Оглавление

[**Глава 1. Клики и их поиск в графе** 2](#_Toc148908856)

[1.1 Алгоритм Брона-Кербоша 2](#_Toc148908857)

[1.1.1 Алгоритм Брона-Кербоша без поворота 3](#_Toc148908858)

[1.1.2 Алгоритм Брона-Кербоша с поворотом 4](#_Toc148908859)

[1.1.3 Алгоритм Брона-Кербоша с порядком вершин 4](#_Toc148908860)

[1.2 Алгоритм MaxCliqueDyn 5](#_Toc148908861)

[1.3 Алгоритм Хансера 6](#_Toc148908862)

[1.4 Алгоритм простой итерации 7](#_Toc148908863)

[1.5 Сравнение алгоритмов 7](#_Toc148908864)

[**Глава 2. Практическая реализация алгоритмов** 8](#_Toc148908865)

[2.1 Класс Graph 8](#_Toc148908866)

[2.2 Реализация алгоритма Брона-Кербоша без поворота 11](#_Toc148908867)

[2.3 Реализация алгоритма Брона-Кербоша с поворотом 12](#_Toc148908868)

[2.4 Основное меню 13](#_Toc148908869)

[2.5 Пробные запуски 14](#_Toc148908870)

[**Глава 3. Оптимизация кода** 15](#_Toc148908871)

[3.1 Оптимизация по производительности 15](#_Toc148908872)

[3.2 Оптимизация по потреблению памяти 16](#_Toc148908873)

[**Заключение** 17](#_Toc148908874)

[**Список литературы** 18](#_Toc148908875)

Глава 1. Клики и их поиск в графе

**Граф –**это топологичекая модель, которая состоит из множества вершин и множества соединяющих их рёбер. В задаче о поиске клик в графе используются невзвешенные и неориентированные графы, то есть графы, ребра которых не имеют направления и веса. Клика в графе – подмножество вершин графа, любые две из которых соединены ребром. Другими словами, клика – это полный подграф первоначального графа. Для нахождения всех возможных образованных в графе клик используются различные алгоритмы решения данной задачи.

1.1 Алгоритм Брона-Кербоша

Алгоритм Брона-Кербоша – алгоритмический метод для нахождения оптимального решения для поиска всех клик неориентированного невзвешенного графа. Был разработан математиками Конрадом Броном и Джоепом Кербошем в 1973 и до сих пор является одним из самых эффективных алгоритмов поиска клик.

В общей сложности алгоритм может применяться для двух различных целей: поиска максимальной клики (основная цель) и поиск некой клики с четко заданным размером. Большинство модификаций и подвидов алгоритма связаны именно с решением задачи, связанной с поиском максимальной клики.

Начиная с одиночной вершины, образующей полный подграф, алгоритм на каждом шаге пытается увеличить уже построенный полный подграф, добавляя в него вершины из множества кандидатов. Алгоритм использует тот факт, что всякая клика в графе является его максимальным по включению полным подграфом.

Высокая эффективность и скорость обеспечивается отсечением при переборе вариантов, которые заведомо не приведут к построению клики, для чего используется дополнительное множество, в которое помещаются вершины, которые уже были использованы для увеличения полного подграфа. Вычислительная сложность линейна относительно количества клик в графе. В худшем случае алгоритм работает за О(), где n – количество вершин в графе.

Хотя другие алгоритмы для решения данной задачи теоретически лучше работают на входах, которые имеют несколько максимальных независимых множеств, алгоритм Брона-Кербоша и последующие его усовершенствования на практике являются более эффективными, нежели другие альтернативы.

Данный алгоритм хорошо известен и широко используется в прикладных областях графовых алгоритмов (например, вычислительная химия).

1.1.1 Алгоритм Брона-Кербоша без поворота

Базовая форма алгоритма Брона-Кербоша представляет собой рекурсивный алгоритм поиска с [возвратом](https://en.wikipedia.org/wiki/Backtracking), который ищет все максимальные клики в данном графе G. В более общем смысле, для трех непересекающихся наборов вершин R, P и X он находит максимальные клики, включающие все вершины в R, некоторые вершины в P и ни одну из вершин в X. Таким образом можно сказать, что множество R – “собираемая” клика с уже отобранными под нее вершинами; P – множество оставшихся кандидатов, которые в дальнейшем будут рассматриваться в качестве возможных членов множества R; X – множество рассмотренных вершин, которые уже были использованы для расширения и не подходят для включения в R. В каждом вызове алгоритма P и X являются непересекающимися множествами, объединение которых состоит из тех вершин, которые образуют клики при добавлении к R. Когда P и X оба пусты, к R нельзя добавить никаких дополнительных элементов, поэтому R является максимальной кликой, и алгоритм выводит в качестве ответа именно R.

Базовая форма алгоритма неэффективна в случае графов с большим количеством немаксимальных клик: он делает рекурсивный вызов для каждой клики, максимальной или нет, что значительно увеличивает время работы алгоритма. В дальнейших доработках Брон и Кербош сами неоднократно исследовали и модифицировали свой базовый алгоритм.

1.1.2 Алгоритм Брона-Кербоша с поворотом

Понимая, что базовый алгоритм (без поворота) тратит слишком много времени на рекурсивный вызов каждой клики в графе, Брон и Кербош ввели вариант алгоритма, включающий некую “осевую вершину” u, выбранную из P. Любая максимальная клика должна включать либо u, либо одного из ее несоседей, иначе клику можно было бы увеличить, добавив к ней u. Поэтому только u и ее несоседи должны быть проверены как варианты для вершины v, которая добавляется к R при каждом рекурсивном вызове алгоритма.

При дополнении базовой формы алгоритма Брона-Кербоша поворотом минимизируется количество рекурсивных вызовов, выполняемых алгоритмом; экономия времени выполнения по сравнению с версией алгоритма без поворота может быть значительной, хотя при небольшом количестве вершин и ребер разница во времени не существенна.

1.1.3 Алгоритм Брона-Кербоша с порядком вершин

Также существует альтернативный метод улучшения базовой формы алгоритма Брона-Кербоша. Под ним подразумевается отказ от поворота на самом внешнем уровне рекурсии, а также тщательный выбор порядка рекурсивных вызовов. Тщательный выбор порядка основан на порядке вырождения графа. Вырожденность графа G – наименьшее число d такое, что каждый подграф графа G имеет вершину степени d или меньше. Порядок вырождения – порядок вершин такой, что каждая вершина имеет d или меньше соседей, которые следуют позже по порядку. Если порядок вершин v, по которым проходит алгоритм Брона-Кербоша, является порядком вырождения, то множество P вершин-кандидатов в каждом вызове (соседи v, которые находятся позже в порядке) гарантированно будут иметь размер не более d.

Порядок вырождения находится за линейное время, многократно выбирая вершину минимальной степени среди оставшихся вершин. Данный метод достаточно эффективен для графов с малым вырождением и используется на практике в разных областях реального мира (например, социальные сети).

1.2 Алгоритм MaxCliqueDyn

Алгоритм MaxCliqueDyn – еще один алгоритм поиска максимальной клики в неориентированном невзвешенном графе. Он основан на базовом алгоритме (алгоритме MaxClique), который находит максимальную клику ограниченного размера. Граница находится с использованием улучшенного алгоритма раскраски. MaxCliqueDyn расширяет алгоритм MaxClique, включая динамически изменяющиеся границы. Алгоритм был разработан Жанезом Концем в 2007 году.

По сравнению с предыдущими алгоритмами MaxCliqueDyn улучшен алгоритмом приближенной окраски (алгоритмом ColorSort) и применением более жестких и более затратных в вычислении верхних границ для пространства поиска. Однако оба улучшения сокращают время на поиск максимальной клики. В дополнение к сокращению времени улучшенный алгоритм раскраски также уменьшает количество шагов, необходимых для поиска максимальной клики.

Базовый алгоритм рекурсивно ищет максимальную клику, добавляя и удаляя вершины из R – набора вершин растущей в данный момент клики.

Алгоритм окраски ColorSort раскрашивает вершины одну за другой в том же порядке, в котором они появляются в наборе вершин-кандидатов P, так что, если следующая вершина v не является смежной со всеми вершинами в некотором цветовом классе, она добавляется в этот класс; и если v смежна хотя бы с одной вершиной в каждом из существующих цветовых классов, она помещается в новый цветовой класс.

На каждом шаге алгоритма MaxClique алгоритм также пересчитывает степени вершин в P относительно вершины, в которой алгоритм находится в данный момент. Затем эти вершины сортируются в порядке убывания относительно их степеней в графе G. Затем алгоритм ColorSort рассматривает вершины в P, отсортированные по их степеням. Таким образом, количество шагов, необходимых для нахождения максимальной клики, сокращается до минимума.

Алгоритм имеет постоянные вычислительные затраты O(n2) на определение степеней и сортировку вершин, где n – количество вершин в P.

1.3 Алгоритм Хансера

Алгоритм Хансера является модификацией алгоритма Брона-Кербоша. В этом алгоритме максимальность найденного множества вершин определяется только невозможностью его расширения, в то время как в методе Брона-Кербоша максимальность проверяется также условием пустоты множества X (множества вершин, которые уже были использованы для расширения). Необходимым и достаточным условием в методе Хансера является то, что множество P на некотором шаге оказывается пустым.

Отличие алгоритма Хансера от алгоритма Брона-Кербоша также состоит в изменении условия для осуществления шага возвращения. На очередном шаге может быть построено множество вершин, которое потенциально может стать наибольшей кликой. Условие возвращения заключается в проверке, является ли текущее «потенциальное множество» подмножеством к какому-нибудь из уже найденных решений. Это условие оказывается более эффективным, чем условие, проверяемое в методе Брона-Кербоша.

Вычислительная сложность алгоритма составляет примерно О().

1.4 Алгоритм простой итерации

На каждом шаге алгоритма ищется вершина v с наименьшей степенью deg(v), и если deg(v) < G-1, где G - количество вершин графа на текущем шаге, то вершина v удаляется. Иначе все вершины графа имеют степень G-1, и полученный подграф является полным подграфом исходного графа.

В большой доле реальных задач этот алгоритм находит максимальный полный подграф, однако это не всегда так. Если существует только одна большая клика, заметно превышающая по размеру остальные, то алгоритм её найдёт. Если таких больших клик несколько, то алгоритм найдёт лишь одну из них, поэтому данный алгоритм не применим в задачах, где требуется максимально точное решение.

Оценка сложности составляет O(n2), где n – количество вершин графа.

1.5 Сравнение алгоритмов

Анализируя работу всех вышеперечисленных алгоритмов по поиску максимальных клик, можно прийти к выводу, что наиболее эффективно себя показывают алгоритмы Брона-Кербоша и их подвиды (алгоритм Хансера).

Алгоритм MaxCliqueDyn состоит сразу из двух алгоритмов: MaxClique и ColorSort. Из-за того, что каждый раз алгоритму требуется делить все входные данные на классы и сортировать, алгоритм тратит на это достаточно много времени.

Алгоритм простой итерации также рассматривает каждый элемент и сортирует по определенному признаку. Кроме того, алгоритм не подразумевает точный результат и выводит ответ лишь частично, находя лишь одну подходящую максимальную клику.

Базовый алгоритм Брона-Кербоша тоже можно отнести к менее эффективным из-за анализа каждой элемента и его соседей.

Модифицированные алгоритмы (алгоритм Хансера) и подвиды алгоритма Брона-Кербоша (с поворотом, с порядком вершин) стараются так или иначе исключить вариант анализа каждого элемента, что позволяет алгоритмам работать намного более эффективно, отсекать большое количество заведомо неподходящих под условия программы элементов и тратить на выполнение задачи гораздо меньше времени.

Можно сделать вывод, что благодаря большому количеству возможных модификаций, ускоряющих процесс выполнения поиска клик в графах, алгоритм Брона-Кербоша является одним из наиболее используемых алгоритмов для решения задач этого типа в совершенно разных областях.

Глава 2. Практическая реализация алгоритмов

2.1 Класс Graph

В базовом классе Graph представлены все необходимые конструкторы, операторы и методы, позволяющие работать с объектами класса. Так как алгоритм работает со множествами чисел, в качестве основного типа данных для реализации класса и алгоритмов выбран тип set, который позволяет реализовать весь функционал графов и их алгоритмов благодаря своей структуре, упорядоченности и свойствам множества, а также позволяет легко обращаться к нужным элементам, сохранять и выводить данные из файлов. Тип map позволяет легко получить доступ к нужному множеству по соответствующему ключу, что также значительно упрощает взаимодействие с элементами. В спецификаторе доступа protected класса Graph находятся член nodes типа set<int> и adjacentList типа map<int, set<int>>. Член nodes – это множество, хранящее в себе все вершины графа в виде целого положительного числа (номер вершины); adjacentList – это ассоциативный массив, который хранит по ключам (вершинам) множества значений (множества соседей данных вершин). Также в protected представлены две рекурсивные функции bronKerboschWOP и bronKerboschWP, отвечающие за соответствующие им способы нахождения клик (алгоритмы Брона-Кербоша без поворота и с поворотом) в графе и вызываемые в public отдельными функциями для вхождения в рекурсию. В самом спецификаторе доступа public представлены основные конструкторы и различные методы.

* Конструктор по умолчанию имеет заданный параметр типа int size, равный 1. Конструктор с помощью цикла добавляет в nodes вершины количеством size с помощью метода addNode().
* Конструктор копий позволяет создавать объект класса на основе уже существующего объекта, копируя его данные из nodes и adjacentList.
* Функция addNode() типа void с параметром index типа int позволяет добавить новую вершину в граф; в nodes и adjacentList добавляется вершина index при условии ее отсутствия в nodes.
* Функция addEdge() типа void с параметрами node1 и node2 типа int позволяет добавить новое ребро в граф; в adjacentList добавляются вершины node1 и node2, если они существуют в nodes и не равны.
* Функция delNode() типа void с параметром index типа int позволяет удалить вершину из графа; из nodes и adjacentList удаляется вершина index и также удаляется из списка соседей adjacentList всех остальных вершин.
* Функция delEdge() типа void с параметрами node1 и node2 типа int позволяет удалить ребро из графа; из adjacentList удаляются вершины node1 и node2, если они существует в adjacentList и не равны.
* Функция areAdjacent() типа bool с параметрами node1 и node2 типа int позволяет проверить, являются ли вершины node1 и node2 соседями (есть ли между ними ребро); возвращает true или false в зависимости от результата поиска node2 в adjacentList[node1].
* Функция getNeighbours() типа set<int> с параметром index типа int позволяет получить множество вершин – соседей вершины index; возвращает множество adjacentList[index].
* Функция getNodes() типа set<int> позволяет получить множество всех вершин nodes; возвращает множество nodes.
* Функция getNumOfNodes() типа int позволяет получить количество вершин графа; возвращает размер множества nodes.
* Функция getNumOfEdges() типа int позволяет получить количество всех ребер в графе; подсчитывает количество соседей в adjacentList() каждой вершины и делит полученное число пополам, так как на каждое ребро приходится по две вершины.
* Функция random() типа void позволяет наполнить граф случайными ребрами; перебираются все пары вершин из nodes и с помощью генератора 0 и 1 и функции areAdjacent() решается, какие вершины будут соединены ребром, а какие – нет.
* Функция allCliquesWOP() типа vector<set<int>> позволяет запустить рекурсивный алгоритм bronKerboschWOP() из protected; создает входные параметры множеств R, P, X для запуска рекурсии при условии ненулевого множества nodes.
* Функция allCliquesWP() типа vector<set<int>> позволяет запустить рекурсивный алгоритм bronKerboschWP() из protected; создает входные параметры множеств R, P, X для запуска рекурсии при условии ненулевого множества nodes.
* Оператор присваивания позволяет присваивать одному объекту класса данные другого объекта класса; копирует данные из nodes и adjacentList одному объекту от другого.
* Дружественный оператор вывода позволяет выводить данные о графе (в виде: <вершина> --> <множество соседей>) в поток и файл; в поток данные графа выводятся перебором множеств из nodes и adjacentList, а в файл сохраняются последовательно: количество вершин, все их номера, затем количество и все соседи каждой из вершин.
* Дружественный оператор ввода позволяет вводить данные о графе из потока и файла: в потоке пользователю выводится номер исследуемой вершины, после чего он должен ввести количество соседей этой вершины и множество всех этих соседей, а при выводе из файла данные считываются в порядке, обратном вводу данных графа в файл с помощью оператора вывода.

2.2 Реализация алгоритма Брона-Кербоша без поворота

В классе Graph представлены две функции, связанные с данным алгоритмом. В public представлена функция allCliquesWOP(), которая задает начальные параметры для запуска рекурсивной функции bronKerboschWOP() в protected.

В allCliquesWOP() задается вектор cliques, который будет состоять из множеств вершин, отобранных в ходе выполнения рекурсивного алгоритма в качестве клик; также при условии наличия ненулевого графа задаются множества R, P = nodes и X, которые соответствуют множествам, описанным Броном и Кербошем в их алгоритме, и запускается сам рекурсивный алгоритм bronKerboschWOP().

В bronKerboschWOP() алгоритм состоит из следующих действий:

1. Если P и X пусты, то в cliques добавляется клика R;
2. Для каждой вершины v из множества P, перебираемого с помощью итератора it:
3. Создается множество N – соседи v;
4. Добавляется v в R;
5. С помощью функции set\_intersection находятся пересечения P с N и X с N;
6. Выполняется рекурсия bronKerboschWOP() с обновленными R, P, X и cliques;
7. Удаляется v из P;
8. Добавляется v в X;

При работе подобного рекурсивного алгоритма в графе перебираются абсолютно все вершины и обновляют вектор cliques, добавляя в него все найденные R.

2.3 Реализация алгоритма Брона-Кербоша с поворотом

В классе Graph представлены две функции, связанные с данным алгоритмом. В public представлена функция allCliquesWP(), которая задает начальные параметры для запуска рекурсивной функции bronKerboschWP() в protected.

В allCliquesWP() задается вектор cliques, который будет состоять из множеств вершин, отобранных в ходе выполнения рекурсивного алгоритма в качестве клик; также при условии наличия ненулевого графа задаются множества R, P = nodes и X, которые соответствуют множествам, описанным Броном и Кербошем в их алгоритме, и запускается сам рекурсивный алгоритм bronKerboschWP().

В bronKerboschWP() алгоритм состоит из следующих действий:

1. Если P и X пусты, то в cliques добавляется клика R;
2. Каждая вершина u в P перебирается и находится u с максимальным количеством соседей adjCount;
3. Создается множество Nu – соседи u;
4. Находится разница между множеством P и множеством Nu;
5. Для каждой вершины v из множества обновленного P, перебираемого с помощью итератора it:
   1. Создается множество N – соседи v;
   2. Добавляется v в R;
   3. С помощью функции set\_intersection находятся пересечения P с N и X с N;
   4. Выполняется рекурсия bronKerboschWOP() с обновленными R, P, X и cliques;
   5. Удаляется v из P;
   6. Добавляется v в X;

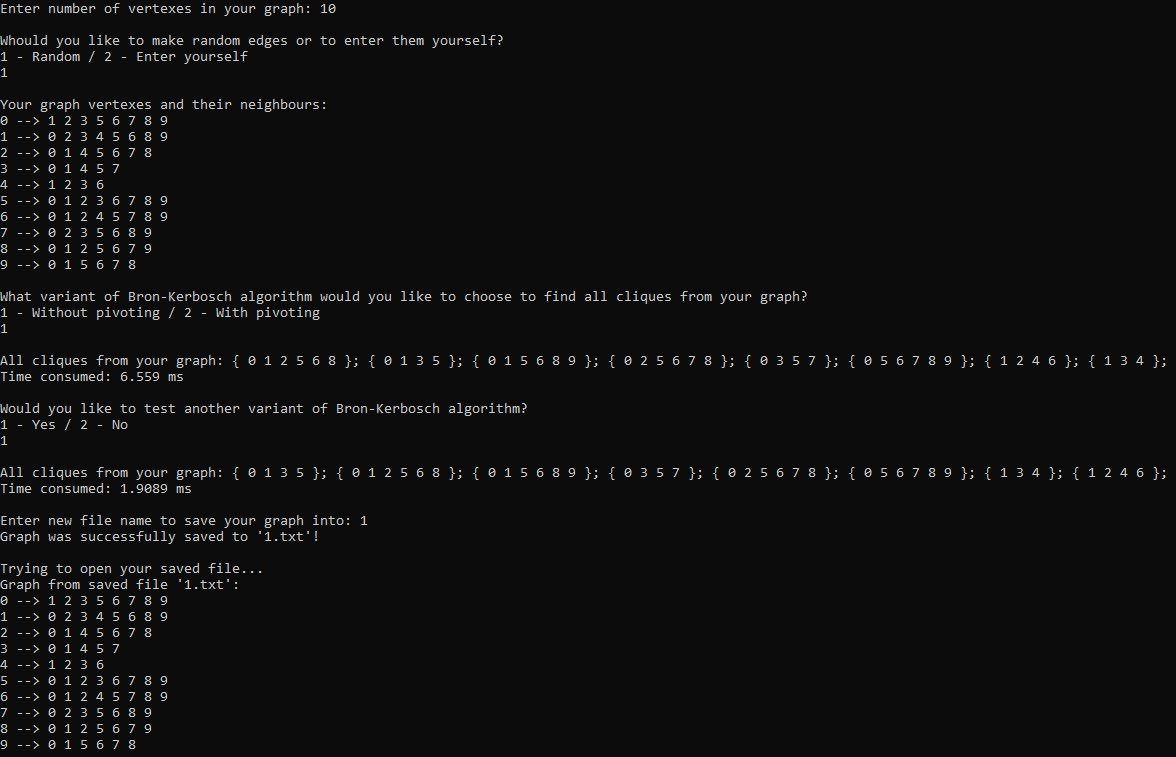
При работе подобного рекурсивного алгоритма в графе перебираются все вершины, отсеивая каждый раз несмежные вершины текущей рассматриваемой вершины и обновляя вектор cliques, добавляя в него все найденные R.

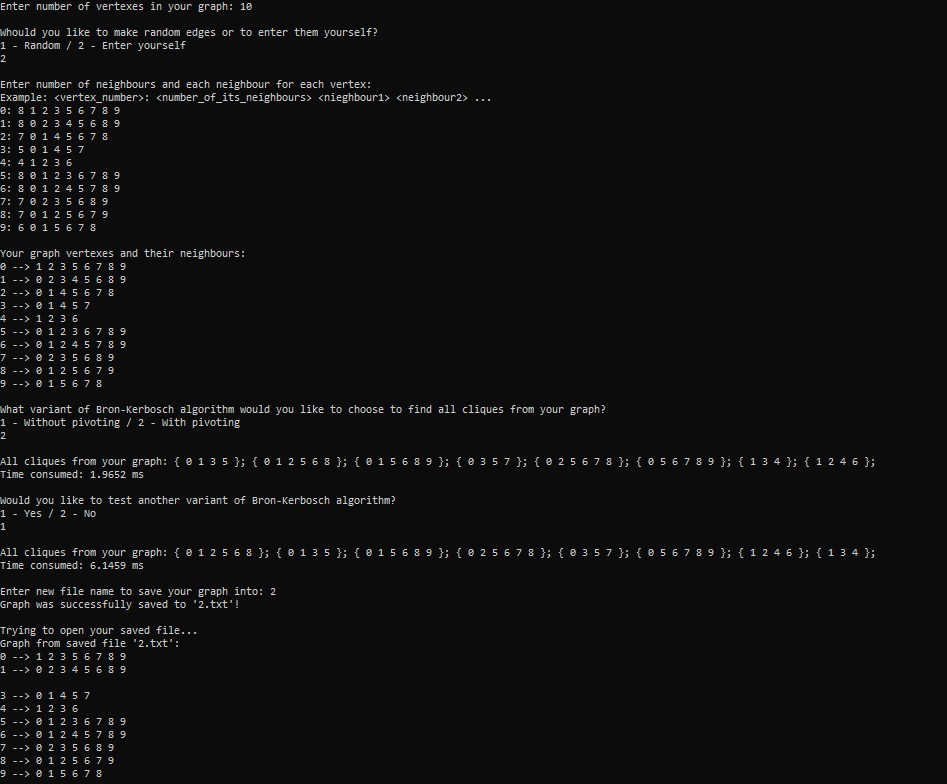
2.4 Основное меню

При старте программы пользователя попросят ввести количество вершин для создания нового графа. Для продолжения работы программы пользователь должен ввести положительное число (>0), иначе программа будет постоянно просить ввести корректное число. Далее программа предложит выбрать вариант ввода ребер между вершинами: 1 – позволить программе наполнить граф случайными ребрами; 2 – ввести все вершины самому. Во втором случае пользователю будет необходимо ввести для каждой вершины сначала количество смежных вершин, а затем и номер каждой такой вершины в графе. Программа покажет заполненный вершинами и ребрами граф и предложит пользователю вариант нахождения клик с помощью алгоритма Брона-Кербоша: 1 – Без поворота; 2 – С поворотом. Далее программа выводит все найденные выбранным пользователем способом клики, а также пишет время в миллисекундах, которое понадобилось программе для вычисления клик в указанном графе. Затем программа предлагает пользователю попробовать найти клики другим способом: 1 – Да; 2 – Нет. При выборе первого варианта алгоритм снова посчитает все клики другим способом и также выведет данные о затраченном времени. После выполнения всех алгоритмов программа попросит пользователя ввести название для создание нового текстового файла (формата .txt) для сохранения в него данных о графе. После чего программа сохранит граф в указанных файл и попробует открыть его для прочтения. В случае успеха на экран будет выведен считанный из файла граф, иначе программа напишет о том, что сохраненный файл был поврежден.

2.5 Пробные запуски

Ниже представлены примеры запуска программы с разными входными данными и возможными выборами пользователя.

Рисунок 1. Пример случайного набора ребер в графе, поиска клик и вывод в файл и из файла

Рисунок 2. Пример пользовательского набора ребер в графе, поиска клик и вывод в файл и из файла

Глава 3. Оптимизация кода

Оптимизация предполагает построение кода, который выполняется быстрее или требует меньше памяти для работы. На оптимизацию можно смотреть как на обобщённые эквивалентные преобразования, гарантированно приводящие к тому же результату, но с меньшими затратами ресурсов.

В данной курсовой работе была использована непосредственно программная оптимизация, которая предполагает построение кода, работающего быстрее или потребляющего меньшее количество памяти.

3.1 Оптимизация по производительности

В данной курсовой работе присутствует большое количество циклов, так как основные алгоритмы работают с помощью перебора большого количества элементов. Можно выделить несколько принципов, согласно которым был оптимизирован и улучшен код программы.

В функции подсчета всех ребер getNumOfEdges() был применен метод развертки (loop unrolling), уменьшивший количество итераций в несколько раз благодаря добавлению дополнительных переменных.

В функциях ввода и вывода класса Graph был применен метод объединения циклов (loop fusion), который позволил значительно сократить количество дополнительных расходов на циклы, независимые друг от друга по их данным.

В рекурсивной функции bronKerboschWP(), а также в функциях ввода и вывода и в main() применен метод циклического преобразования кода (loop-invariant code motion), с помощью которого одноразовые вычисления были вынесены за пределы цикла, благодаря чему перестали производиться множество раз.

В функциях delNode(), а также в функции вывода и в main() применен метод циклического размыкания (loop unswitching), с помощью которого часть условий была вынесена за пределы циклов.

Кроме применения известных методов по оптимизации кода по производительности были отдельно оптимизированы оба реализованных алгоритма Брона-Кербоша (с поворотом и без поворота). В рекурсивных функциях bronKerboschWOP() и bronKerboschWP() было уменьшено количество итераций. Во второй алгоритм был добавлен поиск вершины с максимальной степенью для еще большего уменьшения итераций в цикле, что позволило существенно сократить время работы алгоритма Брона-Кербоша с поворотом.

3.2 Оптимизация по потреблению памяти

В программе есть несколько ключевых мест, которые без оптимизации могут требовать большой объём памяти. Для их оптимизации список клик cliques передается по ссылке, таким образом переменная обновляется, добавляя в свой состав новую клику, а не создается каждый раз снова и снова. Также часть переменных используется несколько раз для различных задач ради экономии памяти. При операциях добавления новых элементов (вершины или ребра) осуществляется проверка на их наличие, чтобы в соответствующих множествах не было массового дублирования элементов.

Ниже представлена таблица с разными замерами времени с помощью дополнительных команд из библиотеки chrono.

Заключение

В процессе работы были изучен алгоритм поиска клик Брона-Кербоша и некоторые его популярные подвиды. Также были рассмотрены и приведены в сравнение несколько других алгоритмов поиска клик: MaxCliqueDyn, алгоритм Хансера и алгоритм простой итерации. Среди представленных алгоритмов Брон-Кербош был отмечен одним из самых эффективных. Был реализован класс невзвешенных и неориентированных графов. В самом классе присутствует множество функций, представленных в качестве инструментов для работы с данным классом. Также реализованы два рекурсивных подвида алгоритма Брона-Кербоша (с поворотом и без поворота). Была введена возможность создания графа со случайным количеством ребер при заданных вершинах. В классе графов присутствует возможность ввода и вывода элементов в консоль, а также хранение данных графа в файле и вывод его содержимого из файла. Присутствует понятный пользователю интерфейс, позволяющий проводить все доступные операции с имеющимися данными. Для повышения эффективности работы программы была проведена оптимизация кода по производительности и по потреблению памяти. В результате применения некоторых методов и анализа кода была успешно усовершенствована работа программы, в частности уменьшилось время выполнения больших алгоритмов.

Список литературы

1. Илларионов, Р. Е. Двухфазный алгоритм решения задачи о клике для разреженных графов большой размерности / Р. Е. Илларионов. – Москва : Изд-во Молодой ученый, 2016. - 256 с.
2. Окулов, С. М. Дискретная математика. Теория и практика решения задач по информатике / С. М. Окулов. – Москва : БИНОМ. Лаборатория знаний, 2012. – 422 с.
3. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – Москва : Вильямс, 2005. – 1296 с.
4. Донец, Г. А., Шор, Н. З. Алгебраический подход к проблеме раскраски плоских графов / Г. А. Донец, Н. З. Шор. – Киев : Наукова думка, 1982. – 144 с.