЁСТКОСТИ В РЕГУЛЯРНЫХ ГРАФАХ

Сразу можно заметить, что для r-регулярного графа G $\tau(G) \leq \dfrac{k(G)}{2} \leq \dfrac{r}{2}$.

Покажем текущее состояние проблемы распознавания жёсткости для регулярных графов.

Теорема. [1] Распознавание 1-жёсткости является NP-трудным для кубических графов.

Доказательство. Докажем, что проблема 1-жёсткости является NP-трудной для кубических графов сведением проблемы 1-жёсткости для произвольного связного графа к данной проблеме.

Пусть $G \neq K_1$ является произвольным связным графом. Построим соответствующий ему граф $H(G)$. Каждой вершине $v \in V(G)$ поставим в соответствие граф H_v (см. Рисунок 3.1[1]). По обе стороны от обозначенной вершины e находится по $d_G(v)$ закрашенных вершин. Обозначим множество закрашенных вершин с одной стороны A_v, а с другой – B_v. Заметим, что каждая вершина не из $A_v \cup B_v$ принадлежит треугольнику.

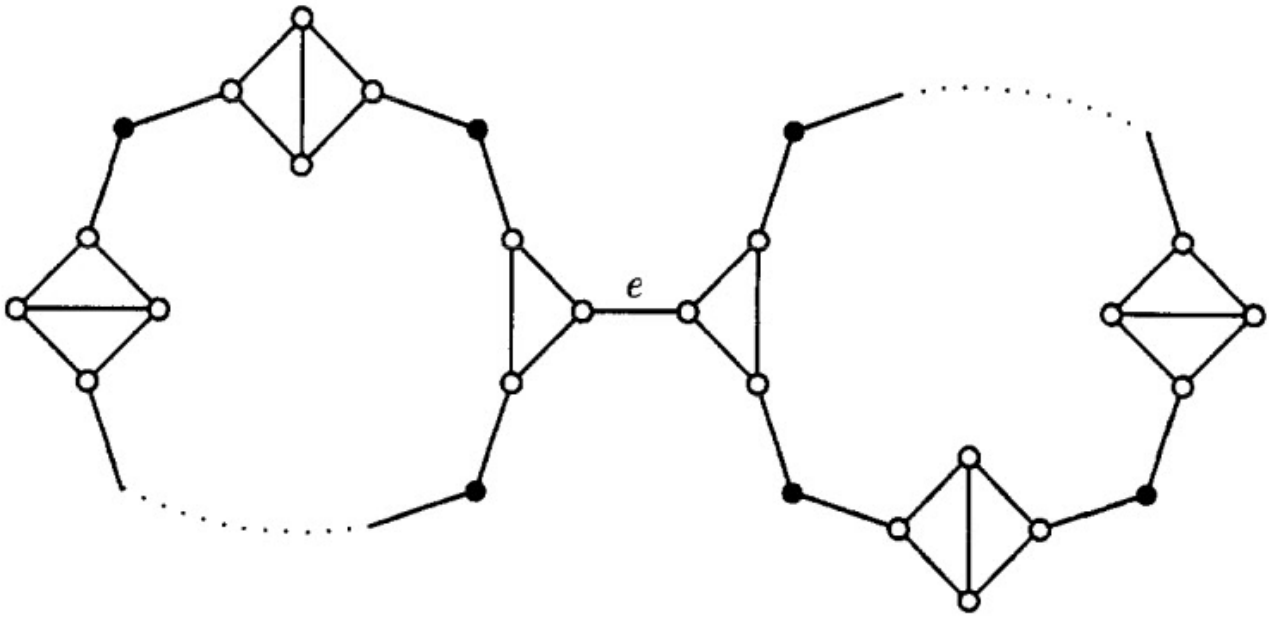

Рисунок 3.1 — Граф H_v

Ребро $uv \in E(G)$ в графе $H(G)$ будет представлено соединением ранее не использованной вершины из A_u с ранее не использованной вершиной из B_v и ранее не использованной вершиной из B_u с ранее не использованной вершиной из A_v.

Нетрудно видеть, что $H(G)$ является двусвязным и кубическим. Для доказательства теоремы теперь достаточно доказать следующее утверждение.

Утверждение. Граф G 1-жёсткий тогда и только тогда, когда $H(G)$ 1-жёсткий.

Доказательство. Предположим, что G не 1-жёсткий. Это значит, что существует непустое $X \subseteq V(G)$ такое, что $|X| < c(G - X)$. Возьмём $Y \subseteq V(H), Y = \bigcup_{v \in X}(A_v \cup B_v)$. Можно заметить, что $|Y| < c(H(G) - Y)$, что значит, что $H(G)$ не 1-жёсткий.

Теперь предположим, что $H(G)$ не является 1-жёстким. Это значит, что существует непустое $Y \subseteq V(H(G))$ такое, что $|Y| < c(H(G) - Y)$.

Лемма. Для любого $y \in Y$ $N_H(y)$ является независимым множеством.

Доказательство. От противного. Предположим, что $N_H(y)$ не являлется независимым множеством для некоторого $y \in Y$. Тогда возьмём $Y' = Y - \{y\}$. $|Y'| = |Y| - 1$ и $c(H - Y') \geq c(H - Y) - 1$. Это значит, что $|Y'| < c(H(G) - Y')$.

Если же $Y' = \emptyset$, то это значило бы, что множество из одного элемента разбивает H на несколько компонент, что противоречит его двусвязности. □

Исходя из этой леммы можно сделать вывод, что множество Y содержит только закрашенные врешины (все остальные принадлежат треугольникам).

Лемма. Для любой вершины $v \in V(G)$ $A_v \cap Y$ равно либо A_v, либо $\emptyset$. То же самое множно сказать и про B_v.

Доказательство. От противного. Предположим, что $A_v \cap Y$ не является ни A_v, ни $\emptyset$. Тогда в A_v, есть две "последовательные"(см. Рисунок 3.2[1]) вершины $a, a' \in A_v$ такие, что $a \in Y, a' \notin Y$. Возьмём $Y' = Y + \{a'\}$. Тогда получим $|Y'| = |Y| + 1$ и $c(H - Y') \geq c(H - Y) + 1$. Это значит, что $|Y'| < c(H(G) - Y')$.

Доказательство для B_v идентично. □

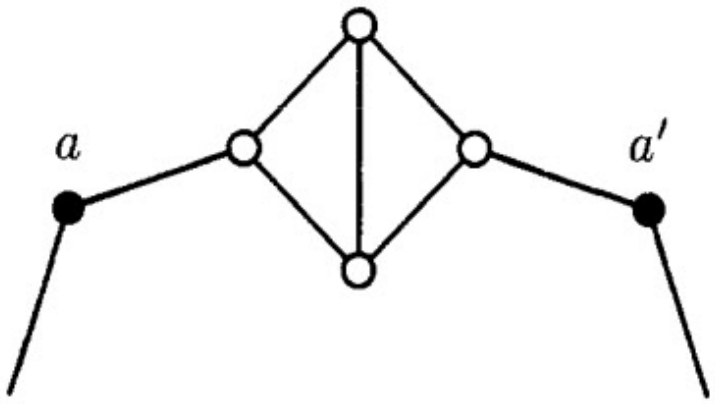

Рисунок 3.2 — "Последовательные"вершины

Определение. Вершина $v \in V(G)$ называется разделённой, если ровно одно из множеств A_v или B_v лежит в Y.

Лемма. Пусть $v \in V(G)$ является разделённой вершиной и $A_v \nsubseteq Y$ и $B_v \subseteq Y$. Тогда для любого $w \in N_G(v)$ $B_w \subseteq Y$.

Доказательство. От противного. Пусть $B_w \nsubseteq Y$. Возьмём $Y' = Y \cup A_v$. Тогда получим $|Y'| = |Y| + d_G(v)$ и $c(H - Y') \geq c(H - Y) + d_G(v)$. Это значит, что $|Y'| < c(H(G) - Y')$. □

Лемма. В графе G нет разделённых вершин.

Доказательство. От противного. Предположим, что S множество разделённых вершин G и $S \neq \emptyset$. Рассмотрим любую компоненту C из множества S в G. И пусть $V(C) = \{v_1, v_2, ..., v_m\}$. Предположим, что для всех v_i $A_{v_i} \nsubseteq Y$ и $B_{v_i} \subseteq Y$. Заметим, что если $wv_i \in E(G), w \notin V(C)$, то $A_w \subseteq Y, B_w \subseteq Y$. Также заметим, что $\bigcup_{i=1}^{m}(H_{v_i} - B_{v_i})$ индуцирует в графе H подграф с $d_G(v_1) + ... + d_G(v_m)$ компонентами связности. Теперь возьмём $Y' = Y \backslash (\bigcup_{i=1}^{m} B_{v_i})$. Пусть $Y = \emptyset$. Если $V(C) \neq V(G)$ то G не является связным, противоречие. Если же $V(C) = V(G)$, то $c(G - Y = |Y|$, потому что каждая вершина $v \in G$ будет порождать по $d_G(v)$ новых компонент связности в H, что в сумме даёт $|Y|$. Это противоречит условию, что $c(G - Y) > |Y|$. Можно считать, что $Y' \neq \emptyset$. Тогда

$|Y'| = |Y| - \sum_{i=1}^{m}(d_G(v))$ и $c(H - Y') = c(H - Y) - (\sum_{i=1}^{m}(d_G(v)) - 1)$.

Используя тот факт, что $c(H - Y) > |Y|$, получим

$c(H - Y') > |Y| - \sum_{i=1}^{m}(d_G(v)) + 1 = |Y'| + 1 > |Y'|$.

Далее можно просто модифицировать Y по всем таким компонентам C. □

Пусть $X = \{v \in V(G)\}$, где $A_v, B_v \subseteq Y$. Поскольку разделённых вершин нет, можно показать, что $c(G - X) > |X|$ и G не 1-жёсткий.

□

Доказательство данного утверждения ведёт к доказательсту теоремы.

□

Теорема. [2] Для любого целого $t \geq 1$ проблема t-жёсткости остаётся NP-трудной для $3t$-регулярных графов.

Теорема. [2] Для любого целого $r \geq 3$ проблема 1-жёсткости является NP-трудной для r-регулярных графов.

Доказательство. Известно, что проблема распознавания 1-жёсткости для кубических графов является NP-трудной. Для $r \geq 4$ докажем теорему сведением к проблеме 1-жёсткости для кубических графов.

Пусть G является кубическим графом. Для $r \geq 4$ построим соответствующий граф $H_r = H_r(G)$. Каждой вершине $v \in V(G)$ будет соответствовать граф $H_{v,r}$ (Рисунки 3.3, 3.4). Структура графа $H_{v,r}$ будет зависеть от чётности r. В любом случае каждые 3 чёрные вершины будут иметь степень $r - 1$.

Первый случай (Рисунок 3.3[2]): $r = 2k, k \geq 2$.

В этом случае мы имеем 4 базовых графа $(B_1^a, B_1^b, B_2^a, B_2^b)$ и один граф связи C_3.

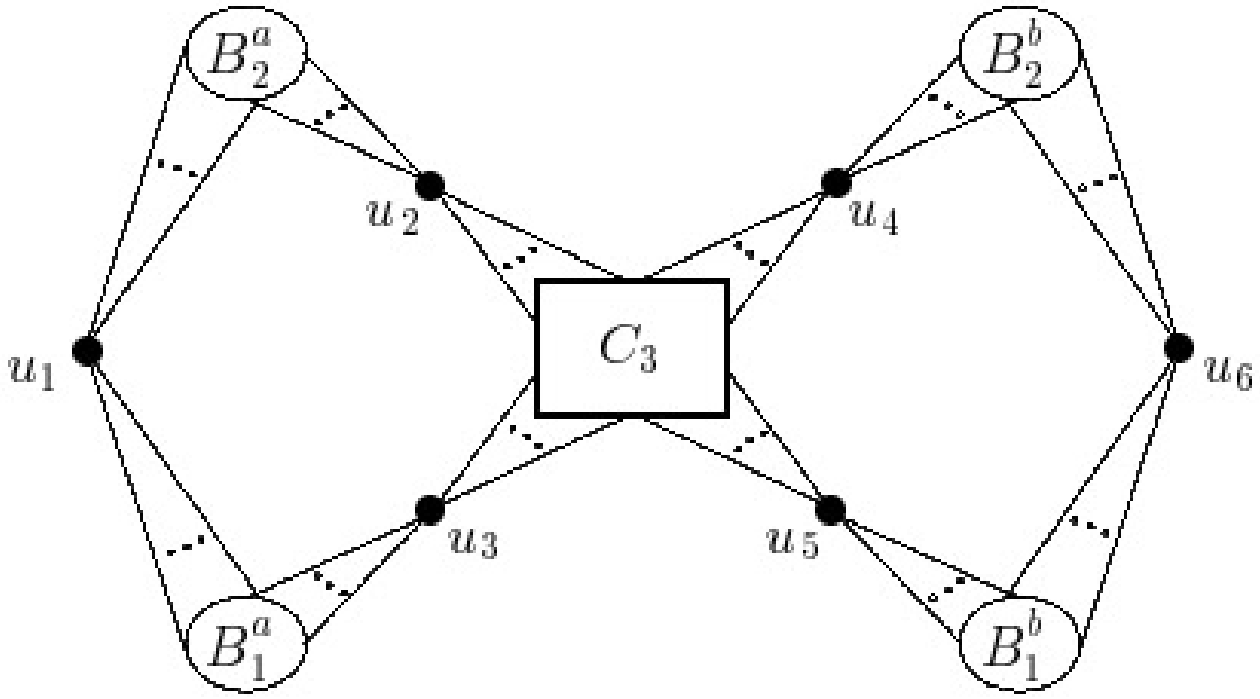

Рисунок 3.3 — r чётное

Для $j \in \{a, b\}$ граф B_1^j является полным графом на $2 * k$ вершинах. Положим $V(B_1^j) = R_1^j \cup S_1^j$, где $R_1^j = \{r_{1,1}^j, r_{1,2}^j, ..., r_{1,k}^j\}$ и $S_1^j = \{s_{1,1}^j, s_{1,2}^j, ..., s_{1,k}^j\}$.

Для $j \in \{a, b\}$ граф B_2^j является полным графом на $2 * k + 1$ вершинах с некоторыми удалёнными рёбрами. Положим $V(B_2^j) = R_2^j \cup S_2^j \cup \{w\}$, где $R_2^j = \{r_{2,1}^j, r_{2,2}^j, ..., r_{2,k}^j\}$ и $S_2^j = \{s_{2,1}^j, s_{2,2}^j, ..., s_{2,k}^j\}$. При этом удаляются рёбра $\{r_{2,1}^j s_{2,1}^j, r_{2,2}^j s_{2,2}^j, ..., r_{2,k-1}^j s_{2,k-1}^j\}$.

C_3 является полным графом на $2k$ вершинах с некоторыми удалёнными рёбрами. Положим $V(_3) = R_3 \cup S_3$, где $R_3 = \{r_{3,1}, r_{3,2}, ..., r_{3,k}\}$ и $S_3 = \{s_{3,1}, s_{3,2}, ..., s_{3,k}\}$. При этом удаляются рёбра $\{r_{3,1}s_{3,1}, r_{3,2}s_{3,2}, ..., r_{3,k-1}s_{3,k-1}\}$.

Теперь рассмотрим 6 чёрных вершин $\{u_1, u_2, ..., u_6\}$.

u_1 соединена с $R_1^a \cap R_2^a - \{r_{2,k}^a\}$,

u_2 соединена с $S_2^a \cap R_3 - \{s_{2,k}^a\}$,

u_3 соединена с $S_1^a \cap R_3 - \{r_{3,k}\}$,

u_4 соединена с $R_2^b \cap S_3 - \{r_{2,k}^b\}$,

u_5 соединена с $R_1^b \cap S_3 - \{s_{3,k}\}$,

u_6 соединена с $S_1^b \cap S_2^b - \{s_{2,k}^b\}$.

Второй случай (Рисунок 3.4[2]): $r = 2k + 1, k \geq 2$.

В этом случае мы имеем 4 базовых графа $(B_4^a, B_4^b, B_4^c, B_4^d)$ и один граф связи C_5.

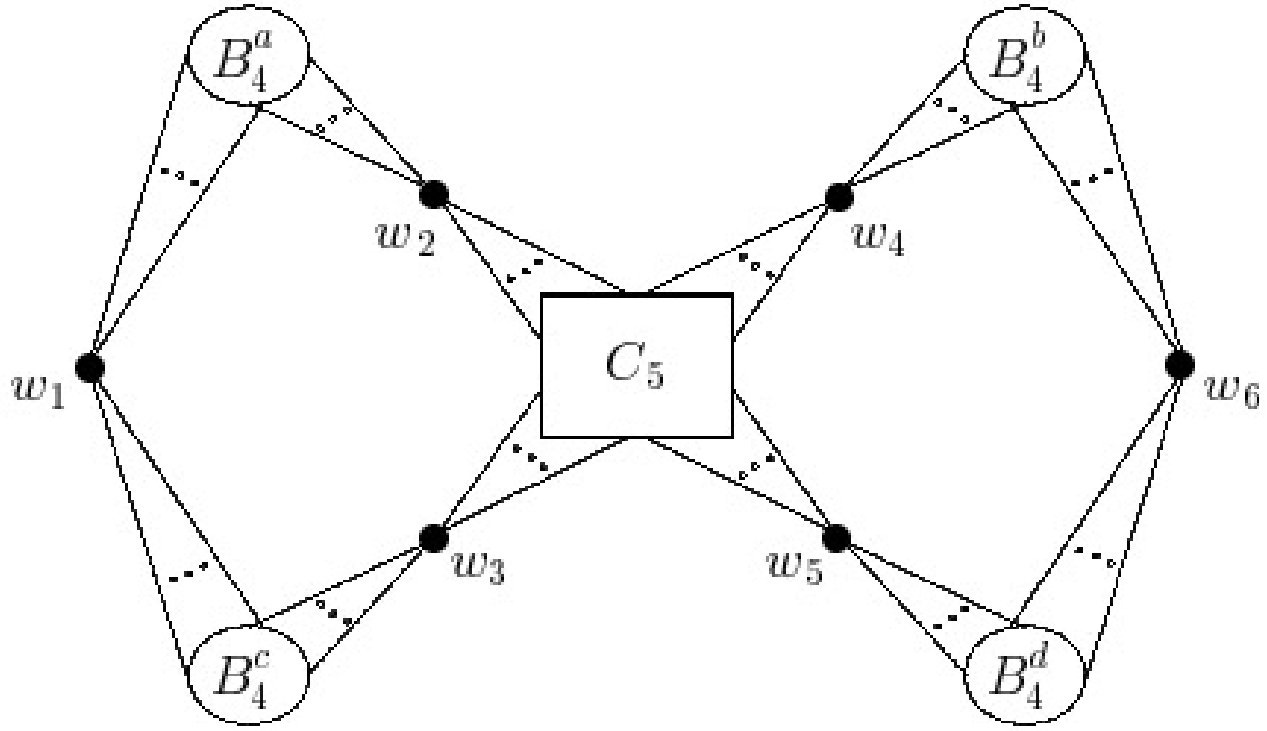

Рисунок 3.4 — r нечётное

Для $j \in \{a, b, c, d\}$ граф B_4^j является полным графом на $2 * k + 2$ вершинах с некоторыми удалёнными рёбрами. Положим $V(B_4^j) = R_4^j \cup S_4^j$, где $R_4^j = \{r_{4,1}^j, r_{4,2}^j, ..., r_{4,k+1}^j\}$ и $S_4^j = \{s_{4,1}^j, s_{4,2}^j, ..., s_{4,k+1}^j\}$. При этом удаляются рёбра $\{r_{4,1}^j s_{4,1}^j, r_{4,2}^j s_{4,2}^j, ..., r_{4,k}^j s_{4,k}^j\}$.

C_5 является полным графом на $2k+2$ вершинах с некоторыми удалёнными рёбрами. Положим $V(_5) = R_5 \cup S_5$, где $R_5 = \{r_{5,1}, r_{5,2}, ..., r_{5,k+1}\}$ и $S_5 = \{s_{5,1}, s_{5,2}, ..., s_{5,k+1}\}$. При этом удаляются рёбра $\{r_{5,1}s_{5,2}, r_{5,2}s_{5,3}, ..., r_{5,k}s_{5,k+1}\}$ и $\{r_{5,2}s_{5,1}, r_{5,3}s_{5,2}, ..., r_{5,k+1}s_{5,k}\}$.

Теперь рассмотрим 6 чёрных вершин $\{w_1, w_2, ..., w_6\}$.

w_1 соединена с $R_4^a \cap R_4^c - \{r_{4,k+1}^a, r_{4,k+1}^c\}$,

w_2 соединена с $S_4^a \cap R_5 - \{s_{4,k+1}^a, r_{5,k+1}\}$,

w_3 соединена с $S_4^c \cap R_5 - \{s_{4,k+1}^c, r_{5,1}\}$,

w_4 соединена с $R_4^b \cap S_5 - \{r_{4,k+1}^b, s_{5,k+1}\}$,

w_5 соединена с $R_4^d \cap S_5 - \{r_{4,k+1}^d, s_{5,1}\}$,

w_6 соединена с $S_4^b \cap S_4^d - \{s_{4,k+1}^b, s_{4,k+1}^d\}$.

Для чётного r обозначим чёрные вершины в $H_{v,r}$ с одной стороны от C_3 как $A_{v,r}$, а те, что на другой стороне, как $B_{v,r}$. Аналогично для нечётного r обозначим чёрные вершины в $H_{v,r}$ с одной стороны от C_5 как $A_{v,r}$, а те, что на другой стороне, как $B_{v,r}$ Таким образом множество чёрных вершин в H_r это $B = \cup_{v \in V(G)} A_{v,r} \cup Bv, r$. Пусть W является оставшимся множеством "белых"вершин в H_r. Ребро vw в G будет представлено в H_r соединением

любой ранее не использованной вершины из $A_{,r}$ с любой ранее неиспользованной вершиной из $B_{w,r}$ и любой ранее неиспользованной вершиной из $A_{w,r}$ с любой ранее не использованной вершиной из $B_{,r}$. Заметим, что H_r является r-регулярным 2-связным графом.

Чтобы доказать теорему, достаточно доказать следующее утверждение:

Граф G является 1-жёстким тогда и только тогда, когда $H_r(G)$ 1-жёсткий.

Доказательство:

Предположим, что G не 1-жёсткий. Тогда существует непустое множество $X \subseteq V(G)$ такое, что $c(G - X) > |X|$. Пусть $Y \subseteq V(H_r)$ и $Y = \cup_{v \in X}(A_{v,r} \cup B_{v,r})$. Легко показать, что $c(H_r - Y) > |Y|$, а это значит, что H_r не 1-жёсткий.

Теперь предположим, что H_r не является 1-жёстким. Тогда существует непустое множество такое, что $c(H_r - Y) > |Y|$. Далее (Леммы 1-4) будут указаны свойства, которым множество Y должно удовлетворять.

Лемма. Можно допустить, что $Y \cap W = \emptyset$

Лемма. Можно допустить, что для всех $v \in V(G)$ $A_{v,r} \cap Y$ будет либо $A_{v,r}$, либо $\emptyset$. Аналогично для $B_{v,r}$.

Лемма. Пусть $v \in V(G)$. Если $A_{v,r}$ не является подмножеством Y,а $B_{v,r} \subseteq Y$, тогда можно предположить для всех $w \in N_G(v)$, что $B_{w,r} \subseteq Y$.

Лемма. Можно предположить, что нет расщеплённых вершин.

Пусть $X = \{v \in V(G) | A_{v,r}, B_{v,r} \subseteq Y$. Нет расщеплённых вершин. Тогда легко показать, что X является непустым подмножеством $V(G)$ таким, что $c(G - X) > |X|$. То есть G не 1-жёсткий. □

Следует заметить, что для любого $r \geq 4$ $H_{v,r}$ содержит 1-фактор. Тогда можно получить следующий результат:

Для любого фиксированного целого $r \geq 3$ проблема 1-жёсткости остаётся NP-трудной для класса 2-связных, r-регулярных графов, содержащих 1-фактор.

Теорема. [2] Для любого целого t и любого фиксированного $r \geq 3t$ проблема распознавания t-жёсткости остаётся NP-трудной для r-регулярных графов.

Теорема. [6] Для любого положительного рационального $t < \frac{2}{3}$ есть полиномиальный алгоритм для распознавания t-жёсткого 3-регулярного графа.

Теорема. [6] Пусть G является кубическим графом. Тогда G является $\frac{2}{3}$-жёстким тогда и только тогда, когда $G = K_4$, $G = K_2 \times K_3$ или является инфляцией 3-связного кубического графа.

Теорема. [6] 1/2-жёсткий 4-регулярный граф можно распознать за полиномиальное время.

ГЛАВА 4
ЭФФЕКТИВНЫЙ АЛГОРИТМ ПОИСКА ЖЁСТКОСТИ ДЛЯ ГРАФОВ С ОГРАНИЧЕННОЙ ДРЕВЕСНОЙ ШИРИНОЙ

4.1 Древесная ширина графа

Определение. [6] Дан граф $G = (V, E)$. 2^V обозначим множество всех подмножеств из V. Древесной декомпозицией графа G является пара $\tau = (X, T)$, где $X = (X_b | b \in B) \subseteq 2^V$, а $T = (B, F)$ является деревом, которое удовлетворяет следующим условиям:

1. $\cup_{b \in B} X_b = V$,
2. Для любых $u, v \in E$ существует $b \in B$ такое, что $u, v \subseteq X_b$,
3. Для любых $i, j, k \in B$, если j лежит на пути от i до k, то $X_i \cap X_k \subseteq X_j$.

Отсюда следует, что для любого $v \in V$ множество вершин $\{b \in B | v \in X_b\}$ индуцирует поддерево в T.

Определение. Шириной древесной декомпозиции $\tau = (X, T)$ является величина, равная $max_{b \in V_T} |X_b| - 1$. Древесной шириной графа G является минимальная ширина древесной декомпозиции по всем декомпозициям G. Обозначается $tw(G)$.

Вычисление древесной ширины для произвольного графа является NP-трудной задачей. Однако, если древесная ширина ограничена, то она может быть вычислена за полиномиальное время[3].

Будем использовать понятие вершина для элементов $v \in V$, нода для элементов $b \in B$, мешок для элементов $X_b \in X$. Будем говорить, что мешок X_b соответствует ноде $b \in B$ в дереве T.

Древесную декомпозицию можно сделать корневой, назначив какую-либо из нод корнем.

Определение. [6] Корневая древесная декомпозиция называется красивой, если каждый мешок $X_i \in X$ в ней относится к одному из следующих типов:

1. Нода i не имеет детей в T (мешок первого типа, листьевой),
2. Нода i имеет в точности одного ребёнка j, и $X_i \subseteq X_j$ и $|X_i| = |X_j| - 1$ (мешок второго типа),
3. Нода i имеет в точности одного ребёнка j, и $X_j \subseteq X_i$ и $|X_i| = |X_j| + 1$ (мешок третьего типа),

4. Нода i имеет в точности двух детей k, j и $X_i = X_j = X_k$ (мешок четвёртого типа).

Каждая нода i имеет тип (от первого до четвёртого), соответствующий X_i.

Красивая древесная декомпозиция называется очень красивой, если все листьевые мешки в ней имеют размер 1.

Дана древесная декомпозиция графа G с древесной шириной $tw(G)$ с $O(n)$ нод. Тогда очень красивая древесная декомпозиция с шириной $tw(G)$ и числом нод $O(4n)$ может быть получена за полиномиальное время[4].

Далее будем считать, что для графа G уже известна очень красивая древесная декомпозиция с шириной $tw(G)$.

4.2 Построение древесной декомпозиции

Теорема. [3] Для любого натурального k существует линейный алгоритм, который проверяет, что древесная ширина графа $G = (V, E)$ не превосходит k, и, если это так, то выдаёт древесную декомпозицию графа G с шириной k.

Опишем суть данного алгоритма из [3].

Основная идея заключается в следующем: вершины графа G разбиваются на 2 подмножества. В первом подмножестве находятся вершины с "маленькой"степенью, во втором – с "большой". Можно показать, что граф с древесной шириной не более k содержит "совсем немного"вершин с большой степенью.

Рассмотрим два случая:

1. "Достаточно много"вершин с маленькой степенью смежны с другими вершинами с маленькой степенью. В этом случае можно показать, что максимальное паросочетание в G содержит достаточно много ($\Omega(n)$) рёбер. Можно получить граф G' путём стягивания всех вершин в максимальном паросочетании. Рекурсивно можно получить древесную декомпозицию с шириной k для графа G' или заключить, что древесная ширина G', а следовательно и G, больше k. Из древесной декомпозиции G' с шириной k можно получить древесную декомпозицию для G с шириной не более $2k + 1$. Из неё можно получить искомую древесную декомпозицию для G.

2. "Совсем немного"вершин с маленькой степенью являются смежными с другими вершинами с маленькой степенью. Можно показать, что некоторые новые рёбра могут быть добавлены в G без увеличения древесной ширины от числа, не превосходящего k, до числа, которое больше k. "Улучшенный граф G в который добавлены новые рёбра, имеет достаточно много ($\Omega(n)$) вершин, которые I-симплицированы, то есть их

соседи образуют клику в улучшенном графе. Получаем граф G' путём удаления всех I-симплицированных вершин. Рекурсивно можно получить древесную декомпозицию с шириной k для графа G' или заключить, что древесная ширина G', а следовательно и G, больше k. Имея древесную декомпозицию графа G' с шириной k можно получить подходящую древесную декомпозицию для G.

В каждом случае объём нерекурсивной работы будет линейным, а размер графа G' будет почти в константу раз меньше размера G, что показывает, что алгоритм работает за линейное время.

Теорема. [4] Пусть дан граф G на n вершинах. Также дана древесная декомпозиция $\mathcal{T} = (X, T)$ графа G с шириной k на $O(n)$ нодах. Тогда за $O(kn)$ время можно получить красивую древесную декомпозицию G на не более $4n$ нодах.

Доказательство. Приведём набросок доказательства.

Выберем произвольную вершину $r \in V(T)$ и сделаем дерево корневым за $O(n)$ (то есть построим списки детей для каждой ноды) с корнем r. Для каждого ребра $uv \in E(T)$ найдём $X_u \backslash X_v$ и $X_v \backslash X_u$. Можно показать, что это можно сделать за $O(k)$. Если $X_u = X_v$, то стянем ребро uv. Иначе подразделим uv $|X_u \backslash X_v| + |X_v \backslash X_u| - 1$ новыми нодами. Для этого на пути от u к v каждая следующая нода должна либо добавлять элемент из $X_v \backslash X_u$, либо удалять элемент из $X_u \backslash X_v$. Пусть у нас есть нода b с $t \geq 2$ детей. Тогда добавим $2t - 2$ детей с таким же мешком, как и у b. Далее построим дерево с корнем в b такое, что каждая вершина a с $X_a = X_b$ имела либо двух детей с такими же мешками, либо одного изначального ребёнка b. Можно показать, что построенная таким образом древесная декомпозиция будет содержать не более $4n$ нод. Добавление каждой новой ноды может быть сделано за $O(k)$, что даёт итоговое время $O(kn)$. □

4.3 Алгоритм

Данный алгоритм распознаёт жёсткость графа G за $O(|V(G)|^3 tw(G)^{2tw(G)})$, используя при этом динамическое программирование[6].

Нам дана очень красивая корневая древесная декомпозиция. Пусть V_b обозначает множество всех вершин графа G, которые находятся в мешках потомков $b \in B$ и $G_b := G[V_b]$.

Требуется вычислить следующую функцию для каждой ноды $b \in B$ "снизу вверх":

$Mnc(b, s, \mathcal{Q}, \mathcal{P})$:= максимальное число компонент связности (maximum number of components) графа $G_b - S$, где максимум берётся для всех множеств $S \subseteq V_b$, которые удовлетворяют следующим свойствам:

1. $|S| = s$,
2. $S \cap X_b = \mathcal{Q}$,
3. $\mathcal{P}$ является разбиением $X_b - \mathcal{Q} = X_b - S$, где 2 вершины принадлежат одному множеству, если они принадлежат одной компоненте связности графа $G_b - S$.

Если не существует подходящего S, то значение функции будем считать нулевым.

Для каждого $b \in B$ вычислим $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ для каждого возможного значения $0 \leq s < n$, для каждого $\mathcal{Q} \subseteq X_b$ и разбиения $\mathcal{P}$ используя ранее посчитанные значения в детях.

В итоге получаем:

$$\tau(G) = min\left\{\frac{s}{Mnc\,(r, s, \mathcal{Q}, \mathcal{P})} \mid 0 \leq s \leq n, Mnc\,(r, s, \mathcal{Q}, \mathcal{P}) \geq 2\right\},$$

где r является корнем T.

4.4 Детали алгоритма и обоснование корректности

Данный алгоритм и обоснование его корректности описаны в [6].

Введём некоторые определения и обозначения для разбиений, которые в дальнейшем пригодятся при доказательствах.

Определение. Разбиением $\mathcal{P}_A = \{A_1, A_2, ..., A_p\}$ множества A является множество непустых подмножеств $A_i \subseteq A$, которые попарно не пересекаются и $\bigcup_{i=1}^{p} A_i = A$. Подмножество A_i назовём блоком.

Для удобства будем считать, что $\mathcal{P}_A \cup \{\emptyset\} = \mathcal{P}_A$.

Определение. Если $x \notin A$, то будем говорить, что разбиение $\mathcal{P}_A$ не покрывает A.

Определение. Разбиение $\{A\}$ назовём тривиальным.

Определение. Если $a \in A$, то $\mathcal{P}_A - a = \{A_1 - \{a\}, A_2 - \{a\}, ..., A_p - \{a\}\}$. Причём изменится (или удалится) только одно подмножество.

Определение. Если $A' \subset A$, то $\mathcal{P}_A$ индуцирует разбиение $\mathcal{P}_{A'} = \{A_1 \cap A', A_2 \cap A', ..., A_p \cap A'\}$.

Определение. Разбиение $\mathcal{Q}_A = B_1, B_2, ..., B_3$ называется уточнением $\mathcal{P}_A$ и обозначается $\mathcal{P}_A \sqsubseteq \mathcal{Q}_A$, если каждое множество B_i является подмножеством некоторого A_i. Единичным уточнением $\mathcal{Q}_A$ можно назвать, если $B_i = A_i$ для всех A_i, кроме одного (или нуля).

Определение. Объединением двух разбиений $\mathcal{P}$ и $\mathcal{Q}$ (обозначим $\mathcal{P} \sqcup \mathcal{Q}$) является разбиение $\mathcal{R}$ такое, что $\mathcal{R}_A \sqsubseteq \mathcal{Q}_A$ и $\mathcal{R}_A \sqsubseteq \mathcal{Q}_A$.

Лемма. Объём информации, которая храниться в каждой ноде дерева, $O\left(|V(G)|tw(G)^{2tw(G)}\right)$

Доказательство. Для каждой вершины b мы храним максимальное количество компонент связности в $G_b - S$ для всех возможных комбинаций $s, \mathcal{Q}, \mathcal{P}$. $s = |S|$, поэтому в худшем случае будем иметь $|V(G)|$ вариантов выбора.

$\mathcal{P}$ является разбиением $X_b - \mathcal{Q}$, поэтому можно считать, что пара $\mathcal{P}, \mathcal{Q}$ является разбиением X_b, где одно из множеств помечено. Известно, что $|X_b| \leq tw(G)+1$. Также известно, что количество независимых разбиений множества и n элементов ограничено числом $\left(\frac{0.792n}{ln(n+1)}\right)^n$. Любое из подмножеств может быть помечено, а всего их может быть не более n.

Получаем, что для каждого s хранится $n\left(\frac{0.792n}{ln(n+1)}\right)^n = O\left((n+1)^{(n+1)}\right) = O(tw(G)^{tw(G)})$ информации. □

Лемма. Если b является листьевой нодой очень красивой древесной декомпозиции, то $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ может быть посчитано за $O(1)$ время.

Доказательство. Нода b является листьевой, поэтому $|X_b| = 1$. Рассмотрим два случая.

$s = 0$.

В таком случае имеем $S = \emptyset, \mathcal{Q} = \emptyset, \mathcal{P} = \{X_b\}$. Тогда очевидно, что $Mnc(b, s, \mathcal{Q}, \mathcal{P}) = 1$.

$s = 1$.

Имеем $S = X_b, \mathcal{Q} = X_b, \mathcal{P} = \emptyset, Mnc(b, s, \mathcal{Q}, \mathcal{P}) = 0$.

Других значений для вычислений нет, поэтому всю информацию для листьевой ноды можно посчитать за константное время. □

Лемма. Если b является нодой второго типа очень красивой древесной декомпозиции, то $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ может быть посчитано за $O(tw(G))$ время.

Доказательство. Пусть b' будет единственным ребёнком b в дереве T. И пусть $X_{b'} = X_b \cup \{f\}, f \in V$. Чтобы вычислить $Mnc(b, s, \mathcal{Q}, \mathcal{P})$, надо рассмотреть все $S \subset V_b$, которые удовлетворяют параметрам. Рассмотрим два случая.

$f \in S$.

В таком случае максимальным количеством компонент связности будет $Mnc(b', s, \mathcal{Q} \cup \{f\}, \mathcal{P})$. Достаточно показать, что в обоих случаях мы имеем дело с одинаковым множеством множеств S, который удовлетворяют параметрам.

Заметим, что $f \in S \cap X_{b'} = \mathcal{Q} \cup \{f\}$ и $\mathcal{P}$ не покрывает f. В $Mnc(b', s, \mathcal{Q} \cup \{f\}, \mathcal{P})$ мы рассматриваем только те S, которые содержат f. $G_b = G_{b'}$, поэтому $C(G_b - S) = C(G_{b'} - S)$. Также $X_b - S = X_{b'} - S$ и параметр $\mathcal{P}$ одинаковый в обеих функциях.

$f \notin S$.

В этом случае $C(G_{b'} - S)$ индуцирует разбиение $\mathcal{P}'$ в $X_{b'} - S = (X_b - S) \cup \{f\}$. Поскольку $C(G_b - S) = C(G_{b'} - S)$, структура $\mathcal{P}$ и $\mathcal{P}'$ должна быть одинаковой. То есть $\mathcal{P} = \mathcal{P}' - f$. Отсюда получаем, что $Mnc(b, s, \mathcal{Q}, \mathcal{P}) = max\{Mnc(b', s, \mathcal{Q}, \mathcal{P}') | \mathcal{P}' - f = \mathcal{P}\}$, поскольку и справа, и слева мы рассматриваем одинаковое множество множеств S.

В первом случае нужное значение вычисляется за константу. Во втором же случае, чтобы получить все подходящее $\mathcal{P}'$, надо к каждому из подмножеств $\mathcal{P}$ добавить f. Поскольку $|\mathcal{P}| \leq tw(G)$, ограничение по времени будет тоже $O(tw(G))$. □

Лемма. Если b является нодой третьего типа очень красивой древесной декомпозиции, то $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ может быть посчитано за $O(tw(G)^{tw(G)})$ время.

Доказательство. Пусть b' будет единственным ребёнком b в дереве T. И пусть $X_{b'} = X_b - \{i\}, i \in V$. Чтобы вычислить $Mnc(b, s, \mathcal{Q}, \mathcal{P})$, надо рассмотреть все $S \subset V_b$, которые удовлетворяют параметрам. Рассмотрим два случая.

$i \in S$.

Докажем, что в данном случае максимальным числом компонент будет $Mnc(b', s - 1, \mathcal{Q} - \{i\}, \mathcal{P})$. Для этого достаточно показать, что любое подмножество из $S \subseteq G_b$ такое, что $i \in S$, удовлетворяет параметрам $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ тогда и только тогда, когда $S' = S - \{i\}$ удовлетворяет параметрам $Mnc(b', s - 1, \mathcal{Q} - \{i\}, \mathcal{P})$. При этом также $c(G_b - S) = c(G_{b'} - S')$.

Заметим, что $|V_{b'} \cap S'| = s - 1$, потому что $|V_b \cap S| = s$ и $i \notin V_{b'}$. Аналогично $X_{b'} \cap S' = \mathcal{Q} - \{i\}$. Также заметим, что $C(G_b - S) = C(G_{b'} - S')$, поэтому $c(G_b - S) = c(G_{b'} - S')$. Это также говорит о том, что $X_b - S = X_{b'} - S'$, поэтому разбиение $\mathcal{P}$ не меняется.

$i \notin S$.

В этом случае S будет полностью входить в $G_{b'}$ с $|S| = s$ и $S \cap X_{b'} = \mathcal{Q}$. Поскольку $i \notin S$, i принадлежит одной из компонент $G_b - S$. Возможно, что удаление i разделит эту компоненту на несколько. То есть мы получим компоненты $G_{b'} - S$ удалением i из её компоненты. Эта компонента может разделиться на несколько, но остальные не будут затронуты.

Пусть $\mathcal{C}_{V_b - S} = \{C_1, C_2, ..., C_P\}$ будет разбиением, которое образуют компоненты связности $G_b - S$. Без ограничения общности предположим, что $i \in C_1$. Теперь пусть $\mathcal{D}_{V_{b'} - S} = \{D_1, D_2, ..., D_q, C_2, ..., C_p\}$ будет разбиением, которое образуют компоненты связности $G_{b'} - S$. Это подразумевает, что в каждом блоке D_i есть вершина, которая является соседом i. Также заметим, что все

соседи i находятся в $X_{b'}$ из-за структуры древесной декомпозиции. $\mathcal{C}$ индуцирует разбиение $\mathcal{P} = \{A_1, A_2, ..., A_{p'}\}$ в $X_b - S$, тогда пусть $\mathcal{D}$ индуцирует разбиение $\mathcal{P}' = \{B_1, B_2, ..., B_{q'}, A_2, ..., A_{p'}\}$. Тут также подразумевается, что в каждом блоке B_i должен находиться хотя бы один сосед i. Назовём разбиение $\mathcal{P}'$ i-уточнением разбиения $\mathcal{P}$, если $\mathcal{P}'$ было получено из $\mathcal{P}$ удалением вершины i и уточнением блока, который содержит эту вершину, так, чтобы каждый подблок содержал хотя бы одного соседа i. Получаем, что максимальным числом компонент в данном случае будет

$max\{Mnc(b', s, \mathcal{Q}, \mathcal{P}')|\mathcal{P}$ является i-уточнением $\mathcal{P}\}$.

Следует уточнить, что в некоторые разбиения $\mathcal{P}'$, которые являются i-уточнениями $\mathcal{P}$ не являются валидными (нет таких S, которые им удовлетворяют). Так, например, если v и w являются соседями i и принадлежат разным блокам B_j, но при этом являются соседями между собой, то не будет такого подмножества S, которое их разделит. Во всех таких случаях $Mnc(b', s, \mathcal{Q}, \mathcal{P}')$ считается неопределённым. Все такие неопределённые значения можно считать нулевыми. Тогда они не повлияют на ответ.

Итоговым значением будет максимум из двух вышеописанных случаев.

Вычислительное время в первом случае будет константным. Во втором случае мы должны найти в i-уточнения $\mathcal{P}$. В худшем случае $|\mathcal{P}| = 1$ и i является соседом каждой вершины. В этом случае нужно будет перебрать все разбиения X_b, размер которого не больше $tw(G)+1$, поэтому количество разбиений ограниченно $O(tw(G)^{tw(G)})$ (доказательство в Лемме 1). □

Лемма. Если b является нодой четвёртого типа очень красивой древесной декомпозиции, то для всех фиксированных s $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ для всех возможных $\mathcal{Q}, \mathcal{P}$ может быть посчитано за $O(|V(G)|tw(G)^{2tw(G)})$ время.

Доказательство. Пусть b' и b'' будут детьми ноды b в дереве T. Поскольку b является нодой 4 типа, мы имеем $X_b = X_{b'} = X_{b''}$ и $V_{b'} \cap V_{b''} = X_b$. В отличае от предыдущих лемм, чтобы вычислить $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ мы не будем рассматривать все подмножества S, которые удовлетворяют параметрам. Пусть $V_{b'} \cap S = S'$ и $V_{b''} \cap S = S''$. По свойствам декомпозиций $S' \cap S'' = S \cap X_b = \mathcal{Q}$. В дальнейшем покажем, что достаточно рассмотреть такие S' и S'', которые являются оптимальными. □

Утверждение. Пусть $\mathcal{Q}$ и $\mathcal{P}$ зафиксированы и пусть S будет таким, что $c(G_b - S)$ будет максимальным среди всех S, которые удовлетворяют параметрам (то есть $S \cap V_b = \mathcal{Q}, C(G_b - S)$ индуцирует разбиение $\mathcal{P}$ в X_b). Это подразумевает, что $c(G_{b'} - S')$ максимальное среди всех подмножеств, которые удовлетворяют параметрам. Аналогичное утверждение работает и для S''.

Доказательство. Предположим, что $c(G_{b'} - S')$ не максимальное. Пусть $R' \subseteq G_{b'}$ будет подмножеством, которое удовлетворяет тем же параметрам, и $c(G_{b'} - R') > c(G_{b'} - S')$. Определим $R = R' \cup S''$, то есть заменим S' на R'

(и соответственно заменим $S' - \mathcal{Q}$ на $R' - \mathcal{Q}$. Заметим, что R соответствует тем же параметрам, что и S ($\mathcal{P}$ и $\mathcal{Q}$).

Есть три типа компонент связности в $G_b - S$: которые находятся полностью в $V_{b'}$; которые находятся полностью в $V_{b''}$; которые находятся частично и там, и там (они также должны пересекать X_b по свойствам древесной декомпозиции).

После замены S на R количество компонент третьего типа не изменится, потому что это количество блоков в разбиении $\mathcal{P}$. Компоненты второго типа (как и их количество) совсем не изменятся. С другой стороны, количество компонент первого типа увеличится из-за предположения об R'. То есть $c(G_b - R) > c(G_b - S)$, что является противоречием. □

Утвержение выше показывает, что можно оптимизировать отдельно обе стороны, пока мы сохраняем общие параметры. Заметим, что разбиение, которое индуцирует $C(G_{b'} - S')$ не является $\mathcal{P}$, но является его строгим уточнением $\mathcal{P} \sqsubset \mathcal{P}'$. (Оно должно быть уточнением, потому что $G_b - S$ содержит больше вершин и рёбер, что может только уменьшить количество компонент связности.) В этому случае также можно показать, что мы можем заменить множество S' на R', которое даёт больше компонент и соответствует тем же параметрам, что и S', то есть $R' \cap X_b = \mathcal{Q}$ и $C(G_b - R')$ индуцирует разбиение $\mathcal{P}$ в X_b. Важно заметить, что $c(G_{b'} - R')$ индуцирует количество блоков в $\mathcal{P}'$ в X_b, что больше, чем количество блоков в $\mathcal{P}$. Это значит, что в $G_{b'}$ ребра из $G_{b''}$ соединяют некоторые компоненты из $C(G_{b'} - R')$. Однако будут связаны те же компоненты, что и в $C(G_{b'} - S')$.

Пусть S' и S'' будут множествами, выбранными из $G_{b'}$ и $G_{b''}$ так, что $S' \cap X_b = S'' \cap X_b = Q$. Пусть $\mathcal{P}'$ и $\mathcal{P}''$ обозначают разбиения индуцированные $C(G_{b'})$ и $C(G_{b''}$ в X_b. Рассмотрим компоненты из $G_b - (S' \cup S'')$. Каждая компонента, которая не пересекает X_b является такой же, как в $C(G_{b'} - S')$ и $C(G_{b''} - S'')$. С другой стороны, некоторые компоненты из $C(G_{b'} - S')$ могут быть соединены рёбрами компонент из $C(G_{b''} - S'')$, и наоборот. Очевидно, что эти компоненты индуцируют разбиение $\mathcal{P}' \sqcup \mathcal{P}''$ в X_b.

Рассуждения выше показывают, что если мы заменим S' на R', то это даст больше компонент в $G_{b'}$, соответствующим пераметрам $\mathcal{Q}$ и $\mathcal{R}'$ таким, что $\mathcal{R}' \sqcup \mathcal{P}'' = \mathcal{P}' \sqcup \mathcal{P}''$, поэтому мы получим больше компонент в $G_b - (R' \cup S'')$. Мы также можем заменить S'' на лучшее R'' таким же способом, пока $\mathcal{R}' \sqcup \mathcal{R}'' = \mathcal{P}' \sqcup \mathcal{P}''$.

$c(G_b - (R' \cup R''))$ может быть посчитано следующим образом. Есть $|\mathcal{R}'|(|\mathcal{R}''|)$ компонент в $G_{b'} - R''(G_{b''} - R'')$, которые пересекают X_b, и $|\mathcal{R}' \sqcup \mathcal{R}''|$ компонент в $G_b - (R' \cup R'')$, которые пересекают X_b. Остальные компоненты не изменяются. В итоге получаем

$$c(G_b - (R' \cup R'')) = c(G_b - R') + c(G_b - R'') - |\mathcal{R}'| - |\mathcal{R}''| + |\mathcal{R}' \sqcup \mathcal{R}''|.$$

Утверждение. Для ноды b 4 типа верно: $Mnc(b, s, \mathcal{Q}, \mathcal{P}) = max\{Mnc(b', s', \mathcal{Q}, \mathcal{P}') + Mnc(b'', s'', \mathcal{Q}, \mathcal{P}'') - |\mathcal{P}'| - |\mathcal{P}''| + |\mathcal{P}' \sqcup \mathcal{P}''|\}$.

При этом должно выполняться следующее:

- $s' + s'' - |\mathcal{Q}| = s$,
- $\mathcal{P}' \sqcup \mathcal{P}'' = \mathcal{P}$,
- $Mnc(b, s', \mathcal{Q}, \mathcal{P}') > 0$,
- $Mnc(b, s'', \mathcal{Q}, \mathcal{P}'') > 0$,

Доказательство. Сначала предположим, что S соответствует параметрам $s, \mathcal{Q}, \mathcal{P}$ и максимизирует $c(G_b - S)$. Пусть $S' = S \cap V_{b'}, S'' = S \cap V_{b''}, s' = |S'|, s'' = |S''|$, и пусть $\mathcal{P}'(\mathcal{P}'')$ будет разбиением, которое $C(G_{b'} - S')(C(G_{b''} - S''))$ индуцирует в X_b. S' соответствует параметрам $b', s', \mathcal{Q}, \mathcal{P}$, и утверждение 1 показывает, что S' оптимально в $G_{b'}$. Таким образом $c(G_{b'} - S') = Mnc(b', s', \mathcal{Q}, \mathcal{P}'$. Аналогично $c(G_{b''} - S'') = Mnc(b'', s'', \mathcal{Q}, \mathcal{P}''$. Мы уже показали, что $c(G_b - S)$ имеет вид, указанный в формуле, и что все условия в правой части формулы выполнены. (Заметим, что если, напимер, $S' = \mathcal{Q}$, то s' и $\mathcal{P}$ определены, то есть и $Mnc(b', s', \mathcal{Q}, \mathcal{P}')$ определено и не равно 0.) Это показывает, что $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ не может быть больше, чем максимум на правой стороне формулы.

Теперь предположим, что S' и S'' выбраны так, чтобы все условия с правой стороны уравнения соблюдались. Тогда $S = S' \cup S''$ будет соответствовать параметрам $b, s, \mathcal{Q}, \mathcal{P}$ и количество компонент связности в $G_b - S$ определяется по формуле в максимуме. Это показывает, что $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ не может быть меньше максимума справа. □

Осталось оценить время работы. Естественная идея заключается в том, что для фиксированных $s, \mathcal{Q}, \mathcal{P}$ можно перебрать все подходящие значения $s', s'', \mathcal{P}', \mathcal{P}''$. Количество возможных пар s', s'' не больше $|V(G)|$; их легко перечислить. Однакао, при фиксированных s', s'' непросто перебрать все подходящие разбиения $\mathcal{P}''$ для данного $\mathcal{P}'$, чтобы $|\mathcal{P}' \sqcup \mathcal{P}'' = \mathcal{P}$ сохранялось.

Для данных s', s'' мы найдём все подходящие комбинации $\mathcal{Q}$ и пар разбиений $\mathcal{P}', \mathcal{P}''$ для $X_b - \mathcal{Q}$. Для разбиений $\mathcal{P}', \mathcal{P}''$ легко вычислить $\mathcal{P}' \sqcup \mathcal{P}''$. Таким образом значение полученное для комбинации $s', s'', \mathcal{P}', \mathcal{P}''$ будет участвовать в расчёте максимума для $Mnc(b, s' + s'' - |\mathcal{Q}|, \mathcal{Q}, \mathcal{P}' \sqcup \mathcal{P}'')$. Поскольку количество возможных пар $\mathcal{Q}, \mathcal{P}'$ и пар $\mathcal{Q}, \mathcal{P}''$ не больше $O(tw(G)^{tw(G)})$, полное время расчёта не больше $O(|V(G)|tw(G)^{2tw(G)})$.

Теорема. Существует алгоритм, который вычисляет жёсткость графа G за время $O(|V(G)|^3 tw(G)^{2tw(G)})$.

Доказательство. Общее число нод в $\mathcal{T}$ $O(|V(G)|)$. Лемма 1 показывает, что общий объём информации в одной ноде $O(|V(G)|tw(G)^{tw(G)})$. Леммы 2, 3, 4, 5 показывают, что вся информация для нод определённого типа может быть посчитана за

- $O(|V(G)|tw(G)^{tw(G)})$ для нод первого типа,
- $O(|V(G)|tw(G)^{tw(G)+1})$ для нод второго типа,
- $O(|V(G)|tw(G)^{2tw(G)})$ для нод третьего типа,
- $O(|V(G)|^2tw(G)^{2tw(G)})$ для нод четвёртого типа.

Сумма значений для каждой ноды будет не больше $O(|V(G)|^3tw(G)^{2tw(G)})$. □

Следствие. Жёсткость может быть найдена за полиномиальное время для графов с ограниченной древесной шириной.

ГЛАВА 5
РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА ЖЁСТКОСТИ ДЛЯ ГРАФОВ С ОГРАНИЧЕННОЙ ДРЕВЕСНОЙ ШИРИНОЙ

В данной секции будут приведены детали реализации алгоритма для определения жёсткости графа. Данный алгоритм принимает на вход граф $G = (V, E)$, а также очень красивую древесную декомпозицию $\mathcal{T} = (X, T)$.

Установим некоторые ограничения на входные данные.

- $tw(G) \leq 4$

  Константа в данном алгоритме сильно растёт вместе с древесной шириной графа. При $tw(G) = 5$ константа будет равна 5^{10}, что уже недопустимо для реальных вычислений.

- $|V(T)| \leq 100$

  Время работы алгоритма имеет кубическую зависимость от количества вершин графа. Также при расчёте количества вычислительных опреаций стоит учитывать константу, которая зависит от древесной ширины графа (её значение при $tw(G) = 4$ может достигать $4^8 = 65536$).

В дальнейшем данные ограничения помогут сделать удобное хеширование для разбиений.

5.1 Преобразования разбиений

В ходе работы алгоритма нам придётся выполнять некоторые операции над разбиениями:

1. Для множества вершин найти все его разбиения.

2. Для множества вершин найти все его подмножества (это пригодится для выделения подмножества $\mathcal{Q}$).

3. Найти все разбиения $\mathcal{P}'$ такие, что $\mathcal{P}' - f = \mathcal{P}$, где $f \in V(G)$ покрывается данным нам разбиением $\mathcal{P}$. Данная операция пригодится для вычисления значений ноды второго типа.

4. Найти все i-уточнения для разбиения $\mathcal{P}$ ($\mathcal{P}$ покрывает i). Данная операция пригодится для вычисления значений ноды третьего типа.

5. Найти объединие двух разбиений $\mathcal{P} \sqcup \mathcal{P}'$. Данная операция пригодится для вычисления значений ноды четвёртого типа.

Первые две операции нетрудно реализовать рекурсией с полным перебором.

Рассмотрим шаг рекурсии для первой операции. У нас есть разбиение первых $k-1$ элементов, состоящее из $p \leq k-1$ подмножеств. Поочерёдно добавляем элемент с номером k в каждое из p подмножеств и делаем рекурсивный вызов для $k+1$. Также нужно сделать один рекурсивный вызов для создания нового множества с элементом под номером k.

Рекурсия для второй операции куда проще. На каждом шаге делаем два рекурсивных вызова: добавляем текущий элемент в текущее множество или нет.

Чтобы избежать лишнего копирования во все рекурсивные вызовы, можно передавать все контейнеры по ссылке, но тогда после каждого рекурсивного вызова следует приводить текущее множество в изначальное состояние.

В линтинге 5.1 приведён пример кода на C++ для выделения всвех разбиений.

```
typedef long long ll;

void RecPartitions(ll index, vector<set<ll>> &cur, vector<vector<set<ll>>> &res,
     const vector<ll> &vec)
{
     if (index == vec.size())
     {
          res.push_back(cur);
          return;
     }
     for (ll i = 0; i < cur.size(); ++i)
     {
          cur[i].insert(vec[index]);
          RecPartitions(index + 1, cur, res, vec);
          cur[i].erase(vec[index]);
     }
     cur.push_back({vec[index]});
     RecPartitions(index + 1, cur, res, vec);
     cur.pop_back();
}

vector<vector<set<ll>>> utils::GetAllPartitions(const vector<ll> &vec)
{
     vector<vector<set<ll>>> res;
     vector<set<ll>> cur;
     RecPartitions(0, cur, res, vec);
     return res;
}
```

Листинг 5.1: Поиск всех разбиений для множества

Третья опрация по своей сути является последним шагом рекурсии для первой операции.

Рассмотрим алгоритм выполнения четвёртой операции. Разбиваем все вершины, которые покрываются данным разбиением на два подмножества: в первом подмножестве A будут соседи вершины i (хотя бы одна вершина из этого подмножества должна быть в каждом подмножестве в итоговом

разбиении), во втором подмножестве B – все остальные вершины, кроме i. Первым шагом находим все разбиения для A, как делалось в первой операции. Теперь надо распределить элементы из B по подмножествам каждого из получившихся разбиений. Это делается с помощью рекурсии аналогично первой операции, но в этом случае элементы из B не могут создавать свои подмножества.

Для поиска объединения разбиений $\mathcal{P} \sqcup \mathcal{P}'$ (выполнения 5 операции) используем другой алгоритм. Будем строить граф $H(V_H, E_H)$. V_H будет множеством всех вершин, которые покрываются вершинами из $\mathcal{P}(\mathcal{P}')$. Для каждого $A_i \in \mathcal{P}$ добавим в граф следующие рёбра: возьмём любую вершину из $f \in A_i$ и добавим в E_H ребра от f до каждой из остальных вершин в A_i. Потом проведём аналогичную операцию с $\mathcal{P}'$. В получившемся графе H выделим компоненты связности. Множество подмножеств вершин, которые находятся в одной и той же комопненте связности, и будет искомым $\mathcal{P} \sqcup \mathcal{P}'$.

Операции добавления рёбер и поиск в ширину для выделения компонент связности в H работают за $O(|V_H|)$, что равно количеству элементов, которые покрываются разбиениями.

5.2 Хэш для разбиений

В ходе работы алгоритма достаточно часто возникает необходимость искать инфромацию, используя разбиение ($\mathcal{P}$) в качестве ключа. Также в качестве ключа будет использоваться множество ($\mathcal{Q}$). Поэтому надо иметь механизм быстрого поиска по этим двум структурам.

При построении хэша будем использовать тот факт, что $tw(G) \leq 4$, а это значит, что любое множество, которое мы будем хешировать, имеет не более 5 элементов. Также заметим, что элементы данных множеств (номера вершин) не превышают 99 (всего вершин не больше 100 при индексации с 0).

Хэш для множества будем строить следующим образом. Сортируем данное множество (или читаем из упорядоченного множества std::set в C++). Результат изначально равен нулю. Перебираем элементы в порядке убывания. На каждой итерации увеличиваем результат в 100 раз и добавляем текущий элемент. Получаем число, которое по сути является конкатенацией всех чисел множества в порядке убывания. Стоит заметить, что при переборе элементов по возрастанию хэши для множеств, $A(0 \notin A)$ и $A \cup \{0\}$ получились одинаковыми. Хэш для пустого множества будет равен -1, чтобы было различие с множеством $\{0\}$.

Заметим, что из такого хэша совсем нетрудно восстаносить само множество. Для этого достаточно добавлять во множество остаток от деления хэша на 100 и делить хэш на 100 без остатка, пока значение хэша больше нуля.

Максимальным значением хэша будет 9998979695, что без проблем помещается в целочисленный тип (long long).

Теперь построим хэш для разбиений.

Сначала построим хэш для каждого множества в разбиении. Далее сортируем эти хэши (теперь порядок множеств в разбиении неважен). Далее можно похожим образом сконкатенировать данные числа, чтобы получить хэш для разбиения, но тогда в некоторых случаях будет стёрта граница между подмножествами. Поэтому можно изменить тип хэша на строку и ставить разделители между хэшами множеств. Таким образом получим следующую конструкцию $hash1\_hash2\_...\_hash3$.

При этом нетрудно видеть, что длина хэша будет не больше 14. Действительно, число разделителей равно числу подмножеств минус 1, а каждое число добавляет не более двух символов в строку.

Также из данного хэша нерудно восстановить само разбиение: достаточно разбить строку по разделителям и для каждой получившейся подстроки найти (предварительно переведя её в число) соответствующее множество.

Возможность быстрого перевода хэша в разбиение (множество) позволяет более экономно хранить информацию для каждой ноды древесной декомпозиции.

5.3 Инициализация данных

Для каждой ноды $b \in T$ надо провести инициализацию данных. Для этого создадим структуру NodeInfo, в которой для каждого возможного размера подмножества s будут храниться все возможные подмножества X_b и для каждого такого подмножества будет храниться отображение из разбиения оставшихся вершин в оптимальный ответ для получившихся $s, \mathcal{Q}, \mathcal{S}$. Изначально оптимальным ответом будет 0. На данном этапе не учитывается соответствие пары $\mathcal{Q}, \mathcal{P}$ с размером множества удаляемых вершин s и структурой графа. Невалидные значения при работе динамического программирования так и останутся нулевыми и не повлияют на конечный ответ.

Заметим, что для каждой ноды b нужно знать количество вершин в V_b. Это можно сделать обходом дерева в глубину, сохраняя для каждой ноды b' множество вершин в её поддереве $V_{b'}$. Данный обход можно сделать за $O(|V(G)|^2)$, что не повлияет на общую асимтотику алгоритма.

5.4 Динамическое программирование

Для листьевой ноды b все значения $Mnc(b, s, \mathcal{Q}, \mathcal{P})$, которых всего 2, вычисляются тривиальным образом.

Для нод второго и третьего типа алгоритм имеет схожую структуру. Будем отдельно перебирать все $s, \mathcal{Q}, \mathcal{P}$. Для $\mathcal{P}$ вычислим все подходящие разбиения, значения для которых будут просматриваться в ребёнке b'. При этом

надо учесть, что храним мы только хэши разбиений, поэтому предварительно строим из хэша искомое разбиение, а потом либо добавляем новую вершину f в разбиение (для ноды второго типа), либо ищем все i-уточнения (для ноды третьего типа). Далее берём максимум по полученным значениям.

Для нод четвёртого типа алгоритм отличается. Мы ищем значения $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ для каждого s отдельно. Также отдельно перебираем пары s', s'' такие, что $s' \leq s, s'' \leq s, s' + s'' - |\mathcal{Q}| = s$. Теперь же мы перебираем не все подходящие значения для $\mathcal{Q}, \mathcal{P}$, а для всех комбинаций, которые подходят по s', s'' модифицируем подходящее $Mnc(b, s, \mathcal{Q}, \mathcal{P})$ с использованием операции объединения разбиений.

Для получения ответа будем перебирать все валидные значения для корневой вершины. Искать наилучшее значение будем не через сравнение действительных чисел (double), а через сравнение рациональных чисел, что будет более логично, учитывая, что ответ также представим рациональным числом. Вывод же результата будет как вещественным (что бывает лучше для практических целей), так и рациональным, что является более точным.

5.5 Время выполнения алгоритма

Приведём временные результаты работы алгоритма в таблице 5.1.

n	10	20	50	80	100
tw(G)					
2	0.0002	0.00642	0.1578	0.7007	1.3082
3	0.0051	0.06386	0.9774	4.0539	7.1549
4	0.4103	3.3245	50.8473	255.3947	583.4243

Таблица 5.1 — время работы алгоритма

Всё время в таблице даётся в секундах.

Заметим, что для $tw(G) = 1$ результатов в таблице нет. Причиной этому служит то, что древесную ширину равную 1 имеют только деревья, для которых жёсткость определяется тривиальным образом (см. Утверждение 1).

Как было показано выше, данный алгоритм работает за $O(|V(G)|^3 tw(G)^{2tw(G)})$ время. Следует заметить, что перебор всех возможных вариантов $s, \mathcal{Q}, \mathcal{P}$ хоть и заставит сделать больше действий в рамках динамического программирования, но при этом избавит от необходимости проверок валидности для каждого случая.

Также следует учесть работу с хэшом. Хранение множеств и разбиений в сжатом виде существенно экономит память, но при это создаёт необходимость перед работой с разбиением (множеством) его рехешировать. При этом хэши хранить необходимо для быстрого поиска по словарю. Получается есть два

варианта: хранить разбиения вместе с хешами, либо хранить только хэши, но при этом каждый раз при необходимости преобразовывать их в разбиения (множества). При этом можно показать, что достаточно восстановить разбиение только один раз. Для второго и третьего типа нод это делается один раз при расчёте $Mnc(b, s, \mathcal{Q}, \mathcal{P})$, для конкретного $\mathcal{P}$. В нодах 4 типа можно делать предрасчёт всех разбиений для конкретных $s, \mathcal{Q}$, что не сильно увеличит общую память алгоритма, особенно если учесть, что в разбиениях и множествах содержится не более 5 элементов ($tw(G) \leq 4$).

Также заметим, что в данной реализации для мноежств используется std::set, который имеет логарифмическое время поиска по ключу. Однако размеры множеств не превышают 5, что позволяет считать время поиска константным. Использование std::set удобно для построения хэша: элементы при обходе сразу идут в отсортированном порядке.

ЗАКЛЮЧЕНИЕ

В ходе работы:

1. Была рассмотрена задача распознавания жёсткости графа.
2. Рассмотрена связь жёсткости с другими параметрами графа.
3. Были рассмотрены общие сведения о сложности распознавания жёсткости графа для различных классов графов и приведены классы, для которых известны полиномиальные алгоритмы нахождения жёсткости.
4. Представлены классы графов, для которых неизвестна сложность распознавания жёсткости.
5. Изучена связь гамильтоновости и жёсткости графа. Приведены классы графов, для которых определённая жёсткость является достаточным условием гамильтоновости.
6. Разобран метод построения жёстких негамильтоновых графов путём клонирования небольших графов.
7. Приведены некоторые доказательства теорем, связанных с жёсткостью в регулярных графах.
8. Рассмотрен алгоритм поиска жёсткости для графов с ограниченной древесной шириной.
9. Реализован алгоритм поиска жёсткости для графов с ограниченной древесной шириной при наличии древесной декомпозиции с минимально возможной шириной. Приведены некоторые делали реализации данного алгоритма, и показаны временные результаты его работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bauer, D. The complexity of recognizing tough cubic graphs / D. Bauer. — Department of Mathematical Sciences Stevens Institute of Technology, Hoboken, NJ 07030, U.S.A., 1997. — 10 с.

2. Bauer, D. The Complexity of Toughness in Regular Graphs / D. Bauer. — Department of Mathematical Sciences Stevens Institute of Technology, Hoboken, NJ 07030, U.S.A., 1998. — 12 с.

3. Bodlaender, L. A linear time algorithm for finding tree-decompositions of small treewidth / L. Bodlaender. — Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, the Netherlands, 1992. — 20 с.

4. Bodlaender, L. The Fine Details of Fast Dynamic Programming over Tree Decompositions / L. Bodlaender, P. Bonsma, D. Lokshtanov. — Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, the Netherlands, 2013. — 15 с.

5. Broersma, H. How tough is toughness? / H. Broersma. — Faculty of EEMCS, University of Twente, The Netherlands, 2019. — 25 с.

6. Gyula, Y. An Efficient Algorithm to Compute the Toughness in Graphs with Bounded Treewidth / Gyula Y. Katona, Humara Khan. — Department of Computer Science and Information Theory, Budapest University of technology and Economics, Budapest, Hungary, 2024. — 11 с.

7. Gyula, Y. Strengthen some complexity results on toughness of graphs / Gyula Y. Katona, Kitti Varga. — Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Hungary, 2019. — 17 с.

8. Kemp, T. An algorithmic approach to a conjecture of Chvátal on toughness and hamiltonicity of graphs / T. Kemp. — Faculty of Electrical Engineering, Mathematics and Computer Science University of Twente, The Netherlands, 2024. — 78 с.

9. Wikipedia: Split graph [Электронный ресурс]. — Режим доступа: https://en.wikipedia.org/wiki/Split_graph. — Дата доступа: 18.05.2025.