МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Связь гамильтоновости и плотности отношения смежности графа

НАУЧНО-ИССЛОДОВАТЕЛЬСКАЯ РАБОТА

студент 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Смирнова Данилы Антоновича

Научный руководитель
д.ф.-м.н., доцент _____ М. Б. Абросимов
подпись, дата

Заведующий кафедрой
д.ф.-м.н., доцент _____ М. Б. Абросимов
подпись, дата

Саратов 2019

Содержание

Определения

Множество — совокупность попарно различных объектов, называемых элементами этого множества.

Под объединением двух множеств A и B подразумевается множество $A \cup B$, состоящее из элементов, принадлежащих хотя бы одному из множеств A или B.

Под пересечением двух множеств A и B подразумевается множество $A \cap B$, состоящее из элементов, принадлежащих обоим множествам A и B.

Под разностью двух множеств A и B подразумевается множество $A \setminus B$, состоящее из элементов, принадлежащих множеству A, которых нет во множестве B.

Под мощностью множества A подразумевается число его элементов, обозначаемое как $|A|$.

Декартово произведение двух множеств — множество всевозможных пар элементов, таких что первый элемент пары принадлежит первому множеству, а второй элемент — второму множеству.

Бинарное отношение — подмножество декартова произведения рассматриваемого множества на себя.

Пусть φ — бинарное отношение на множестве A.

Отношение φ на множестве A называется антирефлексивным, если $(\forall a \in A)((a,a) \notin \varphi)$.

Отношение φ на множестве A называется симметричным, если $(\forall a,b \in A)((a,b) \in \varphi \Rightarrow (b,a) \in \varphi)$.

Отношение между элементами одного и того же множества A называется отношением на множестве A.

Отношение φ на множестве A называется универсальным, если оно совпадает с декартовым произведением $A \times A$.

Ориентированный граф — это пара G=(V,E), где V — конечное непустое множество вершин, E — отношением на множестве V, называемое отношением

смежности или множеством ребер графа.

Неориентированный граф — граф, отношение смежности которого антирефлексивно и симметрично.

Пусть далее по всему тексту работы обозначение n будет определять число вершин в графе, а m — число ребер в графе, если нет другого описания для значений m или n внутри раздела или подраздела.

Плотное бинарное отношение — такое бинарное отношение φ на множестве A, что $(\forall a,c\in A)(a,c)\in\varphi\Rightarrow(\exists b\in A)((a,b)\in\varphi\wedge(b,c)\in\varphi)$.

Две вершины в графе называются смежными, если существует ребро, концами которого эти две вершины являются.

Путь в графе — последовательность вершин $v_1, v_2, \ldots, v_x$, такая что каждая вершина v_i смежна с вершиной v_{i+1} для всех $i=\overline{1,x-1}$.

Пусть задан некоторый путь в графе. Если все вершины пути попарно различны, такой путь называется простым.

Если начальная и конечная вершина пути в графе совпадают, такой путь называется циклом.

Под длиной цикла понимается число вершин этого цикла, причем первая и последняя вершина считаются за одну.

Если кроме начальной и конечной вершин все остальные вершины в цикле попарно различны, то такой цикл называется простым.

Граф называется гамильтоновым, если содержит хотя бы один цикл длины n.

Путь называется гамильтоновым, если состоит из n вершин.

Цикл называется гамильтоновым, если состоит из n вершин.

Под степенью вершины v подразумевается число смежных с вершиной v вершин, обозначаемое как deg(v).

Связный граф — граф, в котором между каждой парой вершин существует хотя бы один путь.

Панциклический граф — граф, в котором есть хотя бы один цикл длины t

для всех $t=\overline{3,n}$.

Немым изображением графа называется его изображение без меток вершин.

Ориентированный граф называется полным, если его отношение смежности универсально.

Символы перевода строк — управляющие символы, имеющие численные значения машинных кодов. Для операционной системы Windows это пара символов с кодами 13,10, для операционных систем Unix, Linux, MacOS только один код 10.

Скрипт — набор системных команд, обрабатываемых операционной системой. В общем случае под скриптом не обязательно подразумевается системный набор команд, но в данной работе используются только системные скрипты.

Конкурентный доступ к данным — ситуация, возникающая, когда к файлу запрашивает доступ более чем одна программа одновременно. Кроме файлов в качестве данных могут выступать и данные, передаваемые от одной программе другой напрямую.

Модули программы — составные части программы, которые созданы, чтобы в будущем быть объединенными в одну общую программу.

Потоки исполнения программы — подпрограммы программы, выполняющиеся отдельно друг от друга и от основной программы.

Введение

Теория графов в современном мире нашла применение в таких областях как компьютерные сети, построение оптимальных маршрутов, строительство железных дорог и мостов, создание схем электронно-вычислительных машин, схем авиалиний, карт звездного неба, построение генеалогического древа.

В связи с этим, важно изучить существующие научные результаты теории графов и, возможно, создать новые.

В данной научной работе было рассмотрено большое число характеристик графов с целью выявить возможную связь каких-либо характеристик.

Основным результатом является гипотеза о панциклических графах.

1 Теоретическая часть

В данном разделе находится доказательство, используемой в работе теоремы Дирака.

1.1 Лемма о длине цикла

Пусть G — произвольный неориентированный граф и δ — минимальная степень его вершин.

Если $\delta \geq 2$, то в графе G существует цикл C длиной $t \geq \delta + 1$.

Доказательство.

Пусть вершины в графе нумеруются с нуля. Рассмотрим путь P максимальной длины из s вершин: $v_0, v_1, \ldots, v_s$. Все вершины, смежные с вершиной v_0, лежат на P. Обозначим за k максимальный номер вершины, смежной с вершиной v_0. Тогда $\delta \leq \deg(v_0) \leq k$. Цикл C состоит из вершин $v_0, v_1, \ldots, v_k, v_0$ и имеет длину $t = k + 1 \geq \delta + 1$. Что и требовалось доказать.

1.2 Теорема Дирака

Пусть G — неориентированный граф и δ — минимальная степень его вершин.

Если $n \geq 3$ и $\delta \geq \frac{n}{2}$, то G — гамильтонов граф.

Доказательство.

Пусть C — это цикл наибольшей длины в графе G. По лемме его длина $t \geq \delta + 1$. Если C гамильтонов, то теорема доказана. Предположим обратное, то есть $G \setminus C \neq \varnothing$. Рассмотрим путь P из вершин x, …, y, такой что $P \cap C = \{y\}$ наибольшей длины m. Заметим, что по условию $\delta \geq \frac{n}{2}$, а значит $\delta \geq n - \delta > n - t = |V(G \setminus C)|$. Поэтому каждая вершина из $G \setminus C$ смежна с некоторыми вершинами из C.

Заметим, что вершина x не может быть смежна:

1) с вершинами из C, расстояние от которых до y (по циклу C) не превышает m. Действительно, пусть вершина $v \in C$ и расстояние от v до y по циклу меньше либо равно m. Тогда этот участок цикла можно заменить на путь (v,x,P,y), длина которого равна m+1. Тогда образуется цикл большей длины, что противоречит предположению о максимальности цикла C.
2) двум смежным вершинам на C. Пусть $u, v \in C$ и $\{(u,v),(u,x),(x,v)\} \in E$. Тогда заменив ребро (u,v) на последовательность вершин (u,x,v), увеличим длину цикла на 1.
3) вершинам из $G \setminus (C \cup P)$, поскольку P максимальный.

Получаем $\deg(x) \le m + \frac{t-2m}{2} = \frac{t}{2} < \frac{n}{2} \le \delta$. Противоречие. Что и требовалось доказать.

Будем считать, что граф удовлетворяет теореме Дирака, если минимальная степень его вершин не менее $\frac{n}{2}$.

2 Практическая часть

В данном разделе описаны файлы и подпрограммы, участвующие в функционировании программы. Также необходимо отметить, что обрабатываемые программой графы генерируются комплексом программ Nauty, распространяемой под лицензией Apache License 2.0.

2.1 Файл конфигурации

Рассмотрим одну из составляющих созданной программы. Программа считывает управляющие данные из файла конфигурации, который задает типы графов и число вершин этих графов.

В файле конфигурации перечислены несколько типов обрабатываемых программой графов. В каждой строке идет сначала название типа, например, connected — связные графы, затем максимальное число обрабатываемых вершин, например 11.

Если указано отрицательное число основная программа пропустит такую строку. Это сделано с целью, если, например, нужно оставить только несколько основных типов графов, или какие-то просто не нужно обрабатывать(например, строка biconnected -5 обработана не будет). Такие строки не считаются программой как ошибка и просто пропускаются. Кроме того это удобно — достаточно всего лишь поставить минус перед числом вершин, а когда такой тип снова потребуется — убрать, и программа будет обрабатывать этот тип как и раньше.

Если считан тип графа, который программе неизвестен, на экран будет выдана ошибка и программа завершится с кодом ошибки -1.

2.2 Формат выходных данных

Результаты работы основной программы сохраняются в таблицу с расширением ”.csv”. Это расширение является самым простым способом представления табличных данных в текстовом виде. Табличный редактор сможет показать такой файл в виде таблицы, если отделять элементы одной строки запятыми, а строки — переводами строк.

В последствии, по нахождении гипотезы, вывод данных в табличном виде был отключен, а оставлен только консольный вывод, на который подавались сообщения о возможных ошибках при проверке гипотезы о панциклических графах.

2.3 Главный управляющий скрипт

Главный управляющий скрипт состоит из нескольких команд. Если в ходе последовательного выполнения хотя бы одна из команд завершается с ошибкой, выполнение прекращается и на экран выдается сообщение об ошибке.

Этапы выполнения:

1) Компиляция основной программы.
2) Запуск основной программы.
3) Открытие результатов в табличном редакторе.

После успешного выполнения всех этапов автоматически запускается табличный редактор с результатами работы подпрограмм.

2.4 Компиляция основной программы

При компиляции основной программы выполняется всего одна команда — вызов компилятора g++. Результатом является бинарный файл программы. Название файла исходного кода подается в качестве аргумента вызова g++.

3 Основная программа

Первым этапом выполнения основной программы является очистка предыдущих результатов. Для этого из программы запускается скрипт, позволяющий очистить соответствующую папку с таблицами результатов.

После этого запоминается текущее время системных часов. Это сделано с целью отслеживания общего времени выполнения программы.

Далее считывается файл конфигурации.

Для каждого типа графов из файла конфигурации рассматриваются все графы с числом вершин с 3 по n, где n — максимальное число вершин данного типа.

Далее происходит формирование заголовка таблицы. В таблицу добавляется разметка по числу вершин и названия исследуемых свойств. Например, количество графов с циклом указанной длины обозначены по английски cc, это обозначение заменяет первый индекс в разметке для экономии места в таблице.

n	cnt	h	e	cc	2	3	4	5	co	2	3	4	5	ch	2	3	4	5
3	2	1	1	0	0	1	T	T	0	1	1	T	T	0	0	1	T	T
4	6	3	1	0	0	1	0	T	0	3	2	1	T	0	1	1	1	T
5	21	8	4	0	0	4	0	0	0	5	12	3	1	0	0	5	2	1

Таблица 1 - пример выходных данных.

После того, как заголовок таблицы сформирован, запускается основной цикл программы, в котором из файла конфигурации считаются тип графа и максимальное число вершин n (например, в таблице 1 максимальное число вершин 5). Для всех возможных значений числа вершин с 3 по n выполняется следующее.

1) Формирование команды компиляции одного из модулей программы, например для подпрограммы подсчета при текущем числе вершин 5 команда имеет вид ”g++ -c -std=c++11 -DT=4 -DN=5 -DNMAX=5 calc/src/prog.cpp -o calc/o/prog.o”. Аргументы команды из примера соответственно следующие:
   1.1) Сам компилятор.
   1.2) Указание компилятору требования только скомпилировать, но не объединять модули программы в одну программу. Это будет сделано позднее.
   1.3) Указание компилятору версии языка программирования.
   1.4) Число потоков исполнения.
   1.5) Число вершин, с которым будет работать подпрограмма подсчета.
   1.6) Максимальное число вершин.
   1.7) Имя файла исходного текста.
   1.8) Флаг выходного файла.
   1.9) Имя выходного бинарного файла модуля.

   Точно таким же образом компилируются другие модули программы: модули для проверки графа на гамильтоновость, наличия цикла заданной длины, плотности отношения смежности и так далее.
2) Объединение модулей в одну программу подсчета. Это выполняется командой ”g++ -lpthread -o calc/bin/prog calc/o/hamilton.o calc/o/cycle.o calc/o/prog.o calc/o/tr.o calc/o/cycdp.o”. Аргументы команды из примера соответственно следующие:
   2.1) Сам компилятор.
   2.2) Флаг многопоточности выходной программы.
   2.3) Флаг выходной программы.
   2.4) Имя выходной программы.
   2.5) Имена модулей, в примере их 5.

3) Запуск программы подсчета. Например, для связных графов команда будет иметь вид ”./geng -c 4 | ./showg -e | ./calc/bin/prog calc/csv/connected.csv”. Аргументы команды из примера соответственно следующие:
   3.1) Генератор графов.
   3.2) Флаги генерации, задающие тип генерируемых графов.
   3.3) Число вершин каждого генерируемого графа(все сгенерированные графы будут иметь данное число вершин).
   3.4) Преобразователь формата графов.
   3.5) Флаги преобразователя формата графов.
   3.6) Бинарный файл программы подсчета.
   3.7) Имя выходной таблицы.

Элемент ” | ” в опциях запуска скрипта является сообщением командному интерпретатору о необходимости направления вывода одной программы на вход другой программы. Например, в команде ”./a | ./b” вывод программы a перенаправляется на ввод программы b.

После запуска сформированных команд компиляции и генерации на экран выводится время, суммарно затраченное на компиляцию и выполнение подпрограммы подсчета. Это позволяет сориентироваться и выбрать число вершин, при котором программу будет легко тестировать(обычно до 9 вершин). На большем числе вершин программа уже запускается для проверки гипотез.

В завершение основной программы выводится суммарное затраченное время всех команд.

4 Программа подсчета

Программа подсчета, как было описано выше, компилируются для каждого числа вершин в отдельности. Это позволяет ускорить ее на меньшем числе вершин, так как процессе отладки было установлено, что рассчитанная на большее, чем необходимо для текущего шага, число вершин программа подсчета работала медленнее, чем программа, скомпилированная под точное число вершин для данного шага.

Разбиение программы на модули также ускоряет программу более чем вдвое, так как сказываются более короткие переходы от одного участка модуля до другого, вместо длинных переходов по всей программе. Переходы подразумеваются на уровне языка ассемблера, в который компилируются программа.

Рассмотрим составляющие программы подсчета.

4.1 Текстовый вывод графа

Данная функция использовалась только для отладки программы и поиска ошибок. Функция выводит список ребер графа на экран.

4.2 Функция подсчета

Это и есть основная по нагрузке функция программы подсчета. Она представляет из себя цикл, прерываемый только по окончании входных данных.

Рассмотрим основные шаги функции:

1) Блокировка доступа. На этом шаге доступ к разделяемым данным блокируется, так как программа работает в нескольких потоках, поэтому конкурентное считывание может привести к ошибкам.
2) Пропуск строк. Используемый программой подсчета генератор графов выводит дополнительную информацию, не используемую в подсчетах. Это текстовое описание текущего сгенерированного графа, например ”Graph 5, order 5” означает, что пятый по счету граф сгенерирован и его порядок равен пяти. Также генератор выдает одну пустую строку перед текстовым описанием графа.

3) В случае, если входные данные закончились, цикл подсчета завершается. Это происходит если не удалось выполнить шаг 3, то есть на вход не было подано ни одной строки данных.
4) Считывание числа вершин и числа ребер графа и самого графа, заданного списком ребер. Этот список ребер считывается и запоминается.
5) Освобождение доступа. Разделяемый ресурс(входной поток данных) освобождается текущим потоком. Время доступа к разделяемому ресурсу сведено к минимуму, так как это ускоряет вычисления. Основная нагрузка уходит на вычисления, а не на ввод данных.
6) Очистка списков смежностей вершин. В списке смежностей для каждой вершины будут храниться вершины, смежные с данной вершиной.
7) Заполнение списков смежностей и подсчет степеней вершин. Для каждой вершины заполняется список смежных с ней вершин. Также для каждой вершины хранится ее степень.
8) Проверка графа на гамильтоновость при помощи теоремы Дирака.
9) Проверка отношения смежности графа на плотность.
10) Проверка гипотезы о панциклических графах.

4.3 Функция проверки отношения смежности графа на плотность

Рассмотрим все ребра графа. Зафиксируем ребро (u,v). Если не существует вершины t, которая смежна с обоим вершинам u,v, то отношение смежности графа не является плотным. Вершину t достаточно перебрать среди всех вершин, смежных вершине u. Общая сложность функции равна O(nm). Это справедливо, так как в случае полного графа каждая вершина имеет n-1 смежную, а число ребер равно m.

4.4 Подпрограмма проверки графа на гамильтоновость

В модуле проверки на гамильтоновость в приложениях также находится и проверка, проверяющая, выполняется ли теорема Дирака для графа. Функция проверки обозначена как dirac_bad и проверяет, выполняется ли условие, описанное в теореме. Сложность функции dirac_bad составляет O(n).

В данном подразделе под m подразумевается двоичное представление некоторого множества, а не числа ребер графа.

Первое, что требуется сделать — проверить выполнения теоремы Дирака гамильтоновости графа. Если минимальная степень вершин графа не менее $\frac{n}{2}$, следовательно граф содержит гамильтонов цикл. Если это не так, выполним проверку при помощи динамического программирования.

Предположим, что в графе есть гамильтонов цикл, заканчивающийся в вершине h. Положим значения $D_{i,m}=0$ для всех $i=\overline{0,n-1}$ и $m=\overline{0,2^n-1}$ и зададим таким образом некоторое множество вершин. В двоичном представлении это означает, что в текущем множестве вершин m есть все вершины графа. Будем последовательно рассматривать все вершины, смежные с вершиной h. Пусть одна из таких вершин — это вершина j. Тогда удалим из текущего множества m вершину i и перейдем в вершину j рекурсивно. В двоичном виде множества представлены следующим образом: например при n=3 и текущем множестве {1,3} двоичное представление имеет вид 101. Тогда чтобы удалить вершину с номером i из множества, достаточно просто вычесть из его двоичного представления число 2^i.

Рекурсивная функция hamilton(m,i), проверяющая гамильтоновость графа, имеет следующий вид:

1) Если $D_{i,m}\neq 0$ завершить выполнение функции.
2) Положить $D_{i,m}=1$.

3) Если $m=2^i$, то выполнить следующее: если вершина h смежна с вершиной i, то завершить выполнение функции с результатом: найден гамильтонов цикл; иначе завершить выполнение функции с результатом: найден гамильтонов путь.
4) Рассмотрим номера всех вершин, смежных с вершиной i. Пусть одна из них — вершина с номером j. Тогда, если $m \& 2^i \neq 0$ (где операция & подразумевает двоичную операцию И), то выполнить следующее:
   4.1) Вызвать функцию $\text{hamilton}(m-2^i, j)$.
   4.2) Если результат вызова — гамильтонов цикл, завершить выполнение функции с результатом: найден гамильтонов цикл.
   4.3) Если результат вызова — гамильтонов путь, запомнить, что на этапе 4.1 был найден хотя бы 1 гамильтонов путь.
5) Если на этапе 4.1 был найден хотя бы 1 гамильтонов путь, завершить выполнение функции с результатом: найден гамильтонов путь. Иначе завершить выполнение функции с результатом: ничего не найдено.

Для того, чтобы узнать, имеет ли граф хотя бы один гамильтонов цикл или хотя бы один гамильтонов путь, необходимо запустить функцию $\text{hamilton}(2^n-1, i)$ для всех $i=\overline{0, n-1}$. Если был найден хотя бы один гамильтонов цикл — значит граф гамильтонов. Если цикл не был найден, но найден хотя бы один гамильтонов путь — значит граф содержит гамильтонов путь. Если не был найден и гамильтонов путь — значит граф не содержит ни гамильтонова пути ни гамильтонова цикла. Общая сложность функции $O((2^n)\cdot n^2)$. Это справедливо, так как заполняема таблица для динамического программирования имеет размер $(2^n)\cdot n$, а в случае полного графа каждая вершина имеет n-1 смежную.

4.5 Подпрограмма нахождения множества длин простых циклов графа

Рассмотрим рекурсивную функцию lengths(i,t):

1) Установим метку вершины i в значение t.
2) Рассмотрим все смежные i вершины. Пусть j — одна из них. Тогда:
   2.1) Если она не имеет метки, вызовем рекурсивно функцию lengths(j,t+1);
   2.2) Иначе, вычислим число length=t-to_label+1, где to_label — метка вершины j. Если length не менее 3, значит в графе есть цикл длины length.
3) Снимем метку вершины i.

Для того, чтобы найти множество всех длин циклов графа достаточно запустить функцию lengths(0,1), так как вершины в программе нумеруются с нуля и метка не может быть нулевой, так как в программе нулевая метка принимается как отсутствие метки. Общая сложность функции O((n-1)!). Это справедливо, так как в случае полного графа первая рассматриваемая вершина имеет n-1 смежную вершину, вторая рассматриваемая вершина имеет n-2 смежную вершину, и так далее.

Данная функция использовалась для поисков ошибок при нахождении цикла заданной длины следующей функцией.

4.6 Функция проверки существования цикла заданной длины

Если добавить в проверку графа на гамильтоновость условие, при котором функция будет завершаться не когда осталась одна вершина в рассматриваемом множестве, а когда осталось n-t вершин, где t — длина цикла, то получится именно функция проверки существования цикла заданной длины. Общая сложность функции $O\left(n\left(C_n^1+C_n^2+...+C_n^t\right)\right)$, так как число возможных подмножеств из 1 элемента C_n^1, число возможных подмножеств из 2 элементов C_n^2, и так далее до числа возможных подмножеств из t элементов C_n^t. Это справедливо, так как алгоритм последовательно рассматривает все

подмножества, пока их мощность не станет равна t. Как только мощность множества становится равной t алгоритм проверяет совпадение последней рассмотренной вершины со стартовой. В случае совпадения результат — в данном графе есть некоторый цикл длины t. Сам цикл в точности находить не требуется, достаточно установить его наличие.

5 Гипотеза о панциклических графах

В ходе работы была получена следующая гипотеза: все графы, удовлетворяющие теореме Дирака и имеющие плотное отношение смежности — панциклические.

Проверка проводилась на всех связных графах с 3 до 11 вершин включительно.

Заключение

В данной работе были исследованы различные характеристики графов. В результате сформирована гипотеза о панциклических графах, являющаяся главным результатом работы. Описаны все методы, используемые при проверках графов.

Список использованных источников

1) Теорема Дирака [Электронный ресурс]: [сайт]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Теорема_Дирака (дата обращения: 05.05.2020). Загл. с экрана. Яз. рус.
2) Глоссарий теории графов [Электронный ресурс]: [сайт]. URL: https://ru.wikipedia.org/wiki/Глоссарий_теории_графов (дата обращения: 05.05.2020). Загл. с экрана. Яз. рус.
3) Гамильтоновы графы [Электронный ресурс]: [сайт]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Гамильтоновы_графы (дата обращения: 05.05.2020). Загл. с экрана. Яз. рус.
4) Путь в теории графов [Электронный ресурс]: [сайт]. URL: https://ru.wikipedia.org/wiki/Путь_(теория_графов) (дата обращения: 05.05.2020). Загл. с экрана. Яз. рус.
5) Плотный порядок [Электронный ресурс]: [сайт]. URL: https://ru.wikipedia.org/wiki/Плотный_порядок (дата обращения: 05.05.2020). Загл. с экрана. Яз. рус.
6) Фляйшнер Г. Эйлеровы графы и смежные вопросы. Пер. с англ. - М.:Мир, 2002, 176 с.
7) Кристофидес Н. Теория графов. Алгоритмический подход. Пер. с анг. - М.:Мир, 1978, 432 с.
8) Майника Э. Алгоритмы оптимизации на сетях и графах. Пер. с англ. - М.:Мир,1981, 328 с.
9) Алексеев В.В., Гаврилов Г.П., Сапоженко А.А. (ред.) Теория графов. Покрытия, укладки, турниры. Сборник переводов - М. : Мир, 1974.— 224 с.
10) Р.Уилсон. Введение в теорию графов. – М.: Мир, 1977

Личная подпись студента _____

Приложения

Приложение А

Каталог графов

В данном разделе находятся изображения панциклических графов, удовлетворяющих гипотезе о панциклических графах.

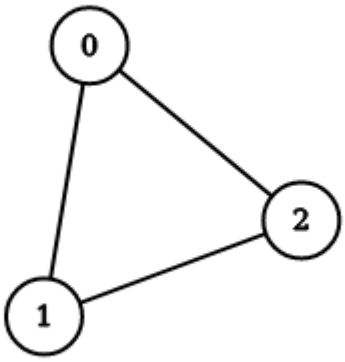

Рисунок 1 - один из графов, удовлетворяющий гипотезе о панциклических графах.

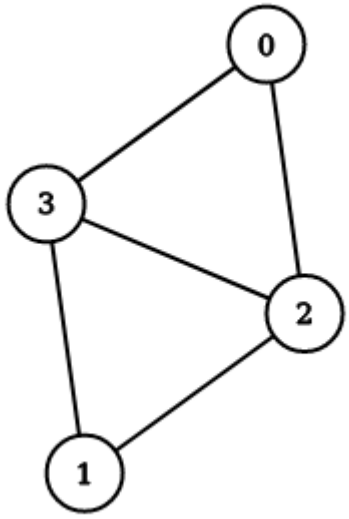

Рисунок 2 - один из графов, удовлетворяющий гипотезе о панциклических графах.

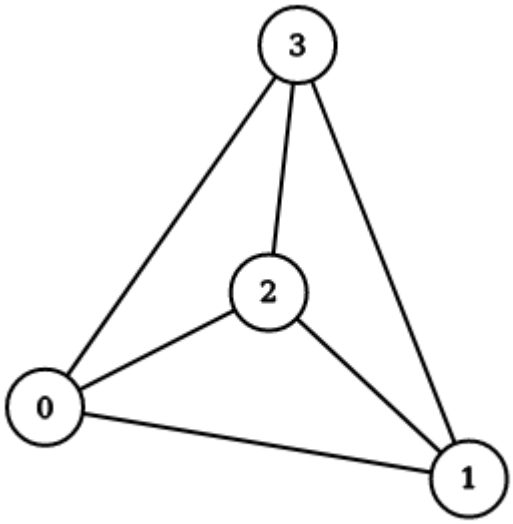

Рисунок 3 - один из графов, удовлетворяющий гипотезе о панциклических графах.

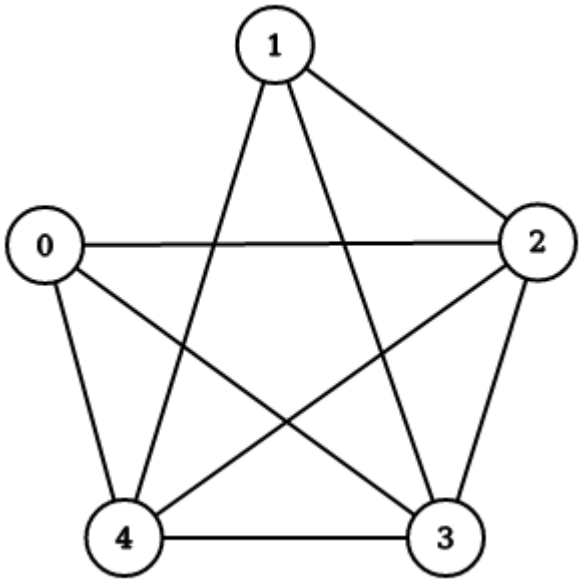

Рисунок 4 - один из графов, удовлетворяющий гипотезе о панциклических графах.

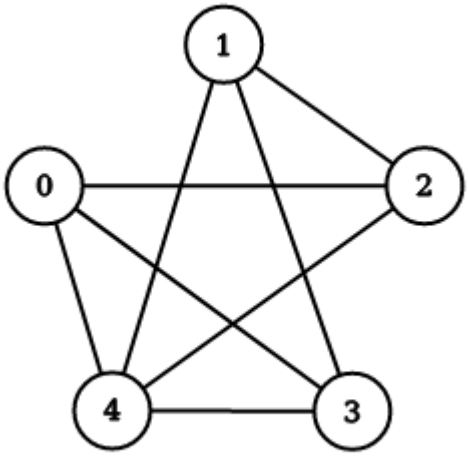

Рисунок 5 - один из графов, удовлетворяющий гипотезе о панциклических графах.

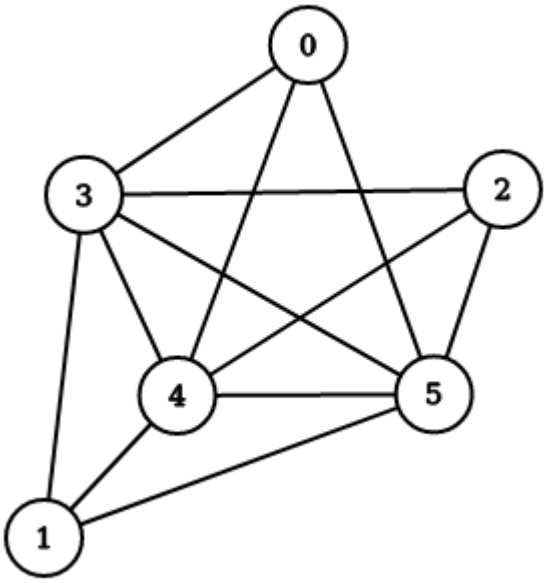

Рисунок 6 - один из графов, удовлетворяющий гипотезе о панциклических графах.

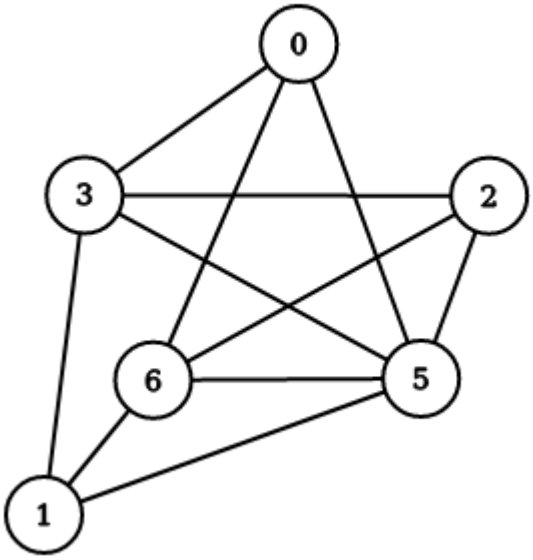

Рисунок 7 - один из графов, удовлетворяющий гипотезе о панциклических графах.

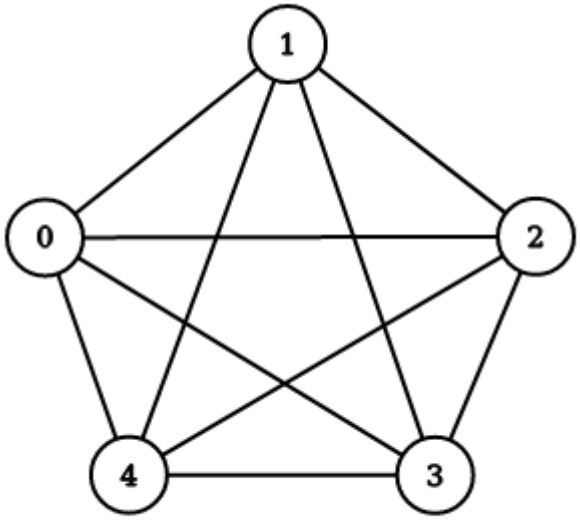

Рисунок 8 - один из графов, удовлетворяющий гипотезе о панциклических графах.

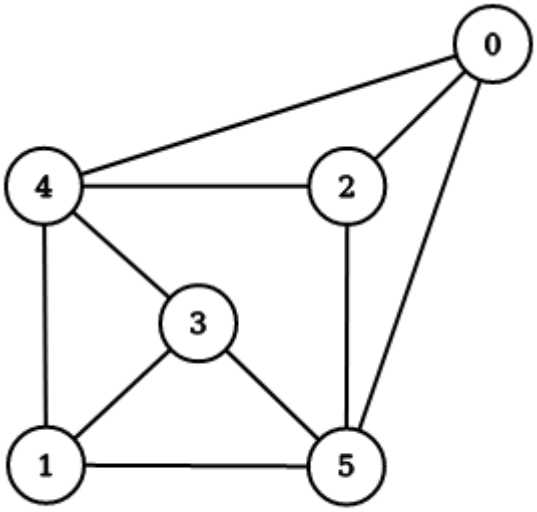

Рисунок 9 - один из графов, удовлетворяющий гипотезе о панциклических графах.

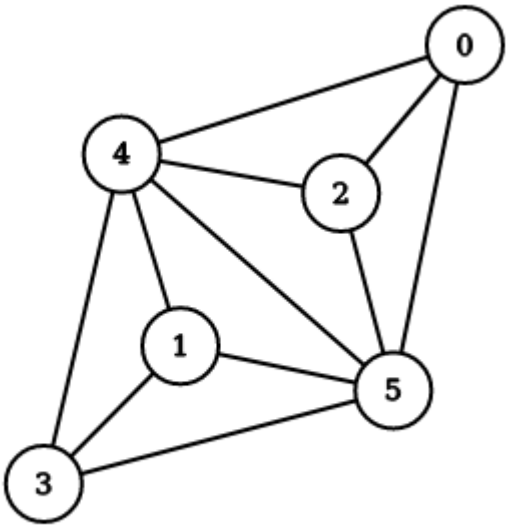

Рисунок 10 - один из графов, удовлетворяющий гипотезе о панциклических графах.

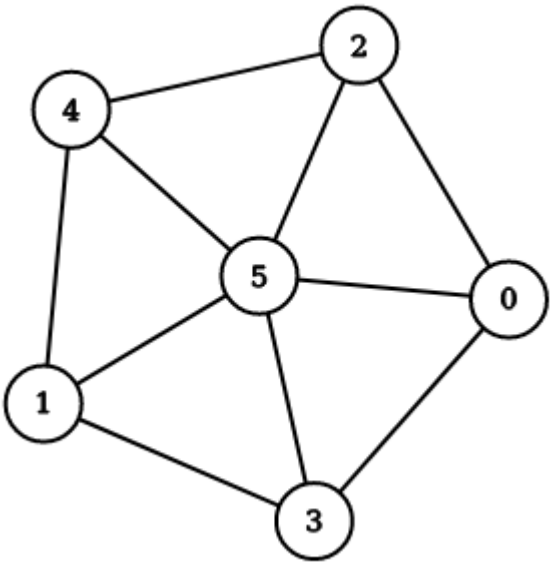

Рисунок 11 - один из графов, удовлетворяющий гипотезе о панциклических графах.

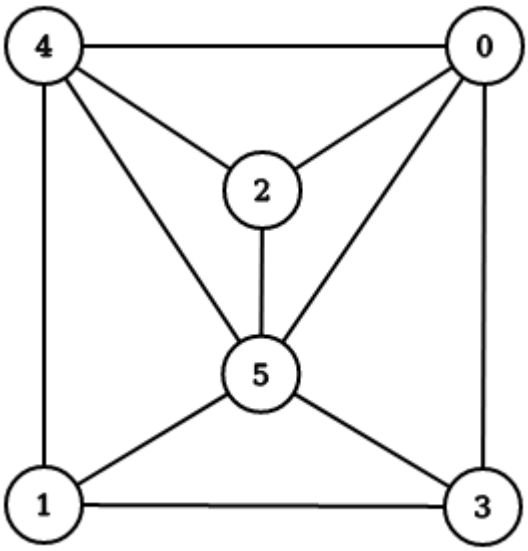

Рисунок 12 - один из графов, удовлетворяющий гипотезе о панциклических графах.

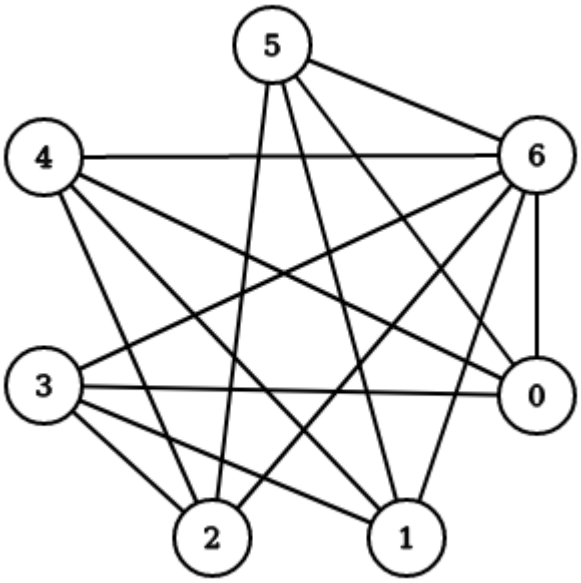

Рисунок 13 - один из графов, удовлетворяющий гипотезе о панциклических графах.

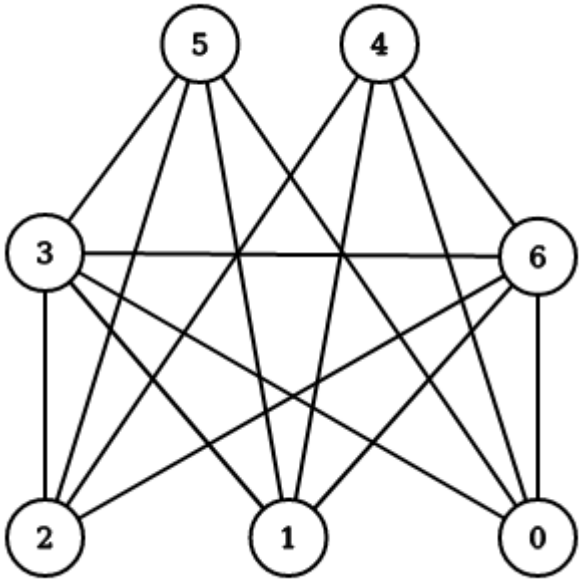

Рисунок 14 - один из графов, удовлетворяющий гипотезе о панциклических графах.

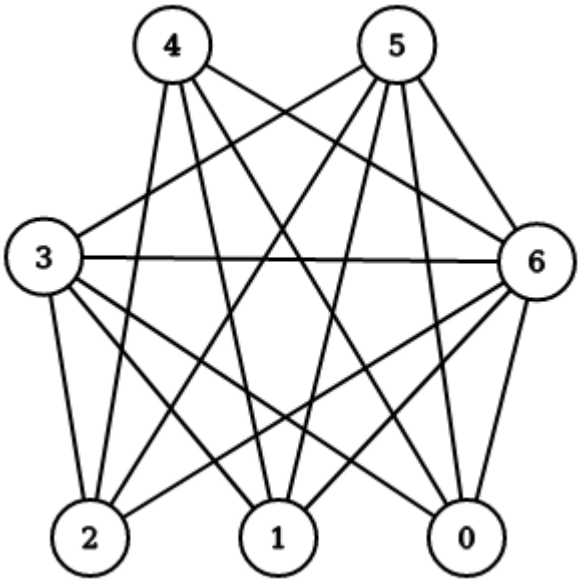

Рисунок 15 - один из графов, удовлетворяющий гипотезе о панциклических графах.

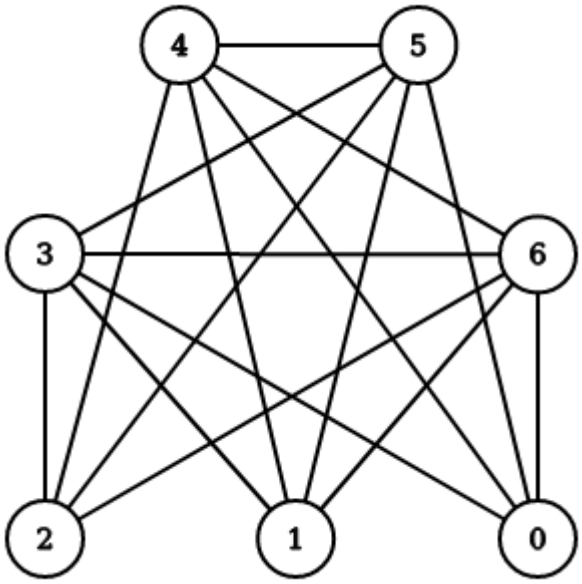

Рисунок 16 - один из графов, удовлетворяющий гипотезе о панциклических графах.

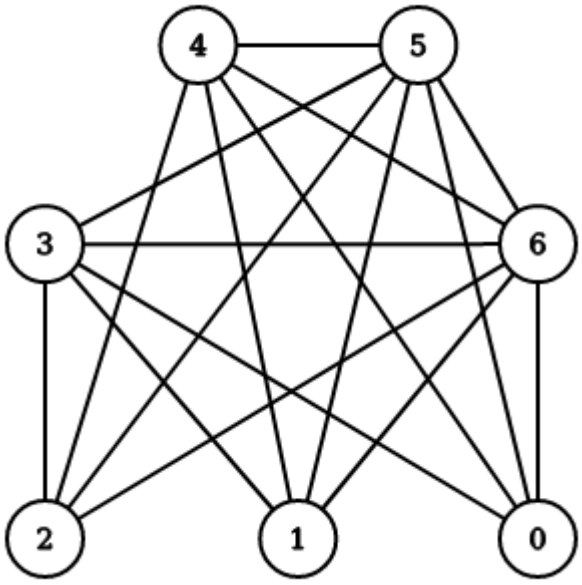

Рисунок 17 - один из графов, удовлетворяющий гипотезе о панциклических графах.

Поскольку с ростом числа вершин становится трудно изобразить граф, опустим числовые метки вершин.

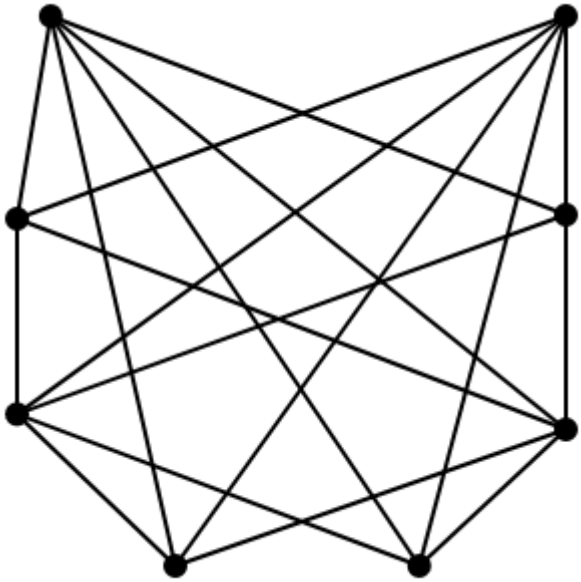

Рисунок 18 - один из графов, удовлетворяющий гипотезе о панциклических графах.

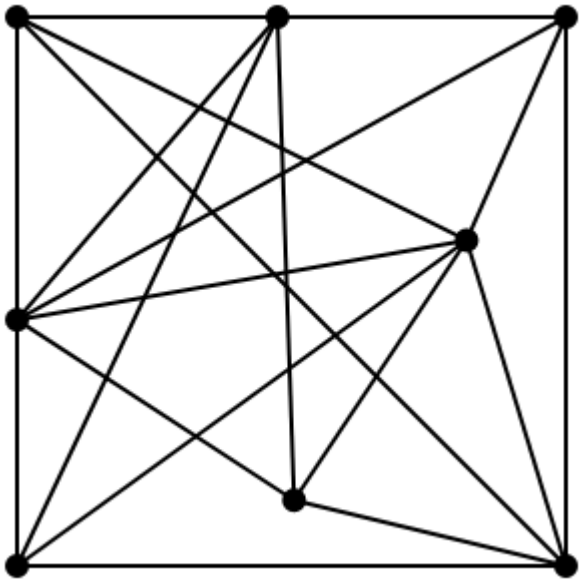

Рисунок 19- один из графов, удовлетворяющий гипотезе о панциклических графах.

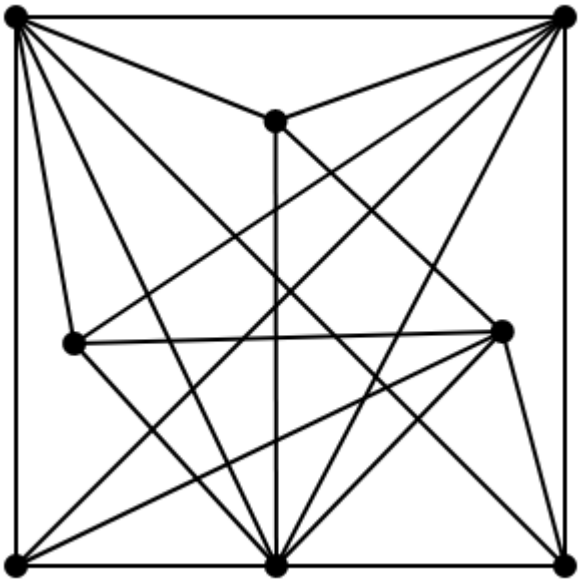

Рисунок 20 - один из графов, удовлетворяющий гипотезе о панциклических графах.

Приложение Б

Программа подсчета

```
#include "lib.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <mutex>
#include <thread>
std::vector<int>g[T][N];
std::vector<std::pair<int,int>>e[T];
int  degree[T][N];
bool a[T][N][N];
std::mutex mtx;
void print(int id)
{
  mtx.lock();
  for(int i=0;i<N;i++)
    printf("%d\n",i);
  for(int i=0;i<e[id].size();i++)
    printf("%d %d\n",e[id][i].first,e[id][i].second);
  printf("\n");
  mtx.unlock();
}
static char ignore_input[1024];
static bool err;
static bool load(int id)
{
  mtx.lock();
  fgets(ignore_input,sizeof(ignore_input),stdin);
  fgets(ignore_input,sizeof(ignore_input),stdin);
  if(feof(stdin))
    {
      mtx.unlock();
      return false;
    }
  int n,m;
```

```
  scanf("%d%d\n",&n,&m);
  e[id].clear();
  for(int i=0;i<m;i++)
    {
      int from,to;
      scanf("%d%d",&from,&to);
      e[id].push_back(std::make_pair(from,to));
    }
  fgets(ignore_input,sizeof(ignore_input),stdin);
  mtx.unlock();
  return true;
}
static void fill(int id)
{
  for(int i=0;i<N;i++)
    g[id][i].clear();
  memset(degree[id],0,sizeof(degree[id]));
  memset(a[id],0,sizeof(a[id]));
  for(int i=0;i<e[id].size();i++)
    {
      int from,to;
      from=e[id][i].first;
      to=e[id][i].second;
      degree[id][from]++;
      degree[id][to]++;
      g[id][from].push_back(to);
      g[id][to].push_back(from);
      a[id][from][to]=true;
      a[id][to][from]=true;
    }
}
static void print_err(int id)
{
  print(id);
  for(int i=0;i<20;i++)
    printf("гипотеза не подтвердилась\n");
}
```

```
static bool set_err(bool t)
{
  mtx.lock();
  err=t;
  mtx.unlock();
}
static bool get_err()
{
  mtx.lock();
  bool t=err;
  mtx.unlock();
  return t;
}
static void calc(int id)
{
  long cnt_local=0;
  time_t timer=time(NULL);
  while(load(id) && !get_err())
    {
      fill(id);

      cnt_local++;
      double minutes=difftime(time(NULL),timer)/60;
      if(!id && minutes>30)
    {
      printf(" >> N: %d process: %ld minutes:
%.1lf\n",N,cnt_local*T,minutes);
      timer=time(NULL);
    }

      if(!dirac_bad(id))
       continue;

      if(!trs(id))
       continue;

      for(int i=3;i<=N;i++)
```

```
        if(!cycdp(id,i))
          {
           set_err(true);
           print_err(id);
           return;
          }

       print(id);
       if(cnt_local>2)
        return;

    }
}
int main(int argc, char * argv[])
{
  std::vector<std::thread> threads;
  for(int i=0;i<T;i++)
    threads.push_back(std::thread(calc,i));
  for(int i=0;i<T;i++)
    threads[i].join();
  return 0;
}
```

Приложение В

Основная программа

```
#include <ctime>
#include <cstdio>
#include <string>
#include <cstdlib>
using namespace std;
int main()
{
  long long_check=1;
  for(int i=0;i<18;i++)
    long_check*=10;
  printf(" >> long check %ld\n",long_check);

  system("rm calc/csv/*.csv");
  time_t timer=time(NULL);
  FILE * in=fopen("calc/conf.txt","r");
  for(;;)
    {
      char buf[100];
      int n;
      if(fscanf(in,"%s%d",&buf,&n)!=2)
       break;
      if(n<3)
       continue;
      string s(buf),s_n=to_string(n);
      FILE * f=fopen(("calc/csv/"+s+".csv").c_str(),"w");
      fprintf(f,"\nn,cnt,h,e");
      for(int i=0;i<n;i++)
       fprintf(f,",%d",i+1);
      fprintf(f,"\n");
      fclose(f);
      for(int i=3;i<=n;i++)
       {
           time_t timer = time(NULL);
           string s_i=to_string(i);
           string command;
```

```
command="g++ -c -std=c++11 -DT=4";
command+=" -DN="+s_i;
command+=" -DNMAX="+s_n;
command+=" calc/src/prog.cpp";
command+=" -o calc/o/prog.o";
printf(" >> %s\n",command.c_str());
if(system(command.c_str()))
    return -1;
command="g++ -c -std=c++11 -DT=4";
command+=" -DN="+s_i;
command+=" calc/src/cycle.cpp";
command+=" -o calc/o/cycle.o";
printf(" >> %s\n",command.c_str());
if(system(command.c_str()))
    return -1;
command="g++ -c -std=c++11 -DT=4";
command+=" -DN="+s_i;
command+=" calc/src/hamilton.cpp";
command+=" -o calc/o/hamilton.o";
printf(" >> %s\n",command.c_str());
if(system(command.c_str()))
    return -1;
command="g++ -c -std=c++11 -DT=4";
command+=" -DN="+s_i;
command+=" calc/src/tr.cpp";
command+=" -o calc/o/tr.o";
printf(" >> %s\n",command.c_str());
if(system(command.c_str()))
    return -1;
command="g++ -c -std=c++11 -DT=4";
command+=" -DN="+s_i;
command+=" calc/src/cycdp.cpp";
command+=" -o calc/o/cycdp.o";
printf(" >> %s\n",command.c_str());
if(system(command.c_str()))
    return -1;
command="g++ -lpthread -o calc/bin/prog";
```

```
            command+=" calc/o/hamilton.o";
            command+=" calc/o/cycle.o";
            command+=" calc/o/prog.o";
            command+=" calc/o/tr.o";
            command+=" calc/o/cycdp.o";
            printf(" >> %s\n",command.c_str());
            if(system(command.c_str()))
                return -1;
            if(s=="connected")
                command="./geng -c "+s_i;
            else if(s=="biconnected")
                command="./geng -C "+s_i;
            else if(s=="bipartite")
                command="./geng -cb "+s_i;
            else if(s=="triangle_free")
                command="./geng -ct "+s_i;
            else if(s=="four_cycle_free")
                command="./geng -cf "+s_i;
            command+=" | ./showg -e | ./calc/bin/prog";
            printf(" >> %s\n",command.c_str());
            if(system(command.c_str()))
                return -1;
            double minutes=difftime(time(NULL),timer)/60;
            fprintf(stderr," >> %s%d %.1lf min\n",buf,i,minutes);
        }
    }
  double minutes=difftime(time(NULL),timer)/60;
  fprintf(stderr," >> TOTAL TIME %.1lf min\n", minutes);
  fclose(in);
  return 0;
}
```

Приложение Г

Подпрограмма проверки отношения смежности графа на плотность

```
#include "lib.h"
#include <cstring>
#include <cstdio>
#include <cstdlib>
static bool bad(int edge, int id)
{
  int from=e[id][edge].first,to=e[id][edge].second;
  for(int i=0;i<g[id][from].size();i++)
    {
      int v=g[id][from][i];
      if(a[id][v][to])
       return false;
    }
  return true;
}
bool trs(int id)
{
  for(int i=0;i<e[id].size();i++)
    if(bad(i,id))
      return false;
  return true;
}
```

Приложение Д

Подпрограмма нахождения множества длин циклов графа

```
#include <vector>
#include <cstring>
#include <cstdio>
#include "lib.h"
static int  label[T][N];
static bool s[T][N];
void lengths(int i,int l,int id)
{
  label[id][i]=l;
  for(int j=0;j<g[id][i].size();j++)
    {
      int to=g[id][i][j],to_label=label[id][to];
      if(!to_label)
       lengths(to,l+1,id);
      else
       {
        int length=l-to_label+1;
        if(length>=3)
           s[id][length-1]=true;
       }
    }
  label[id][i]=0;
}
std::vector<int> lengths_main(int id)
{
  memset(s[id],0,sizeof(s[id]));
  memset(label[id],0,sizeof(label[id]));
  lengths(0,1,id);
  std::vector<int>v;
  for(int i=0;i<N;i++)
    if(s[id][i])
      v.push_back(i+1);
  return v;
}
```

Приложение Ж

Подпрограмма проверки гамильтоновости графа

```
#include "lib.h"
#include <cstring>
#define nothing 0
#define hamilton_path 1
#define hamilton_cycle 2
static bool dp[T][N][1<<N];
static int h_start[T];
bool dirac_bad(int id)
{
  int min_degree=degree[id][0];
  for(int i=1;i<N;i++)
    min_degree=std::min(min_degree,degree[id][i]);
  return min_degree>=((N+1)>>1);
}
static int hamilton(int i,int m,int id)
{
  if(dp[id][i][m])
    return nothing;
  dp[id][i][m]=true;
  if((1<<i)==m)
    {
      for(int j=0;j<g[id][i].size();j++)
       if(g[id][i][j]==h_start[id])
         return hamilton_cycle;
      return hamilton_path;
    }
  int ans=nothing;
  for(int j=0;j<g[id][i].size();j++)
    if( m & (1<<g[id][i][j]) )
      {
       int tmp=hamilton(g[id][i][j],m^(1<<i),id);
       if(tmp==hamilton_cycle)
         return hamilton_cycle;
       if(tmp==hamilton_path)
         ans=hamilton_path;
```

```
      }
  return ans;
}
int hamilton_main(int id)
{
  if(dirac_bad(id))
    return hamilton_cycle;
  memset(dp[id],false,sizeof(dp[id]));
  int ans=nothing;
  for(int i=0;i<N;i++)
    {
      h_start[id]=i;
      int tmp=hamilton(i,(1<<N)-1,id);
      if(tmp==hamilton_cycle)
    return hamilton_cycle;
      if(tmp==hamilton_path)
    ans=hamilton_path;
    }
  return ans;
}
```

Приложение И

Заголовочный файл

```
#ifndef LIB
#define LIB
extern void print(int id);
extern int  hamilton_main(int id);
extern bool dirac_bad(int id);
extern bool cycdp(int id,int l);
extern bool trs(int id);
extern int  degree[T][N];
extern bool a[T][N][N];
#include <mutex>
extern std::mutex mtx;
#include <vector>
extern std::vector<int> lengths_main(int id);
extern std::vector<int> g[T][N];
extern std::vector<std::pair<int,int>> e[T];
#endif
```

Приложение К

Подпрограмма нахождения цикла заданной длины

```
#include "lib.h"
#include <cstring>
static bool dp[T][N][1<<N];
static int  start[T];
static bool f(int i,int m,int l,int cnt,int id)
{
  if(dp[id][i][m])
    return false;
  dp[id][i][m]=true;
  if(cnt==l)
    {
      for(int j=0;j<g[id][i].size();j++)
       if(g[id][i][j]==start[id])
        return true;
      return false;
    }
  for(int j=0;j<g[id][i].size();j++)
    if( m & (1<<g[id][i][j]) )
      if(f(g[id][i][j],m^(1<<i),l,cnt+1,id))
    return true;
  return false;
}
bool cycdp(int id,int l)
{
  memset(dp[id],false,sizeof(dp[id]));
  for(int i=0;i<N;i++)
    {
      start[id]=i;
      if(f(i,(1<<N)-1,l,1,id))
       return true;
    }
  return false;
}
```