Министерство науки и высшего образования Российской Федерации  
Федеральное государственное образовательное учреждение высшего образования

«Национальный исследовательский технологический университет «МИСИС»

Институт компьютерных наук (ИКН)

Кафедра автоматизированных систем управления (АСУ)

**Курсовая работа**

по дисциплине «Алгоритмы дискретной математики»

На тему: "Алгоритм Брона-Кербоша"

Выполнил:  
студент группы БИВТ-22-СП-1

Евдокимов В.М.

Проверила:   
Валова А.А.

Москва, 2025

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc191816447)

[Актуальность 3](#_Toc191816448)

[Цели работы 3](#_Toc191816449)

[Задачи работы 3](#_Toc191816450)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ 4](#_Toc191816451)

[Пример работы алгоритмы 5](#_Toc191816452)

[Начало алгоритма 6](#_Toc191816453)

[Итог 9](#_Toc191816454)

[РЕАЛИЗАЦИЯ 11](#_Toc191816455)

[Структура программы 12](#_Toc191816456)

[Основная часть программы 15](#_Toc191816457)

[ЗАКЛЮЧЕНИЕ 17](#_Toc191816458)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc191816459)

[ПРОГРАММНЫЕ ПРИЛОЖЕНИЯ 19](#_Toc191816460)

# ВВЕДЕНИЕ

Алгоритм Брона — Кербоша представляет собой метод для поиска максимальных клик (а также максимальных по включению независимых множеств вершин) в неориентированных графах. Разработанный Клодом Броном и Хоацином Кербошем в 1973 году, этот алгоритм получил широкое применение в теории графов и компьютерной науке. Он является основным инструментом для решения множества задач, таких как социальные сети, биоинформатика и анализ сетевой безопасности, где необходимо выявлять полностью связанные подграфы.

## Актуальность

Современные приложения в различных областях науки и технологий требуют эффективных методов для анализа сложных сетевых структур. В частности, задача нахождения максимальных клик является критически важной в социальном анализе для выявления тесно связанных групп, в биоинформатике для поиска функциональных кластеров белков, и в сетевой безопасности для обнаружения потенциально уязвимых сегментов сети. Алгоритм Брона — Кербоша, благодаря своей эффективности и точности, остается одним из самых популярных методов для решения этих задач, что подчеркивает его актуальность в современных исследованиях и практических применениях.

## Цели работы

Целью данной курсовой работы является детальное изучение алгоритма Брона — Кербоша, его теоретических основ и практических аспектов применения. В рамках работы планируется рассмотреть структуру и основные шаги алгоритма, проанализировать его эффективность и ограничения, а также исследовать возможности оптимизации и модификации для улучшения производительности в различных сценариях применения.

## Задачи работы

Реализация алгоритма Брона — Кербоша

# МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Алгоритм Брона — Кербоша используется для нахождения всех максимальных клик в неориентированных графах. Клика K в графе G = (V,E) — это подмножество вершин V′ ⊆ V, такое что для любых u, v ∈ V′, (u, v) ∈ E. Максимальная клика — это клика, которая не может быть расширена за счет включения в нее еще одной вершины [1].

Основные этапы алгоритма можно описать следующими математическими компонентами и шагами [2]:

1. **Входные данные:**
   * G = (V, E): неориентированный граф, где V — множество вершин, E — множество рёбер.
   * R: множество вершин текущей клики (начально пустое).
   * P: множество кандидатов на добавление в клику (начально все вершины V).
   * X: множество вершин, уже рассмотренных и исключённых из множества кандидатов (начально пустое).
2. **Рекурсивная функция *bron\_kerbosch*(R, P, X):**
   * Если P и X пусты, то R является максимальной кликой, и её нужно сохранить.
   * Иначе, выбираем произвольную вершину v из P ∪ X.
   * Для каждой вершины u из P ∖ N(v) (где N(v) — множество соседей вершины v):
     + Вызываем рекурсивно *bron\_kerbosch*(R ∪ {u}, P ∩ N(u), X ∩ N(u)).
     + Перемещаем u из P в X.

Пример с кодом на Python:

def bron\_kerbosch(R, P, X, graph):  
 if not P and not X:  
 print("Maximal clique:", R)  
 return  
 for v in list(P):  
 newR = R.union([v])  
 newP = P.intersection(graph[v])  
 newX = X.intersection(graph[v])  
 bron\_kerbosch(newR, newP, newX, graph)  
 P.remove(v)  
 X.add(v)  
  
# Пример графа  
graph = {  
 1: {2, 3},  
 2: {1, 3},  
 3: {1, 2, 4},  
 4: {3}  
}  
  
# Начальные множества  
R = set()  
P = set(graph.keys())  
X = set()  
  
# Запуск алгоритма  
bron\_kerbosch(R, P, X, graph)

*Листинг 1 – Функция, реализующая алгоритм Брона-Кербоша*

## Пример работы алгоритмы

Рассмотрим граф с вершинами и рёбрами, описанными следующим образом:

* Вершины: V = {1, 2, 3, 4}
* Рёбра: E = {(1, 2), (1, 3), (2, 3), (3, 4)}. Граф можно визуализировать следующим образом:

1

/ \

2 - 3 - 4

Граф представлен в виде смежных списков:

graph = {  
 1: {2, 3},  
 2: {1, 3},  
 3: {1, 2, 4},  
 4: {3}  
}

*Листинг 2 – Представление графа в программном коде*

### Начало алгоритма

Алгоритм начинается с вызова *bron*\_*kerbosch*(R, P, X, graph) с пустым множеством R, множеством всех вершин P={1, 2, 3, 4} и пустым множеством X.

#### Шаг 1: Первое погружение в рекурсию

1. Вызов *bron\_kerbosch*(R = {}, P = {1, 2, 3, 4}, X = {}).
2. Выбираем вершину 1 из P.

**Переход к обработке вершины 1:**

* R = {1}
* P = {2, 3} (соседи вершины 1)
* X = ∅

#### Шаг 2: Второе погружение в рекурсию

1. Вызов *bron\_kerbosch*(R = {1}, P = {2, 3}, X = {}).
2. Выбираем вершину 2 из P.

**Переход к обработке вершины 2:**

* R = {1, 2}
* P = {3} (соседи вершины 2, которые также являются соседями 1)
* X = ∅

Шаг 3: Третье погружение в рекурсию

1. Вызов *bron\_kerbosch*(R = {1, 2}, P = {3}, X = {}).
2. Выбираем вершину 3 из P.

**Переход к обработке вершины 3:**

* R = {1, 2, 3}
* P = ∅ (соседи вершины 3, которые также являются соседями 1 и 2)
* X = ∅

Так как P и X пусты, R = {1, 2, 3} является максимальной кликой. Она выводится.

1. Возвращаемся к вызову *bron\_kerbosch*(R = {1, 2}, P = {3}, X = {}).
2. Перемещаем вершину 3 из P в X:

* P = ∅
* X = {3}

Так как P пусто, дальнейших рекурсивных вызовов не будет.

Возвращение ко второму уровню рекурсии

1. Возвращаемся к вызову *bron\_kerbosch*(R = {1}, P = {2, 3}, X = {}).
2. Перемещаем вершину 2 из P в X:

* P = {3}
* X = {2}

1. Выбираем следующую вершину из P. Пусть это будет вершина 3.
2. Переходим к обработке вершины 3:

* R = {1,3}
* P = ∅ (соседи вершины 3, которые также являются соседями 1)
* X = {2}

Так как P и X пусты, R = {1, 3} является максимальной кликой. Она выводится.

Возвращение к первому уровню рекурсии

1. Возвращаемся к вызову *bron\_kerbosch*(R = {}, P = {1, 2, 3, 4}, X = {}).
2. Перемещаем вершину 1 из P в X:

* P = {2,3,4}
* X = {1}

1. Выбираем следующую вершину из P. Пусть это будет вершина 2.
2. Переходим к обработке вершины 2:

* R = {2}
* P = {3} (соседи вершины 2)
* X = ∅

Второй уровень рекурсии (продолжение)

1. Вызов *bron\_kerbosch*(R = {2}, P = {3}, X = {}).
2. Выбираем вершину из P. Пусть это будет вершина 3.
3. Переходим к обработке вершины 3:

* R = {2, 3}
* P = ∅ (соседи вершины 3, которые также являются соседями 2)
* X = ∅

Так как P и X пусты, R = {2,3} является максимальной кликой. Она выводится.

1. Возвращаемся к вызову *bron\_kerbosch*(R = {2}, P = {3}, X = {}).
2. Перемещаем вершину 3 из P в X:

* P = ∅
* X = {3}

Так как P пусто, дальнейших рекурсивных вызовов не будет.

Возвращение к первому уровню рекурсии (продолжение)

1. Возвращаемся к вызову *bron\_kerbosch*(R = {}, P = {2, 3, 4}, X = {1}).
2. Перемещаем вершину 2 из P в X:

* P = {3,4}
* X = {1,2}

1. Выбираем следующую вершину из P. Пусть это будет вершина 3.
2. Переходим к обработке вершины 3:

* R = {3}
* P = {4} (соседи вершины 3)
* X = {1,2}

Второй уровень рекурсии (продолжение)

1. Вызов *bron\_kerbosch*(R = {3}, P = {4}, X = {1, 2}).
2. Выбираем вершину из P. Пусть это будет вершина 4.
3. Переходим к обработке вершины 4:

* R = {3,4}
* P = ∅ (соседи вершины 4, которые также являются соседями 3)
* X = ∅

Так как P и X пусты, R = {3, 4} является максимальной кликой. Она выводится.

1. Возвращаемся к вызову *bron\_kerbosch*(R = {3}, P = {4}, X = {1, 2}).
2. Перемещаем вершину 4 из P в X:

* P = ∅
* X = {1, 2, 4}

Так как P пусто, дальнейших рекурсивных вызовов не будет.

Возвращение к первому уровню рекурсии (завершение)

1. Возвращаемся к вызову *bron\_kerbosch*(R = {}, P = {3, 4}, X = {1, 2}).
2. Перемещаем вершину 3 из P в X:

* P = {4}
* X = {1, 2, 3}

1. Выбираем следующую вершину из P. Пусть это будет вершина 4.
2. Переходим к обработке вершины 4:

* R = {4}
* P = ∅ (соседи вершины 4)
* X = {3}

Так как P и X пусты, R = {4} является максимальной кликой. Она выводится.

## Итог

Алгоритм завершает работу, найдя все максимальные клики в графе:

* {1, 2, 3}
* {3, 4}

Изображение выглядит как текст, диаграмма, число, Шрифт

Автоматически созданное описаниеРисунок 1 – Пример алгоритма с графом

Изображение выглядит как круг, снимок экрана, бильярдный шар

Автоматически созданное описание

Рисунок 2 – Граф алгоритма

# РЕАЛИЗАЦИЯ

import networkx as nx  
import matplotlib.pyplot as plt  
  
  
def bron\_kerbosch(R, P, X, graph, cliques):  
 *"""  
 Алгоритм Брона-Кербоша для поиска максимальных клик.  
  
 Параметры:  
 R - текущее множество вершин в клике  
 P - кандидаты на расширение клики  
 X - уже обработанные вершины, которые не могут быть в текущей клике  
 graph - граф (объект NetworkX)  
 cliques - список всех найденных максимальных клик  
 """* if not P and not X:  
 cliques.append(R)  
 return  
 for vertex in P.copy():  
 bron\_kerbosch(R.union({vertex}),  
 P.intersection(graph[vertex]),  
 X.intersection(graph[vertex]),  
 graph, cliques)  
 P.remove(vertex)  
 X.add(vertex)  
  
  
def find\_maximal\_cliques(graph):  
 *"""  
 Нахождение всех максимальных клик в графе.  
  
 Параметры:  
 graph - граф (объект NetworkX)  
  
 Возвращает:  
 Список всех максимальных клик  
 """* cliques = []  
 P = set(graph.nodes())  
 bron\_kerbosch(set(), P, set(), graph, cliques)  
 return cliques  
def visualize\_graph(graph, cliques):  
 *"""  
 Визуализация графа и его максимальных клик.  
  
 Параметры:  
 graph - граф (объект NetworkX)  
 cliques - список максимальных клик  
 """* pos = nx.spring\_layout(graph)  
 plt.figure(figsize=(8, 6))  
  
 # Рисуем граф  
 nx.draw(graph, pos, with\_labels=True, node\_color='lightblue', node\_size=3000, font\_size=15, font\_weight='bold')  
  
 # Раскраска клик  
 colors = ['red', 'green', 'blue', 'orange', 'purple']  
 for i, clique in enumerate(cliques):  
 nx.draw\_networkx\_nodes(graph, pos, nodelist=clique, node\_color=colors[i % len(colors)], node\_size=3000,  
 alpha=0.6)  
  
 plt.title("Граф и его максимальные клики")  
 plt.show()  
  
  
# Создание графа  
G = nx.Graph()  
edges = [(1, 2), (1, 3), (2, 3), (3, 4)]  
G.add\_edges\_from(edges)  
  
# Нахождение максимальных клик  
maximal\_cliques = find\_maximal\_cliques(G)  
print("Максимальные клики:", maximal\_cliques)  
  
# Визуализация графа и клик  
visualize\_graph(G, maximal\_cliques)

*Листинг 3 – Весь программный код реализации алгоритма Брона-Кербоша*

## Структура программы

Программа состоит из нескольких функций [3]:

1. *bron\_kerbosch* - рекурсивная функция, реализующая алгоритм Брона-Кербоша.
2. *find\_maximal\_cliques* - функция, инициализирующая процесс поиска максимальных клик.
3. *visualize\_graph* - функция для визуализации графа и найденных клик.

#### **Импорт библиотек**

В начале программы импортируются необходимые библиотеки:

import networkx as nx  
import matplotlib.pyplot as plt

networkx используется для создания и манипулирования графами, а matplotlib.pyplot - для визуализации графов.

Функция *bron\_kerbosch* реализует сам алгоритм поиска максимальных клик.

def bron\_kerbosch(R, P, X, graph, cliques):  
 if not P and not X:  
 cliques.append(R)  
 return  
 for vertex in P.copy():  
 bron\_kerbosch(R.union({vertex}),  
 P.intersection(graph[vertex]),  
 X.intersection(graph[vertex]),  
 graph, cliques)  
 P.remove(vertex)  
 X.add(vertex)

*Листинг 4 – Функция bron\_kerbosch(…)*

* R - текущее множество вершин в клике.
* P - кандидаты на расширение клики.
* X - уже обработанные вершины, которые не могут быть в текущей клике.
* graph - объект графа (NetworkX).
* cliques - список всех найденных максимальных клик.

Функция сначала проверяет, пусты ли множества P и X. Если да, то R является максимальной кликой, и она добавляется в список cliques. Затем, для каждого вершины из P, функция рекурсивно вызывается для обновленных множеств R, P и X. После рекурсивного вызова вершина удаляется из P и добавляется в X [4].

#### Функция find\_maximal\_cliques

Эта функция инициализирует поиск максимальных клик.

def find\_maximal\_cliques(graph):  
 cliques = []  
 P = set(graph.nodes())  
 bron\_kerbosch(set(), P, set(), graph, cliques)  
 return cliques

*Листинг 5 – Функция инициализации поиска максимальных клик*

* graph - объект графа (NetworkX).

Функция создает пустой список для хранения клик и множество всех вершин графа P. Затем она вызывает функцию bron\_kerbosch с начальными пустыми множествами R и X и множеством вершин P. Возвращается список всех максимальных клик.

#### Функция visualize\_graph

Эта функция визуализирует граф и найденные максимальные клики.

def visualize\_graph(graph, cliques):  
 pos = nx.spring\_layout(graph)  
 plt.figure(figsize=(8, 6))  
  
 # Рисуем граф  
 nx.draw(graph, pos, with\_labels=True, node\_color='lightblue', node\_size=3000, font\_size=15, font\_weight='bold')  
  
 # Раскраска клик  
 colors = ['red', 'green', 'blue', 'orange', 'purple']  
 for i, clique in enumerate(cliques):  
 nx.draw\_networkx\_nodes(graph, pos, nodelist=clique, node\_color=colors[i % len(colors)], node\_size=3000,  
 alpha=0.6)  
  
 plt.title("Граф и его максимальные клики")  
 plt.show()

*Листинг 6 – Функция визуализации графа и максимальных клик*

* graph - объект графа (NetworkX).
* cliques - список максимальных клик.

Функция создает расположение вершин графа с помощью spring\_layout. Затем создает графическое окно и рисует весь граф с узлами и ребрами. После этого каждую клику визуализирует разным цветом. Наконец, отображает граф с помощью plt.show().

## Основная часть программы

В основной части программы создается граф, ищутся максимальные клики и визуализируется результат [5].

# Создание графа

try:  
 G = nx.Graph()  
 edges = [(1, 2), (1, 3), (2, 3), (3, 4)]  
 G.add\_edges\_from(edges)  
 print("Граф создан успешно")  
  
 # Нахождение максимальных клик  
 maximal\_cliques = find\_maximal\_cliques(G)  
 print("Максимальные клики:", maximal\_cliques)

# Визуализация графа и клик  
 visualize\_graph(G, maximal\_cliques)  
except Exception as e:  
 print(f"Ошибка: {e}")

*Листинг 7 – Создание графа*

1. Создается объект графа G.
2. Добавляются ребра в граф.
3. Выводится сообщение об успешном создании графа.
4. Вызывается функция find\_maximal\_cliques для поиска максимальных клик.
5. Выводится список найденных клик.
6. Вызывается функция visualize\_graph для визуализации графа и клик.
7. В случае ошибки выводится сообщение об ошибке.

Изображение выглядит как снимок экрана, текст, линия, круг

Автоматически созданное описание

Рисунок 3 – Визуализация графа

# ЗАКЛЮЧЕНИЕ

В данной документации описана реализация алгоритма Брона-Кербоша для поиска максимальных клик в графе на языке Python с использованием библиотек NetworkX и Matplotlib. Программа включает создание графа, поиск клик и их визуализацию. Такой подход позволяет наглядно представить результаты работы алгоритма и может быть полезен для дальнейшего изучения и анализа структуры графов.

# СПИСОК ЛИТЕРАТУРЫ

[1] Bron-Kerbosch Algorithm // Wikipedia. – URL: https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch\_algorithm (дата обращения: 01.03.2025).

[2] Алгоритм Брона — Кербоша // Wikipedia. – URL: https://ru.wikipedia.org/wiki/Алгоритм\_Брона\_—\_Кербоша (дата обращения: 01.03.2025).

[3] Графы и пути: алгоритм Брона-Кербоша – максимальные клики // Medium. – URL: https://medium.com/nuances-of-programming/%D0%B3%D1%80%D0%B0%D1%84%D1%8B-%D0%B8-%D0%BF%D1%83%D1%82%D0%B8-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-%D0%B1%D1%80%D0%BE%D0%BD%D0%B0-%D0%BA%D0%B5%D1%80%D0%B1%D0%BE%D1%88%D0%B0-%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5-%D0%BA%D0%BB%D0%B8%D0%BA%D0%B8-857d7b7e829d (дата обращения: 01.03.2025).

[4] Bron-Kerbosch Algorithm // OpenGenus. – URL: https://iq.opengenus.org/bron-kerbosch-algorithm/ (дата обращения: 01.03.2025).

[5] Bron-Kerbosch Algorithm – Видеообзор // YouTube. – URL: https://www.youtube.com/watch?v=132XR-RLNoY (дата обращения: 01.03.2025).

# ПРОГРАММНЫЕ ПРИЛОЖЕНИЯ

[1] Replit  – облачная среда разработки Python // Replit.com. – URL: https://replit.com/@pdasha598/kurs#main.py (дата обращения: 01.03.2025).

[2] Python 3.10 – интерпретатор языка программирования Python / Разработчик: Python Software Foundation. – URL: https://www.python.org/ (дата обращения: 01.03.2025).

[3] PyCharm Community Edition 2023.2 – интегрированная среда разработки Python / Разработчик: JetBrains. – URL: https://www.jetbrains.com/pycharm/ (дата обращения: 01.03.2025).